МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

УНИВЕРСИТЕТ ИТМО

С.А. Вакуленко, А.А.Жихарева

ПРАКТИЧЕСКИЙ КУРС ПО НЕЙРОННЫМ СЕТЯМ

Учебное пособие

РЕКОМЕНДОВАНО К ИСПОЛЬЗОВАНИЮ В УНИВЕРСИТЕТЕ ИТМО
по направлению подготовки 13.04.02
в качестве учебного пособия для реализации основных профессиональных образовательных программ высшего образования магистратуры

Санкт-Петербург

2018

Вакуленко С.А., Жихарева А.А. Практический курс по нейронным сетям – СПб: Университет ИТМО, 2018. – 71 с.

Рецензент: Казаков А.Я., доктор физико-математических наук, профессор, Санкт-Петербургский университет технологии и промышленного дизайна

В пособии излагается основное содержание дисциплины «Нейронные сети для систем управления автоматизированным электроприводом»; рассматриваются основные модели теории нейронных сетей: однослойный персептрон, многослойный персептрон, сети радиальных базисных функций и машины опорных векторов. Кроме того, кратко описано глубокое обучение и модели рекуррентных нейронных сетей. В теории рекуррентных сетей изложены некоторые оригинальные результаты авторов и их применения к робототехнике. В связи с сетями РБФ также кратко описаны вейвлеты и их применения при анализе сигналов. Для каждой модели в пособии описываются процедура обучение этой модели при помощи системы Matlab. Приведены соответствующие скрипты с подробными комментариями.
Для понимания пособия достаточны элементарные познания в линейной алгебре и анализе.

УНИВЕРСИТЕТ ИТМО

Университет ИТМО – ведущий вуз России в области информационных и фотонных технологий, один из немногих российских вузов, получивших в 2009 году статус национального исследовательского университета. С 2013 года Университет ИТМО – участник программы повышения конкурентоспособности российских университетов среди ведущих мировых научно-образовательных центров, известной как проект «5 в 100». Цель Университета ИТМО – становление исследовательского университета мирового уровня, предпринимательского по типу, ориентированного на интернационализацию всех направлений деятельности.

© Университет ИТМО, 2018

© Вакуленко С.А., Жихарева А.А., 2018

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ 5

Задачи теории искусственного интеллекта (ИИТ) (Artificial intellect theory) 5
Общие принципы функционирования нейронных сетей 6
Задача классификации объектов 8
Основные модели теории нейронных сетей 9

ОДНОСЛОЙНЫЙ ПЕРСЕПТРОН 11

Геометрическая интерпретация 12
Моделирование логических функций однослойным персептроном 13
Пример Минского 14
Алгоритм обучения однослойного персептрона 14
Иллюстрация работы алгоритма 16
Построение нейронной сети для операции конъюнкции в пакете Matlab 17
Задача о моделировании черного ящика в общей постановке 19
Построение нейронной сети для классификации множеств точек плоскости в пакете Matlab 19
Упражнения 23

МНОГОСЛОЙНЫЙ ПЕРСЕПТРОН 25

Архитектура многослойного персептрона 26
Построение нейронной сети для операции XOR в пакете Matlab 27
Упражнения 31

СЕТИ РАДИАЛЬНЫХ БАЗИСНЫХ ФУНКЦИЙ (RADIAL BASIS FUNCTIONS NETWORK (RBF)) 32

Построение нейронной сети для операции XOR в пакете Matlab 34
Упражнения 37

МЕТОД ОПОРНЫХ ВЕКТОРОВ (SUPPORT VECTOR MASHINE (SVM)) 38

Построение нейронной сети для разделения внешности и внутренности эллипса в пакете Matlab 41

Упражнения 45

ОБРАБОТКА СИГНАЛОВ С ПОМОЩЬЮ ВЕЙВЛЕТОВ (WAVELETS) 46

Вейвлеты 46
Пространство сигналов и изображений 47
Ортонормальные базисы 48
Сдвиги и сжатия вейвлетов 48
Алгоритмы аппроксимации сигналов и подавления шума 49
Упражнения 56

DEEP LEARNING 57

Фундаментальные концепции 57
Краткое обсуждение структуры глубоких нейронных сетей 58
Обратное распространение ошибки 58
Трудности с глубокими моделями 59
Сверточные нейронные сети (CNN, Convolutional neural networks) 59
Автоматическое распознавание речи (Automatic speech recognition) 59
Распознавание образов (Image recognition) 61
Понимание естественного языка 61

РЕКУРРЕНТНЫЕ СЕТИ 62

Применения к созданию управляющего сигнала для роботов 63

ЗАКЛЮЧЕНИЕ 65

ОБЩИЕ ИСТОЧНИКИ И КЛАССИЧЕСКИЕ РАБОТЫ 66

Специальная литература 66

НАПРАВЛЕНИЕ "ЭЛЕКТРОЭНЕРГЕТИКА И ЭЛЕКТРОТЕХНИКА" 68

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 такие 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. Подробнее о неразрешимых глобальных проблемах (которые являются захватывающе интересными) читатель может прочитать в книгах известного английского физика Роджера Пенроуза.

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

Задача обработки сигналов и изображений

При обработке изображения (сигнала) часто необходимо отделить полезную (информативную) компоненту сигнала от шума. Такие задачи получили название задач фильтрации. Задачи подобного рода важны, в частности, в полиграфии. Например, нужно выделить наиболее существенную для пользователя информацию и убрать все остальное. Это сложная задача теории искусственного интеллекта.

Задача распознавания образов. Распознавание речи

Пусть, например, на конвейере движется продукция. Необходимо создать систему технического зрения, позволяющую автоматически различить бракованную и нормальную продукцию.

Другой пример: курсы валют, акций обычно сильно колеблются. Эти колебания связаны с тем, что на рынке имеется много агентов, действующих, как правило, несогласованно. Трейдер решает, в частности, такую задачу — выделить среди этих колебаний так называемый основной тренд, чтобы понять, стоит ли вкладывать свои деньги в акции. Основной тренд представляет собой некую усредненную кривую, где мелкие колебания (шум) должны быть устранены в результате фильтрации.

Еще одна популярная ныне и крайне важная задача о выделении основного тренда — это задача анализа климатических данных с целью выяснения, есть глобальное потепление или нет.

Еще несколько важных задач — это распознавание спама в Интернете, автоматический перевод, распознавание почерка и так далее. Задача распознавания речи также относится к классу задач ИИТ.

Теория систем искусственного интеллекта (ИИ) использует следующие разделы математики:

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

Знать все это не обязательно, хотя и полезно.

Общие принципы функционирования нейронных сетей

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

Итак, что мы знаем о мозге? Очень кратко:

- 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 так называемых сигмоидальных функций. Типичный график такой функции имеет вид, представленный на рис. 1, 2.

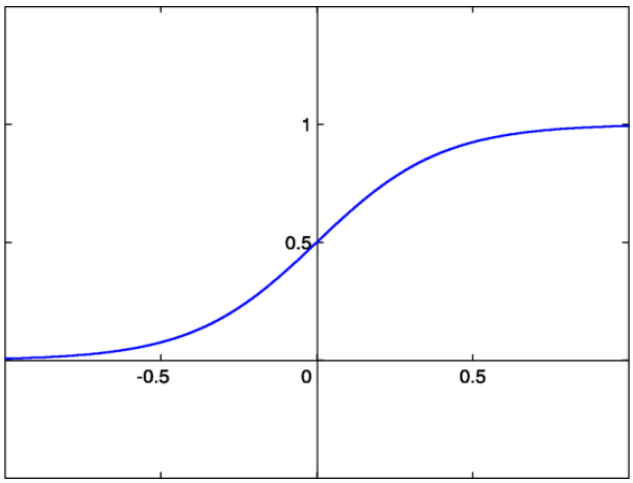

Рисунок 1. Пример сигмоидальной функции

Приведем в качестве примера несколько функций подобного рода:

- $\sigma = \frac{1}{1+e^{-ax}}$, где x — амплитуда входного сигнала, который получает нейрон от других нейронов; σ — выходной сигнал нейрона (рис. 1);
- $\sigma = H(x) = \begin{cases} 1, x > 0 \\ 0, x \le 0 \end{cases}$ — функция скачка, или функция Хевисайда (рис. 2);
- кусочно-линейная (функция $\sigma = max(x,0)$);
- кусочно-полиномиальная.

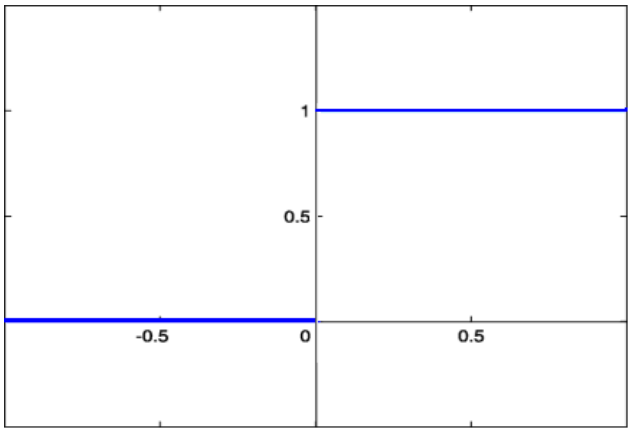

Рисунок 2. Функция Хевисайда

При распознавании образов в мозге происходят сложные процессы. Упрощенно систему распознавания образов можно представить как многослойный персептрон.

Сигналы из многослойного персептрона обрабатываются кортексом; математически это можно рассмотреть как рекуррентную нейросеть.

Силу связи между нейронами описывает матрица синаптических связей (W). Известно, что вся долговременная фундаментальная информация хранится в этой матрице. Матрица синаптических связей медленно меняется со временем в процессе обучения мозга. Итак, в мозге 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. Дональд Хебб — канадский физиолог, который экспериментально показал, что если два нейрона оба часто одновременно активны, то сила связи между ними растет. Аналогичные исследования проводил Е. Конорский. 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, например, преобразование Фурье (С. Вакуленко, 2002).

Задача классификации объектов

Рассмотрим общие принципы функционирования нейронных сетей на примере задачи классификации.

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

Предположим, что мы характеризуем объект с помощью набора признаков $(x_1, x_2, \ldots, x_n)$. 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 n компонентами, или точку k-мерного Эвклидова пространства $X = (x_1, x_2, \ldots, x_n)$. Тогда задача классификации сводится к следующей математической задаче: разделить два множества точек A и B n-мерного эвклидова пространства некоторой гиперповерхностью размерности $n-1$.

Сделаем некоторые важные комментарии. Выбор признаков для классификации является исключительно сложной задачей, которую мы пока не рассматриваем. Ранее эта задача решалась вручную, в последнее время появились эффективные методы автоматического нахождения наиболее эффективных признаков (так называемый Deep Learning, о котором будет отдельно сказано ниже). 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 $X(1), X(2), \ldots, X(P)$, для которых принадлежность объекта к классу A или классу B известна. Чтобы математически формализовать этот факт, определим индикатор:

$$D(X) = \begin{cases} 1, X \in A \\ 0, X \in B \end{cases}.$$

По накопленному в результате обучения «опыту» строим нейросеть (систему ИИ), которая проводит разделяющую поверхность.

Математически этот процесс может быть описан как поиск некоторой функции

$$y = F(X, W),$$

где W — набор параметров нейронной сети (или другой системы ИИ). Для нейросетей эти параметры, в частности, задают силу связи между нейронами и подбираются так, чтобы ошибка обучения (*error training*)

была бы минимальной (как можно ближе к нулю). В качестве ошибки обучения обычно рассматривают функцию

$$E_{train}(W) = \sum_j |F(X(j), W) - D(X(j))|,$$

где $X(j), j = \overline{1,P}$ берутся из обучающего множества.

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

$$E_{test}(W) = \sum_j |F(X(j), W) - D(X(j))|,$$

где $X(j)$ взяты из тестового множества.

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

Основные модели теории нейронных сетей

Выбор модели — скорее искусство, чем наука.

Вначале перечислим основные модели теории нейронных сетей, выделив отдельно два разных класса сетей:

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

Одна из первых моделей 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, в частности, выбор числа слоев, нейронов и связей между ними не так прост.

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 X и на выходе выдает число

$$y = \sigma(S),$$

где

$$S = \sum_k w_k x_k - h = w_1 x_1 + w_2 x_2 + \ldots w_n x_n - h.$$

Входной вектор X состоит из n компонент $X = (x_1, x_2, \ldots, x_n)$, каждая из которых представляет собой численную характеристику анализируемого нейронной сетью объекта.

Через $w_i, i = \overline{1,n}$ и h обозначены параметры персептрона — *синаптические веса* и *порог (сдвиг)* соответственно. Синаптические веса указывают силу влияния конкретной характеристики на выходное значение. Значения этих параметров могут быть как положительными, так и отрицательными.

Отметим, что переменные $x_i, i = \overline{1,n}$ могут быть булевскими, то есть принимать значения 0 или 1. Это означает, что объект обладает данным признаком, если $x_i = 1$, и не обладает, в случае, если $x_i = 0$. В этом случае функция $\sigma(S) = H(S) = \begin{cases} 1, S > 0 \\ 0, S \le 0 \end{cases}$ — функция Хевисайда или функция скачка.

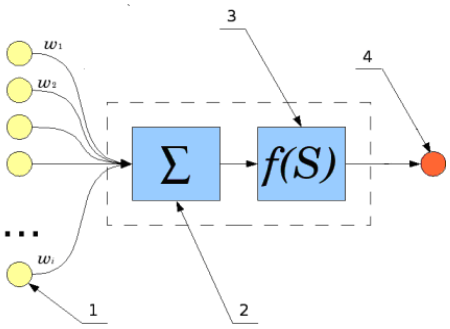

Рисунок 3. Модель однослойного персептрона

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 $X(t)$, между которыми можно провести разделяющую эти множества гиперплоскость).

Геометрическая интерпретация

Однослойный персептрон работает как классификатор объектов, которые математически могут быть заданы как элементы двух разных множеств в многомерных линейных пространствах. Чтобы понять, как это происходит, рассмотрим случай, когда сигмоидальная функция есть функция скачка $\sigma = H(S)$. Предположим, требуется классифицировать объекты в два непересекающихся класса объектов *A* и *B*. Для этого, определим выходные значения: положим $D = 1$, если объект принадлежит к классу *A*, $D = 0$, если объект принадлежит к классу *B*:

$$D(X) = \begin{cases} 1, X \in A \\ 0, X \in B \end{cases},$$

Тогда, из определения функции скачка $\sigma = H(S)$ следует, что объект лежит в классе A при условии $S > 0$ и объект лежит в классе *B*, если $S < 0$.

Следовательно, уравнение $S = 0$ может быть рассмотрено как некий разделитель множеств объектов разных классов.

С другой стороны, уравнение $S = 0$ равносильно уравнению

$$w_1 x_1 + w_2 x_2 + \ldots w_n x_n - h = 0,$$

которое определяет гиперплоскость в *n*-мерном линейном пространстве, состоящем из векторов (строк) $X = (x_1, x_2, \ldots, x_n)$.

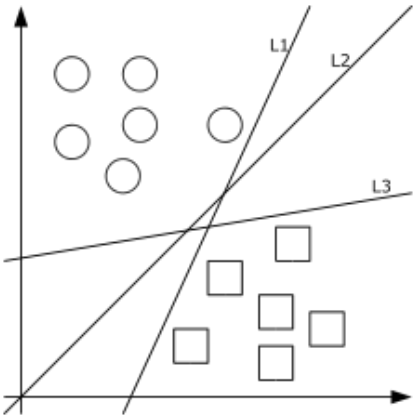

Рисунок 4. Геометрическая интерпретация однослойного персептрона

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

Иллюстрация на рис. 4 демонстрирует вышесказанное на примере игры в квадратики (класс *A*) и шарики (класс *B*) в случае *n=2*, то есть на плоскости. Прямые L_1, L_2, L_3 — примеры «разделителей» этих классов.

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 x_1 и x_2. Представим значения этих переменных как вектор входных значений $X=(x_1,x_2)$, а результат как значение функции скачка, то есть $y=H(S)=H(w_1x_1+w_2x_2-h)$.

Поясним сказанное на примере функции конъюнкции — булевской функции, определенной следующей таблицей:

X_1	0	1	0	1
X_2	0	0	1	1
$X_1 \wedge X_2$	**0**	**0**	**0**	**1**

Наша задача — представить эту логическую функцию с помощью однослойного персептрона. Рассмотрим вектор входных значений (x_1,x_2) как точку плоскости, а результат логической операции — как форму точки: квадрат, в случае, когда результат операции истинен ($y=1$), и круг, если результат операции ложен ($y=0$). Тогда наша задача становится эквивалентной такой задаче классификации: найти прямую, разделяющую два класса объектов — квадраты и круги.

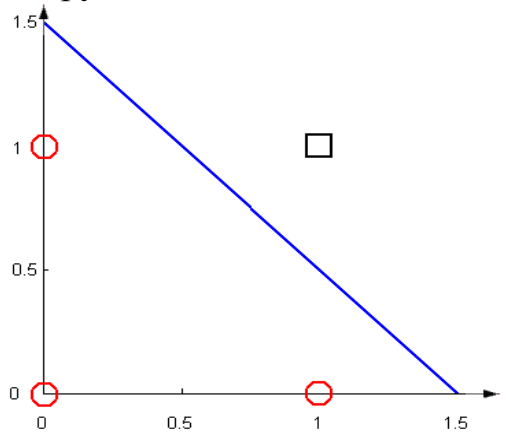

Рисунок 5. Моделирование конъюнкции однослойным персептроном

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

$$x_1+x_2-1.5=0$$

Это означает, что синаптические веса $w_1 = 1$, $w_2 = 1$, а порог $h = 1.5$.

Иллюстрация вышесказанного приведена на рис. 5.

Пример Минского

Однако оказывается, что иногда квадраты и круги неразделимы.

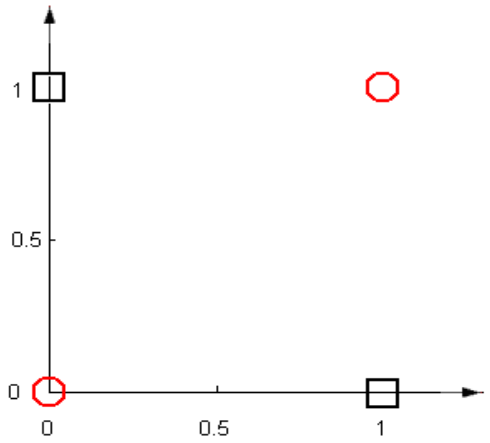

Рисунок 6. Пример Минского

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

$$XOR(x, y) = x + y \ (mod\ 2).$$

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

X_1	0	1	0	1
X_2	0	0	1	1
$XOR(X_1, X_2)$	**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 квадратов, в случае, когда результат операции истинен ($y = 1$), и круга, если результат операции ложен ($y = 0$). Очевидно, что эти два множества (квадраты и круги) не разделимы 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 $X(1), X(2), \ldots, X(P)$, в которых ответ — принадлежность к классу A или B, — известен. Определим индикатор D следующим образом: положим $D(X) = 1$, если X из класса A, и положим $D(X) = 0$, если x из класса B, то есть

$$D(X) = \begin{cases} 1, X \in A \\ 0, X \in B \end{cases}, \quad (1)$$

где всякий вектор x состоит из n компонент: $X = (x_1, x_2, \ldots, x_n)$.

Задача обучения персептрона состоит в нахождении таких параметров $w_1, w_2, \ldots, w_n$ и h, что на каждом обучающем примере персептрон выдавал бы правильный ответ, то есть

$$y(X(i)) = D(X(i)), i = \overline{1,P} \quad (2)$$

Интуитивно кажется очевидным, что если персептрон обучен на большом числе корректно подобранных примеров, и равенство (2) выполнено для почти всех $X(i), i = \overline{1,P}$, то в дальнейшем персептрон будет с близкой к единице вероятностью проводить правильную классификацию для остальных примеров. Этот интуитивно очевидный факт был впервые математически доказан (при некоторых предположениях) в основополагающей работе наших соотечественников В. Вапника и А. Червоненскиса еще в 1960-х годах.

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.

Рассмотрим подробнее алгоритм обучения персептрона.

Шаг 1. Инициализация синаптических весов и смещения.

Значения всех синаптических весов модели полагают равными нулю: $w_i = 0, i = \overline{1,n}$; смещение нейрона h устанавливают равным некоторому малому случайному числу. Ниже, из соображений удобства изложения и проведения операций будем пользоваться обозначением $w_0 = -h$.

Обозначим через $w_i(t), i = \overline{1,n}$ 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 $X = (x_1, x_2, \ldots, x_n)$ 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:

$$w_i(t+1) = w_i(t) + r \cdot (D(t) - y(t)) \cdot x_i(t), i = \overline{0, n-1}$$

где под $D(t)$ 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

$$R(t) = \sum_{t=1}^{T} (D(t) - F(X(t), w(t)))^2 ,$$

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. В результате мы получим указанную выше формулу перенастройки весов.

Иллюстрация работы алгоритма

В процессе обучения вычисляется ошибка

$$\delta(t) = D(t) - y(t).$$

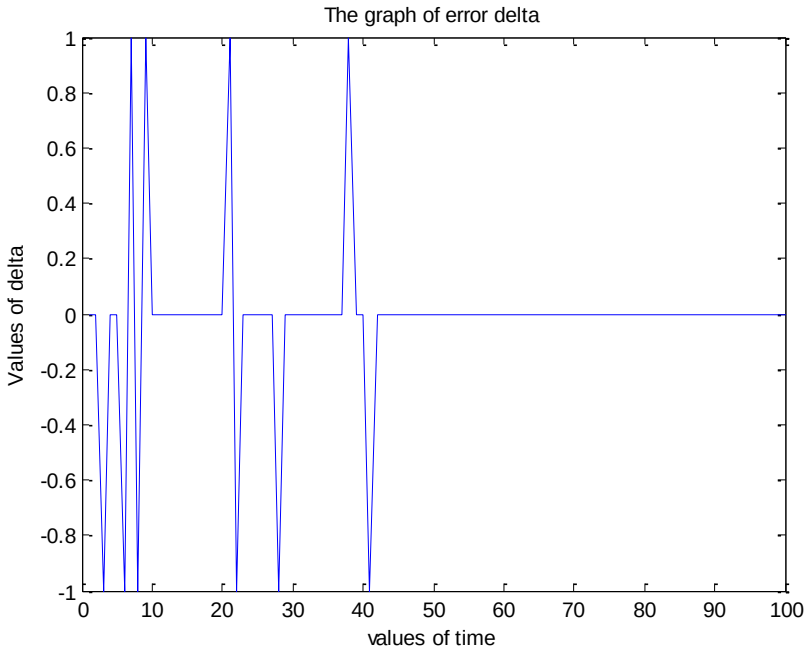

Рисунок 7. График изменения ошибки в процессе обучения нейросети

Построим график, показывающий, как меняется эта ошибка (рис. 7) в ходе обучения сети и адаптации весов. На нем

хорошо видно, что, начиная с некоторого шага, величина $\delta(t)$ равна нулю. Это означает, что персептрон обучен.

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 X_1, X_2 для всех обучающих примеров и представляет собой первые две строки этой таблицы. Целевой вектор T содержит значения индикаторной функции $D(X)$ для всех примеров X (в данном случае 4 примера) и представляет собой последнюю строку этой таблицы. Приведенный ниже скрипт выполняет построение нейронной сети для операции конъюнкции и для удобства содержит подробные комментарии.

```
% аппроксимация конъюнкции однослойным персептроном
% построение функции y = 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
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
net = train(net,P,T);
% симуляция
% вычисление значений аппроксимирующей функции в точках P
y = sim(net, P)
% синаптические веса 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 (2), число уровней персептрона (1), число выходных нейронов (1), число итераций при обучении сети (в нашем случае — 5), время обучения сети.

В то же время согласно инструкциям скрипта в окне команд будет выведена информация по результатам симуляции (вектор y) и значения весовых коэффициентов ($w_1 = 1$, $w_2 = 2$) и коэффициента сдвига ($w_0 = -3$):

```
y =
0     0     0     1
A{1} =
1     2
w_0 =
[-3]
```

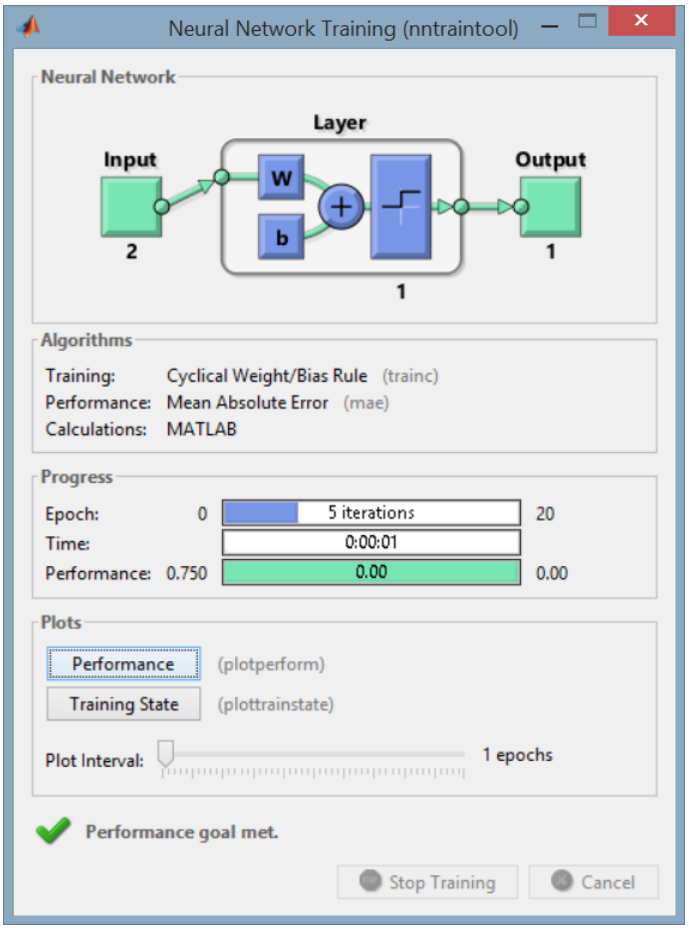

Рисунок 8. Нейронная сеть, реализующая конъюнкцию

Таким образом, получена модель персептрона, разделяющая классы объектов и определяемая уравнением

$$x_1 + 2x_2 - 3 = 0,$$

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

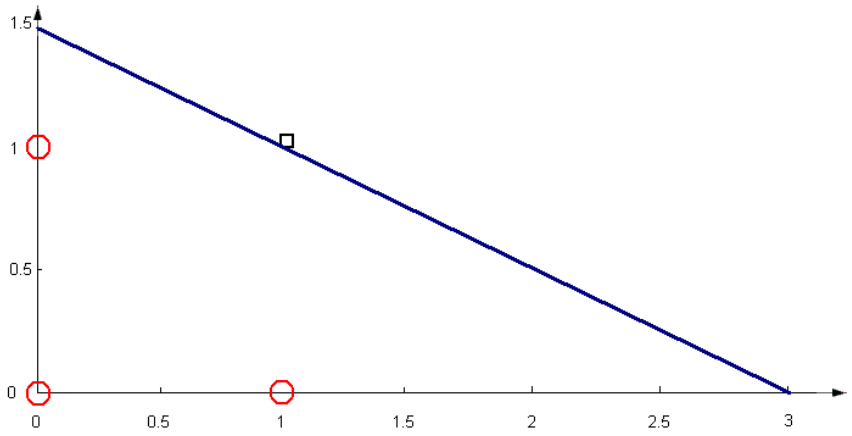

Рисунок 9. Моделирование конъюнкции однослойным персептроном в Matlab

Задача о моделировании черного ящика в общей постановке

Пусть известен реальный выход устройства $D(t)$ при входе $X(t)$, где $X(t)$ — вектор с компонентами $(x_1, x_2, \ldots, x_n)$, t — номер проводимого опыта, $t = \overline{1,T}$ (максимальное число опытов T заранее определено).

Необходимо найти параметры модели $w = (w_0, w_1, \ldots w_n)$ такие, что выход модели $F(X, w)$ и реальный выход устройства $D(t)$ были бы как можно ближе в среднеквадратичном смысле, то есть

$$R(t) = \sum_{t=1}^{T} (D(t) - F(X(t), w))^2 \to min$$

Функцию $R(t)$ 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 *A* и *B* — точки, находящиеся внутри эллипса, лежат в классе *A* и точки, находящихся вне эллипса, лежат в классе *B*. Эллипс определен общим уравнением кривой второго порядка вида

$$a \cdot x^2 + b \cdot xy + c \cdot y^2 + d \cdot x + e \cdot y = 1.$$

Ниже приведен скрипт, реализующий данную классификацию для эллипса $1.5 \cdot x^2 + 2 \cdot xy + 1.1 \cdot y^2 - 2 \cdot x - 3 \cdot y = 1$. 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 (x,y) в пространство размерности 5 следующим образом:

$$(x,y)\to\left(a\cdot x^2,b\cdot x\cdot y,c\cdot y^2,d\cdot x,e\cdot y\right).$$

```
% 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)
% матрица входов
P = [a*x.^2; b*x.*y; c*y.^2; d*x; e*y];
% сформируем вектор выходов T, 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
net = train(net,P,T);
% симуляция
% вычисление значений аппроксимирующей функции в точках P
t = sim(net, P)
% синаптические веса 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);
% построение точек вне эллипса
plot(x_out, y_out, '*r');
clear k;
k = find( t == 1); % индексы точек внутри эллипса
x_out = x(k); y_out = y(k);
% построение точек внутри эллипса
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 (50), число слоев персептрона (1) и используемые модели слоев, число выходных нейронов (1), число итераций при обучении сети (в нашем случае — 50), время обучения сети (в нашем случае — 15 с).

В окне команд будут выведены значения весовых коэффициентов (w_1=*-138.9032*, w_2=*-164.07*, w_3=*-179.1546*, w_4=*-153.6901*, w_5=*-188.0759*) и коэффициента сдвига (w_0=*151*):

```
A{1} =
-138.9032   -164.0700   -179.1546   -153.6901   -188.0759
b =
[151]
```

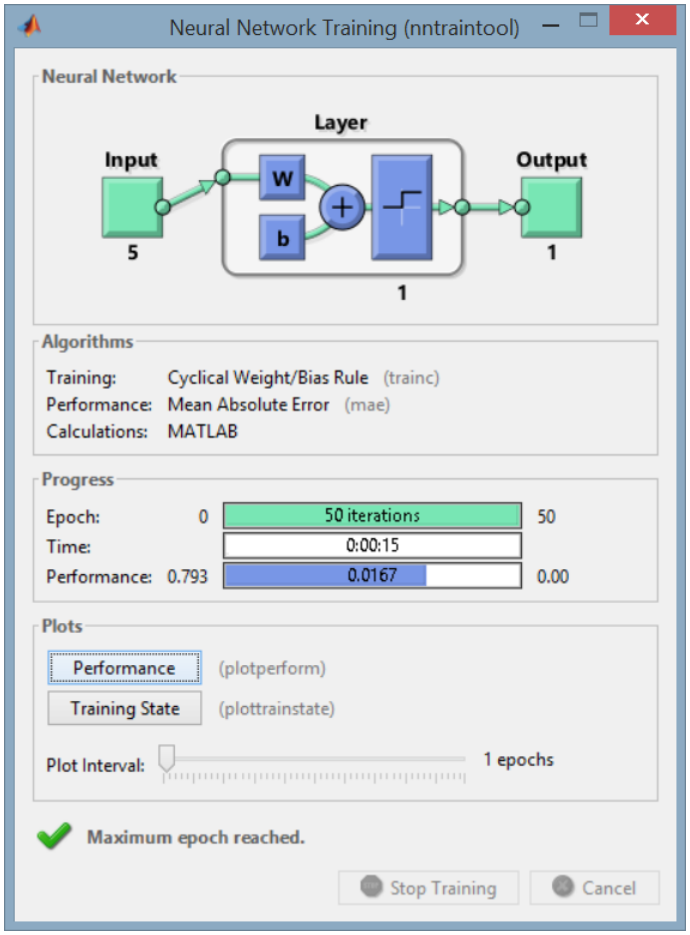

Рисунок 10. Однослойный персептрон

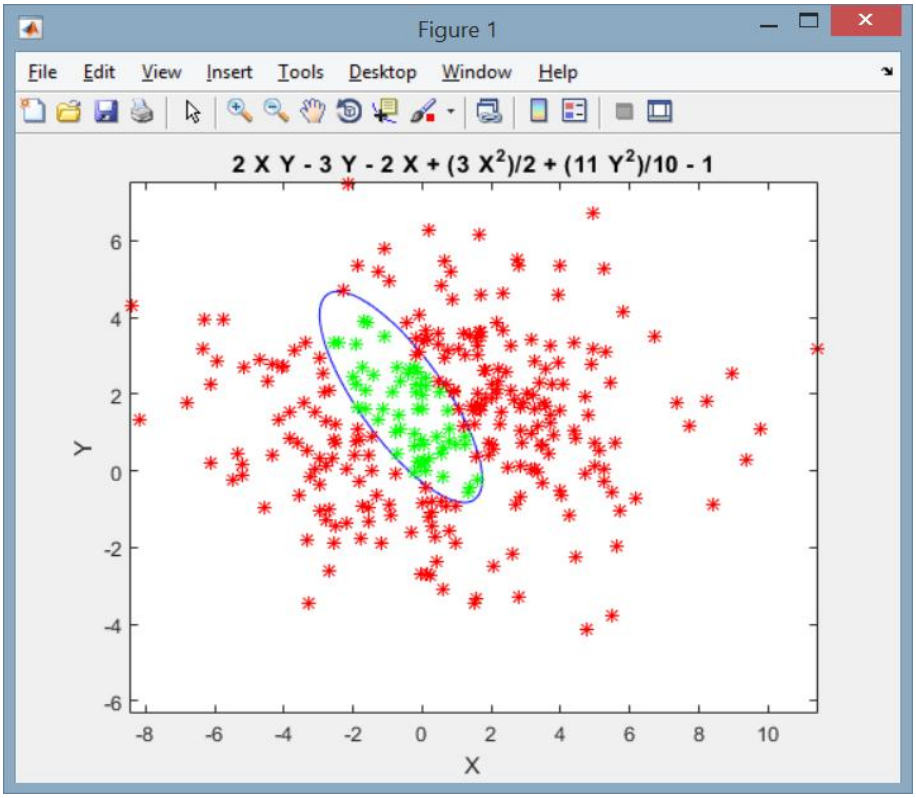

Рисунок 11. Классификация точек однослойным персептроном

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

$$y = H(S) = H(w_1 x_1 + w_2 x_2 + \ldots + w_5 x_5 + w_0) =$$

$$= H(-138.90 \cdot x_1 - 164.07 \cdot x_2 - 179.15 \cdot x_3 - 153.69 \cdot x_4 - 181.07 \cdot x_5 + 151)$$

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

Упражнения

1. Сколько существует булевских функций от двух переменных? От трех?
2. Найти булевские функции от двух переменных, не представимые 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:
   a. в качестве третьего параметра функции *newp* укажите другие модели персептрона (сигмоидальные функции):
      - *'tansig'* — гиперболический тангенс;
      - *'logsig'* — логистический сигмоид;
      - *'purelin'* — линейный сигмоид;
   b. измените максимальное число итераций для проведения обучения сети;
   c. 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;
   d. измените параметры исходного эллипса (коэффициенты кривой).
8. Рассмотрим параболу, заданную в виде общего уравнения кривой второго порядка $a \cdot x^2 + b \cdot xy + c \cdot y^2 + d \cdot x + e \cdot y = 1$. Сгенерируйте случайное множество на плоскости (100 точек). Используя нелинейное отображение $\psi(x,y)$ 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 «снаружи» ветвей гиперболы.

МНОГОСЛОЙНЫЙ ПЕРСЕПТРОН

Пусть известен реальный выход устройства $y(t)$ при входе $X(t)$, где $X(t)$ — вектор с компонентами $(x_1, x_2, \ldots, x_n)$, t — номер проводимого опыта, $t = \overline{1,T}$ (максимальное число опытов T заранее определено).

Необходимо найти параметры модели $w = (w_0, w_1, \ldots w_m)$ и $V_k = (V_{1k}, V_{2k}, \ldots V_{nk})$, h_k, $k = \overline{1,m}$ такие, что выход модели $F(X, V, w)$ и реальный выход устройства $y(t)$ были бы как можно ближе друг к другу. Эта задача всегда разрешима так называемыми многослойными персептронами. Эту модель мы рассматриваем ниже.

Связь между входом и выходом двухслойного персептрона устанавливают следующие соотношения:

$$Z_k = \sigma(V_{1k} x_1 + V_{2k} x_2 + \ldots V_{nk} x_n - h_k), k = \overline{1,m},$$
$$y = \tilde{\sigma}(w_1 Z_1 + w_2 Z_2 + \ldots w_m Z_m + w_0).$$

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-е годы XX века.

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

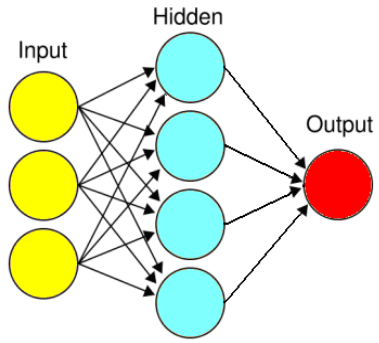

Рисунок 12. Модель многослойного персептрона с одним скрытым слоем (N=1)

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 число нейронов может различаться.

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 в интуитивной формулировке

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

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

Фундаментальная теорема о разделимости классов

Для любой непрерывной функции $f(X)=f(x_1,x_2,\ldots,x_n)$, *определенной на ограниченной замкнутой области, и каждого* $\varepsilon>0$ *существуют коэффициенты* $w=(w_0,w_1,\ldots w_m)$ *и* $V_k=(V_{1k},V_{2k},\ldots V_{nk})$, h_k, $k=\overline{1,m}$, *такие, что*

$$\left| f(X)-\left(\sum_{k=1}^{m} w_k \cdot \sigma(V_{1k}x_1+V_{2k}x_2+\ldots V_{nk}x_n-h_k)+w_0 \right) \right| < \varepsilon .$$

Заметим, что выражение

$$\sum_{k=1}^{m} w_k \cdot \sigma(V_{1k}x_1+V_{2k}x_2+\ldots V_{nk}x_n-h_k)+w_0$$

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 ($n=1$), в этом случае вектор входов представляет собой скаляр: $X=(x_1)$. Это может быть сделано разложением по всплескам (вейвлетам) 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. $n=1$, $X=(x_1)$. Для скалярного x имеем

$$\left| f(X)-\sum_{k=1}^{m} w_k \cdot \sigma(V_k X-h_k) \right| < \varepsilon$$

Мы приближаем производную

$$\left| f'(X) - \sum_{k=1}^{m} w_k \cdot \sigma'(V_k X - h_k) \right| < \varepsilon$$

Введем вейвлет

$$\psi(X) = \sigma'(X) - \sigma'(X-1)$$

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

$$\left| f'(X) - \sum_{k=1}^{m} c_k \cdot \psi(a_k(X - b_k)) \right| < \varepsilon$$

Пункт В. $n > 1$, $X = (x_1, x_2, \ldots, x_n)$, $f(X) = f(x_1, x_2, \ldots, x_n)$.

Из теории рядов Фурье следует, что любую функцию многих переменных $f(X) = f(x_1, x_2, \ldots, x_n)$ можно представить в виде суммы функций одной переменной, то есть

$$f(X) \approx \sum_{k \in D} F_k \cdot e^{i(k,X)},$$

где $k \in D$, D — некоторое дискретное множество векторов $k=(k_1, k_2, \ldots, k_n)$ с целыми координатами. Через (k,X) обозначено скалярное произведение векторов k и X: $(k,X) = k_1 x_1 + k_2 x_2 + \ldots k_n x_n$.

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

$$\sigma(V_k X - h_k) = \tfrac{1}{2}(\sigma(V_k X - h_k) + \sigma(V_k X - h_k)).$$

Замечание.

В доказательстве теоремы одна функция векторного аргумента со скалярными значениями приближалась с помощью двухслойного (входной слой и скрытый слой) персептрона и одним выходом. Векторнозначную 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* (*back propagation error*).

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 скрытом 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;
% скорость обучения
net.trainParam.lr = 0.05;
% точность обучения сети
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 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 (2), число слоев персептрона (2) и моделей слоев, число выходных нейронов (1), число итераций при обучении сети (в нашем случае — 667), время обучения сети.

Замечание. Через поле *IW* структуры сети в *Matlab* обозначен массив ячеек, каждая ячейка *IW{i,j}* которого содержит матрицу весов от входного вектора *j* (источник сигнала) к слою *i* (получатель сигнала). Аналогично, каждая ячейка *LW{i,j}* массива ячеек *LW* содержит матрицу весов от слоя *j* к слою *i*. Например, ячейка *LW{2,1}* содержит матрицу весов W от слоя *2* к слою *1*. При этом, индексы строк элементов матрицы $W=\{w_{k,l}\}$ указывают адресатов весов

нейронов, а индексы столбцов – источник. Так, например, элемент матрицы весов $w_{1,2}$ определяет значение коэффициента для второго элемента входа (слоя-источника) при передаче его на первый нейрон (слоя-приемника).

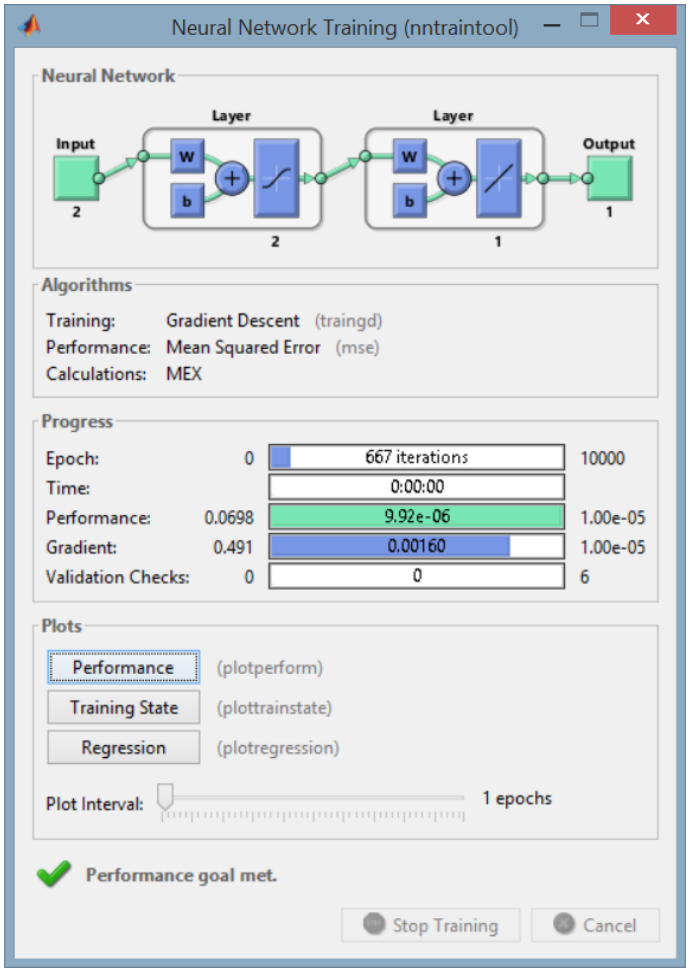

Рисунок 13. Нейронная сеть, реализующая XOR

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):

$$z_1 = \sigma(-1.17 \cdot x_1 - 3.66 \cdot x_2 + 4.51)$$
$$z_2 = \sigma(1.64 \cdot x_1 + 3.49 \cdot x_2 - 0.96)$$
$$y = \tilde{\sigma}(0.99 \cdot z_1 + 0.75 \cdot z_2 - 0.43).$$

Согласно приведенному скрипту в качестве сигмоидальной функции σ для вычисления активности нейронов скрытого слоя использована функция *'tansig'* — гиперболический тангенс; а в качестве сигмоидальной функции $\tilde{\sigma}$ для вычисления активности выходного нейрона — линейный сигмоид (функция *'purelin'*).

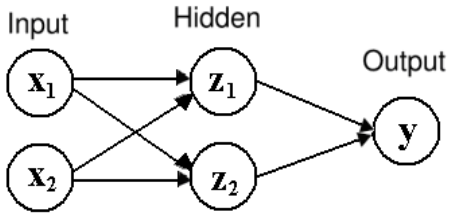

Рисунок 14. Модель персептрона для реализации XOR

Управление числом скрытых слоев персептрона, числом нейронов в каждом из них, моделями каждого слоя, а также методом коррекции весов производится посредством указания соответствующих аргументов встроенной 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 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 приведенном выше скрипте и проанализируйте полученные результаты:
   a. измените модели слоев персептрона, используя сигмоидальные функции
      - *'hardlim'* — функция Хевисайда;
      - *'tansig'* — гиперболический тангенс;
      - *'logsig'* — логистический сигмоид;
      - *'purelin'* — линейный сигмоид;
   b. 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;
   c. измените максимальное число итераций для проведения обучения сети.
2. Найти многослойный персептрон минимальной архитектуры, моделирующий *XOR*.

СЕТИ РАДИАЛЬНЫХ БАЗИСНЫХ ФУНКЦИЙ (RADIAL BASIS FUNCTIONS NETWORK (RBF))

Введем так называемые радиальные базисные функции (RBF). Радиальные базисные функции представляют собой специальный класс функций, характерным свойством которых является наличие центральной точки и монотонное убывание с удалением от этой центральной точки:

$$g(X)=G(\lambda|X-t|),$$

где $X=(x_1,x_2,\ldots,x_n)$, t — некоторая точка, λ — параметр локализации, отвечающий за «крутость» кривой в некоторой окрестности центра кривой t. Если значение параметра λ достаточно велико (больше 3), то график этой функции напоминает узкий «пик», локализованный около центра t.

Такими свойствами обладает, например, функция Гаусса (рис. 15)

$$g(X)=\frac{1}{\sigma\sqrt{2\pi}}\cdot e^{-\frac{(X-t)^2}{2\sigma^2}}.$$

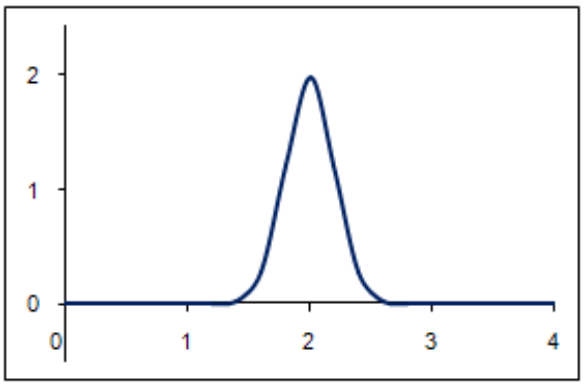

Рисунок 15. Функция Гаусса с центром в точке t=2 и параметром λ=12.5

Система радиальных базисных функций

$$g_j(X)=G(\lambda\cdot\|X-t_j\|),\ j=\overline{1,M}$$

с центрами в точках t_j и общим параметром λ позволяет аппроксимировать любую функцию $f(X)$ с помощью линейной комбинации

$$f(X)=\sum_{j=1}^{M}w_j g_j(X)=F(X,w,M).$$

Последнее выражение определяет сеть радиально-базисных функций (RBF). Такие сети, также как и многослойные персептроны, являются универсальными разделителями и аппроксиматорами, что и утверждается в следующей теореме:

Теорема

Функции $F(X,w,M)$ *являются универсальными аппроксиматорами в компактных областях D,* $X \in D$.

Эта теорема может быть доказана разными сложными методами; здесь мы опустим ее доказательство, однако опишем процедуру поиска неизвестных коэффициентов w_j при фиксированных центрах t_j. Эта процедура, так же как и в случае персептрона, основана на минимизации риска.

Итак, пусть для некоторой неизвестной функции $f(X)$, чье приближение требуется найти, известны ее значения в некотором множестве точек X_j, $j=\overline{1,N}$ — $f(X_j)$. Каждое X_j представляет собой вектор из n компонент: $X_j=(x_{j,1},x_{j,2},\ldots,x_{j,n})$.

В простейшем случае, когда $M=N$ (то есть число радиальных базисных функций совпадает с числом заданных точек, что соответствует числу точек, в которых известно значение признака), в качестве центров радиальных базисных функций возьмем заданные точки, то есть положим $t_j=X_j$, $j=\overline{1,N}$.

Тогда функцию $f(X)$ можно определить как результат интерполяции на узлах X_j, $j=\overline{1,m}$, то есть коэффициенты w_j определяются равенствами

$$F(X_j,w)=f(X_j),\ j=\overline{1,M}$$

что равносильно системе

$$L\cdot w=b$$

где

$$L_{kj}=g_j(X_k),\ b_k=f(X_k),\ j=\overline{1,M},\ k=\overline{1,M}$$

Последняя система легко решается (в том числе средствами пакета *Matlab*).

Более важный и интересный случай представляет собой случай, когда $M<N$. В этом случае функцию $f(X)$ определим с помощью уравнения регрессии, для построения которого воспользуемся методом наименьших квадратов.

В качестве центров возьмем произвольные точки t_j и минимизируем по коэффициентам w_j риск

$$R=\sum_{j=1}^{N}\left(f(X_j)-F(X_j,w)\right)^2.$$

В этом случае также получим линейную систему относительно w_j

$$L \cdot w = b$$

где

$$L_{kj} = \sum_{i=1}^{N} g_j(X_i) \cdot g_k(X_i),\ j = \overline{1,M},\ k = \overline{1,M}$$

$$b_k = \sum_{i=1}^{N} g_k(X_i) \cdot f(X_i),\ k = \overline{1,M}.$$

Как и в предыдущем случае, полученная система легко решается.

Отметим, что в случае фиксированных центров описанная процедура аппроксимации выполняется существенно быстрее по сравнению с обучением многослойного персептрона, поскольку она не требует никаких итераций. Более сложным является случай, когда центры радиальных базисных функций — точки t_j не являются фиксированными. В общем случае эти точки можно искать или выбирать случайно. Известно также, что оптимальный выбор числа радиальных базисных функций M имеет вид $O\left(\sqrt[3]{N}\right)$; скорость аппроксимации сетей RBF — порядка $O\left(\frac{1}{M}\right)$ [1].

Построение нейронной сети для операции XOR в пакете Matlab

В качестве практического примера рассмотрим уже известную нам по предыдущим параграфам задачу реализации функции *XOR*.

Ниже приведен скрипт, решающий эту задачу с помощью радиальных базисных функций средствами пакета *Matlab*. Скрипт снабжен подробными комментариями. Использование радиальных базисных функций предполагает определение дополнительных параметров; каждый из параметров также имеет в скрипте необходимые пояснения.

```
% аппроксимация XOR при помощи 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 RBF
goal = 1e-07;

% параметр λ, определяющий "крутизну" радиальных базисных функций
spread = 3;
```

```
% функция построения сети с помощью RBF
net = newrb(P, T, goal, spread);
% число нейронов в скрытом слое
n_hidden_neurons = net.layers{1}.size

% симуляция
% вычисление значений аппроксимирующей функции в точках P
y = sim(net, P)
% веса входного слоя после обучения сети
A_IW = net.IW;
celldisp(A_IW)
% веса уровней после обучения сети
A_LW = net.LW;
celldisp(A_LW)
% коэффициенты сдвига b после обучения сети
b = net.b;
celldisp(b)
% отображение структуры сети
view(net)
```

В окно команд будет выведена следующая информация:

```
n_hidden_neurons =
3
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
```

Поясним результат. Вектор *y* представляет собой результаты симуляции построенной нейронной сети. Как можно заметить, он в точности совпадает с исходным вектором выходов *T*. Далее следуют весовые коэффициенты и коэффициенты сдвига.

Следует отметить, что радиальная базисная сеть общего вида — это двухслойная нейронная сеть, то есть сеть с одним скрытым слоем, причем число нейронов в скрытом слое, как правило, больше числа входных нейронов. В силу сказанного выведенная в окне команд информация содержит значения весовых коэффициентов для нейронов входного слоя (*A_IW{1}*), нейронов скрытого слоя (*A_LW{2,1}*) и коэффициентов сдвига в этих слоях (*b{1}*, *b{2}*). Число нейронов скрытого слоя — три при двух нейронах на входе сети.

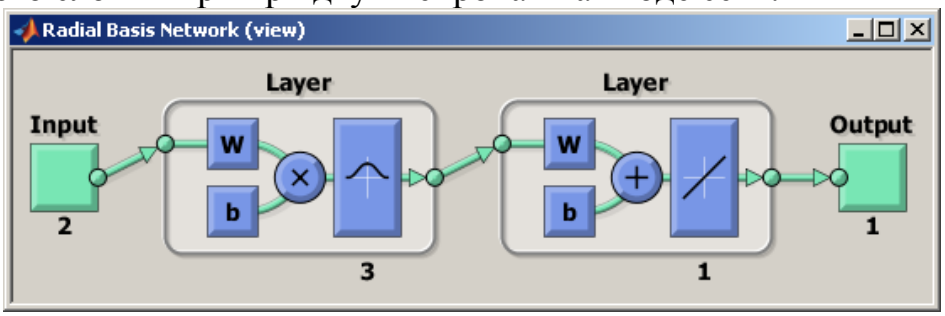

Рисунок 16. Структура построенной радиальной базисной сети

Скрипт заканчивается вызовом встроенной функции

view(net)

которая показывает структуру построенной сети *net* в отдельном графическом окне (см. рис. 16).

Для построения радиальной базисной сети нами была использована встроенная функция *newrb*, имеющая в общем случае следующий синтаксис:

newrb(P, T, goal, spread, MN)

Эта функция добавляет нейроны к скрытому слою радиальной базовой сети до тех пор, пока не достигнет заданного значения среднеквадратичной ошибки *goal* (по умолчанию *0.0*). Параметр *spread* является параметром локализации; чем больше значение *spread*, тем более гладкая будет аппроксимация, однако, очень большие значения этого параметра могут вызвать вычислительные сложности. Параметр *MN* задет максимальное число нейронов в скрытых слоях (по умолчанию это значение в точности соответствует размерности вектора *T*). Обязательными аргументами функции *newrb* являются массив входов (*P*) и вектор выходов (*T*).

Упражнения

1. Воспроизведите приведенный в качестве примера скрипт.
2. Проведите изменения параметров *goal*, *spread* функции *newrb* и проанализируйте полученные результаты.

МЕТОД ОПОРНЫХ ВЕКТОРОВ (SUPPORT VECTOR MASHINE (SVM))

Метод опорных векторов в задачах классификации предложен В. Вапником и его коллегами и получил широкое распространение в начале 2000 гг. [1]. Рассмотрим задачу классификации на два непересекающихся класса, в которой объекты описываются n-мерными вещественными векторами, выход

$$D=\{-1;+1\}.$$

Значение 1 соответствует принадлежности одному классу, а -1 — другому.

Будем строить линейный пороговый классификатор:

$$y(X)=sign(w\cdot X+w_0),$$

где $X=(x_1,x_2,\ldots,x_n)$ — признаковое описание объекта X; вектор $w=(w_1,\ldots w_n)$ и скалярный порог w_0 являются параметрами алгоритма; $sign$ — функция знака. Напомним, что уравнение $w\cdot X+w_0=0$ 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:

- оптимальная разделяющая гиперплоскость;
- возможно неточное разделение, но за ошибки в разделении платится штраф (математический, конечно);
- нелинейное отображение данных.

Рассмотрим каждую из этих идей подробнее.

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 среднее удаление разделяющей плоскости от разделяемых точек пропорционально квадрату длины вектора неизвестных весов W. Задача квадратичного программирования, вообще говоря, трудная (значительно более сложная, чем линейное

программирование), но во многих практических случаях успешно решается.

Штраф за ошибки в разделении

Вторая важная идея — можно применять метод даже тогда, когда множества линейно неразделимы. В этом случае предлагается ввести понятие штрафа и за ошибки в разделении платить этот штраф. Это классическая идея в оптимизации, идущая еще от Лагранжа.

Пусть X^i — i-й пример. Тогда система уравнений метода SVM принимает вид:

$$\|w\| + C\sum_i T\xi_i \to min$$

$$y_i\left(w \cdot X^i + w_0\right) \geq 1 - \xi_i$$

В этой системе неизвестными являются ξ_i и коэффициенты w.

Неотрицательные переменные ξ_i описывают штрафные санкции за то, что пример X^i неправильно классифицирован; здесь $w \cdot X^i$ — скалярное произведение.

Если $\xi_i = 0$, то мы получаем обычную задачу разделения для персептрона, где ищем оптимальную гиперплоскость:

$$\|w\| \to min$$

$$y_i\left(w \cdot X^i + w_0\right) \geq 1$$

Последнее условие означает, что выход персептрона $\left(w \cdot X^i + w_0\right)$ и реальный выход y_i имеют одинаковый знак.

Это сложная задача квадратичного программирования. При наличии штрафов имеется еще параметр C, который надо подбирать. Для этой, вообще говоря, непростой задачи, разработаны методы ее решения.

Напомним некоторые факты о трудных задачах.

Пусть имеется некая задача с входными данными X. Размер этих данных в битах обозначим $|X|$.

Принадлежность задачи к классу трудных задач означает, что, вообще говоря, чтобы найти решение в общем случае, нужно не менее $e^{a|X|}$ шагов, где $a > 0$ — некоторая постоянная.

Однако, во многих случаях, если решение найдено, то проверить это можно быстрее — за полиномиальное число шагов, например, $|X|^3$.

Заметим, что многие практически важные задачи, включая квадратичное программирование, являются трудными, тем не менее, на практике они успешно решаются специальными методами и с помощью эвристических идей.

Нелинейное отображение данных. Ядра (kernels)

Нелинейное отображение в другое пространство с другим скалярным произведением может превращать линейно неразделимые множества в линейно разделимые: $x \to \psi(x)$.

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

Как это происходит?

Пусть имеются два множества A и B. Они могут быть неразделимы гиперплоскостью. Рассмотрим их образы $\psi(A)$ и $\psi(B)$ в результате действия некоторого нелинейного отображения $x \to \psi(x)$.

В качестве примера рассмотрим разделение внутренности A и внешности B эллипса, определенного уравнением.

$$2x^2 + 3y^2 - xy = 1$$

Множества A и B линейно неразделимы в исходном множестве (точек плоскости). Однако, они разделимы в пространстве ψ, к которому можно перейти с помощью отображения

$$\psi = (\psi_1, \psi_2, \psi_3) = (x^2, y^2, xy)$$

Уравнение плоскости в пространстве ψ для классификатора принимает вид

$$2\psi_1 + 3\psi_2 - \psi_3 = 1.$$

Общий подход к применению нелинейных отображений основан на *теореме Ковера о разделимости*. Неформально говоря, эта теорема утверждает, что нелинейное преобразование данных сложной задачи классификации образов в пространство более высокой размерности повышает вероятность линейной разделимости образов (графическая иллюстрация теоремы представлена на рис. 17).

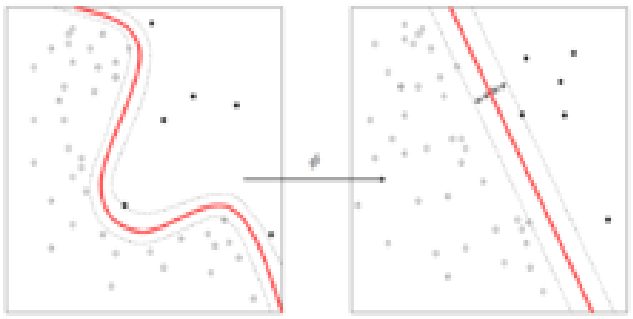

Рисунок 17. Иллюстрация теоремы Ковера

Ядра

Выяснилось, что после нелинейного преобразования ψ во всех основных формулах классификации используется выражение $\psi(x) \cdot \psi(y)$.

Именно, с помощью функции ψ строится так называемое ядро вида

$$K(u,v)=\psi(u)\cdot\psi(v)$$

Данная форма является канонической, тем не менее, также используют и другие ядра. Например:

- $K(u,v)=(u\cdot v+1)^p$
- гауссовская радиальная функция (Gaussian Radial Basis Function)

$$K(u,v)=exp\left(-\frac{(u-v)^2}{2\sigma^2}\right)$$

- гиперболическая разделяющая поверхность (Sigmoidal)

$$K(u,v)=tanh(k\cdot u\cdot v-\delta)$$

К сожалению, четких методов поиска ядер *K* нет.

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

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.

Построение нейронной сети для разделения внешности и внутренности эллипса в пакете Matlab

В качестве практического примера рассмотрим приведенную выше задачу о разделении внутренности и внешности произвольного эллипса.

Ниже приведен скрипт, решающий эту задачу с помощью метода опорных векторов средствами пакета *Matlab*; и используемая в нем пользовательская функция *ellipse_Train_SVM_test*. Оба приведенных скрипта (основной и m-функция) снабжены подробными комментариями. Основной скрипт заканчивается проверкой нейронной сети, то есть проведением классификации произвольно взятой точки плоскости с помощью построенной сети. Также для большей наглядности приведены соответствующие построения на плоскости — исходная кривая, точки двух множеств (помеченные различным образом), построенная сетью разделяющая кривая, тестовая точка.

Напомним, что эллипс задается уравнением

$$a\cdot x^2+b\cdot xy+c\cdot y^2+d\cdot x+e\cdot y=1.$$

Для удобства задания коэффициентов скрипт предполагает их ввод через диалоговое окно *Input*.

```
% разделим точки плоскости на два класса:
% внутри и вне произвольного эллипса ax^2 + bxy + cy^2 + dx + ey = 1

% N - число точек для обучения
N = 300;

% ввод коэффициентов эллипса с помощью диалогового окна
prompt = {'a','b','c','d','e'};
dlg_title = 'Input'; num_lines = 1;

% зададим эллипс
def={'1.5', '2', '1.1', '-2','-3'}; % значения коэффициентов по умолчанию
% V - параметры эллипса (массив ячеек)
% в каждой ячейке содержится соответствующий коэффициент
% окно для ввода параметров эллипса
V = inputdlg(prompt, dlg_title, num_lines, def);

% преобразование к типу double
A = str2double(V);
% запись каждого коэффициента в соответствующую переменную
a = A(1); b = A(2); c = A(3); d=A(4); e = A(5);
% 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');

% определим ядро SVM-сети
kernel = 'quadratic';
% обращение к пользовательской функции построения SVM-сети
net_SVM = ellipse_Train_SVM_test(N, A, kernel)

% зададим координаты произвольной точки плоскости
new_dot = [randn(1)*(a) - d/(2*a), randn(1)*(2*c) - e/(c)]
% проведем классификацию точки с помощью построенной сети
output = svmclassify(net_SVM,new_dot,'Showplot',true)

function [SVMStruct] = ellipse_Train_SVM_test(N, A, kernel);
% метод опорных векторов
% строит SVM для классификации точек плоскости
% в соответствии с заданным эллипсом
% входные данные:
```

```
% A - массив коэффициентов произвольного эллипса
% N - число точек для обучения
% kernel - ядро

% выходные данные
% svmStruct - построенная SVM-сеть

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

% генерируем исходные данные для обучения: координаты точки (x,y),
x = randn(1,N)*(2*a) - d/(2*a); y = randn(1,N)*(2*c) - e/(2*c);
data_training = [x; y];
% формируем классификатор для обучения,
% приписывая -1/+1 точкам вне и внутри эллипса
the_group = ones(1,N);
k = find( (a*x.^2 + b*x.*y + c*y.^2 + d*x + e*y) >= 1);
the_group(k) = -1;

% обучение SVM-сети
% с графическим построением групп данных и разделяющей линии
SVMStruct = svmtrain(data_training, the_group, 'kernel_function', kernel,
'ShowPlot',true);
end
```

В окне команд будет выведена следующая информация: координаты произвольно взятой точки *new_dot* и результат классификации: *1* или *0*, где *1* означает, что точка *new_dot* лежит внутри эллипса, *0* — вне эллипса. Кроме того, будет построена графическая иллюстрация процесса (рис. 18), а именно: тренировочное множество точек отображается красными «крестиками» (точки вне эллипса) и зелеными «звездочками» (точки внутри эллипса); голубой «звездочкой» отмечена произвольно взятая для классификации точка; черный эллипс — построенная с помощью SVM разделяющая кривая; синий эллипс — заданный эллипс.

```
new_dot =
1.0000    -0.5000
output =
1
```

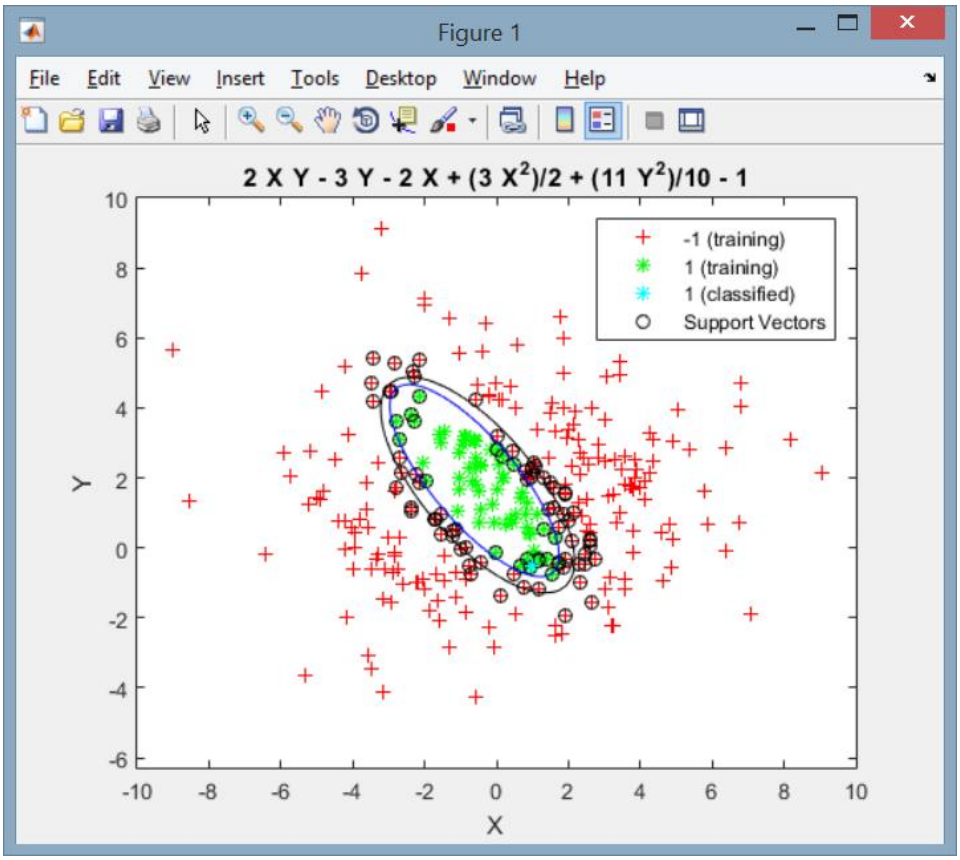

Рисунок 18. Классификация точек плоскости с помощью SVM

Для обучения SVM-классификатора нами была использована встроенная функция *svmtrain*, имеющая в общем случае следующий синтаксис:

svmtrain(Training, Group, Name, Value)

где *Training* — тренировочное множество, для каждой строки которого определено значение классификатора (элемент массива *Group*). При необходимости могут быть заданы дополнительные параметры, которые записываются парами — имя параметра, значение параметра. Так, параметр '*kernel_function*' позволяет определить функцию ядра SVM-сети и может принимать одно из следующих значений

- *'linear'* — линейное ядро (по умолчанию);
- *'quadratic'* — квадратичное ядро;
- *'polynomial'* — полиномиальное ядро (по умолчанию 3-го порядка), данное значение параметра требует указания порядка полинома посредством определения параметра *polyorder* (то есть пары *'polyorder'*, *значение*);
- *'rbf'* — гауссовская радиальная функция (Gaussian Radial Basis Function) с параметром σ=1; при необходимости указания другого значения σ, требуется определить параметр *'rbf_sigma'* (то есть пару *'rbf_sigma'*, *значение*);

- *'mlp'* — многослойное ядро персептрона со шкалой [-1; 1] (по умолчанию); при необходимости указания другого значения необходимо определить параметр *'mlp_params'* (то есть пару *'mlp_params', значение*).

Упражнения

1. Воспроизведите приведенный в качестве примера скрипт. Измените ядро SVM-сети (*'linear'*, *'polynomial'*, *'rbf'*), проведите классификацию в каждом случае, проанализируйте полученные результаты.
2. Проведите классификацию точек плоскости для произвольно заданной гиперболы: точки плоскости, лежащие между двумя ветвями гиперболы, и точки плоскости, лежащие «снаружи» ветвей.

ОБРАБОТКА СИГНАЛОВ С ПОМОЩЬЮ ВЕЙВЛЕТОВ (WAVELETS)

Вейвлеты

Ранее была рассмотрена задача обработки зашумленных сигналов с помощью радиальных базисных функций. Наиболее эффективная современная техника обработки и анализа сигналов и изображений связана с вейвлетами (*wavelets*) – функциями $g(x)=\psi(x)$ с некими свойствами локализации.

Вейвлетом называется функция $\psi(x)$, такая, что интеграл от этой функции равен нулю и интеграл по всему пространству от модуля этой функции сходится, то есть выполнены соотношения

$$\int \psi(x)dx = 0 \ , \ \int |\psi(x)|dx < \infty ,$$

где интегралы взяты по всей оси.

Например, можно взять

$$\psi(x) = x \cdot e^{-\frac{x^2}{2}} .$$

Это не что иное, как производная гауссовской функции. Таким образом, обнаруживаем связь между вейвлетами и радиальными базисными функциями.

Более сложный пример показан на рисунке 19 (значения функции $\psi(x)$ показаны на вертикальной оси)

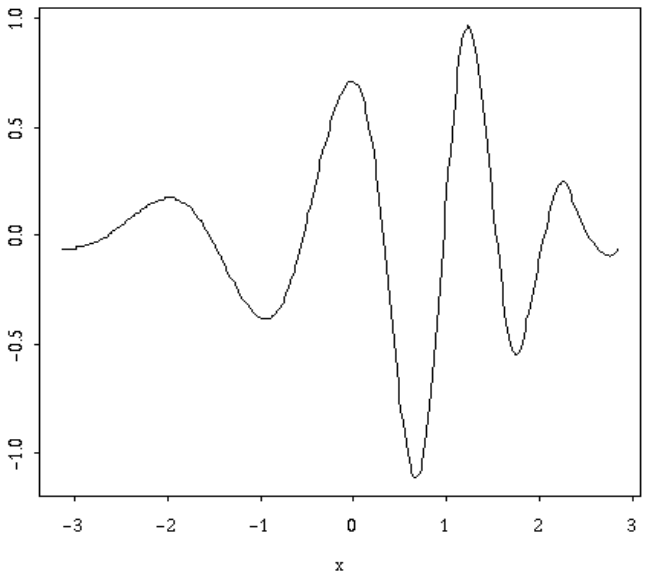

Рисунок 19. Пример вейвлета

Кроме непрерывных и гладких функций, рассматривают их дискретные (кусочно-гладкие) аналоги. На рис. 20 показан вейвлет *db16*, предложенный известным специалистом в этой области Ингрид Добеши (I. Daubechies).

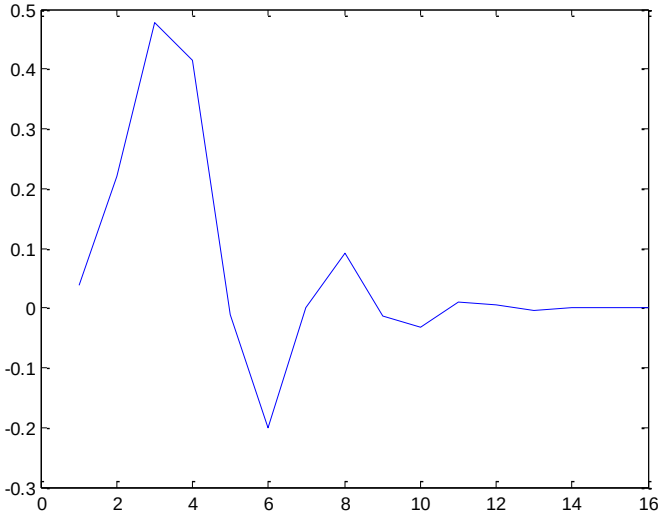

Рисунок 20. Wavelet db16 (by I. Daubechies)

Чтобы понять, как сигналы обрабатываются при помощи вейвлетов, мы напомним некоторые факты из теории Фурье аппроксимаций.

Пространство сигналов и изображений

Сигнал $f(t)$ может быть представлен как функция временного аргумента *t*, определенная на некотором интервале. Если речь идет об изображениях, областью определения является подобласть плоскости.

В общем случае, мы имеем пространство функций *f*, *g*,…, определенных на некоторой области *D*. В этом пространстве определим линейные операции

$$f + g$$
$$C \cdot f$$

и скалярное произведение функций

$$(f, g) = \int_D f(x) \cdot g(x) dx .$$

Так функции становятся векторами – элементами линейного пространства со скалярным произведением

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 функция неинтегрируема, то такие функции мы исключаем из нашего пространства.

Ортонормальные базисы

Пространство функций (обычно они фиксируются с помощью набора некоторых свойств) устроено как обычное евклидово пространство, но бесконечномерно. В нем имеются счетные базисы $e_1, e_2, \ldots, e_n$, такие, что

$$(e_i, e_j) = \delta_{ij}$$

где δ_{ij} — символ Кронекера, равный нулю, если индексы i и j различны, и равный 1, если индексы i и j равны.

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

$$f = \Sigma(f, e_j) e_j \quad f = \sum (f, e_j) e_j$$

Наиболее известный базис – это *базис Фурье*

$$(sin(nx), cos(nx))$$

для периодических функций на $[0; 2\pi]$.

Однако, в 1990-е годы, используя достижения математического анализа и теории рядов Фурье, открыли новый базис, который имеет ряд преимуществ по сравнению с базисом Фурье. Чтобы понять, как построен этот базис, рассмотрим некоторые операции над вейвлетами.

Сдвиги и сжатия вейвлетов

Пусть $\psi(x)$, где аргумент x — скаляр, или даже вектор — произвольный вейвлет. Вейвлеты можно сдвигать и локализовать. Положим

$$\psi_{a,h}(x) = a^{-\frac{n}{2}} \cdot \psi\left(\frac{x-h}{a}\right)$$

Параметр h определяет сдвиг вейвлета по оси X. Параметр a определяет локализацию вейвлета. Если, например, $a=2$ функция $\psi_{a,1}(x)$ сжата в два раза по сравнению с исходной функцией $\psi(x)$, и ее график сдвинут на единицу вправо.

Имеется тождество Кальдерона, которое показывает, что линейные комбинации функций $\psi_{a,h}(x)$ приближают все функции в квадратичной норме L_2.

Тождество Кальдерона в интерпретации Гроссмана-Морле

Определим вейвлет-коэффициенты $F(a,h)$ функции $f(x)$ как

$$F(a,h) = (f, \psi_{a,h}) = \int f(x) \cdot \psi_{a,h}(x) dx .$$

Тогда мы можем восстановить функцию по этим коэффициентам

$$f(x) = \int a^{-(n+1)} da \left(\int F(a,h) \cdot \psi_{a,h}(x) dh \right)$$

Эта формула может быть доказана с помощью преобразования Фурье.

Особо замечательный ортонормальный базис вейвлетов

В результате многолетних исследований выяснилось, что имеются такие особые функции $\psi(x)$, что функции $\psi_{a,h}(x)$ образуют ортонормированный базис, где $a = 2^n$ и $h = k$, k, n — целые.

Преимущество этого базиса связано с тем, что параметр a является степенями двойки. Это позволяет построить эффективные быстрые алгоритмы обработки сигналов.

Разложение функции по таким особым вейвлетам имеет вид

$$f(x) = \sum_{n,k \in Z} \beta(n,k) \cdot 2^{\frac{n}{2}} \cdot \psi\left(2^n x - k\right)$$

Коэффициенты $\beta(n,k)$ (*wavelet coefficients)* определяются при помощи равенства

$$\beta(n,k) = \int f(x) \cdot 2^{\frac{n}{2}} \cdot \psi\left(2^n x - k\right) dx.$$

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 некоей глобальной медленной эволюцией рынка, зашумлено мелкими колебаниями, связанными с активностью множества действующих на рынке агентов.

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 вейвлетов они работают так же, как аналогичные алгоритмы для Фурье базиса.

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

Опишем алгоритм подавления шума.

Для подавления шума выберем некий порог и выбросим из разложения все коэффициенты вейвлета, 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. Этот факт позволил совершить революцию в обработке данных. Мы можем также

использовать вейвлеты в сетях RBF. Ниже приведены примеры такого анализа и аппроксимации.

В состав *Matlab* входит стандартный набор вейвлетов и сигналов. Для использования каждого из них, необходимо загрузить требуемый вейвлет и/или сигнал с помощью команды

load <имя_вейвлета>;

Так, следующая команда загружает вейвлет Добеши 4 (*Ingrid Daubechies*), после чего выполняется построение этого вейвлета (рис. 21):

```
load db4;
plot(db4); title('db4')
```

Аналогично можно загрузить вейвлеты *db1*, …, *db10*.

Аналогичным образом при помощи базы *Matlab* загружаются сигналы для анализа. Например, следующая команда загружает синусоидальный зашумленный сигнал *noissin*, чей график приведен на рис. 22.

```
load noissin;
```

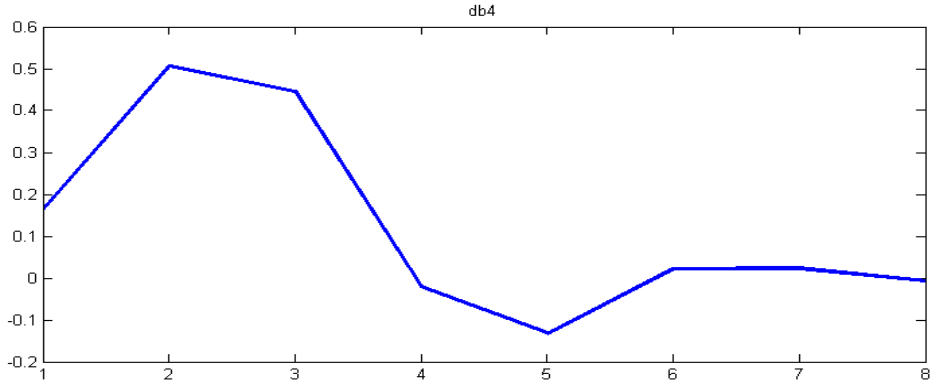

Рисунок 21. Вейвлет db4 (Daubechies 4)

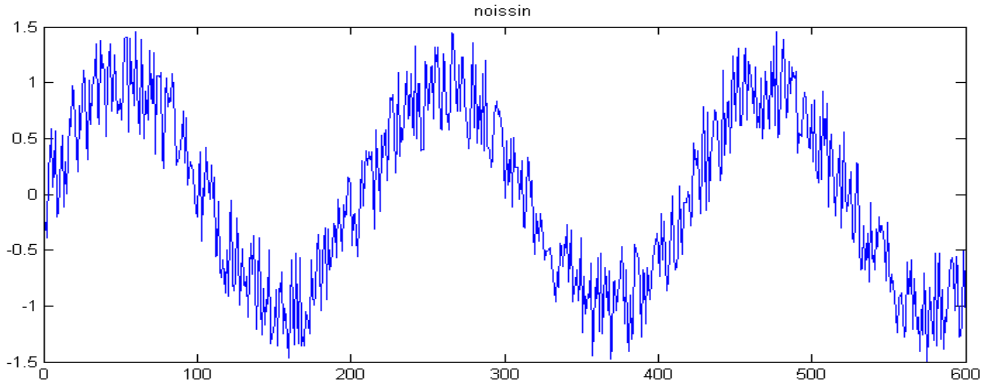

Рисунок 22. Сигнал noissin

Выполним непрерывное вейвлет-преобразование при помощи вейвлета *db4*:

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

Аргументы функции *cwt* определяют исследуемый сигнал, шкала анализа и используемый для преобразования вейвлет. Результатом приведенной команды является массив коэффициентов этого преобразования, каждая строка которого соответствует одной шкале. Добавив в команду *cwt* четвертый аргумент *'plot'*, построим графики абсолютных значений коэффициентов непрерывного вейвлет-преобразования (рис. 23), при этом используем другую шкалу для получения более четкого представления поведения сигнала:

```
C = cwt(noissin,2:4:128,'db4','plot');
```

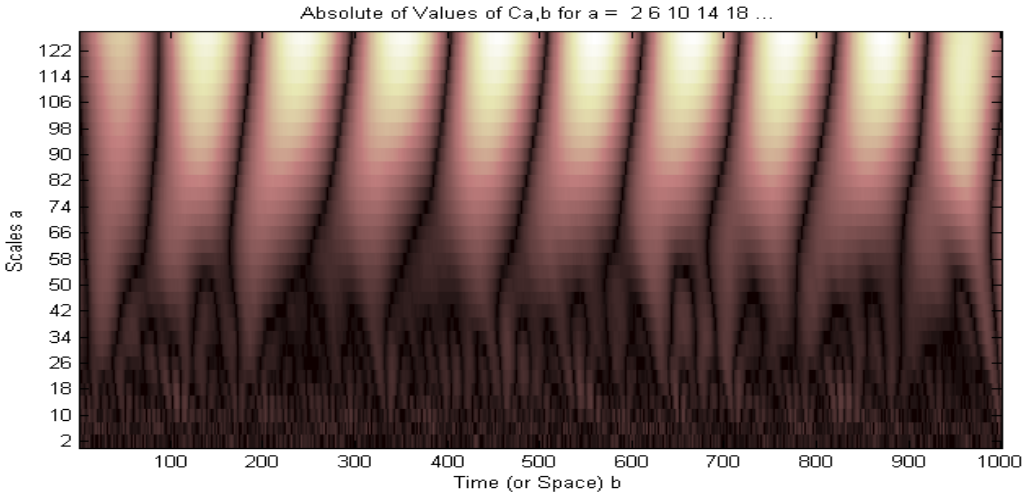

Рисунок 23. Абсолютные значения коэффициентов вейвлет-преобразования

В качестве четвертого аргумента команды *cwt* можно указать параметр *'scal'* или *'scalCNT'*. В этом случае, помимо набора коэффициентов, мы получим построение анализируемого сигнала и масштабированное изображение *scalogram* (в первом случае, рис. 24) или контурное представление *scalogram* (во втором случае, рис. 25).

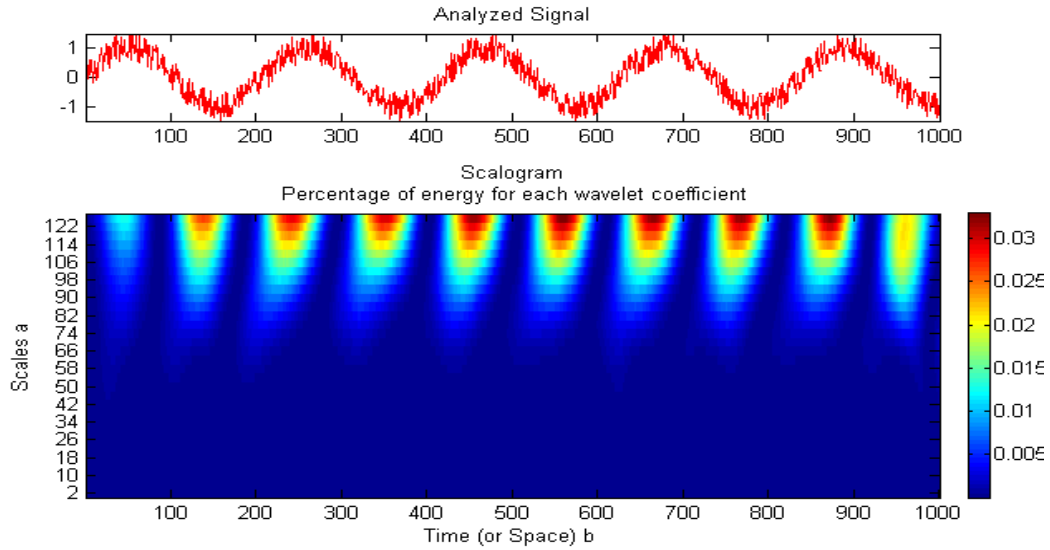

Рисунок 24. Scalogram (масштабированное изображение)

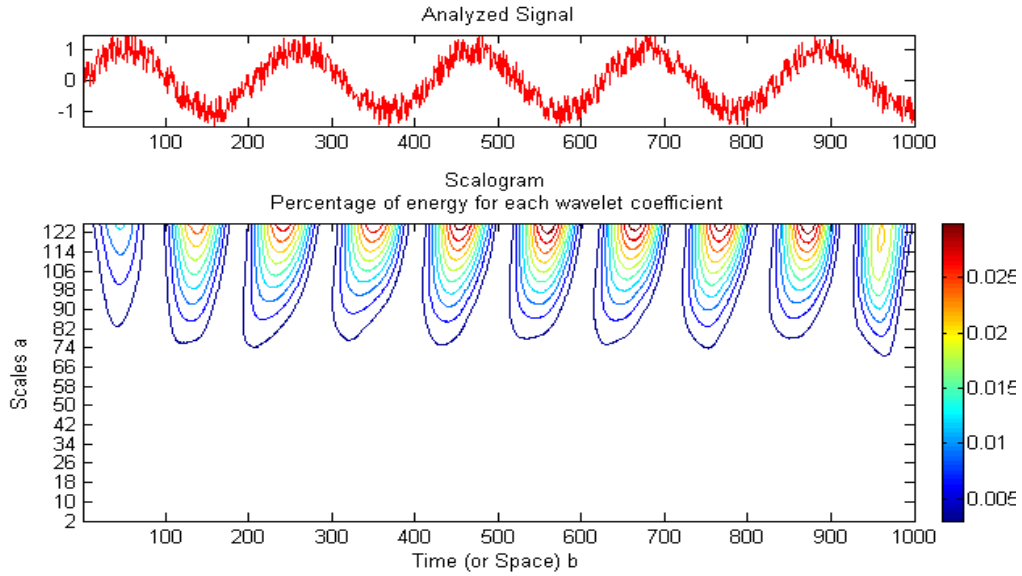

Рисунок 25. Scalogram (контурное представление)

Теперь обсудим дискретное вейвлет-преобразование сигналов. Для этого загрузим сигнал *noissin* (данный сигнал отражает реальное потребление электроэнергии, измеренное в течение трех дней):

```
load noissin;
```

По графическому представлению сигнала (рис. 22) очевидно, что имеет место зашумление сигнала, возникающее вследствие сбоя измерительной аппаратуры.

Проведем дискретный вейвлет-анализ, убирающий этот шум.

Выделим из сигнала *noissin* некоторую часть, из которой и будем убирать шумы:

```
s = noissin (1:600);
l_s = length(s);
```

Для дискретного одноуровневого преобразования сигналов в *Matlab* предназначена функция *dwt*, имеющая в простейшем случае два аргумента — сигнал и вейвлет, и возвращающая в качестве результата два вектора — вектор коэффициентов аппроксимации (*approximation*) и вектор коэффициентов шумов (*detail*).

Выполним одноуровневое разложение исходного сигнала с использованием вейвлета *db4*. Результат запишем в вектора *cA1* (*approximation*) и *cd1* (*detail*):

```
[cA1,cD1] = dwt(s,'db1');
```

Представим полученные результаты графически (см. рис. 26), предварительно восстановив сглаженный сигнал и шумы в соответствующие вектора исходной размерности:

```
A1 = upcoef('a',cA1,'db1',1,ls);
D1 = upcoef('d',cD1,'db1',1,ls);
subplot(1,2,1); plot(A1); title('Approximation A1')
subplot(1,2,2); plot(D1); title('Detail D1')
```

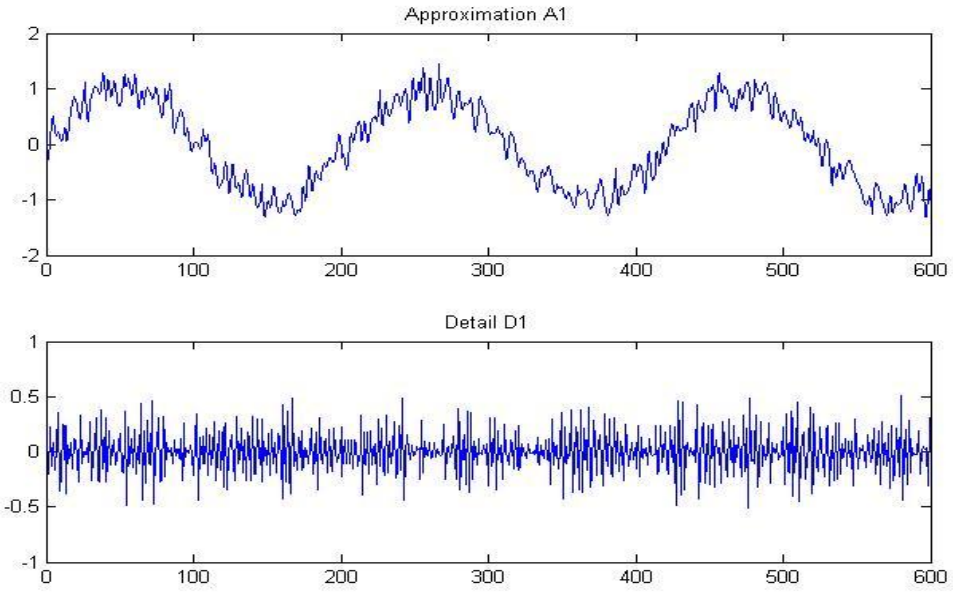

Рисунок 26. Одноуровневое дискретное вейвлет-преобразование

По полученным коэффициентам аппроксимации и шумов можно также провести восстановление исходного сигнала командой

```
A0 = idwt(cA1,cD1,'db1',l_s);
plot(A0); title('A0')
```

Как видно на рис. 27, восстановленный сигнал полностью идентичен исходному, представленному на рис. 22.

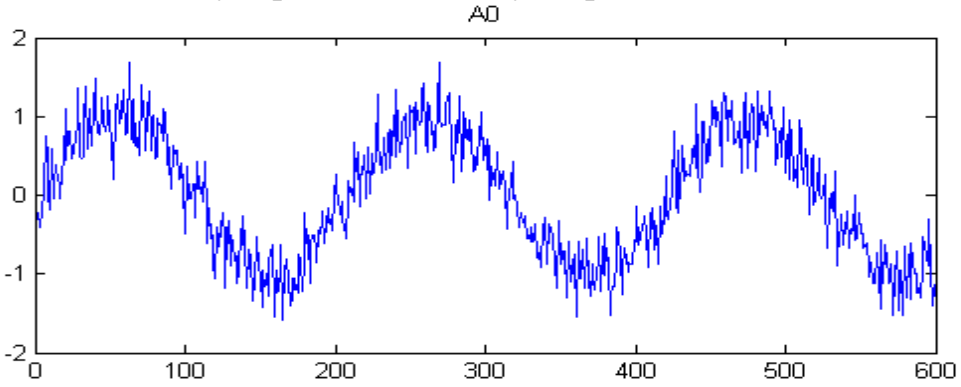

Рисунок 27. Восстановленный сигнал

Возможно также выполнение и многоуровневого вейвлет-преобразования. В этом случае на 1-м уровне из исходного сигнала *S* будут получены вектора коэффициентов аппроксимации и шумов — *cA1*, *cD1*; далее вектор аппроксимации 1-го уровня *cA1* «разделяют» на вектор коэффициентов аппроксимации и шумов 2-го уровня — *cA2*, *cD2*; и т. д. (см. рис. 28). Таким образом, в результате вейвлет-преобразования уровня *N*, мы будем иметь (*N+1*) вектор коэффициентов — вектор коэффициентов аппроксимации уровня *N* и *N* векторов шумов, выделенных на каждом уровне.

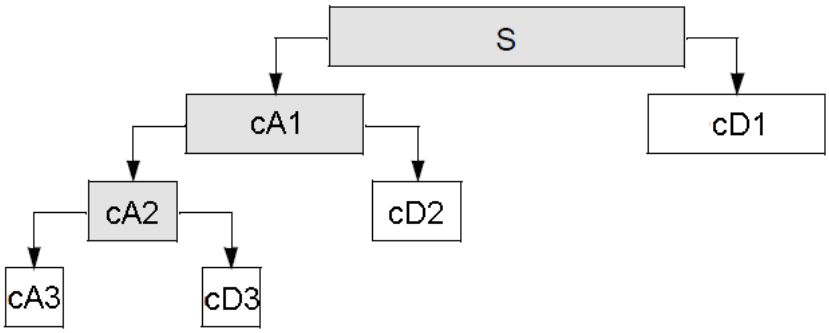

Рисунок 28. Формирование векторов коэффициентов при многоуровневом дискретном вейвлет-преобразовании

В *Matlab* такое преобразование выполняет функция *wavedec*, в качестве аргументов которой необходимо указать исследуемый сигнал, уровень разложения и используемый для преобразования вейвлет. Возвращаемыми значениями являются вектор *C* коэффициентов всех компонент разложения (то есть «склеенные» в одно целое вектор коэффициентов аппроксимации уровня *N* и *N* векторов коэффициентов шумов) и вектор *L* длин каждой из компонент.

В качестве примера проведем с тем же сигналом *noissin* преобразование 3-го уровня, используя уже знакомый вейвлет *db4*.

```
[C,L] = wavedec(s, 3, 'db4');
```

Для извлечения коэффициентов аппроксимации уровня *N* и коэффициентов шумов соответствующих уровней воспользуемся функциями *appcoef* и *detcoef* соответственно. Например,

```
% извлечение коэффициентов аппроксимации 3-го уровня
cA3 = appcoef(C, L, 'db4', 3);
% извлечение «шумовые» коэффициентов 2-го уровня
cD2 = detcoef(C, L, 2);
```

Восстановим сглаженный сигнал и шумы всех уровней в соответствующие вектора исходной размерности:

```
% Reconstructing the Level 3 Approximation
A3 = wrcoef('a', C, L, 'db4', 3);
% Reconstructing the Level 1, 2, and 3 Details
D1 = wrcoef('d', C, L, 'db4', 1);
D2 = wrcoef('d', C, L, 'db4', 2);
D3 = wrcoef('d', C, L, 'db4', 3);
% графические построения
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')
subplot(2,2,3); plot(D2); title('Detail D2')
subplot(2,2,4); plot(D3); title('Detail D3')
```

После чего построим графическую интерпретацию полученных результатов. Так на рис. 29 показан сглаженный после аппроксимации

3-го уровня сигнал (*A1*) и шумы, выделенные (убранные) на каждом из уровней аппроксимации (*D1*, *D2*, *D3*).

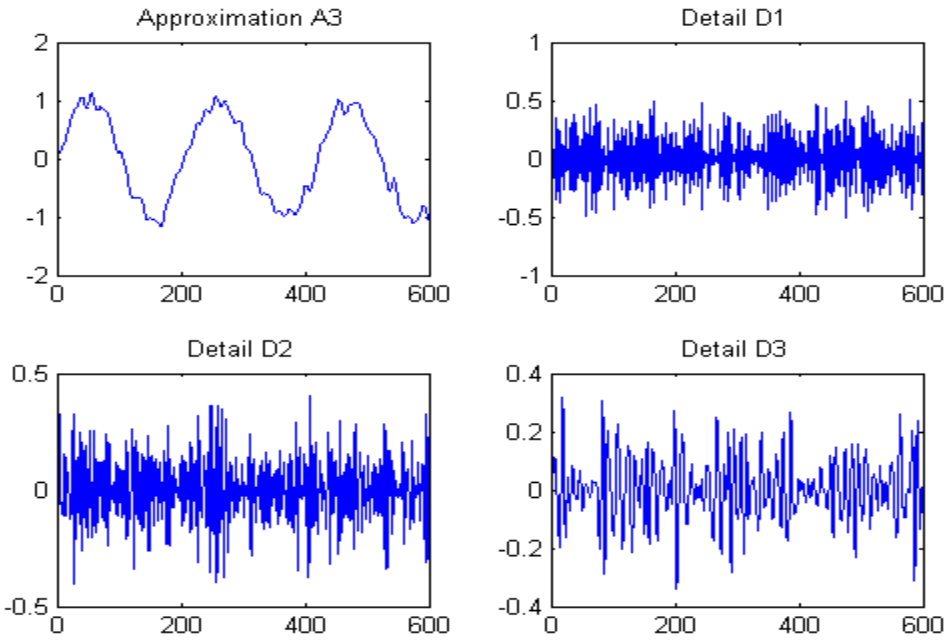

Рисунок 29. Многоуровневое дискретное вейвлет-преобразование

Для лучшего понимания результата сглаживания сигнала построим в одном графическом окне исходный сигнал и результаты аппроксимации 1-го уровня (полученный ранее) и аппроксимации 3-го уровня (рис. 30).

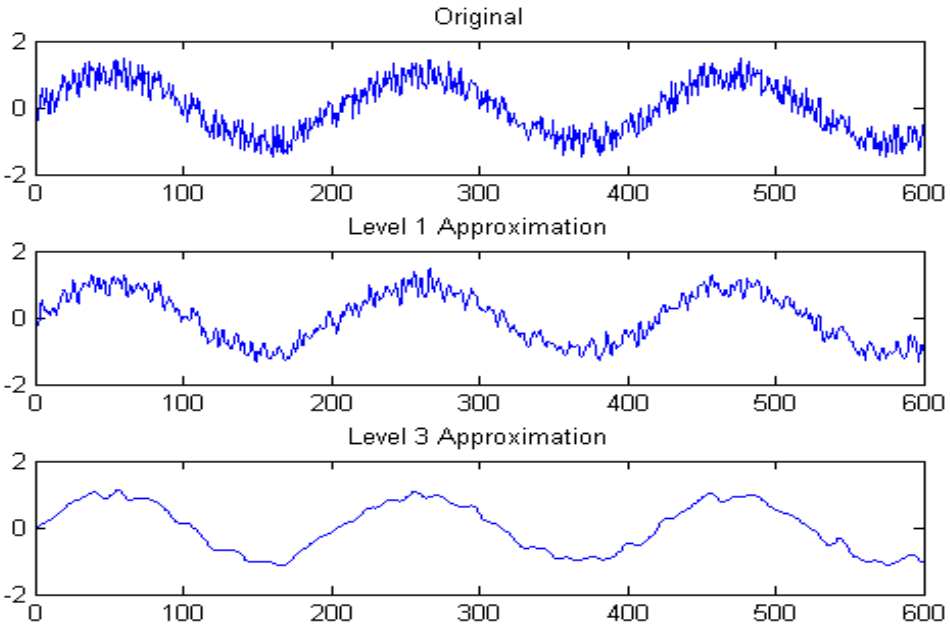

Рисунок 30. Исходный сигнал и результаты вейвлет-преобразований первого и третьего уровней

Восстановление исходного сигнала по полученным ранее коэффициентам аппроксимации и шумов можно провести командой
A0 = waverec(C, L, 'db4');

В заключении приведем перечень семейств вейлетов, доступных в *Matlab*:

Сокращенное название семейства	*Семейство вейвлетов*
'haar'	Haar wavelet
'db'	Daubechies wavelets
'sym'	Symlets
'coif'	Coiflets
'bior'	Biorthogonal wavelets
'rbio'	Reverse biorthogonal wavelets
'meyr'	Meyer wavelet
'dmey'	Discrete approximation of Meyer wavelet
'gaus'	Gaussian wavelets
'mexh'	Mexican hat wavelet
'morl'	Morlet wavelet
'cgau'	Complex Gaussian wavelets
'shan'	Shannon wavelets
'fbsp'	Frequency B-Spline wavelets
'cmor'	Complex Morlet wavelets

Более подробная информация о каждом семействе вейвлетов будет выведена после вызова команды
waveinfo (<имя_вейвлета>);

Упражнения

1. Выполните непрерывное и дискретное преобразование сигнала *leleccum* с помощью 7 различных вейвлетов. Проанализируйте полученные результаты.
2. Выполните непрерывное и дискретное преобразование сигнала *vonkoch* с помощью 7 различных вейвлетов. Проанализируйте полученные результаты.

DEEP LEARNING

Глубокое обучение является частью более широкого семейства методов машинного обучения на основе изучения представлений данных. Наблюдения, например изображение, могут быть представлены во многих формах, таких как вектор значений интенсивности на пиксель, или в более абстрактной форме в виде набора ребер, областей определенной формы и т. д. Некоторые представления лучше других, что позволяет упростить задачу обучения (например, распознавания лиц или распознавание выражения лица). Одно из преимуществ глубокого обучения состоит в автоматическом выборе признаков.

Исследования в этой области нацелены на выбор наилучшего представления объектов. Некоторые из представлений используют достижения в области нейробиологии, в частности, основываются на интерпретациях обработки информации и коммуникационных связей в нервной системе, таких как нейронное кодирование, которое пытается определить взаимосвязь между различными стимулами и связанные с ними нервные реакции в головном мозге.

Различные специальные нейросетевые архитектуры, такие как глубокие нейронные сети (DNN), сверточные нейронные сети (CNN), были успешно применены в таких областях, как компьютерное зрение, автоматическое распознавание речи, обработка естественного языка, биоинформатика. Недавно появилась фундаментальная монография о Deep Learning [12].

Фундаментальные концепции

Глубокие модели основаны на распределенных представлениях. Основным предположением теории распределенных представлений является то, что наблюдаемые данные генерируются с помощью взаимодействия факторов, организованных в слои. Глубокое обучение предполагает, что эти слои факторов соответствуют уровням абстракции или композиции. Изменяя количество слоев и размеров слоев, мы можем достичь различных уровней абстракции.

Глубокое обучение использует идею иерархических объясняющих факторов, где более высокий уровень, более абстрактные понятия, извлекаются из нижних уровней. Эти архитектуры часто строятся слой за слоем. Глубокое обучение помогает распутать эти абстракции и обнаружить, какие функции могут быть полезны для изучения.

Важное преимущество глубоких моделей — это то, что для них были развиты алгоритмы обучения без супервайзера (учителя).

Краткое обсуждение структуры глубоких нейронных сетей

Глубокая нейронная сеть (DNN) является искусственной нейронной сетью (ИНС) с несколькими скрытыми слоями между входным и выходным слоями. Подобно обычной ИНС, DNN может моделировать сложные нелинейные отношения. DNN архитектуры, например, для обнаружения объекта и синтаксического анализа, генерируют композиционные модели, где объект выражается в виде слоистой композиции примитивов изображения.

DNN, как правило, выполнены в виде сетей прямого распространения, но здесь также очень успешно применяются рекуррентные нейронные сети.

Сверточные нейронные сети (CNN) используются в области компьютерного зрения. CNN также были применены к акустическому моделированию для автоматического распознавания речи (ASR), где они были успешны по сравнению с предыдущими моделями (был период, когда в этой области доминировали скрытые марковские модели). Мы кратко рассмотрим историю методов обучения DNN ниже.

Обратное распространение ошибки

DNN может быть обучена стандартным алгоритмом обратного распространения ошибки. Согласно различным источникам, основы непрерывного обратного распространения были получены в контексте теории управления Генри Дж. Келли в 1960 г. и Артуром Э. Брайсоном в 1961 г., который применил принципы динамического программирования. В 1962 г. Стюарт Дрейфус опубликовал более простой вывод, основанный только на цепном правиле. В 1970 г. Сеппо Линнайнмаа опубликовал общий метод автоматического дифференцирования дискретных связанных сетей с помощью вложенных дифференцируемых функций (AD). Это соответствует современной версии метода обратного распространения ошибки, который является эффективным, даже когда сети слабо связаны. В 1973 г. Стюарт Дрейфус используется алгоритм обратного распространения для адаптации параметров контроллеров пропорционально градиентам ошибок. В 1974 г. Пол Уэрбос упомянул о возможности применения этого принципа в искусственных нейронных сетей, и в 1982 г., он применил метод AD Linnainmaa к нейронным сетям, который широко используется в настоящее время. В 1986 г. Дэвид Э. Румельхарт, Джеффри Э. Хинтон и Рональд Дж. Уильямс показали, с помощью компьютерных экспериментов, что этот метод может генерировать полезные внутренние представления входных данных в скрытых слоях нейронных сетей. В 1993 г. Эрик А. Ван был первым, кто выиграл международный конкурс

распознавания образов с помощью обратного распространения ошибки.

Трудности с глубокими моделями

Как и в случае ИНС, могут возникнуть трудности с обучением DNN, если они обучаются примитивными наивными способами. Две основные проблемы — переобучение и большое время вычислений.

Доминирующий метод для обучения структур DNN — это коррекция ошибок обучения (например, обратного распространение ошибки с градиентного спуском) благодаря простоте реализации и тенденции сходиться к лучшему локальных оптимумов, чем другие методы обучения. Тем не менее, эти методы могут быть вычислительно дороги, особенно для DNN (то есть требовать большого времени обучения или памяти). Есть много параметров обучения, которые необходимо учитывать при обучении DNN, такие как размер (количество слоев и количество единиц на слой), скорость обучения и начальных весов. Охват всего пространства параметров для оптимальных параметров может оказаться невозможным из-за затрат времени и вычислительных ресурсов. Было показано, что различные «трюки», такие как использование мини-пакетирования (вычисление градиента на нескольких обучающих примерах сразу, а не на отдельных примерах), полезны для ускорения вычислений. Большая пропускная способность обработки графических процессоров произвела значительные ускорение в обучении в силу векторно-матричного характера вычислений, которые здесь необходимы и хорошо выполняются графическими процессорами.

Сверточные нейронные сети (CNN, Convolutional neural networks)

CNN удобны для обработки визуальных и других двумерных данных. CNN состоит из одного или более сверточных слоев с полностью соединенными нейронами. По сравнению с другими глубокими архитектурами, сверточные нейронные сети показали превосходные результаты в обработке изображений и речевых приложениях. Они также могут быть обучены с помощью стандартного обратного распространения. CNN легче обучаются, чем другие регулярные, глубокие нейронное сети и имеют много меньше параметров для оценки, что делает их весьма привлекательными.

Автоматическое распознавание речи (Automatic speech recognition)

Распознавание речи было революционизировано DNN, и особенно сетями *Long short-term memory networks* (LSTM). Это

специальная реккурентная нейронная сеть, модель, построенная в 1997 г. В 2003 г. LSTM стали конкурировать с традиционными речевыми распознавателями за выполнение определенных задач. В конце 90-х годов считалось, что скрытые марковские модели (HMM) более эффективны в распознавании речи, чем нейронные модели. В них речевой сигнал моделируется марковской моделью с неизвестными параметрами, значения которых находятся в процессе обучения.

В 2007 г. LSTM достигли отличных результатов в некоторых приложениях, хотя компьютеры были гораздо медленнее, чем сегодня. В 2015 г. эти результаты стали применяться в смартфонах.

Первоначальный успех глубокого обучения в распознавании речи, однако, был основан на задачах небольшой сложности. Результаты, приведенные в таблице ниже, предназначены для автоматического распознавания речи на популярном наборе данных TIMIT. Это обычный набор данных, используемый для начальных оценок глубоких учебных архитектур. Весь набор содержит речь 630 спикеров на восьми основных диалектах американского варианта английского языка, где каждый спикер читает 10 предложений. Доля ошибок в процентах приведена в таблице.

Метод	*%*
Randomly Initialized RNN	26.1
Bayesian Triphone GMM-HMM	25.6
Hidden Trajectory (Generative) Model	24.8
Monophone Randomly Initialized DNN	23.4
Monophone DBN-DNN	22.4
Triphone GMM-HMM with BMMI Training	21.7
Monophone DBN-DNN on fbank	20.7
Convolutional DNN	20.0
Convolutional DNN w. Heterogeneous Pooling	18.7
Ensemble DNN/CNN/RNN	18.2
Bidirectional LSTM	17.9

В 2010 г. исследователи, работающие в промышленности, расширили методы глубокого обучения от малого словаря TIMIT до большого словаря распознавания речи.

Одним из основополагающих принципов глубокого обучения является принцип не ручного подбора набора признаков, а их автоматического поиска (авторы благодарны Петру Семенову за обсуждение этой проблемы).

Крупномасштабное автоматическое распознавание речи является первым и наиболее убедительным успешным примером применения DNN в новейшей истории, как в промышленности, так и в науке, что еще более важно. Все основные коммерческие системы распознавания речи (например, Microsoft Кортана, Xbox, Skype Переводчик, Amazon Alexa, Google Now, Apple Siri, Baidu и IFLYTEK голосового поиска, а также ряд речевых продуктов Nuance и т. д.) основаны на DNN.

Распознавание образов (Image recognition)

Обычной оценкой набора для классификации изображений является набор данных базы данных MNIST. MNIST состоит из рукописных цифр и включает в себя 60000 примеров для обучения и 10000 тестовых примеров. Как и в случае TIMIT, его небольшой размер позволяет создавать несколько конфигураций для тестирования. В настоящее время лучший результат на MNIST является коэффициент ошибки 0,23%, достигнутый Ciresan и др. в 2012 г.

Одним из примеров приложений DNN является обучение с углубленным изучением, что может позволить автомобилю сделать полный обзор своего окружения. Другим примером является технология, известная как дисморфологии лица, используемая для анализа случаев пороков развития человека и связанная с анализом большой базой данных генетических синдромов.

Понимание естественного языка

Нейронные сети были использованы для моделей языка с начала 2000-х годов. Рекуррентные нейронные сети, особенно LSTM, являются наиболее подходящими для анализа последовательных данных, таких как язык. LSTM помогли улучшить машинный перевод и моделирование языка. LSTM в сочетании с CNN также улучшили автоматический ввод субтитров изображения и множество других приложений.

РЕКУРРЕНТНЫЕ СЕТИ

Наиболее простая и фундаментальная модель рекуррентной сети может быть записана в виде:

$$U_i(t+1) = \sigma\left(\sum_j W_{ij} \cdot U_j(t) - h_i \right) \qquad (3)$$

где σ — сигмоидальная функция, $U_j(t)$ — состояние j–го нейрона в момент времени t, W_{ij} — матрица связей между нейронами, всего имеем N нейронов, $i, j = 1,\ldots,N$. Величины h_i — пороги.

Модель (3) определяет динамическую систему с дискретным временем t. В ней нет слоев. Вектор состояния $U(t)$ блуждает в фазовом пространстве, образуя структуры, называемые аттракторами.

Начиная с 1991 г., был получен ряд фундаментальных результатов об этих сетях. Мы формулируем их в сокращенном и нестрогом виде, поскольку нет возможности строго определить здесь основные понятия (алгоритма и аттрактора), использованные в формулировках.

Аналог системы (3) с непрерывным временем имеет вид

$$\frac{dU_i}{dt} = \sigma\left(\sum_j W_{ij} \cdot U_j(t) - h_i \right) - \lambda_i U_i \qquad (4)$$

Теорема 1

Сети (1) способны реализовать любые алгоритмы (машины Тьюринга).

Первое доказательство этой теоремы получено в работах [11, 12]. Более короткое, но весьма трудное для понимания доказательство приведено в [13].

Если, как некоторые полагают, мозг — это машина Тьюринга, тогда модель (3) способна моделировать мозг и процесс мышления.

Отметим, что в некоторых случаях достаточно всего двух нейронов, чтобы получить все машины Тьюринга. Итак, сеть с двумя нейронами может вычислить все! Этот поразительный результат получен для кусочно-линейных сигмоидальных функций французским математиком П. Койраном с коллегами [13]. Вопрос, однако, в том, как ее научить решать конкретные задачи.

Теорема 2

Сети (3) и (4) могут порождать сколь угодно сложную динамику и иметь сколь угодно сложные аттракторы при достаточно большом числе нейронов.

Остановимся кратко на второй теореме и некоторых ее приложениях.

Теорема 2 была доказана в [14,17]. Одно из ее доказательств любопытно и метод доказательства имеет приложения в биологии. Рассмотрим случай (4).

Рассмотрим дифференциальное уравнение

$$\frac{dq}{dt} = f(q).$$

Если гладкая ограниченная функция $f(q)$ имеет несколько корней q, то типичное поведение решений $q(t)$ — это стремление к одному из корней. В этом случае говорят, что уравнение имеет точечные локальные аттракторы (*rest point attractor*). Такое поведение встречается во многих физических, биологических, и социальных системах и соответствует стабилизации поведения системы со временем.

Более сложная ситуация возникает, если мы рассмотрим систему двух дифференциальных уравнений

$$\begin{cases} \frac{dq_1}{dt} = f_1(q_1, q_2) \\ \frac{dq_2}{dt} = f_2(q_1, q_2) \end{cases}$$

Такие системы могут описывать не только стремление к равновесию, но и колебательные режимы, когда предельное решение – периодическая функция времени.

Чтобы построить нейронную сеть (4), имитирующую заданную колебательную динамику, мы берем два нейрона-центра и $N>>1$ нейронов–сателлитов. Каждый сателлит действует на оба центра, и в свою очередь, центры действуют на все сателлиты. Однако сателлиты не взаимодействуют между собой. Путем настройки связей сателлит — центр и центр — сателлит мы можем получить заданную колебательную динамику центров.

Чтобы получить хаотическую динамику, можно использовать три центра.

Применения к созданию управляющего сигнала для роботов

Сути с такой структурой способны генерировать сигналы, имитирующие динамику биологических объектов, например, людей. С помощью специальных датчиков, закрепленных на теле человека (рис. 31), мы получаем координаты соответствующих точек тела в зависимости от дискретных моментов времени. Затем мы аппроксимируем эти координаты сетью с несколькими центрами и примерно 100 сателлитами. Результат приведен на рис. 32, где красная линия — это экспериментальные данные и зеленая линия — это аппроксимация при помощи нейронной сети.

Рисунок 31. Объект с установленными датчиками

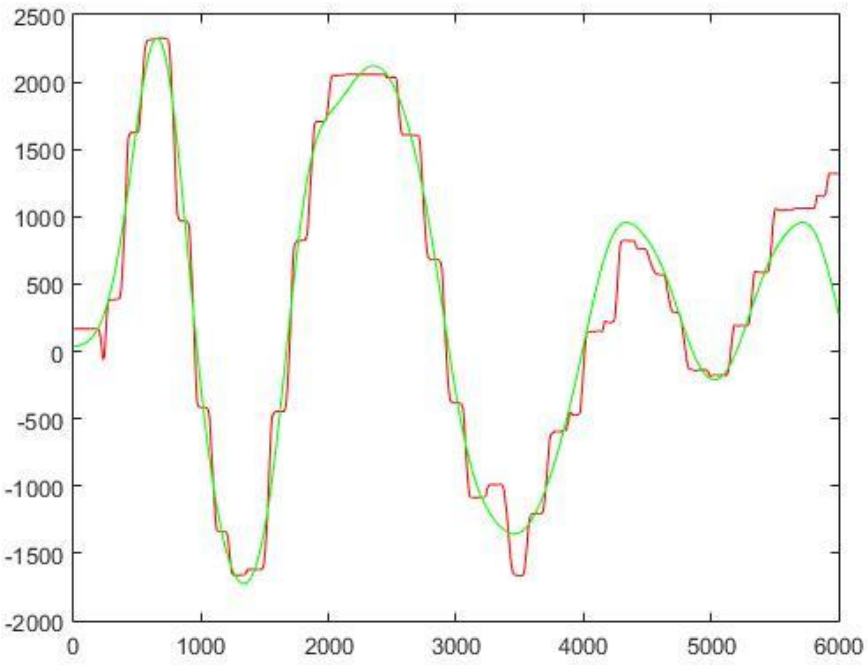

Рисунок 32. Аппроксимация траектории движения исследуемого объекта

ЗАКЛЮЧЕНИЕ

В данном пособии сформулированы основные понятия и модели теории нейронных сетей, а также приведены примеры их использования в пакете *Matlab*.

История современных моделей нейронных сетей началась в середине 50-х годов, когда Фрэнк Розенблатт придумал модель персептрона и заявил, что эта модель способна решить все проблемы теории искусственного интеллекта. В 1964 году наши соотечественники В. Вапник и А. Червоненскис создали первую математическую теорию, которая объяснила, почему нейронные сети могут успешно работать.

В дальнейшем выяснилось, что оптимизм Розенблатта был чрезмерным, но есть другие модели, например, многослойный персептрон, которые могут успешно решать задачи классификации. Была доказана фундаментальная теорема о том, что рекуррентные нейронные сети способны реализовать любой алгоритм. Тем не менее, в начале 2000-х гг. среди некоторых специалистов существовал определенный пессимизм по поводу того, смогут ли нейросетевые модели успешно конкурировать с человеческим мозгом в таких задачах, как распознавание речи, почерка, образов, игра го, шахматы и т. д.

В настоящее время благодаря методам Deep Learning достигнуты большие практические успехи. Оказывается, глубокое обучение на миллионах примеров способно творить чудеса.

Новые алгоритмы, такие как альфа зеро, показали невероятные возможности. Например, они воспроизвели в процессе самообучения всю шахматную теорию всего за 4 часа.

В будущем вы увидите еще более невероятные вещи. Возможно, вы уже больше не будете водить машину – это сделает нейронная сеть и абсолютно безопасно. Мы надеемся, что это пособие поможет вы понять некоторые принципы функционирования этих суперсистем Будущего.

ОБЩИЕ ИСТОЧНИКИ И КЛАССИЧЕСКИЕ РАБОТЫ

1. Хайкин С. Нейронные сети. М. : Вильямс, 2006.
2. Kohonen T., Honkela T. Kohonen Network. Scholarpedia. 2007.
3. Kohonen T. Self-Organized Formation of Topologically Correct Feature Maps // Biological Cybernetics.1982. 43 (1): 59–69. doi:10.1007/bf00337288.
4. McCulloch W., Pitts W. A Logical Calculus of Ideas Immanent in Nervous Activity // Bulletin of Mathematical Biophysics. 1943. 5 (4): 115–133. doi:10.1007/BF02478259.
5. Hebb D. The Organization of Behavior. New York : Wiley, 1949.
6. Farley B. G., Clark W. A. Simulation of Self-Organizing Systems by Digital Computer // IRE Transactions on Information Theory. 1954. 4 (4): 76–84. doi:10.1109/TIT.1954.1057468.
7. Rochester N., Holland J. H., Habit L. H., Duda W. L. Tests on a cell assembly theory of the action of the brain, using a large digital computer // IRE Transactions on Information Theory. 1956. 2 (3): 80–93. doi:10.1109/TIT.1956.1056810.
8. Rosenblatt F. The Perceptron: A Probabilistic Model For Information Storage And Organization In The Brain // Psychological Review. 1958. 65 (6): 386–408. doi:10.1037/h0042519. PMID 13602029.
9. Werbos P. J. Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. 1975.
10. Minsky M., Papert S. Perceptrons: An Introduction to Computational Geometry. MIT Press, 1969.
11. Rumelhart D. E; McClelland J. Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Cambridge : MIT Press, 1986.
12. Goodfellow I., Bengia Y., Courville A. Deep Learning. London ; Cambridge : MIT Press, 2016.

Специальная литература

13. Siegelmann H. T., Sontag E. D. Turing computability with neural networks // Appl. Math. Lett. 1991. Vol. 4, 6.
14. Siegelmann H. T., Sontag E. D. On the computational power of neural nets // J. Comp. Syst. Sci. 1995. Vol. 50. P. 132–150.
15. Koiran P., Moore C. Closed-form analytic maps in one and two dimensions can simulate Turing machines // Theoretical Computer Science. 1999. Volume 210, Issue 1. P. 217–223.
16. Vakulenko S. A.; Gordon P. V. Neural networks with prescribed large time behavior // J. Phys. A. Math. Gen. 31. 1998. No. 47. P. 9555–9570.

17. Vakulenko S., Grigoriev D. Complexity of gene circuits, Pfaffian functions and the morphogenesis problem // C. R. Acad. Sci. Ser I. 2003. 337. P. 721–724.
18. Vakulenko S., Radulescu O. Flexible and Robust Patterning by Centralized Gene Networks // Fundamenta Informaticae. 2012. Vol. 119, P. 1–25. DOI 10.3233/FI-2012-712 IOS Press

Миссия университета – генерация передовых знаний, внедрение инновационных разработок и подготовка элитных кадров, способных действовать в условиях быстро меняющегося мира и обеспечивать опережающее развитие науки, технологий и других областей для содействия решению актуальных задач.

НАПРАВЛЕНИЕ "ЭЛЕКТРОЭНЕРГЕТИКА И ЭЛЕКТРОТЕХНИКА"

В текущее время направление "Электроэнергетика и электротехника" реализуется в рамках факультета Систем управления и робототехники. Исторически это направление локализовалось на кафедре Электротехники и прецизионных электромеханических систем.

В 1930 году техникум точной механики и оптики был реорганизован в учебный комбинат, состоящий из института, техникума и ФЗУ в системе Всесоюзного объединения оптико-механической промышленности.

В те годы электротехническую подготовку в нашем институте проводили кафедры «Электротехники» и «Электроизмерительных приборов». Кафедрой «Электротехники» руководил проф. Салтыков Л.Н., а кафедрой «Электроизмерительных приборов» проф. Шишелов Л.П.

С сентября 1933 года исполнять обязанности заведующего кафедрой «Электротехники» нашего института начинает Рукавишников Н.Н., а с ноября 1937 года, на заведование кафедрой назначается Солодовников А.А., известный специалист в области электротехники, электроизмерительных приборов и оборудования.

Во время войны при эвакуации ЛИТМО в г. Черепаново кафедрой руководил доц., к.т.н. Березниковский С.Ф.; штатное расписание кафедры в те годы насчитывало всего 4 человека.

После возвращения ЛИТМО из эвакуации в 1944 году кафедрой заведует Березниковский С.Ф., которого 25 января 1945 года освобождают от обязанностей заведующего кафедрой «Общей и специальной электротехники» и назначают заведующим этой кафедрой профессора Зилитенкевича С.И.

В послевоенные годы в целом по стране и в Ленинграде ощущался дефицит опытных преподавателей высшей школы и руководство институтом пригласило в качестве заведующего кафедрой «Общей и специальной электротехники» известного ученого, педагога и методиста Пиотровского Л.М. Большинство учебников по электрическим машинам в ту пору было написано Пиотровским Л.М. лично или в соавторстве с другими видными учеными.

В 1948 году на базе кафедры «Общей и специальной электротехники» образуются кафедры: «Общей электротехники и электрических машин» зав.каф. доц. Березниковский С.Ф., «Теоретических основ электротехники» зав. каф. проф. Слепян Л.Б. и «Электроизмерительных приборов» исполняющий обязанности зав. каф. проф. Слепян Л.Б.

В 1951 году кафедры «Электротехники» и «ТОЭ» объединяют в единую кафедру «Электротехники и ТОЭ» под руководством доц. Березниковского С.Ф. в составе Радиотехнического факультета.

В 1956 году на радиотехническом факультете вновь образуются две кафедры – «ТОЭ» зав. каф. доц. Сочнев А.Я. и «Электрических машин» зав. каф. доц. Березниковский С.Ф.

В июле 1958 года доц. Сочнева А.Я. освобождают от обязанностей зав. каф. «ТОЭ», а доц. Фунтова Н.М. назначают в.и.о. зав. каф. и избирают по конкурсу на должность заведующего в 1960 году.

В 1961 году в ЛИТМО на должность заведующего кафедрой «Электрических машин» приглашают профессора Сахарова А.П.

В 1965 году на должность заведующего кафедрой «Электрических машин» избирается доц., к.т.н. Глазенко Т.А.

В 1968 году кафедры «ТОЭ» и «Электрических машин» объединяются в единую кафедру «Электротехники» под руководством заслуженного деятеля науки и техники Российской Федерации, почетного члена Электротехнической академии России проф. Глазенко Т.А., которая двадцать пять лет возглавляла кафедру. Она являлась видным, творчески активным ученым, автором более 200 научных работ.

В 1990 году на должность заведующего кафедрой избирается профессор, д.т.н. Герман–Галкин С.Г.

В 1996 году кафедра «Электротехники» была переименована в кафедру «Электротехники и прецизионных электромеханических систем».

С 1991 года кафедрой руководит доцент, кандидат технических наук, Томасов Валентин Сергеевич.

С 1992 по 2005 годы на кафедре работал заслуженный деятель науки и техники Российской Федерации, действительный член

Международной Энергетической академии, профессор, д.т.н., Сабинин Ю.А.

Сегодня на кафедре работают: профессор, проф. Вакуленко С.А.; доценты: Абдуллин А.А., Борисов П.В., Горшков К.С., Ильина А.Г., Ловлин С.Ю., Лукичев Д.В., Никитина М.В., Поляков Н.А., Субботин Д.А., Толмачев В.А., Томасов В.С., Усольцев А.А.; ст. преподаватель: Денисов К.М.; ассистенты: Демидова Г.Л., Кононова М.Е., Цветкова М.Х., тьюторы Гурьянов В.А., Махин И.Е.

На кафедре реализуются многочисленные опытно-конструкторские проекты по созданию электроприводов прецизионных следящих систем наведения телескопов траекторных измерений. Кафедра электротехники и прецизионных электромеханических систем Университета ИТМО, является головным предприятием Российского космического агентства в части исследования, проектирования, изготовления и ввода в эксплуатацию электросиловых приводов на базе высокоэффективных п/п преобразователей и компьютерных систем управления для квантово-оптических систем нового поколения. Разработанные электропривода используются в составе высокоточных оптико-электронных и лазерных комплексов для космической навигации и геодезии (система «Глонасс»), обзора и контроля космического пространства, полигонных испытаний и сертификации ракетного вооружения, а также контроля испытаний стратегических ракет, предусмотренного международными договорами в области ограничения стратегических вооружений.

На кафедре работает аспирантура и ведётся большая научно-исследовательская работа в том числе в созданной международной научной лаборатории «Силовая электроника и автоматизированный электропривод» (руководители: Валентин Сергеевич Томасов, доцент, кандидат технических наук, Рышард Стжелецки, профессор, Dr. habil. технических наук).

Основные направления исследований:

- Разработка алгоритмов прецизионного управления оптическими телескопами в том числе на инфранизких скоростях в условиях переменных внешних воздействий с учетом упругости механизмов и нелинейного характера сил сопротивления движению;
- Разработка и создание специального программно-математического обеспечения, реализующего современные алгоритмы управления, а также создание автоматизированных систем дистанционного управления электроприводами;
- Разработка методов идентификации сложных механических объектов, быстрого прототипирования прецизионных электромеханических систем, управляющих ими;

- Разработка и создание специализированных высокопроизводительных микропроцессорных систем многоконтурного управления электроприводами;
- Разработка и создание блоков силовой электроники для управления моментными электродвигателями в том числе энергоэффективных полупроводниковых преобразователей с возможностью двустороннего обмена энергией;
- Разработка устройств бесконтактной передачи энергии.