# ПРАКТИЧЕСКИЙ КУРС ПО НЕЙРОННЫМ СЕТЯМ

**Вакуленко С.А., Жихарева А.А.**

Содержание

1. Введение
2. Общие принципы функционирования нейронных сетей.
3. Основные модели (типы) теории нейронных сетей.
4. Однослойный персептрон.
	1. Алгоритм обучения однослойного персептрона.
	2. Практические примеры.
5. Многослойный персептрон.
	1. Структура многослойного персептрона.
	2. Теорема об универсальной аппроксимации.
	3. Практические примеры.
6. Сети радиальных базисных функций.
	1. Структура сети.
	2. Теорема об универсальной аппроксимации.
	3. Практические примеры.
7. Машины опорных векторов (SVM).
	1. Основные идеи. Нелинейные преобразования данных.
	2. Практические примеры.
8. Deep Learning
9. Обработка сигналов при помощи всплесков.

# Введение. Задачи теории искусственного интеллекта (ИИТ) (artificial intellect theory)

Примерно сорок-пятьдесят лет назад, когда теория искусственного интеллекта начала развиваться, люди надеялись смоделировать человеческий мозг и научить компьютер решать все те же сложные задачи, которые решает человеческий мозг. В настоящее время эта глобальная цель выглядит не очень достижимой, однако, в ряде важных конкретных задач мы продвинулись. По поводу неразрешимых глобальных проблем (которые являются захватывающе интересными), читатель может обратиться к книгам известного английского физика Роджера Пенроуза.

В данном курсе мы будем обсуждать модели, которые могут, с тем или иным успехом, решать следующие конкретные проблемы

*Задача обработки сигналов и изображений*

При обработке изображения (сигнала) часто необходимо разделить полезную (информативную) компоненту сигнала от шума. Эти задачи получили название задач фильтрации.

Такие задачи важны, в частности, в полиграфии. Например, нужно выделить наиболее существенную для пользователя информацию и убрать все остальное. Это сложная задача ИИТ.

*Задача распознавания образов*

Например, имеем конвейер, по которому идет продукция. Можно ли создать систему технического зрения, позволяющую автоматически различить бракованную и нормальную продукцию.

*Распознавание речи также относится к классу задач ИИТ*

Другой пример — курсы валют, акций обычно сильно колеблются. Эти колебания связаны с тем, что на рынке имеется много агентов, действующих обычно несогласованно. Трейдер решает простую, казалось бы, задачу — выделить среди этих колебаний так называемый основной тренд, чтобы понять стоит ли вкладывать свои деньги в акции. Основной тренд представляет из себя некую усредненную кривую, где мелкие колебания (шум) должен быть устранен в результате фильтрации.

Другая популярная ныне и крайне важная задача о выделении основного тренда, это задача анализа климатических данных с целью выяснить, есть глобальное потепление или нет.

Другие важные задачи — это распознавание спама в Интернете, автоматический перевод, распознавание почерка и так далее.

Что надо знать и используется из математики

* теория многомерных векторных пространств;
* алгоритмы, оценка быстродействия алгоритмов; жадные алгоритмы, алгоритм градиентного спуска;
* динамическое программирование;
* динамические системы и аттракторы; функции Ляпунова.

# Общие принципы функционирования нейронных сетей

Первый принцип, который используют нейронные модели это использовать аналогии с мозгом и реальными нейронами. В связи с этим сформулируем основные факты о мозге и нейронах.

Итак, что мы знаем о мозге?

* Мозг состоит из нейронов
* Нейроны связаны друг с другом
* Нейроны обмениваются сигналами
* Сигналы имеют булевскую природу
* Система связанных нейронов - это стохастическая динамическая система

Упрощенно, нейрон — это пороговая система, которая получает входные сигналы от других нейронов, суммирует их и если эта сумма превышает некий порог, генерирует выходной сигнал.

Идеализированная модель такой пороговой системы может быть построена при помощи сигмоидальных функций. Типичный график такой функции имеет следующий вид:

*Рисунок* 1*. Пример сигмоидальной функции*

Приведем в качестве примера несколько функций подобного рода:

* , где *x* — амплитуда входного сигнала, который получает нейрон от других нейронов; *σ* — выходной сигнал нейрона (рис. 1);
*  — функция скачка или функция Хевисайда (рис. 2);

*Рисунок* 2*. Функция Хевисайда*

* Кусочно-линейная (функция )
* Кусочно-полиномиальная

При распознавании образов в мозге происходят сложные процессы. Упрощенно систему распознавания образов можно представить как многослойный персептрон (см. часть 1).

Сигналы из многослойного персептрона обрабатываются кортекс - это рекуррентная нейросеть (см. часть 2).

Силу связи между нейронами описывает матрица синаптических связей *W*. Известно, что вся долговременная фундаментальная информация хранится в этой матрице. Матрица синаптических связей *W* медленно меняется со временем. Итак, в сети две динамики — быстрая динамика нейронов, описываемая динамической системой, и медленная, связанная с изменением силы связи между нейронами. Медленная динамика упрощенно описывается так называемым правилом Хебба. Хебб — канадский физиолог, который экспериментально показал, что если два нейрона оба часто одновременно активны, то сила связи между ними растет. Возможны разные варианты математической записи такого правила.

Если воспользоваться простейшим вариантом правила Хебба, предложенным С. Фузи, Н. Брюнелем и др., то можно показать, что это правило в совокупности с шестислойным персептроном реализует все важнейшие алгоритмы (преобразование Фурье и т.д.) (Вакуленко, 2002).

Рассмотрим общие принципы функционирования нейронных сетей на примере задачи классификации.

Предположим, мы хотим создать автоматическую систему, которая различает два типа объектов, *А* и *B*. Системы технического зрения позволяют записать данные об объектах (признаки объекта) в цифровом виде.

Предположим, что мы характеризуем объект с помощью набора признаков . Признаки могут быть выражены с помощью целых чисел или даже булевских переменных, то есть «есть признак» — «нет признака».

Совокупность признаков можно рассматривать как вектор с *k* компонентами, или точку *k*-мерного Эвклидова пространства . Тогда задача классификации сводится к следующей математической задаче: разделить два множества точек *А* и *B* *k*-мерного Эвклидова пространства некоторой гиперповерхностью размерности *k-1*.

Сделаем некоторые важные комментарии. Выбор признаков для классификации является исключительно сложной задачей, которую мы пока не рассматриваем. Ранее это задача решалась вручную, в последнее время появились эффективные методы автоматического нахождения наиболее эффективных признаков (так называемый Deep Learning). Отметим, что выбор правильных признаков важен для успешности последующей обработки системы признаков методами, которые мы описываем ниже.

Обучение нейронной сети в задачах классификации происходит на наборе обучающих примеров , для которых принадлежность объекта к классу *А* или классу *B* известна. Кроме того, определим индикатор:

.

По накопленному в результате обучения «опыту» строим сеть, которая проводит разделяющую поверхность.

Математически этот процесс может быть описан как поиск некоторой функции

,

где *W* - набор параметров нейронной сети. Эти параметры, в частности, задают силу связи между нейронами и подбираются так, чтобы ошибка обучения (*error training*)

,

где  берутся из обучающего множества, была бы минимальной (как можно ближе к нулю).

Для проверки эффективности обучения нейронной сети берут тестовое множество объектов и вычисляют

,

где  взяты из тестового множества.

После того, как система обучена (что иногда требует большого процессорного времени), она решает автоматически для любого поданного на вход системы объекта , к какому классу он относится.

# Основные модели теории нейронных сетей

*Выбор модели — скорее искусство, чем наука.*

СЕТИ ПРЯМОГО РАСПРОСТРАНЕНИЯ

* Однослойный персептрон.
* Многослойный персептрон.
* Сети радиальных базисных функций (RBF)
* Машины опорных векторов (SVM)

РЕКУРРЕНТНЫЕ СЕТИ

* Реккурентные сети.
* Аттракторные нейронные сети.

*Обзор моделей*

Первая модель нейронной сети была предложена Ф.Розенблаттом в 50-е годы XX века и получила название однослойного персептрона. В этой модели входной сигнал проходил всего один слой нейронов и после этого формировался выходной сигнал при помощи сигмоидальной функции.

В дальнейшем выяснилось, что возможности таких простейших систем ограничены. Было предложено обобщение — модель многослойного персептрона, отличие которого от однослойного заключается в прохождении входного сигнала через несколько слоев.

Многослойные персептроны могут моделировать любую систему «вход-выход», то есть любой черный ящик. Однако, они имеют ряд недостатков.

Такие системы, как сети радиальных базисных функций и машины опорных векторов, можно рассматривать как специальные модификации многослойных персептронов с целью преодоления недостатков последних.

Все перечисленные модели характеризуются наличием слоев, через которые проходит сигнал. В рекуррентных нейронных сетях невозможно выделить отдельные слои. Сигналы могут циркулировать по сети во всех направлениях, образуя сложную пространственно–временную структуру (*pattern*). Такие сети обладают колоссальными возможностями. Так, например, известно, что они могут моделировать любую динамическую систему или машину Тьюринга, то есть реализовать любой алгоритм.

# Однослойный персептрон

Однослойный персептрон представляет собой простейшую модель нейронной сети.

Однослойный персептрон получает на вход сигнал, заданный вектором  и на выходе выдает число

,

где

.

Входной вектор  состоит из *n* компонент , каждая из которых представляет собой численную характеристику анализируемого нейронной сетью объекта.

Через  и  обозначены параметры персептрона*— синаптические веса и порог (сдвиг).* Синаптические веса указывают силу влияния конкретной характеристики на выходное значение. Их значения могут быть как положительными, так и отрицательными.

Отметим, что переменные  могут быть булевскими, то есть принимать значения *0* или *1*. Это означает, что объект обладает данным признаком, если , и не обладает, в случае, если . В этом случае функция  — функция Хевисайда или функция скачка.

*Рисунок* 3*. Модель однослойного персептрона*

На рис. 3 приведена модель однослойного персептрона, цифрами 1-4 помечена последовательность действий при работе алгоритма.

Преимущество персептрона заключается в его простоте. Укажем и его недостаток — персептрон классифицирует только линейно разделимые объекты (то есть только такие множества векторов , между которыми можно провести разделяющую эти множества гиперплоскость).

## Геометрическая интерпретация

Однослойный персептрон работает как классификатор объектов двух разных множеств. Чтобы понять, как это происходит, рассмотрим случай, когда сигмоидальная функция есть функция скачка . Предположим, требуется классифицировать объекты в два непересекающихся класса объектов *A* и *B*. Для этого, определим выходные значения: положим , если объект принадлежит к классу *A*, , если объект принадлежит к классу *B*:

,

Тогда, из определения функции скачка  следует, что объект лежит в классе A при условии  и объект лежит в классе *B*, если .

Следовательно, уравнение  может быть рассмотрено как некий разделитель множеств объектов разных классов.

С другой стороны, уравнение  равносильно уравнению

,

которое определяет гиперплоскость в *n*-мерном линейном пространстве.

Напомним, что в двумерном пространстве гиперплоскость — это прямая, а в трехмерном — обычная плоскость.

Следующая иллюстрация (рис. 4) демонстрирует вышесказанное на примере игры в квадратики (класс *A*) и шарики (класс *B*) в случае *n=2*, то есть на плоскости. Прямые , ,  — примеры «разделителей» этих классов.

*Рисунок* 4*. Геометрическая интерпретация однослойного персептрона*

Таким образом, однослойный персептрон есть линейный разделитель.

## Моделирование логических функций однослойным персептроном

Однослойный персептрон позволяет несложным образом реализовать ряд логических функций таких, как

* конъюнкция
* дизъюнкция
* отрицание
* демократическое голосование

Результат каждой из этих функций определяется значением двух булевских переменных  и . Представим значения этих переменных, как вектор входных значений , а результат, как значение функции скачка, то есть .

Поясним сказанное на примере функции конъюнкции. Известно, что

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 0 | 1 | 0 | 1 |
|  | 0 | 0 | 1 | 1 |
|  | **0** | **0** | **0** | **1** |

Рассмотрим вектор входных значений  как точку плоскости, а результат логической операции как форму точки: квадратик, в случае, когда результат операции истинен (), и шарик, если результат операции ложен (). Тогда задача классификации сводится к задаче отыскания прямой, разделяющей два класса объектов — квадратики и шарики.

Прямой, разделяющей эти множества объектов, является, например, прямая



Это означает, что синаптические веса , , а порог .

Иллюстрация вышесказанного приведена на рис. 5.

*Рисунок* 5*. Моделирование конъюнкции однослойным персептроном*

## Пример Минского

Однако оказывается, что иногда квадратики и шарики неразделимы.

*Рисунок* 6*. Пример Минского*

Марвин Минский показал, что возможности персептрона ограничены, потому что он разделяет множества с помощью гиперплоскости. В самом деле, есть такие явно различимые классы объектов, для которых прямая как их разделитель, не подходит. Например, рассмотрим логическую функцию «исключающее или» (*XOR*) определяемую формулой

.

Для лучшего понимания приведем также таблицу значений функции *XOR* для различных значений переменных:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 0 | 1 | 0 | 1 |
|  | 0 | 0 | 1 | 1 |
|  | **0** | **1** | **1** | **0** |

Эта функция не реализуема однослойным персептроном, то есть не существует такой прямой, которая бы разделила эти два класса объектов. На рис. 6 проиллюстрирован результат этой логической операции в виде квадратиков, в случае, когда результат операции истинен (), и шарик, если результат операции ложен (). Очевидно, что эти два множества (квадратиков и шариков) неразделимы никакой прямой.

## Алгоритм обучения однослойного персептрона

Имеются итерационные алгоритмы, которые находят параметры , если исходные объекты линейно разделимы.

Обучение нейронной сети в задачах классификации происходит на наборе обучающих примеров , в которых ответ — принадлежность к классу *А* или *B*, — известен. Определим индикатор *D* следующим образом: положим , если  из класса *А*,и положим , если  из класса *B*, то есть

|  |  |
| --- | --- |
| (1) | , |

где всякий вектор  состоит из *n* компонент: .

Задача обучения персептрона состоит в нахождении таких параметров  и , что на каждом обучающем примере персептрон выдавал бы правильный ответ, то есть

|  |  |
| --- | --- |
| (2) |  |

Интуитивно кажется очевидным, что если персептрон обучен на большом числе корректно подобранных примеров, и равенство (2) выполнено для почти всех , то в дальнейшем персептрон будет с близкой к *1* вероятностью проводить правильную классификацию для остальных примеров. Этот интуитивно очевидный факт был впервые математически доказан (при некоторых предположениях) в основополагающей работе наших соотечественников Вапника и Червоненскиса.

На практике, однако, оценки по теории Вапника—Червоненскиса иногда не очень удобны, особенно для сложных моделей нейронных сетей. Поэтому практически, чтобы оценить ошибку классификации, часто поступают следующим образом: множество обучающих примеров разбивают на два случайно выбранных подмножества, при этом обучение идет на одном множестве, а проверка обученного персептрона — на другом.

 (????? Нужна ли картинка)

Рассмотрим подробнее алгоритм обучения персептрона

*Шаг 1. Инициализация синаптических весов и смещения.*

Значения всех синаптических весов модели полагают равными нулю: , смещение (порог) нейрона *h* устанавливаются равными некоторым малым случайным числам. Ниже, из соображений удобства изложения и проведения операций будем пользоваться обозначением *.*

Обозначим через  вес связи от *i*-го элемента входного сигнала к нейрону в момент времени *t*.

*Шаг 2. Предъявление сети нового входного и желаемого выходного сигналов.*

Входной сигнал  предъявляется нейрону вместе с желаемым выходным сигналом *D*.

*Шаг 3. Адаптация (настройка) значений синаптических весов. Вычисление выходного сигнала нейрона.*

Перенастройка (адаптация) синаптических весов проводится по следующей формуле:



где под  подразумевается индикатор, определенный равенством (1), а *r*— параметр обучения, принимающий значения меньшие *1*.

Описанный выше алгоритм — это алгоритм градиентного спуска, который ищет параметры, чтобы минимизировать ошибку. Алгоритм итеративный. Формула итераций выводится следующим образом. Введем риск

,

где суммирование идет по числу опытов (*t* — номер опыта), при этом задано максимальное число опытов — *T*.

Подставим вместо F формулу для персептрона, вычислим градиент по w. Получим указанную выше формулу перенастройки весов.

## Иллюстрация работы алгоритма

В процессе обучения вычисляется ошибка



Построим график, показывающий как меняется эта ошибка (рис. 7). На нем хорошо видно, что начиная с некоторого шага величина  равна нулю. Это означает, что персептрон обучен.

*Рисунок* 7*.*

## Построение нейронной сети для операции конъюнкции в пакете Matlab

Построим нейронную сеть для логической операции конъюнкции с помощью пакета Matlab. Таблица данной логической операции приведена выше. В данной реализации массив входов *P* задает значения  для всех обучающих примеров и представляет собой первые две строки этой таблицы. Целевой вектор *T* содержит значения индикаторной функции  для всех примеров  (в данном случае 4 примера) и представляет собой последнюю строку этой таблицы. Приведенный ниже скрипт содержит подробные комментарии.

% аппроксимация конъюнкции однослойным персептроном

% построение функции T = H(w\_1\*X\_1 + w\_2\*X\_2 + w\_0)

% (P,T) задают таблицу истинности

% зададим вектор входов

P = [ 0 1 0 1; 0 0 1 1 ];

% зададим вектор выходов = целевой вектор

T = [ 0 0 0 1 ];

% встроенная функция построения сети

% с помощью персептрона,

% по умолчанию использована модель 'hardlim'

net = perceptron;

% определим максимальное число итераций для обучения сети

net.trainParam.epochs = 20;

% обучение сети

% (в процессе обучения происходит коррекция весов w\_i)

net = train(net,P,T);

% вычисление значений аппроксимирующей функции в точках P

y = sim(net, P) % симуляция

% синаптические веса: коэффициенты w\_i после обучения сети

A = net.IW;

celldisp(A)

% коэффициент сдвига b после обучения сети

w\_0= net.b

В процессе работы скрипта *Matlab* отображает текущую информацию о модели и параметрах обучения в специальном окне *Neural Network Training* (рис. 8). В частности, в этом окне показана структура нейронной сети — число входных нейронов, число уровне персептрона, число выходных нейронов (*Neural Network*), число итераций при обучении сети (в нашем случае — 5), время обучения сети.

*Рисунок* 8*. Нейронная сеть, реализующая конъюнкцию*

В то же время согласно инструкциям скрипта в окне команд будет выведена информация по результатам симуляции (вектор *y*) и значения весовых коэффициентов (, ) и коэффициента сдвига ():

y =

0 0 0 1

A{1} =

1 2

w\_0 =

[-3]

Таким образом, получена модель персептрона, определяемая уравнением

,

что геометрически будет означать разбиение двух множеств заданной прямой (рис. 9).

*Рисунок* 9*. Моделирование конъюнкции однослойным персептроном в Matlab*

## Задача о моделировании черного ящика в общей постановке

Пусть известен реальный выход устройства  при входе , где  — вектор с компонентами , *t* — номер проводимого опыта,  (максимальное число опытов T заранее определено).

Необходимо найти параметры модели  такие, что выход модели  и реальный выход устройства  были бы как можно ближе в среднеквадратичном смысле, то есть



Функцию  называют *эмпирическим риском*.

## Построение нейронной сети для классификации множеств точек плоскости в пакете Matlab

В качестве другого примера применения однослойного персептрона приведем пример классификации точек плоскости на два класса — точки, находящихся внутри эллипса, и точки, находящихся вне эллипса. Эллипс определен общим уравнением кривой второго порядка вида

.

Ниже приведен скрипт, реализующий данную классификацию для эллипса . Скрипт снабжен подробными комментариями. Кроме того, для большей наглядности приведены соответствующие построения на плоскости — исходная кривая, точки двух множеств, окрашенные соответственно в разные цвета.

В этом примере применяется одна из важнейших идей теории нейронных сетей. А именно, в исходном пространстве двух переменных однослойный персептрон не может разделить внешность и внутренность эллипса. Но мы можем сделать нелинейное преобразование данных, отобразив их в пространство высшей размерности. Здесь применяется так называемая теорема Ковера (ее формулировка будет приведена в одном из следующих разделов), которая утверждает, что после такого отображения данные могут стать линейно разделимыми.

Применим в нашем случае нелинейное отображение точки плоскости  в пространство размерности *5* следующим образом:

.

% построение однослойного персептрона для классификации

% точек плоскости в соответствии с заданным эллипсом

% (внутри и вне эллипса (ax^2 + bxy + cy^2 + dx + ey = 1))

% Результат работы нейронной сети:

% вектор значений t:

% 0 - точка вне эллипса, 1 - точка внутри эллипса

N = 300; % число точек для обучения

% коэффициенты заданного эллипса

a = 1.5; b = 2; c = 1.1; d = -2; e = -3;

% построим график исходного эллипса

syms X Y; f = a\*X.^2 + b\*X.\*Y + c\*Y.^2 + d\*X + e\*Y-1;

ellips = ezplot(f); set (ellips,'Color','b'); hold on

% сгенерируем исходные данные для обучения:

% точки плоскости (x,y), чьи координаты имеют

% нормальное распределение на интервалах [-2a;2a] и [-2c;2c]

x = randn(1,N)\*(2\*a) - d/(2\*a); y = randn(1,N)\*(2\*c) - e/(2\*c);

% определим отображение

% (x1,x2)->(a\*x1^2,b\*x2^2,c\*x1\*x2,d\*x1,e\*x2)

P = [a\*x.^2; b\*x.\*y; c\*y.^2; d\*x; e\*y]; % матрица входов

% формируем T = вектор выходов, приписывая значения 0 и 1

$ для точек, лежащих вне и внутри эллипса соответственно

T = ones(1,N);

k = find( sum(P) >= 1);

T(k) = 0;

% функция построения сети с помощью персептрона,

% по умолчанию использована модель 'hardlim'

net = newp(P,T);

% определим максимальное число итераций для обучения сети

net.trainParam.epochs = 50;

% функция обучения сети; при этом происходит коррекция весов W\_i

net = train(net,P,T);

% вычисление значений аппроксимирующей функции в точках P

t = sim(net, P) % симуляция

% синаптические веса: коэффициенты w\_i после обучения сети

A = net.IW;

celldisp(A)

% коэффициент сдвига b после обучения сети

b = net.b

% геометрические построения

% распознавание точек плоскости после обучения сети

k = find( t == 0); % индексы точек вне эллипса

x\_out = x(k); y\_out = y(k);

plot(x\_out, y\_out, '\*r');

clear k;

k = find( t == 1); % индексы точек внутри эллипса

x\_out = x(k); y\_out = y(k);

plot(x\_out, y\_out, '\*g');

В процессе работы скрипта *Matlab* отображает текущую информацию о модели и параметрах обучения в специальном окне *Neural Network Training* (рис. 10). В частности в этом окне показана структура нейронной сети — число входных нейронов, число слоев персептрона и используемые модели слоев, число выходных нейронов (*Neural Network*), число итераций при обучении сети (в нашем случае — 50), время обучения сети.

Рисунок .

В окне команд будут выведены значения весовых коэффициентов (*w1=-138.9032, w2=-164.07, w3=-179.1546, w4=-153.6901, w5=-188.0759*) и коэффициента сдвига (*w0=151*):

A{1} =

-138.9032 -164.0700 -179.1546 -153.6901 -188.0759

b =

[151]

Таким образом, построенная модель имеет вид (у всех весовых коэффициентов указано только два десятичных знака):



Результат классификации множества точек после обучения персептрона представлен графически на рис. 11; точки «внутренности» и «внешности» эллипса красным изображены зеленым и красным цветом соответственно.

Рисунок .

## Упражнения

1. Сколько существует булевских функций от двух переменных? От трех?
2. Найти булевские функции от двух переменных, непредставимые однослойным персептроном.
3. Представим ли штрих Шеффера однослойным персептроном?
4. Представима ли стрелка Пирса однослойным персептроном.
5. Построить с помощью встроенной функции *newp(P,T)* однослойный персептрон, реализующий стрелку Пирса.
6. Построить с помощью встроенной функции *newp(P,T)* однослойный персептрон, реализующий штрих Шеффера.
7. Проверьте работу скрипта из второго примера для классификации точек плоскости на множества точек «внешности» и «внутренности» эллипса. Проведите следующие изменения в данном скрипте и проанализируйте полученные результаты:
	1. в качестве третьего параметра функции *newp* укажите другие модели персептрона (сигмоидальные функции):
		* *'tansig'* — гипебролический тангенс;
		* *'logsig'* — логистический сигмоид;
		* *'purelin'* — линейный сигмоид;
	2. измените максимальное число итераций для проведения обучения сети;
	3. проверьте корректность работы построенной нейронной сети, для чего проведите классификацию нескольких тестовых точек плоскости; для наглядности выполните построение этих точек на той же плоскости; координаты тестовых точек задайте самостоятельно;
	4. измените параметры исходного эллипса (коэффициенты кривой).
8. Рассмотрим параболу, заданную в виде общего уравнения кривой второго порядка . Сгенерируйте случайное множество на плоскости (100 точек). Используя нелинейное отображение *ψ(x,y)* в 5-мерное пространство и однослойный персептрон в 5-мерном пространстве, разделите точки множества на два класса —- точки, лежащие «вне» и «внутри» параболы.
9. Решите предыдущую задачу в случае гиперболы, при этом точки сгенерированного множества разделите на точки, лежащие между ветвями гиперболы и точки, лежащие «внутри» ветвей гиперболы.

**Многослойный персептрон**

Пусть известен реальный выход устройства  при входе , где  — вектор с компонентами , *t* — номер проводимого опыта,  (максимальное число опытов *T* заранее определено).

Необходимо найти параметры модели  и , ,  такие, что выход модели  и реальный выход устройства  были бы как можно ближе. Эта задача всегда разрешима так называемыми многослойными персептронами. Эту модель мы рассматриваем ниже.

Связь между входом и выходом двухслойного персептрона устанавливают следующие соотношения:

,

.

К явным преимуществам многослойного персептрона следует отнести возможность с его помощью классифицировать любые объекты, разделимые в пространстве признаков. Этот результат был получен лишь в 90-e годы XX века.

К недостаткам относят сложность и медленность настройки, а также плохая способность к обобщению. Чтобы преодолеть эту трудность, применяются эвристические приемы и методы, развитые в теории Deep Learning.

Рисунок . Модель многослойного персептрона с одним скрытым слоем (N=1)

Рисунок 12 иллюстрирует архитектуру двухслойного персептрона, где число скрытых слоев *N=1*. Трехслойный персептрон, очевидно, имеет два скрытых слоя (то есть два ряда «голубых» шариков), и т.д. Необходимо отметить, что в каждом скрытом слое может быть разное число нейронов.

## Архитектура многослойного персептрона

Многослойный персептрон способен на абсолютное разделение, то есть может провести любую разделяющую гиперповерхность. Сформулируем это утверждение неформальным образом в следующей теореме:

**Фундаментальная теорема**

*Пусть имеется два произвольных непересекающихся множества в многомерном линейном пространстве. Тогда с помощью многослойного персептрона всегда можно разделить эти два множества, то есть провести гиперповерхность, разделяющую их.*

В формальных терминах приведенная теорема об универсальной аппроксимации может быть изложена так:

**Фундаментальная теорема о разделимости классов**

*Для любой непрерывной функции , определенной на ограниченной замкнутой области, и каждого  существуют коэффициенты  и , , , такие, что*

.?

Заметим, что выражение



описывает персептрон с одним скрытым слоем. Таким образом, теорема утверждает, что одного скрытого слоя достаточно, чтобы аппроксимировать все (то есть любую непрерывную функцию, определенную на ограниченном множестве).

*Основные идеи доказательства.*

Доказательство теоремы проводится сначала для одномерного входа (), в этом случае вектор входов представляет собой скаляр: . Это может быть сделано разложением по всплескам (вэйвлетам) методом Фурье (пункт *A*). Затем рассматривается случай произвольной размерности вектора входов (пункт *B*); при этом применяется разложение функции с помощью Фурье по плоским волнам, что сводит задачу к пункту *A*.

*Пункт* *A.* , . Для скалярного  имеем



|f(x) - Σ*m=1,… M* a*m* σ (w*m* x - h*m* )|< ε

Мы приближаем производную



|f’(x) - Σm=1,… M am σ’ (bm (x - hm ))|< ε

Введем всплеск (вейвлет)



тогда последнее неравенство, как следует из теории вейвлетов (она будет изложена ниже), может быть записано как



f’(x) - Σm=1,… к сm ψ (bm (x - hm ))|< ε

*Пункт* *B.* , , .

Из теории рядов Фурье следует, что любую функцию многих переменных можно представить в виде суммы функций одного переменного, то есть

,

где .

Заметим, что увеличение числа нейронов может подавить шум с сохранением той же точности аппроксимации:

.

σ(bm (x - hm )) =0.5 [σ(bm (x - hm )) + σ(bm (x - hm )) ]

*Замечание*.

В доказательстве теоремы одну функция векторного аргумента со скалярными значениями приближалась с помощью двухслойного (входной слой и скрытый слой) персептрона и одним выходом. Векторно-значную функцию можно аппроксимировать с помощью нескольких параллельных персептронов или даже при помощи одного персептрона (соединив параллельные вместе).

*Обучение многослойного персептрона* происходит по алгоритму градиентного спуска, аналогичному однослойному. Один из знаменитых вариантов этого алгоритма получил название *метод обратного распространения ошибки* (error back propagation).

## Построение нейронной сети для операции XOR в пакете Matlab

Рассмотрим пример многослойного персептрона. Выше мы отмечали, что логическая функция *XOR* нереализуема однослойным персептроном. Сейчас мы увидим, что она реализуема многослойным персептроном, причем, в тайном слое достаточно всего двух нейронов.

% аппроксимация XOR при помощи многослойного персептрона

% (P,T) задают таблицу истинности

P = [ 0 1 0 1; 0 0 1 1 ]; % вектор входов

T = [ 0 1 1 0 ]; % вектор выходов = целевой вектор

% встроенная функция построения сети с помощью персептрона

net = newff(minmax(P),[2,1], {'tansig', 'purelin'}, 'traingd');

% определим максимальное число итераций для обучения сети

net.trainParam.epochs = 10000;

net.trainParam.lr = 0.05; % скорость обучения

% точность обучения сети

% (как только она достигнута, обучение будет закончено)

net.trainParam.goal = 1e-5;

% функция обучения сети; при этом происходит коррекция весов

net = train (net,P,T)

% вычисление значений аппроксимирующей функции в точках P

y = sim(net, P) % симуляция

% синаптические веса входного слоя —

% коэффициенты V\_ik после обучения сети

A\_IW = net.IW;

celldisp(A\_IW)

% синаптические веса скрытых уровней —

% коэффициенты w\_i после обучения сети

A\_LW = net.LW;

celldisp(A\_LW)

% коэффициенты сдвига после обучения сети

b = net.b;

celldisp(b)

В процессе работы скрипта Matlab отображает текущую информацию о модели и параметрах обучения в специальном окне *Neural Network Training* (рис. 13). В частности, в этом окне показана структура нейронной сети — число входных нейронов, число слоев персептрона и моделей слоев, число выходных нейронов (Neural Network), число итераций при обучении сети (в нашем случае — *667*), время обучения сети.

Рисунок .

В окне команд будет выведена информация по результатам симуляции (вектор *y*) и значения весовых коэффициентов для нейронов входного слоя (*A\_IW{1}*) и нейронов скрытого слоя (*A\_LW{2,1})* и коэффициентов сдвига в этих слоях (*b{1}, b{2}*):

y =

-0.0008 1.0038 0.9953 0.0018

A\_IW{1} =

-1.1765 -3.6643

1.6428 3.4941

A\_IW{2} =

[]

A\_LW{1,1} =

[]

A\_LW{2,1} =

0.9963 0.7527

A\_LW{1,2} =

[]

A\_LW{2,2} =

[]

b{1} =

4.5146

-0.9600

b{2} =

-0.4366

Модель построенного персептрона приведена на рис. 14, нейроны скрытого и выходного слоев определяются следующими соотношениями (у всех весовых коэффициентов указано только два десятичных знака для краткости записи):



.

Согласно приведенному скрипте в качестве сигмоидальной функции  для вычисления нейронов скрытого слоя использована функция *'tansig'* — гипебролический тангенс; а в качестве сигмоидальной функции  для вычисления выходного нейрона — линейный сигмоид (функция*'purelin'*).

Рисунок .

Управление числом скрытых слоев персептрона, числом нейронов в каждом из них, моделями каждого слоя, а также методом коррекции весов производится посредством указания соответствующих аргументов функции *newff*. Так, ее второй аргумент предназначен для указания числа нейронов в каждом из слоев персептрона и в общем случае для *k* скрытых слоев имеет вид:

[*n\_1,n\_2,…, n\_k,n\_out*]*,*

где *n\_1* — число нейронов в первом скрытом слое, *n\_2* — число нейронов во втором скрытом слое, *n\_k* — число нейронов в *k*-м скрытом слое, *n\_out* — число нейронов в выходном слое.

При необходимости изменить модели слоев соответствующие имена сигмоидальных функций их перечисляют в фигурных скобках на месте третьего аргумента функции *newff*:

{*model\_hidden\_1,model\_hidden\_2,…, model\_hidden\_k,model\_exit*}

Последний аргумент функции *newff* предполагает указание метода коррекции синаптических весов модели. В приведенном примере использован параметр *'traingd'*, определяющий коррекцию весов по методу градиентного спуска.

## Упражнения

1. Воспроизведите приведенный в качестве примера скрипт. Проведите следующие изменения в данном скрипте и проанализируйте полученные результаты:
	1. измените модели слоев персептрона, используя сигмоидальные функции
		* *'hardlim’* — функция Хевисайда;
		* *'tansig'* — гипебролический тангенс;
		* *'logsig'* — логистический сигмоид;
		* *'purelin'* — линейный сигмоид;
	2. измените метод коррекции весов
		* *'traingda'* — метод градиентного спуска с адаптивной скоростью обучения;
		* *'traingdm'* — метод градиентного спуска с импульсом;
		* *'traingdx'* — метод градиентного спуска с импульсом и адаптивной скоростью обучения;
	3. измените максимальное число итераций для проведения обучения сети.
2. Найти многослойный персептрон минимальной архитектуры, моделирующий *XOR*.

# Сети радиальных базисных функций (NETWORK OF RADIAL BASIC FUNCTIONS (RBF))

Введем так называемые базисные функции gj (**x**), **x**=(x1 , …, xk)

*Удобный выбор таких функций* g*j* (**x**)=G( λ|x - t*j* |), где G — гауссовская функция, t*j* - некоторые точки и λ параметр сжатия. Если этот параметр велик тогда график этой функции напоминает узкий пик локализованный около центра t*j*.

С помощью системы базисных функций мы можем аппроксимировать любую *заданную функцию f(x)*. *C помощью линейных комбинаций*

f(**x**) ≈ Σ*j=1,… M* w *j* g*j* (**x**)= F(**x**, **w, M**)

Это выражение определяет сеть RBF. Такие сети также, как и многослойные персептроны, являются универсальными разделителями и аппроксиматорами.

**Теорема**

*Функции F(x,w,M) являются универсальными аппроксиматорами в компактных областях D, x∈D.*

Эта теорема может быть доказана разными сложными методами и мы опускаем доказательство.

Опишем процедуру поиска неизвестных коэффициентов w*j* при фиксированных центрах t *j*.

Она основана на минимизации риска Мы ищем приближение неизвестной функции f(**x**). Предположим, мы знаем значения f(**x***j*) в некоторых точках **x***j*.

Рассмотрим риск

R=[ Σ*j=1,… M* (f(**x***j*) - F(**x**, **w**) ) 2 ]/M.

Простейший случай **M=N**

M=N и t*j* = x*j* Полагаем (интерполяция)

f(**x***j*) = F(**x**, **w)**

Получаем линейную систему для w *j*

L **w**= b,

где

L kj = gj (**x**k) , bk= f(**x**k)

Система легко решается, например, в пакете Matlab.

Более важный и интересный случай — это **M < N**

Здесь применяем метод наименьших квадратов для построения уравнения регрессии. Берем произвольные точки t*j* и минимизируем риск по w *j* . Получаем линейную систему для w *j*

L **w**= **b**,

где L *kj =* Σ*p=1,… N* g*j* (**x***p*)g*k* (**x***p*),

b*k*= Σ*p=1,…N* g*k* (**x***p*)f(**x***p*) j, k =1,…,M

Снова система легко решается. Отметим, что эта процедура очень быстрая по сравнению с обучением многослойного персептрона (в случае фиксированных центров), она не требует никаких итераций.

**Более сложные подходы**

До сих пор мы предполагали, что точки t*j* фиксированы. Их тоже можно искать или выбирать случайно.

Известно, что оптимальный выбор числа центров *M* дан формулой ≈ O(*N* 1/3) и скорость аппроксимации — О(1/ M)

Приведем практический пример — решим уже известную нам задачу реализации функции XOR.

% аппроксимация XOR при помощи радиальных базисных функцй (RBF)

% (P,T) задают таблицу истинности

P = [ 0 1 0 1; 0 0 1 1 ]; % вектор входов

T = [ 1 0 1 0 ]; % вектор выходов = целевой вектор

% среднеквадратичная ошибка построения РБФ; по умолчанию = 0.0

goal = 1e-07;

% параметр, определяющий "крутизну" всплеска; по умолчанию = 1;

spread = 3;

% чем больше значение spread, тем более гладкой будет аппроксимация;

% обычно, выбирается большим, чем шаг разбиения интервала в обучающей последовательности, но меньшим размера самого интервала

% функция построения сети с помощью RBF

net = newrb(P,T,goal,spread)

% вычисление значений аппроксимирующей функции в точках P

y = sim(net, P) % симуляция

% веса входного слоя - коэффициенты w\_i после обучения сети

A\_IW = net.IW;

celldisp(A\_IW)

% веса уровней - коэффициенты w\_i после обучения сети

A\_LW = net.LW;

celldisp(A\_LW)

b = net.b; % коэффициенты сдвига b после обучения сети

celldisp(b)

В окне команд будет выведена информация по результатам симуляции (вектор y) и значения весовых коэффициентов (w1=-138.9032, w2=-164.07, w3=-179.1546, w4=-153.6901, w5=-188.0759) и коэффициента сдвига (w0=151) в скрытом и выходном слоях:

y =

1.0000 -0.0000 1.0000 -0.0000

A\_IW{1} =

0 0

1 0

0 1

A\_IW{2} =

[]

A\_LW{1,1} =

[]

A\_LW{2,1} =

7.0050 0.0000 7.0050

A\_LW{1,2} =

[]

A\_LW{2,2} =

[]

b{1} =

0.2775

0.2775

0.2775

b{2} =

-12.4907

# Метод опорных векторов (Support Vector Mashine (SVM))

Метод опорных векторов в задачах классификации. Рассмотрим задачу классификации на два непересекающихся класса, в которой объекты описываются n-мерными вещественными векторами, выход

Y={-1,+1}.

Будем строить линейный пороговый классификатор:

Y(x)= sign( **w∙x** - w *0* )

где **x** = (x *1*, . . . , x *n*) признаковое описание объекта **x**; вектор **w** = (w *1*, . . . ,w *n* ) и скалярный порог w*0* являются параметрами алгоритма; sign — функция знака

Гиперплоскость

Уравнение w ∙x - w 0 =0 описывает гиперплоскость, разделяющую классы в n — мерном пространстве.

Критерий и методы настройки параметров в SVM радикально отличаются от персептронных (градиентных) методов обучения.

Метод опорных векторов использует три основные идеи: а) оптимальная разделяющая гиперплоскость; б) возможно неточное разделение, но за ошибки в разделении платится штраф (математический конечно) и в) нелинейное отображение данных.

а) Оптимальная разделяющая гиперплоскость

Предположим, что выборка линейно разделима, то есть существуют такие значения параметров w, w 0, при которых функционал числа ошибок принимает нулевое значение. Но тогда разделяющая гиперплоскость не единственна, поскольку существуют и другие положения разделяющей гиперплоскости, реализующие то же самое разбиение выборки. Идея метода заключается в том, чтобы разумным образом распорядиться этой свободой выбора. Потребуем, чтобы разделяющая гиперплоскость максимально далеко отстояла от ближайших к ней точек обоих классов.

Это приводит к задаче квадратичного программирования, поскольку ширина плоскости пропорциональна квадрату длины вектора неизвестных весов W.

Она, вообще говоря, трудная (значительно сложнее, чем линейное программирование), но во многих практических случаях успешно решается.

Б) Вторая важная идея — можно применять метод даже тогда, когда множества линейно неразделимы. За ошибки в разделении платить штраф.

Это классическая идея в оптимизации, идущая еще от Лагранжа.

ШТРАФ и разделение

||**w**|| **2** + C Σ *i* **T** ξ *i* → min

y *i* (**w** **x i** -w *0* ) ≥ 1- ξ *i*

Неотрицательные переменные ξ *i* описывают штрафные санкции за то, что пример x **i** с номером i неправильно классифицирован. **w** **x -** это скалярное произведение

Случай ξ i =0

Тогда мы получаем обычную задачу разделения для персептрона, где однако мы ищем оптимальную гиперплоскость ||w|| 2 → min

y i (w x i -w 0 ) ≥ 1

Второе условие означает что выход персептрона w x i-w 0 и еальный выход y i имеют одинаковый знак

Это сложная задача квадратичного программирования. При наличии штрафов имеется еще параметр С, который надо подбирать. Однако разработаны методы ее решения, хотя вообще говоря, она из класса NP.

***NP problems***

Пусть имеется задача со входными данными X. Размер этих данных в битах обозначим |X|.

Принадлежность задачи к классу NP означает, что, вообще говоря, чтобы найти решение нужно не менее Exp( a |X|) шагов, где a > 0 — некоторая постоянная

Если решение найдено, то проверить это можно быстро , в полиномиальное число шагов например, |X| **3**

Многие практически важные задача включая квадратичное программирование в классе NP тем не менее на практике они успешно решаются специальными методами и с помощью эвристических идей

В) Третья идея — ядра(kernels)

Нелинейное отображение в другое пространство с другим скалярным произведением может превращать линейно неразделимые множества в линейно разделимые **x → ψ(x)**

Вообще говоря, если размерность ψ выше, чем х, то мы можем получить линейное разделение образов гиперплоскостью в пространстве ψ**.**

Пространство **x** и пространство **ψ**

Пусть имеются два множества A и В. Они могут быть не разделимы гиперплоскостью. (например, внутренность и внешность эллипса)

Рассмотрим их образы ψ (A) и ψ (B) в результате действия отображения x → ψ (x)

Вообще говоря, если размерность ψ выше чем х, то два образа уже могут быть разделены.

Как это происходит? Рассмотрим пример.

Пример 2 — разделение внутренности А и внешности B эллипса, определенного уравнением.

2x **2** + 3y **2** - xy =1

Множества А и B линейно не разделимы, но в пространстве ψ они разделимы

Вектор **x**=(x, y) мы отображаем так

(E1) Ψ= ( x **2** , y **2** , xy)

Уравнение плоскости в пространстве Ψ для классификатора принимает вид

2Ψ1 + 3Ψ2 - Ψ3 =0

Рисунок .

Общий подход основан на теореме Ковера о разделимости:

**Теорема Ковера о разделимости**

(переведем на русский)

Cover's Theorem is a statement in [*computational learning theory*](https://en.wikipedia.org/wiki/Computational_learning_theory) and is one of the primary theoretical motivations for the use of non-linear [*kernel methods*](https://en.wikipedia.org/wiki/Kernel_methods) in [*machine learning*](https://en.wikipedia.org/wiki/Machine_learning) applications. The theorem states that given a set of training data that is not [*linearly separable*](https://en.wikipedia.org/wiki/Linearly_separable), one can with high probability transform it into a training set that is linearly separable by projecting it into a higher-dimensional space via some non-linear transformation.

Рисунок . Иллюстрация теоремы Ковера

Ядра

Выясняется что во всех основных формулах классификации используется выражение **ψ (x) ψ (y)**.

Поэтому ввели так называемые ядра

Using a kernel, the training procedure estimates w in the low dimension space, but the membership test is done on the sign of (K(w, x) + b) rather than (w · x + b) which was used for the linear case.

C помощью функции **ψ** cтроится так называемое ядро —

K(u, v) = **ψ** (u) · **ψ** (v)

Популярные ядра-

K(u, v) = (u · v + 1) **p**

Gaussian Radial Basis Function:

K(u, v) = exp(− (u − v ) **2** /2σ **2** )

Sigmoidal (hyperbolic separating surface):

K(u, v) = tanh(κ u · v − δ)

Y= sign(w 1 x **2** + w 2 y **2** + w 3 xy - w 0 )

Комментарии

Образы отображения ψ в 3 мерном пространстве - красные точки это внутренность эллипса; зеленые - это внешность. К сожалению, нет четких методов поиска отображений ψ.

Преимущества SVM. Принцип оптимальной разделяющей гиперплоскости приводит к максимизации ширины разделяющей полосы между классами, следовательно, к более уверенной классификации. Градиентные нейросетевые методы выбирают положение разделяющей гиперплоскости произвольным образом, как придётся.

Недостатки. Метод опорных векторов неустойчив по отношению к шуму в исходных данных. Если обучающая выборка содержат шумовые выбросы, они будут существенным образом учтены при построении разделяющей гиперплоскости.

% "разделим" точки плоскости на два класса -

% внутри и вне произвольного эллипса (ax^2 + bxy + cy^2 + dx + ey = 1):

% 1. построение SVM-сети (метод опорных векторов) по указанным данным и классификатору

% 2. классификация новых данных по построенной сети

clear, clc

N = 300; % N - число точек для обучения

% ввод коэффициентов эллипса с помощью диалогового окна

% формируется структура типа cell (массив ячеек), в каждой из которых содержится соответствующий коэффициент

prompt = {'a','b','c','d','e'}; dlg\_title = 'Input'; num\_lines = 1;

% эллипс

def={'1.5', '2', '1.1', '-2','-3'}; % значения коэффициентов по умолчанию

% V - параметры эллипса (массив ячеек)

V = inputdlg(prompt,dlg\_title,num\_lines,def);

% преобразование в типу double

[a b c d e] = deal(V{:}); % копирование содержимого каждой ячейки массива ячеек V в соответствующую переменную

% каждая переменная имеет тип char, требуется преобразование в double

str = strvcat(a,b,c,d,e); % объединение переменных типа char в вектор-столбец

A = str2num(str); % преобразование str в числовой вектор-столбец A (типа double)

% class(A);

a = A(1); b = A(2); c = A(3); d=A(4); e = A(5);

% обращение к функции (пользовательской) построения SVM-сети

kernel = 'quadratic'; % ядро SVM-сети

net\_SVM = ellipse\_Train\_SVM\_test(N, A, kernel)

% !!! классификация совсем не всегда верна c линейным ядром - 'linear' !

% !!! хороший результат дают ядра:

% 'quadratic', 'polynomial', 'rbf' - Gaussian Radial Basis Function,

% зададим координаты произвольной точки плоскости

new\_dot = [1,-0.5]

% проведем ее классификацию с помощью построенной сети и нанесение этой точки на график

output = svmclassify(net\_SVM,new\_dot,'Showplot',true) % классификация новой точки (встроенная ф-я)

% svmclassify - Classify using support vector machine (SVM)

% построим график исходного эллипса

hold on, syms X Y; f = a\*X.^2 + b\*X.\*Y + c\*Y.^2 + d\*X + e\*Y-1;

ellips = ezplot(f); set (ellips,'Color','b');

Рисунок .

new\_dot =

1.0000 -0.5000

output =

1

Обработка сигналов с помощью всплесков (wavelets)

**Всплески**

Наиболее эффективная современная техника обработки и анализа сигналов и изображений связана с всплесками (wavelets) – функциями g( x) =ψ(x) с некими свойствами локализации :

 Для простоты мы рассмотрим проблемы обработки сигналов

Вспслеском называется функция ψ(x), такая, что Интеграл от ψ(x) равен 0 и интеграл по всему пространству от | ψ(x) | сходится то есть выполнены соотношения

 ∫ ψ(x) dx=0, ∫ |ψ(x)| dx < ∞

Где интегралы взяты по всей оси.

Например, можно брать что то в роде

Ψ=x exp(-x 2 /2)

Более сложный пример показан на следующем рисунке (значения функции ψ(x) показаны на вертикально оси)

 Кроме непрерывных и гладких функций рассматривают их дискретные

(кусчоно гладкие) аналоги Ниже показан всплеск

wavelet db16 (by I. Daubechies) предложнный извсетным специалистом в этой области Ингрид Добечиз.

Чтобы понять, как сигналы обрабытываются при помощи всплесков, мы напомним некоторые факты из теории Фурье аппроксимаций

**Пространство сигналов и изображений**

Сигнал f(t) может быть представлен как функция времнного аргумента t определенная на некотором интервале. Если речь идет об изображениях, область определения - это подобласть плоскости.

В общем случае, мы имеем пространство функций f, g,… определенных на некоторой области D

 H= L*2*[D]

В нем определяем линейные операции

f +g, c f

И скалярное произведение

 (f,g)= ∫*D* f(x) g(x) dx.

Так функции становятся векторами- элементами линейного пространства со скалярным произведением

**Норма функции**

Норма функции || f|| определяется как

|| f|| 2 = (f, f)

Если область это интервал, аргумент- время, а функция – это сигнал, то квадрат нормы - это мощность сигнала.

Если мощность бесконечна, или интеграла нет, то такие функции мы исключаем из нашего пространства.

**Ортонормальные Базисы**

Пространство функций устроено как обычное евклидово, но бесконечномерно.

В нем возможны счетные базисы

 (basis) e*1* , e*2* … , e*n  , …*

 такие, что (e*i*, e*j*) =δ*ij*

где δ*ij* символ Кронекера равный нулю, если индексы I, j различны

 и равный 1, если индексы I, j равны.

Любую функцию с ограниченной нормой можно разложить по базису, то есть преставить ее в виде суммы бесконечного ряда

i.e., **f= Σ (f, e*j*) e*j***

**Базис Фурье**

Наиболее знаменитый базис – это Базис Фурье

 sin(nx), cos(nx)

для периодических функций на [0, 2 π]

Однако, в 1990-е годы используя достижения матанализа и теории рядов Фурье, открыли новый базис, который имеет ряд преимуществ

По сравнению с базисом Фурье.Чтобы понять, как построен базис, рассмотрим некоторые операции над всплесками

**Сдвиги и сжатия всплесков**

Пусть ψ(**x**), где аргумент **x** - скаляр, или даже вектор - произвольный всплеск. Всплески можно сдвигать и локализовать. Положим

 ψ*a,* ***h***( **x**)= a **- n/2** ψ((x-**h**)/a)

Параметр **h** определяет сдвиг всплеска по оси **x**. Параметр **a**  определяет локализацию всплеска. Если, например, **a=2** функция

ψ*a,* ***1***( **x**) сжата в два раза по сравнению с исходной функцией ψ(**x**), и ее график сдвинут на 1 направо

Имеется тождество Кальдерона, которое показывает, что линейные комбинации функций ψ*a, h*( x) приближают все функции в норме L\_2

**Тождество Кальдерона** в интерпретации ***Гроссмана- Морле***

Определим всплеск-коэффициенты F(a,h) функции f(**x**) by

F(a,**h**)= (f, ψ*a,* ***h***) = ∫ f(**x**) ψ*a,* ***h***( **x**)d**x**.

Тогда мы можем восстановить функцию по этим коэффициентам

 f(**x**)= ∫ a **– (n+1)** da (∫ F(a,**h**) ψ*a,* ***h***( **x**)d**h)**

 Эта формула может быть доказана с помощью преобразования Фурье

**Особо замечательный ортонормальный базис всплесков**

В результате многолетних исследований

**выяснилось** , что имеются такие особые функции ψ(x), что функции ψ*a, h*( x) образуют

**ортонормированный базис** , где а=2 n и h=k, k, n-целые

Преимущество этого базиса связано с тем, что параметры a, h

являются степенями двойки. Это позволяет построить эффетивные быстрые алгоритмы обработки сигналов.

 Разложение функции по таким особым всплескам имеет вид

***f(x) =* Σ*n, ,…k € Z* β(n, k)2 n/2 ψ(2 n *x − k*)**

 *Коэффициенты* β(n, k) (*wavelet coefficients)* определяются при помощи

**β(n, k)= ∫ f(x) 2 n/2 ψ(2 n *x − k*)dx**

**Алгоритмы аппроксимации сигналов и подавления шума**

Многие реальные сигналы и изображения содержат шумовую компоненту, которая искажает их Например, при изменении цен акций медленное изменение цены, связанное с некими глобальными медленными эволюцией рынка, зашумлено мелкими колебаниями связанными с активностью множества агентов, действующих на рынке

Аналогичный шум имеется в сигналах в электрических цепях и других системах. Поэтому важны алгоритмы, позволяющие обработать сигналы и убрать шумовую компоненту.

Для многих сигналов для этой цели разработаны эффективные алгоритмы. При использовании базиса всплесков они работают также, как аналогичные алгоритмы для Фурье базиса.

Однако всплески ( wavelet analysis ) имеют важное преимущество по сравнению с Фурье анализом (Fourier analysis) Именно, они позволяют компактно представлять сигналы с особенностями (сингуляностями), например, скачками, углами и другими. Для таких сигналов Фурье ряды медленно сходятся и численно не эффективны.

Опишем алогритм подавления шума

 Для подавления шума мы выбираем некий порог и выбрасываем из разложения все

wavelet coefficients которые меньше этого порога

(см Fourier Analysis and Wavelet

Analysis *James S. Walker, 1997)*

При этом для аппроксимации функций с сингулярностями нужно намного меньше членов, чем в Фурье анализе. Это совершило революцию в обработке данных. Мы можем также использовать всплески в сетях РБФ.

Ниже мы приводим примеры такого анализа и аппроксимации

Всплески можно загружать при помощи стандартного набора сигналов

Матлаба. Например,

load db4; При этом загружается всплеск Добечис 4 ( Ingrid Daubechies)

 Также можно загрузить всплески db1 … db10.

Сигналы для анализа также можно загружать при помощи базы Матлаба. Например

load noissin;

 Команда

c = cwt(noissin,1:48,'db4');

делает непрерывное вэйфлет преобразование при помощи всплеска db4

Массив с содержит коэффициенты преобразования

load leleccum;

s = leleccum(1:3920);

ls = length(s);

Perform a one-step decomposition of the signal using the db1 wavelet. Type:

» [cA1,cD1] = dwt(s,'db1');

This generates the coefficients of the level 1 approximation (cA1) and detail

(cD1).

To construct the level 1 approximation and detail (A1 and D1) from the

coefficients cA1 and cD1, type:

» A1 = upcoef('a',cA1,'db1',1,ls);

» D1 = upcoef('d',cD1,'db1',1,ls);

**Displaying the Approximation and Detail.**

**5** To display the results of the level-one decomposition, type:

» subplot(1,2,1); plot(A1); title('Approximation A1')

» subplot(1,2,2); plot(D1); title('Detail D1')

To perform a level 3 decomposition of the signal (again using the db1

wavelet), type:

» [C,L] = wavedec(s,3,'db1');

To extract the level 3 approximation coefficients from C, type:

» cA3 = appcoef(C,L,'db1',3);

**9** To extract the levels 3, 2, and 1 detail coefficients from C, type:

» cD3 = detcoef(C,L,3);

» cD2 = detcoef(C,L,2);

» cD1 = detcoef(C,L,1);

**Reconstructing the Level 3 Approximation.**

**10** To reconstruct the level 3 approximation from C, type:

» A3 = wrcoef('a',C,L,'db1',3);

**Reconstructing the Level 1, 2, and 3 Details.**

**11** To reconstruct the details at levels 1, 2 and 3, from C, type:

» D1 = wrcoef('d',C,L,'db1',1);

» D2 = wrcoef('d',C,L,'db1',2);

» D3 = wrcoef('d',C,L,'db1',3);

cA3 = appcoef(C,L,'db1',3);

>> cD3 = detcoef(C,L,3);

>> cD2 = detcoef(C,L,2);

>> cD1 = detcoef(C,L,1);

>> D3 = wrcoef('d',C,L,'db1',3);

>> D1 = wrcoef('d',C,L,'db1',1);

>> subplot(2,2,1); plot(A3); title('Approximation A3')

Undefined function or variable 'A3'.

>> A3 = wrcoef('a',C,L,'db1',3);

>> subplot(2,2,1); plot(A3); title('Approximation A3')

>> subplot(2,2,1); plot(A3); title('Approximation A3')

>> subplot(2,2,2); plot(D1); title('Detail D1')

>> subplot(2,2,3); plot(D2); title('Detail D2')

Undefined function or variable 'D2'.

>> D2 = wrcoef('d',C,L,'db1',2);

>> subplot(2,2,3); plot(D2); title('Detail D2')

>> subplot(2,2,4); plot(D3); title('Detail D3')

>> A0 = waverec(C,L,'db1');

>> plot(A0)

>> subplot(2,1,1);plot(s);title('Original'); axis off

>> subplot(2,1,2);plot(A3);title('Level 3 Approximation');

axis off

>> A0 = waverec(C,L,'db1');

>> plot(A0)

>> subplot(2,1,1);plot(s);title('Original'); axis off

>> subplot(2,1,2);plot(A3);title('Level 3 Approximation');

axis off