Национальный технический университет Украины

"Киевский политехнический институт"

Физико-технический факультет

**Контрольная работа № 1**

По дисциплине "Интеллектуальные системы"

**по теме: "**Нейронная сеть Хемминга**"**

Киев-2008

Содержание

1. Реализация сети. Входные символы

2. Обучение сети

3. Локализация и масштабирование изображения

4 Искажение изображения

5. Алгоритм распознавания

**Цель: Создать программную реализацию статической нейронной сети Хемминга, распознающей символы текста.**

**Описание реализации алгоритма.**

## 1. Реализация сети. Входные символы

Для определения общих параметров НС - несколько констант:

const int iSourceSize=100; // Размер стороны исходного изображения const int iDestSize=40; // Размер стороны изображения для распознавания const int N=iDestSize\*iDestSize; // Сколько входов const int M=10; // Сколько образов const float e=- (1/M) /2; // Вес синапсов второго слоя

Размер входного образа (bmp-файл) должен быть 100х100 точек (параметр iSourceSize), а перед обработкой он приводится к размеру 40x40 (параметр iDestSize). Таким образом, количество входов сети - 1600 (один вход - одна точка изображения; параметр N). Количество выходов M совпадает с количеством цифр и равняется 10. Вес отрицательной обратной связи второго слоя был принят равным - 0.05.

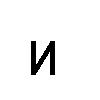


Рис 1 Пример исходного образа

Каждый нейрон представляется в виде простой структуры. Слой - это просто массив нейронов. Первый слой:

struct InputNeiron // Тип - нейрон первого слоя { float fW [N] ; // Весовые коэффициенты синапсов float fOutput; // Выход };

InputNeiron InputRow [M] ; // Это - первый слой

Второй слой реализуется так:

struct Neiron // Тип - нейрон второго слоя { float fOutput; // Выход float fSum; // Взвешенная сумма входов };

Neiron SecondRow [M] ; // Это - второй слой

Здесь уже нет смысла хранить значения отдельных весов, т.к они не меняются на протяжении всей работы сети.

## 2. Обучение сети

После процедуры обучения списки весовых коэффициентов синапсов первого слоя будут содержать образы эталонных символов.

**Алгоритм обучения сети Хэмминга, адаптированный для данной задачи**, выглядит так:

1. Выбирается i-й входной образ.

2. Изображение локализуется и приводится к нужному масштабу.

3. Образ поточечно подаётся на входы i-го нейрона. Если k-я точка образа чёрная, то весу k-го входа присваивается значение 0.5, в противном случае - 0.5 (см. формулу 5 п.3.2.5).

4. Переход на шаг 1, пока не исчерпаны все эталонные образы.

Исходный код алгоритма обучения выглядит так:

TImage\* I1=new TImage (this); // Сюда загружаем эталонный образ TImage\* I2=new TImage (this); // А здесь будет масштабированный I2->Width=iDestSize;

I2->Height=iDestSize;

for (int i=0; i<M; i++) { I1->Picture->LoadFromFile (AnsiString (i) +". bmp");

vScale (I2, I1); // Читаем и масштабируем образ for (int x=0; x<iDestSize; x++) for (int y=0; y<iDestSize; y++) // Перебираем поточечно if (I2->Canvas->Pixels [x] [y] ==clBlack) InputRow [i]. fW [x\*iDestSize+y] =0.5;

else InputRow [i]. fW [x\*iDestSize+y] =-0.5;

} delete I1;

delete I2;

## 3. Локализация и масштабирование изображения

Для успешной работы нужно выяснить точный размер и местоположение образа цифры. После локализации - приводим образ к размеру 40\*40 (масштабирование). Эти операции выполняются как на этапе обучения сети, так и на этапе распознавания. Всё это проделывает функция void vScale (TImage\*, TImage\*).

Для эталона, ввиду отсутствия помех, локализацию провести очень просто: достаточно последовательно просмотреть все точки образа и найти границы цифры:

float iMinX, iMinY, iMaxX, iMaxY; // Границы локализованной цифры iMinX=iMinY=iSourceSize+1;

iMaxX=iMaxY=-1;

for (int x=0; x<iSourceSize; x++) for (int y=0; y<iSourceSize; y++) if (Source->Canvas->Pixels [x] [y] ==clBlack) { if (x<iMinX) iMinX=x;

if (y<iMinY) iMinY=y; if (x>iMaxX) iMaxX=x;

if (y>iMaxY) iMaxY=y;

}

В данной реализации используется стандартный алгоритм обратного масштабирования без интерполяции (Source - исходный образ, Dest - отмасштабированный):

const float fScaleX= (iMaxX-iMinX) /iDestSize; // Коэффициенты сжатия const float fScaleY= (iMaxY-iMinY) /iDestSize;

for (int x=0; x<iDestSize; x++) for (int y=0; y<=iDestSize; y++) Dest->Canvas->Pixels [x] [y] = Source->Canvas->Pixels [x\*fScaleX+iMinX] [y\*fScaleY+iMinY] ;

## 4 Искажение изображения

Чтобы не распознавать эталонные образы - введем искажение. Воспользуемся двумя способомами искажения:

Добавление N-процентного шума.

for (int x=0; x<IBroke2->Width; x++)

for (int y=0; y<IBroke2->Height; y++)

// Добавляем белый X% -й шум

if (random (100) <iPercent) // iPercent - это процент шума, кот // пользователь может указать через интерфейс диалогового окна

if (IBroke2->Canvas->Pixels [x] [y] ==clBlack)

IBroke2->Canvas->Pixels [x] [y] =clWhite;

else

IBroke2->Canvas->Pixels [x] [y] =clBlack;

## 5. Алгоритм распознавания

Общий алгоритм распознавания для сети Хэмминга состоит из четырёх частей:

1. подача распознаваемого образа на входы сети;

2. передача данных с первого слоя на второй;

3. обработка данных вторым слоем;

4. выбор распознанного образа.

**1. Алгоритм работы первого этапа выглядит так**:

1. Выбирается очередной нейрон;

2. Обнуляется его выход;

3. Локализуется и приводится изображение к нужному масштабу (см. п 3.3.3);

4. Локализованный образ поточечно подаётся на входы i-го нейрона. Если k-я точка образа чёрная, то к значению выхода прибавляется значение веса k-го входа, в противном случае это значение вычитается;

5. Значение выхода пропускается через функцию линейного порога;

6. Переход на шаг 1, пока не исчерпаны все нейроны первого слоя.

Исходный код:

for (int i=0; i<M; i++) { InputRow [i]. fOutput=0;

// Подаём образ на нейроны первого слоя for (int x=0; x<iDestSize; x++) for (int y=0; y<iDestSize; y++) if (Source->Canvas->Pixels [x] [y] ==clBlack) InputRow [i]. fOutput+=InputRow [i]. fW [x\*iDestSize+y] ;

else InputRow [i]. fOutput+=-InputRow [i]. fW [x\*iDestSize+y] ;

// Выход - через функцию линейного порога if (InputRow [i]. fOutput>=N/2) InputRow [i]. fOutput=N/2;

}

Теперь надо передать данные с выходов первого слоя на входы второго и в список результатов предыдущего прохода распознавания. Это совсем просто:

float fOutputs [M] ; // Копия выходов предыдущего прохода for (int i=0; i<M; i++) { SecondRow [i]. fOutput=fOutputs [i] =InputRow [i]. fOutput;

SecondRow [i]. fSum=0;

}

Теперь может начинать работу второй слой. Рассмотрим алгоритм его работы:

1. Обнуляется счётчик итераций;

2. Запоминаются выходы нейронов в списке результатов предыдущего прохода;

3. Перебирается сеть по нейронам;

4. i-й нейрон посылает свой выход на i-й вход каждого нейрона;

5. Каждый нейрон, принимая значения, накапливает их, предварительно умножив на коэффициент e (кроме случая, когда нейрон принимает своё же значение - тогда он должен умножить его на 1);

6. Переход на шаг 4, пока не будут обработаны все нейроны;

7. Накопленные суммы каждый нейрон посылает на свой выход;

8. Переход на шаг 2, пока выходы нейронов на текущей итерации не совпадут с выходами на предыдущей или пока счётчик числа итераций не превысит некоторое значение.

Теоретически, второй слой должен работать пока его выходы не стабилизируются, но на практике количество итераций искусственно ограничивают. Исходный код:

int iCount=0;

do { for (int i=0; i<M; i++) // Знаения предыдущей утерации { fOutputs [i] =SecondRow [i]. fOutput;

SecondRow [i]. fSum=0;

} for (int i=0; i<M; i++) // Один шаг работы второго слоя for (int j=0; j<M; j++) // С его выходов на его же входы if (i==j) SecondRow [j]. fSum+=SecondRow [i]. fOutput;

else SecondRow [j]. fSum+=SecondRow [i]. fOutput\*e;

for (int i=0; i<M; i++) // Функция - линейная SecondRow [i]. fOutput=SecondRow [i]. fSum;

} while (! bEqual (fOutputs) && ++iCount<25); // Пока выходы не стабилизируются

Последний шаг - выбор из нейронов второго слоя с наибольшим значением на выходе. Его номер и есть номер распознанного образа:

float fMax=-N;

int iIndex;

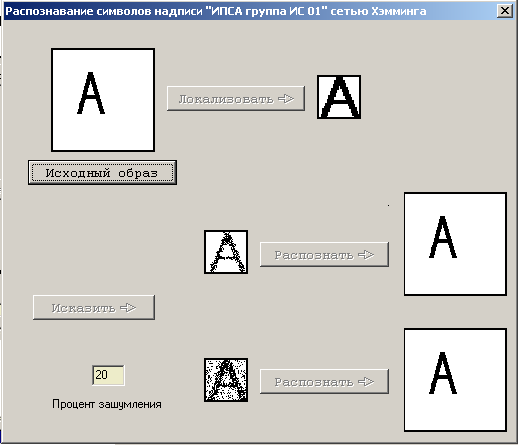
for (int i=0; i<M; i++) if (SecondRow [i]. fOutput>fMax) { fMax=SecondRow [i]. fOutput;

iIndex=i;

}

**2. Описание программы.**

**Руководство пользователя.**



При загрузке формы происходит обучение сети. Данная реализация рассчитана на работу с девятью образами. Для изменения числа образов необходимо изменить значение константы const int M=9 на необходимое число образов и перекомпиллировать программу. Образы в процессе загрузки читаются из директории запускаемой программы из файлов \*. bmp. В данной случае там находятся 9 файлов с образами, имя файла должно складываться из номера образа (начало отсчета - 0) и расширения ". bmp". Если программа не найдет нужного кол-ва файлов - возникает необрабатываемая исключительная ситуация.

Сразу после загрузки формы становится доступной кнопка "Исходный образ", которая открывает стандартное окно диалога и предлагает открыть файл, который содержит изображение одного из образов, на которых обучалась сеть для распознания.

После выбора изображения становиться доступной кнопка "Локализовать", которая запускает процедуру локализации, определяет границы собственно изображения в пустой области фона и центрует изображение с разрешением 40\*40. Локализованное изображение появляется в поле объекта типа TImage.

После этого становиться доступной кнопка "Исказить". Прежде, чем ее нажать, пользователь может изменить значение в поле процент зашумления (0-100%). Кнопка запускает процедуру искажения изображения, в результате которой формируется два искаженных изображения. Первый методом анизотропной апертурной фильтрации (искажаются границы символа, как это бывает при искажениях печати символа на бумаге). Второй способ искажает изображение добавлением N-процентного белого шума, т.е. случайно инвертирует N-процентов общего кол-ва точек, составляющих вектор изображения. При 100% происходит полная инверсия символа. И хотя пользователь в таком случае может различить символ - данная реализация программы не умеет этого делать, т.к считает, что символ изображается черными точками. В форме отображаются оба результата.

На этом этапе становятся доступными две кнопки "распознать" - для первого и второго искаженного образа.

После активизации обеих кнопок "Распознать" можно выбрать либо новый образ, либо поэкспериментировать с текущим, варьируя процент зашумления.

Исследование проводилось таким образом: последовательно увеличивая зашумленность образов (буква И), они подавались на вход сети Хемминга. Результаты работы приведены в таблице 1:

|  |  |  |  |
| --- | --- | --- | --- |
| Тестируемый образ | Процент зашумления образа | Вид искаженного образа | Результат распознавания |
|  | 10% |  |  |
| 20% |  |  |
| 30% |  |  |
| 35% |  |  |
| 40% |  |  |
| 45% |  |  |
| 50% |  |  |
| 60% |  |  |
| 70% |  |  |
| 80% |  |  |
| 90% |  |  |

Таким образом, НС Хемминга прекрасно справляется с задачей распознавания образов для экспериментов с искажением на 0 - 45%. В это диапазоне все эталоны распознаются без ошибок.

При 50% зашумления образы распознаются нестабильно, часто возникает “перепутывание” и на выходе НС появляется совершенно другой эталон.

При 80-90% зашумления образ начинает инвертироваться и при зашумлении 100% мы видим полностью инвертированное изображение исходного. Хотя, человек может распознать с легкостью инвертированный символ, программа не умеет этого делать, т.к считает, что сам символ изображается только черными точками.

При зашумлении 80-100% НС Хемминга перепутывает подаваемые символы с наиболее похожими эталонными. Например, при 100% зашумления буква "П" похожа на толстую полосу и программа распознает символ как "1"