Санкт-Петербургский Государственный Университет
Математико-механический факультет

Кафедра системного программирования

Определение расстояния между точкой и множеством, представленным бинарной диаграммой решений

Курсовая работа студента 345 группы
Зубаревича Дмитрия Александровича

Научный руководитель …… Д.Ю. Бугайченко

Санкт-Петербург
2012

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 конструирующие подходы ведут к сильному усложнению алгоритма обработки данных.

Альтернативой к указанным выше подходам является представление с помощью бинарных диаграмм решений (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. 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 [8] (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 должен хранить 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.

Последний пример поясняет целесообразность темы данной работы. Её целью является разработка алгоритма нахождения расстояния от точки до множества, представленного БДР, программная реализация этого алгоритма в виде библиотеки, предоставляющей интерфейс для работы с искусственными нейронами, описанными в примере, а также тестирование программной реализации на простейшем примере распознавания печатных цифр, однослойной ИНС, составленной из нейронов, реализованных на основе БДР.

Постановка задачи

Задачей данной курсовой работы является:

- Разработка алгоритма нахождения расстояния от точки до множества, представленного бинарной диаграммой решений;
- Реализация алгоритма на основе библиотеки *BDDFunctions* [1], предоставляющей API для работы с БДР;
- Тестирование алгоритма на примере распознавания печатных цифр.

Краткие сведения о бинарных диаграммах решений

Формально, упорядоченная БДР функции вида $f : \{0,1\}^n \to S$ есть ориентированный корневой ациклический граф с множеством вершин $V = T \cup N$, $T \cap N = \emptyset$. Вершины множества N называются *нетерминилами*, и для 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 i соответствуют аргументам определяемой функции x_i. Вершины множества T называются *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 $f_v : \{0,1\}^n \to S$:

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

Существует несколько разновидностей решающих диаграмм. В задачах связанных с использованием булевых функций вида $f : \{0,1\}^n \to \{0,1\}$ и конечные множества широкое распространение получили бинарные решающие диаграммы (BDD) [2]. В задачах, связанных с использованием функций вида $f : \{0,1\}^n \to S$, 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 1 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

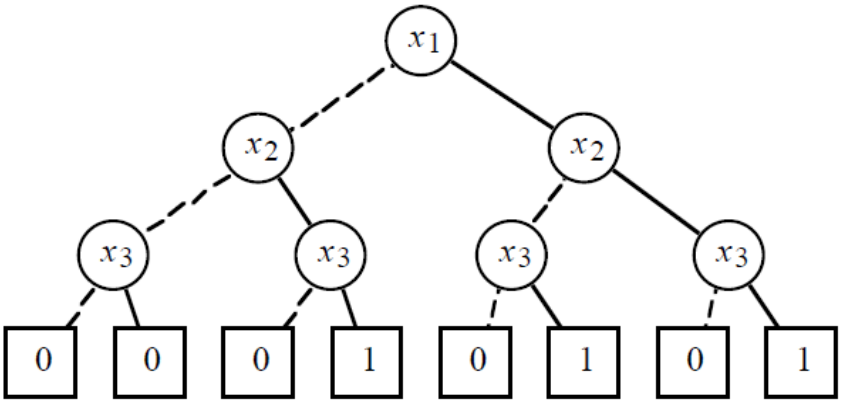

Рис. 1: Табличное задание функции (слева) и бинарное дерево решений (справа).

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 u и v 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:

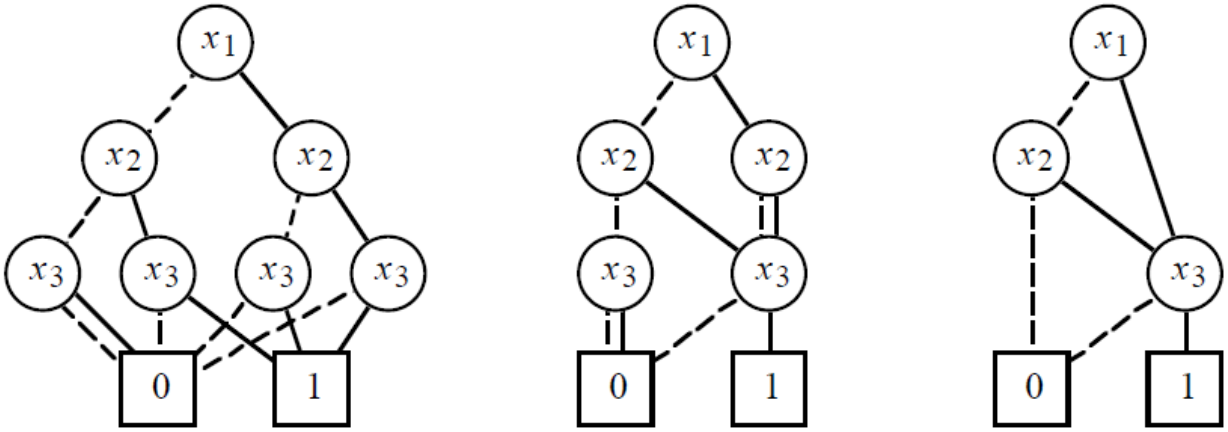

Рис. 2: БДР после применения первого (слева), второго (в центре) и третьего (справа) правил редукции.

На рисунке 2 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.

Алгоритм определения расстояния от точки до множества

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.

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 x индексирует 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 вектора A на матрицу B, так как количество строк матрицы B равно количеству столбцов вектора A. Проводя умножение, получим $(\chi_S \times \rho) = A * B = C \in R^{1 \times |T|}$ – матрица, C_{1y} элемент которой будет 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 $\delta_{\rho,S}$.

Вернёмся к примеру, описанному в ведении. Там была сформулирована идея использования БДР для создания искусственного нейрона с явным представлением данных. Нейрон в таком случае состоял бы из множества, хранящего шаблоны, и метрики, определяющей расстояние до этого множества. Пусть ИНС с такими нейронами используется для распознавания графических образов, к примеру, 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.

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, имеющие готовую реализацию в библиотеке *BDDFunctions*, такие как сложение, умножение, композиция функций и некоторые другие;
- Операции, через которые выражается функция ρ, должны быть наименее трудоемкими, например, сложение менее трудоемко, чем умножение, возведение в квадрат менее трудоемко, чем умножение и т.д.

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 Хэмминга ρ_H. Чтобы определить его, введём функцию $bin: T \to \{0,1\}^m$, которая переводит точку из множества T в 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 -ого бита i-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 $\beta(j,i,x,y) = \begin{cases} 0, \alpha(j,i,x) = \alpha(j,i,y) \\ 1, \alpha(j,i,x) \neq \alpha(j,i,y) \end{cases}$.

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 [3], для снижения размерности множества, представляющего образы, что является весьма важным аспектом при применении БДР, так как БДР является очень требовательными к затратам по памяти.

Программная реализация алгоритма

Программная реализация общего подхода к нахождению расстояния от точки до множества, представленного БДР, строится, исходя из наиболее перспективного применения этого подхода. Исходный код программы оформлен в виде библиотеки *BddNeuron*. В её основе лежит класс *Neuron*, описывающий искусственный нейрон. Он содержит множество “хороших” шаблонов (*обучающее множество* в терминах ИНС), представленное классом *Pattern*, и функцию, позволяющую находить расстояние от тестовой точки до этого множества, представленную классом *FunctionOfDistance*. Ниже представлена диаграмма классов проекта:

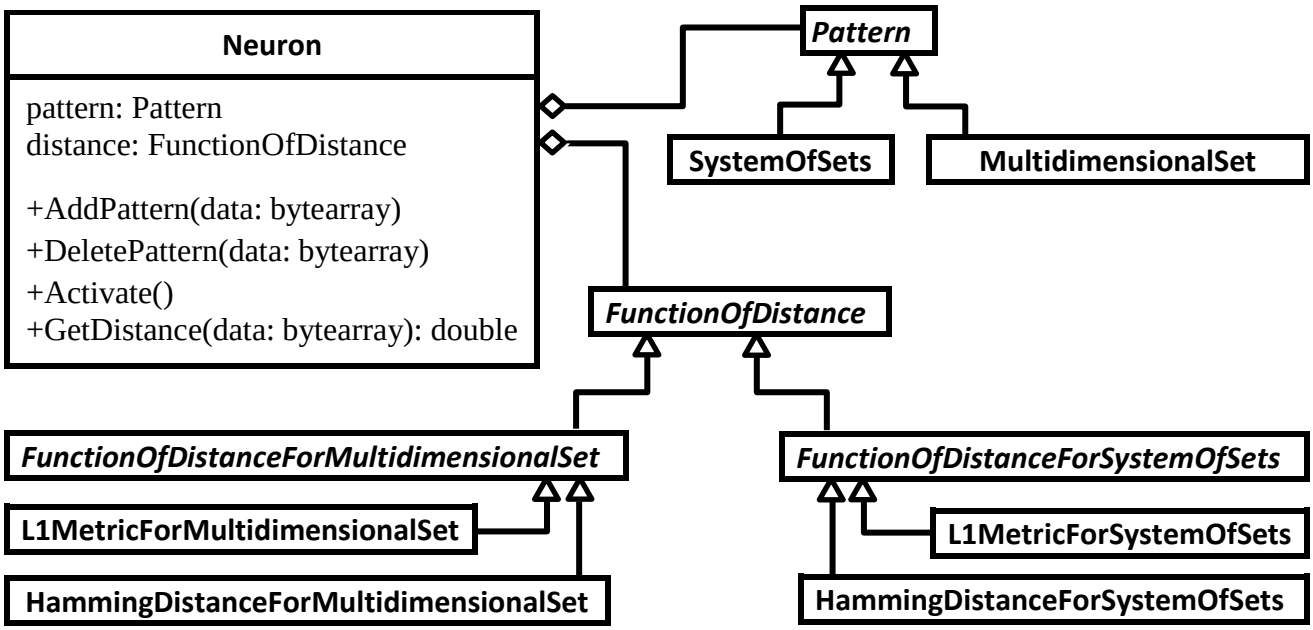

Диаграмма классов библиотеки BddNeuron.

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 n одномерных независимых множеств Set, где n – размерность исходного множества;
- *MultidimensionalSet* – хранит множество в виде 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*.

Имея эти базовые классы, несложно реализовать класс конкретной функции, определяющей расстояние от точки до множества. Для этого нужно в классе-наследнике реализовать абстрактный метод, стоящий функцию, вычисляющую расстояние между двумя точками. Таким способом были реализованы классы, позволяющие работать с L_1- метрикой и расстоянием Хэмминга. Это легко можно видеть на диаграмме классов библиотеки.

Таким образом, класс, представляющий искусственный нейрон, зависит от представления множества шаблонов и от конкретной функции вычисления расстояния от точки до множества, поэтому он реализован как шаблонный класс. При создании конкретного нейрона его конструктору передаётся вектор, содержащий размеры компонент элементов множества, указанные в битах. При этом размер одной компоненты должен занимать от одного до четырёх байт, в зависимости от того, сколько занимает кодировка конкретной компоненты элемента. В случае, если в одной компоненте требуется разместить более четырёх байт, необходимо разбить её на несколько компонент. В случае же необходимости размещения в одной компоненте менее одного байта, необходимо недостающие биты дополнить нулями.

Организация работы с нейроном можно разбить на несколько этапов. Опишем подробно каждый из них.

Инициализация – при создании нейрона происходит построение функции ρ, вычисляющей расстояние между тестовой точкой и точкой множества, и соответствующей ей матрицы (см. предыдущий раздел). Скорость выполнения построения нейрона зависит от трудоёмкости построения функции ρ.

Обучение – после создания нейрона к нему добавляется поэлементно обучающее множество. На этом этапе при добавлении нового шаблона кроме расширения множества шаблонов, хранящегося в нейроне, ничего не происходит, так что этот процесс происходит сравнительно быстро.

Активация – после того как всё обучающее множество находится в нейроне, необходимо выполнить его активацию, вызовом соответствующего метода. При этом происходит первоначальное построение функции, вычисляющей расстояние до множества шаблонов, заключающееся в построении матрицы, соответствующей функции χ_S и умножении её на матрицу, построенную на этапе инициализации. Естественно этот процесс занимает достаточно длительное время по сравнению с построением обучающего множества, так как операция матричного умножения очень трудоёмкая. После активации нейрон формально готов к использованию.

Корректировка – может понадобиться в процессе работы нейрона, если выяснится, что обучающее множество требует исправления. Последнее можно произвести, добавив или удалив необходимую точку из множества шаблонов. Заметим, что это повлечёт за собой перестройку функции, вычисляющей расстояние, а значит, займёт примерно такое же время, как и сама активация. Это означает, что корректировка нейрона после его активации – операция допустимая, но дорогостоящая.

Тестирование – включает в себя прогонку через нейрон тестового множества в терминах ИНС (обычно тестирование удобнее проводить для ИНС в целом, а не для одного конкретного нейрона). Получая результаты тестирования можно понять, что обучающее множество нейрона нуждается в корректировке и произвести её. Этот этап необходим для снижения вероятности возникновения необходимости проводить корректировку нейрона при его использовании на практике, поэтому после тестирования считается, что нейрон полностью готов к работе.

Использование – подразумевает под собой, что большинство запросов к нейрону будут направлены на вычисление расстояния, хотя, как и на этапе тестирования, в случае необходимости можно провести корректировку обучающего множества.

В заключение данного раздела, проиллюстрируем работу с нейроном на простом примере:

```
//создание вектора с размерами компонент элементов множества
vector<size_t> sizes;
//добавление размера (32 бита) одной компоненты
sizes.push_back(32);
//Создание нейрона с системой множеств для хранения шаблонов
//и L1-метрикой для вычисления расстояния
Neuron<SystemOfSets, L1MetricForSystemOfSets> neuron =
        new Neuron<SystemOfSets, L1MetricForSystemOfSets>(sizes);
//бинарная кодировка трёх точек
bytearray val1 = TypedToBinary<int>(10);
bytearray val2 = TypedToBinary<int>(20);
bytearray val3 = TypedToBinary<int>(0);
//Добавляем две точки в обучающее множество
neuron.AddPattern(val1);
neuron.AddPattern(val2);
//Активируем нейрон
neuron.Activate();
//Получаем расстояние от третьей точки до обучающего множества
//В этом случае оно будет равно 15
double distance = neuron.GetDistance(val3);
//Корректируем обучающее множество
neuron.AddPattern(val3);
//Получаем расстояние от третьей точки до обучающего множества
//В этом случае оно будет равно 0
distance = neuron.GetDistance(val3);
```

Тестирование алгоритма

В качестве тестирования программной реализации была выбрана задача распознавания образов печатных цифр. Для её решения была реализована простейшая однослойная ИНС состоящая только из нейронов, применяющих БДР. Для распознавания одного символа выделен один нейрон. Поэтому для распознавания цифр требуется десять нейронов. Работа с ИНС разбита на три этапа:

- Обучение – добавление шаблонных точек в соответствующие множества;
- Активация – активация всех нейронов сети, т.е. построение функций, находящих расстояние от тестовых точек до множеств шаблонов;
- Тестирование – нахождение минимального расстояния среди всех нейронов от тестовой точки до шаблонного множества нейрона.

В качестве кодировки изображений была предпринята попытка использовать перцептивный хэш [3], для уменьшения размерности множеств шаблонов и, соответственно, ресурсоёмкости ИНС. Коротко идея такого хэширования заключается в следующем:

- Берём чёрно-белое изображение цифры и масштабируем его до размеров 8×8 пикселей;
- Если пиксель белый, сопоставляем ему ноль, иначе единицу, тем самым получаем 64 бита;
- Переводим полученные биты в два числа типа unsigned int.

При таком подходе элемент множества (хэш) будет состоять из двух компонент по 4 байта. Что вполне приемлемо для БДР.

При таком подходе, как уже говорилось в разделе, описывающем разработку алгоритма, нахождения расстояния от точки до множества, в качестве функции, определяющей расстояние между двумя образами, хорошо подходит расстояние Хэмминга, которое работает с хэшированным представлением образов.

К сожалению, на практике масштабирование изображения до размеров 8×8 не оправдало себя, показав не самые хорошие результаты распознавания.

Чтобы улучшить результаты, было принято решение масштабировать образы символов до размера 16×16. Тогда результирующий хэш можно было бы разместить в восьми числах типа unsigned int, однако, так как в первых двух и последних двух битовых строках изображения всегда получались нули, хэш можно разместить в шести числах типа unsigned int.

Для того, чтобы проще проводить обучение и тестирование, было решено применить язык C#. Это позволило проводить наглядную генерацию графических образов цифр, просто вычислять их хэшированное представление и проводить генерацию C++ кода, для методов, реализующих обучение и тестирование ИНС.

При обучении ИНС на хэшированных, описанным выше образом, изображениях цифр, отрисованных шрифтами Bookman “Old Style”, “Century”, “Cambria” и “Calibri” успешно произошло распознавание цифр, отрисованных шрифтом “Times New Roman”.

Кроме того, на этом примере тестирования библиотеки *BddNeuron* были проведены замеры времени, затраченного на выполнение основных этапов работы нейронов на процессоре с тактовой частотой 1,8 ГГц:

- Инициализация – построение функции нахождения расстояния между двумя точками и построение соответствующей ей матрицы занимает около 300мс.
- Обучение – построение обучающего множества из четырёх элементов, каждый из которых кодируется шестью байтами, занимает в среднем порядка 18 мс.
- Активация – построение функции поиска расстояния от точки до обучающего множества занимает 10-15 с.
- Тестирование – нахождение расстояния от множества до одной тестовой точке занимает около 6мс.

При тестировании использовалось представление множества шаблонов в виде системы множеств и функция нахождения расстояния от точки до множества, построенная на основе расстояния Хэмминга.

Использованные технологии

Библиотека *BddNeuron* реализована на основе библиотеки *BddFunctions*, предоставляющей гибкий объектно-ориентированный C++ интерфейс для работы с функциями, множествами и матрицами, представленными БДР. В свою очередь библиотека *BDDFunctions* основана на *пакете решающих диаграмм – Colorado University Decision Diagram Package* (CUDD) [4], 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 GNU C++ с использованием компилятора MinGW и интегрированной среды разработки Code::Blocks [5].

Для покрытия исходного кода тестами использована библиотека с открытым исходным кодом *Google C++ Testing Framework* (GTest) [6]. GTest позволяет легко проводить модульное тестирование и обладает дружественным интерфейсом.

Генерация тестового и обучающего множеств для тестового примера, т.е. генерация изображений цифр и получение их хэшей, реализована посредством языка C#, с использованием среды разработки Microsoft Visual Studio 2010 [7]. Для прозрачности процесса, генератор разработан с использованием технологии Windows Forms.

Итоги

В ходе проделанной работы достигнуты следующие результаты:

- Разработан общий подход к построению алгоритма нахождения расстояния от точки до множества, представленного БДР, позволяющий определять расстояние до множества, как среднее арифметическое расстояний от точки до точек множества, которые вычисляются с помощью произвольно заданной метрики. Кроме того были рассмотрены возможности построения БДР для двух метрик: L_1- метрики и расстояния Хэмминга;
- Изучена библиотека *BDDFunctions*, предоставляющая высокоуровневый API для работы с БДР.
- Разработанный подход реализован на языке GNU C++ с использованием компилятора MinGW и интегрированной среды разработки Code::Blocks на основе библиотеки *BDDFunctions*. Исходный код оформлен в виде библиотеки *BddNeuron*, позволяющей строить искусственные нейроны, на основе БДР.
- Реализованы два подхода к представлению многомерных множеств шаблонов с помощью БДР: посредствам системы одномерных множеств и посредствам многомерного множества.
- Библиотека *BddNeuron* успешно апробирована на примере распознавания образов цифр, отрисованных различными шрифтами.

Дальнейшие планы

Перечислим некоторые направления в дальнейшей работе по данной теме:

- Получение большего числа экспериментальных данных по применению искусственных нейронов, в основе которых лежит БДР;
- Разработка программной реализации ИНС, позволяющей строить, многослойные сети с построенными в данной работе нейронами, а также комбинированные сети, состоящие как из нейронов нового типа, так и из классических нейронов;
- Разработка алгоритмов обучения для указанных выше типов ИНС, и их соответствующее применение на примерах распознавания образов;
- Сравнение ИНС нового типа с классическими;
- Распараллеливание библиотеки *BDDFunctions*, с целью повышения скорости её работы;
- Исследование возможности применения методов, которые позволили бы понижать размерность множества шаблонов (в частности, перцептивного хэширования) в задачах распознавания, с целью снижения трудоемкости при применении БДР.

Таким образом, работа в данном направлении подразумевает решение достаточно большого количества сложных и интересных задач, которые, в конечном счёте, могут привести к достаточно хорошим результатам.

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

[3] P. 293–318.Ализар А. “Выглядит похоже”. Как работает перцептивный хэш. URL: http://habrahabr.ru/post/120562/

[4] Somenzi F. CUDD: Colorado University Decision Diagram Package. URL: http://vlsi.colorado.edu/ fabio/CUDD.

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

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

[7] Microsoft Visual Studio 2010. URL: http://www.microsoft.com/visualstudio/ru-ru.

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