Лабораторная работа №4

Исследование адаптивного линейного элемента

1 Цель работы

Углубление теоретических знаний в области архитектуры нейронных сетей с линейной активационной функцией, исследование свойств квадратичной целевой функции и LMS-алгоритма обучения, приобретение практических навыков обучения однослойной сети линейных адаптивных элементов при решении задачи классификации и адаптивной фильтрации.

2 Основные теоретические положения

2.1 Структура однослойной сети из адаптивных линейных элементов

Адаптивный линейный элемент (АЛЭ) (ADALINE - Adaptive Linear Element) представляет собой нейрон с линейной функцией преобразования purelin. Общая структурная схема однослойной сети из АЛЭ изображена на рисунке 4.1.

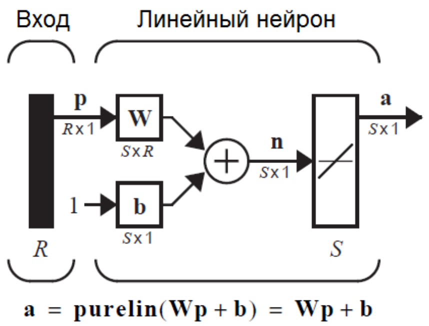

Рисунок 4.1 ─ Схема однослойной сети из адаптивных линейных элементов

Выходное значение отдельного нейрона сети вычисляется в соответствии с выражением

$$a_i = purelin(n_i) = purelin({}_i\mathbf{w}^T\mathbf{p} + b_i) = {}_i\mathbf{w}^T\mathbf{p} + b_i \qquad (4.1)$$

2.2 Классификация с использованием АЛЭ

Рассмотрим АЛЭ с 2-мя входами и 1 выходом (рисунок 4.2.).

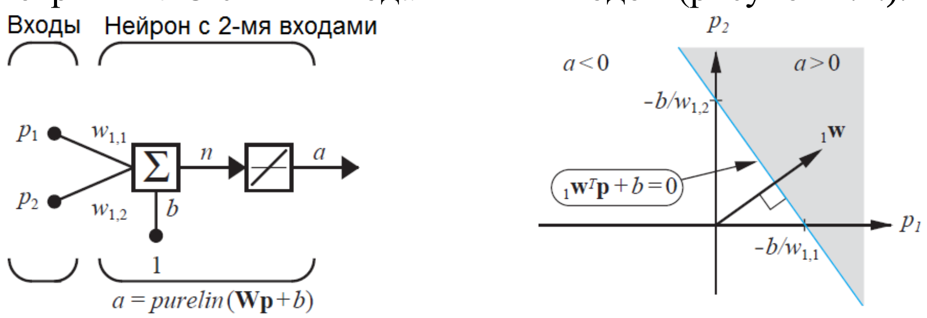

Рисунок 4.2 ─ АЛЭ и его граница решения

С учетом только 2-х входов выходной сигнал нейрона будет равен

$$a = {}_1\mathbf{w}^T\mathbf{p} + b = w_{1,1}p_1 + w_{1,2}p_2 + b\,. \tag{4.2}$$

Граница решения определится из условия

$$n = {}_1\mathbf{w}^T\mathbf{p} + b = w_{1,1}p_1 + w_{1,2}p_2 + b = 0\,. \tag{4.3}$$

АЛЭ разделяет входное пространство на две области, где $a>0$ и $a<0$ (см. рисунок 4.2). Таким образом, АЛЭ (как и персептрон) может использоваться для классификации **линейно сепарабельных** объектов**.**

2.3 Целевая функция нейронной сети

Задача обучения нейронных сетей сводится к поиску весов и смещений, которые обеспечивают необходимую эффективность сети. Количественная мера эффективности сети определяется **целевой функцией** или **критерием эффективности**. Целевая функция зависит от параметров сети и имеет малые значения, когда сеть функционирует хорошо, и большие значения, когда сеть функционирует плохо. Поэтому целью обучения является поиск параметров сети, которые обеспечивают минимум целевой функции.

Пусть $F(\mathbf{x})$ непрерывная целевая функция, где $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$ **—** вектор столбец параметров сети в n-мерном евклидовом пространстве. Необходимо найти минимум $F(\mathbf{x})$ по **x.** В точке $\mathbf{x}=\mathbf{x}^*$ будет минимум целевой функции, если градиент функции (вектор первых частных производных $F(\mathbf{x})$ по **x**) будет равен нулю в этой точке (условие *первого порядка*)

$$\nabla F(\mathbf{x})\big|_{\mathbf{x}=\mathbf{x}^*} = \mathbf{0}\,. \tag{4.4}$$

Точки **x**, удовлетворяющие этому условию, называются *стационарными*. В точке x* будет существовать строгий минимум, если (*условие2-го порядка)*

$$\Delta\mathbf{x}^T\nabla^2 F(\mathbf{x})\big|_{\mathbf{x}=\mathbf{x}^*}\Delta\mathbf{x} > 0\,, \tag{4.4}$$

где $\nabla^2 F(\mathbf{x})$– квадратная *матрица Гессе* целевой функции (матрица вторых частных производных F(**x**) по **x,** называемая гессианом**)**, $\mathbf{\Delta x=x - x^*}$ – окрестность точки минимума. Чтобы условие (4.4) выполнялось для любых $||\mathbf{\Delta x}||>\mathbf{0}$ достаточно, чтобы матрица Гессе была *положительно определена*. Для того чтобы в точке **x*** находился минимум (строгий или слабый) достаточно, чтобы матрица Гессе была *положительно полуопределена*. Напомним, что матрица положительно определена, если все собственные числа λ матрицы положительны. Если все собственные числа матрицы не отрицательны, то матрица положительно полуопределена.

При обучении нейронных сетей часто используют *квадратичную целевую функцию*. Общая форма квадратичной целевой функции

$$F(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} + \mathbf{d}^T\mathbf{x} + c, \tag{4.5}$$

где A – симметричная матрица. Градиент и гессиан квадратичной *целевой функции* $F(\mathbf{x})$ соответственно равны:

$$\nabla F(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{d}, \quad \nabla^2 F(\mathbf{x}) = \mathbf{A}. \tag{4.6}$$

2.4 Алгоритм наискорейшего спуска

Цель оптимизации заключается в поиске вектора параметров сети $\mathbf{x}^*$, который минимизирует целевую функцию $F(\mathbf{x})$. Для поиска $\mathbf{x}^*$ применяют алгоритмы последовательного приближения – *итеративные алгоритмы*. В соответствии с такими алгоритмами поиск начинается с начального значения $\mathbf{x}_0$ и на каждом шаге k вектор параметров корректируют в соответствии с формулой

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k \tag{4.7}$$

или

$$\Delta\mathbf{x}_k = (\mathbf{x}_{k+1} - \mathbf{x}_k) = \alpha_k \mathbf{p}_k, \tag{4.8}$$

где $\mathbf{p}_k$ – вектор, определяющий направление поиска; α_k – скорость обучения, определяющая длину шага $\Delta\mathbf{x}_k$.

Итеративные алгоритмы оптимизации отличаются выбором вектора направления поиска $\mathbf{p}_k$ и способами вычисления значений скорости обучения. Так, в *алгоритме наискорейшего спуска* (**SDA– steepest descent algorithm**) направление поиска обратно направлению вектора градиента, т.е. $\mathbf{p}_k = -\mathbf{g}_k$, где $\mathbf{g}_k = \nabla F(\mathbf{x})$ – значение вектора градиента целевой функции на k–ой итерации алгоритма. Соответственно процедура SDA определяется выражением:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{g}_k. \tag{4.9}$$

Если целевая функция $F(\mathbf{x})$ является увадратичной и имеет сильный минимум, то все собственные значения матрицы Гессе **A** будут положительными и условие схождения алгоритма SDA запишется в виде неравенства:

$$\alpha < \frac{2}{\lambda_{max}}, \tag{4.10}$$

где λ_{max} — максимальное собственное число матрицы Гессе. Так как собственные числа матрицы Гессе квадратичной целевой функции определяют кривизну поверхности целевой функции, то в соответствии с (4.10) *скорость обучения устойчивого SDA обратно пропорциональна максимальной кривизне целевой функции.*

2.3 Решение минимума среднего квадрата ошибки

Рассмотрим АЛЭ с одним выходом. Для упрощения анализа введем следующие обозначения: $\mathbf{x}^T = [_1\mathbf{w}\ \mathbf{b}]$ и $\mathbf{z}^T = [\mathbf{p}\ \mathbf{1}]$. Тогда выходной сигнал АЛЭ можно представить в виде скалярного произведения $a = \mathbf{x}^T \mathbf{z}$.

В ходе обучения с учителем параметры (веса и смещения) АЛЭ корректируются таким образом, чтобы выходной сигнал a имел наилучшее приближение к желаемой реакции t. Для этого вычисляют ошибку приближения $e=(t-a)$ и минимизирует **средний квадрат ошибки** (СКО, MSE – mean square error). С учетом введенных обозначений *целевая функция* $F(\mathbf{x})$, заданная в виде СКО, будет равна

$$F(\mathbf{x}) = E[e^2] = E[(t-a)^2] = E[(t-\mathbf{x}^T\mathbf{z})^2], \qquad (4.11)$$

где E – символ математического ожидания. Раскроем (4.11):

$$F(\mathbf{x}) = E[t^2 - 2t\mathbf{x}^T\mathbf{z} + \mathbf{x}^T\mathbf{z}\mathbf{z}^T\mathbf{x}] = E[t^2] - 2\mathbf{x}^T E[t\mathbf{z}] + \mathbf{x}^T E[\mathbf{z}\mathbf{z}^T]\mathbf{x} .$$

СКО целевая функция может быть переписана в квадратичной форме

$$F(\mathbf{x}) = c - 2\mathbf{x}^T\mathbf{h} + \mathbf{x}^T\mathbf{R}\mathbf{x}, \text{ где } c = E[t^2], \mathbf{h} = E[t\mathbf{z}], \mathbf{R} = E[\mathbf{z}\mathbf{z}^T]. \qquad (4.12)$$

Здесь $\mathbf{h}$ – вектор кросс-корреляции, $\mathbf{R}$ – корреляционная матрица.

Сравним целевую функцию (4.12) с квадратичной целевой функцией общего вида (4.5). Получим, что

$$\mathbf{d} = -2\mathbf{h}, \ \mathbf{A} = 2\mathbf{R}, \qquad (4.13)$$

т.е. матрица Гессе целевой функции (4.12) равна **2R**. Точка минимума целевой функции $F(\mathbf{x})$ определится из условия 1-го порядка (4.4):

$$\nabla F(\mathbf{x}) = \nabla\left(c + \mathbf{d}^T\mathbf{x} + \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x}\right) = \mathbf{d} + \mathbf{A}\mathbf{x} = -2\mathbf{h} + 2\mathbf{R}\mathbf{x} = \mathbf{0} . \qquad (4.14)$$

Если корреляционная матрица положительно определена, то будет существовать единственная точка минимума целевой функции (4.12)

$$\mathbf{x}^* = \mathbf{R}^{-1}\mathbf{h}. \qquad (4.15)$$

Решение (4.15), позволяющее вычислять параметры АЛЭ через матрицу **R** и вектор **h**, называют **решением минимума СКО (оптимальным решением Винера).** На практике точные значения **h** и **R** обычно неизвестны. Однако, если входные данные, представляемые вектором **p**, соответствуют стационарному эргодическому процессу, то, используя усреднение по времени, можно оценить **h** и **R**. Если на k-м шаге имеются оценки матрицы **R** и вектора **h**, обозначаемые как $\mathbf{R}_k$ и $\mathbf{h}_k$, то для поиска винеровского решения можно использовать алгоритм наискорейшего спуска. Подставив значение градиента из (4.14) в выражение (4.9), можно получить алгоритм наискорейшего спуска, сходящийся к решению (4.15):

$$\mathbf{g}_k = -2\mathbf{h}_k + 2\mathbf{R}_k\mathbf{x}_k , \qquad (4.16)$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \boldsymbol{\alpha}_k\, \mathbf{g}_k, \tag{4.17}$$

Отметим, что **существование решения (4.15) целиком зависит от R**. Следовательно, характеристики входного вектора **p** определяют возможность существования единственного решения.

2.4. LMS алгоритм

На практике вычислять оценки матрицы **R** и вектора **h** затратно. Поэтому выполняют аппроксимацию алгоритма наискорейшего спуска, в которой используют приближенные оценки градиента. Идея заключается в аппроксимации среднего квадрата ошибки (4.11) более простым выражением

$$\hat{F}(\mathbf{x}) = (t(k) - a(k))^2 = e^2(k), \tag{4.18}$$

в котором вместо математического ожидания E квадрата ошибки используется просто квадрат мгновенной ошибки. Тогда оценка градиента, называемая в этом случае *стохастическим (случайным) градиентом*, будет равна

$$\hat{\nabla} F(\mathbf{x}) = \nabla e^2(k). \tag{4.19}$$

Так как $a = \mathbf{x}^{\mathrm{T}}\mathbf{z}$ и $e=(t-a)$, то стохастический градиент будет равен

$$\hat{\nabla} F(\mathbf{x}) = \nabla e^2(k) = -2e(k)\mathbf{z}(k). \tag{4.20}$$

Подставив стохастический градиент (4.19) в алгоритм SDA, получим получим **LMS-алгоритм** (LMS- least mean square, алгоритм наименьшего среднего квадрата, также называемый **правилом обучения Уидроу-Хоффа)**

$$\mathbf{x}_{k+1} = \mathbf{x}_k + 2\alpha e(k)\mathbf{z}(k), \tag{4.21}$$

или

$${}_1\mathbf{w}(k+1) = {}_1\mathbf{w}(k) + 2\alpha e(k)\mathbf{p}(k), \quad b(k+1) = b(k) + 2\alpha e(k). \tag{4.22}$$

Если используется слой из АЛЭ, то LMS-алгоритм записывается в матричной форме:

$$\mathbf{W}(k+1) = \mathbf{W}(k) + 2\alpha \mathbf{e}(k)\mathbf{p}^T(k), \quad \mathbf{b}(k+1) = \mathbf{b}(k) + 2\alpha \mathbf{e}(k). \tag{4.21}$$

Поскольку в выражении (4.21) используются неточные оценки градиента, то в процессе обучения возникает шум, в результате чего траектория движения вектора параметров $\mathbf{x}_k$ к вектору **x*** в LMS-алгоритме не совпадает с траекторией движения в алгоритме наискорейшего спуска. LMS-алгоритм привлекает своей простотой.

Условие сходимости SDA (4.9) определялось значением собственных чисел матрицы Гессе **A** квадратичной целевой функции. Для LMS справедливо, что **A=2R** . Поэтому по аналогии можно записать условие сходимости LMS-алгоритма

$$\alpha < 1/\lambda_i \quad \text{для всех} \quad i \qquad (4.22)$$

или

$$0 < \alpha < 1/\lambda_{max} \qquad (4.23)$$

где λ_i - собственные числа корреляционной матрицы **R**. Если это условие выполняется, то можно показать, что LMS алгоритм, обрабатывающий на каждой итерации один входной вектор, сходится к решению, *совпадающему с решением минимума СКО* (4.15) .

2.5 Адаптивная фильтрация

На основе АЛЭ реализуют адаптивные фильтры (рисунок 4.3). Для этого на входе АЛЭ размещают линию задержки (tapped delay line – TDL). Один элемент линии задержки, обозначенный на рисунке 4.3 буквой D, представляет собой регистр, который запоминает входное значение на время, равное шагу дискретизации. В этом случае вектор **p** , поступающий на вход АЛЭ, соответствует совокупности значений на выходе линии задержки – $y(k), y(k-1), \ldots, y(k-R+1)$.

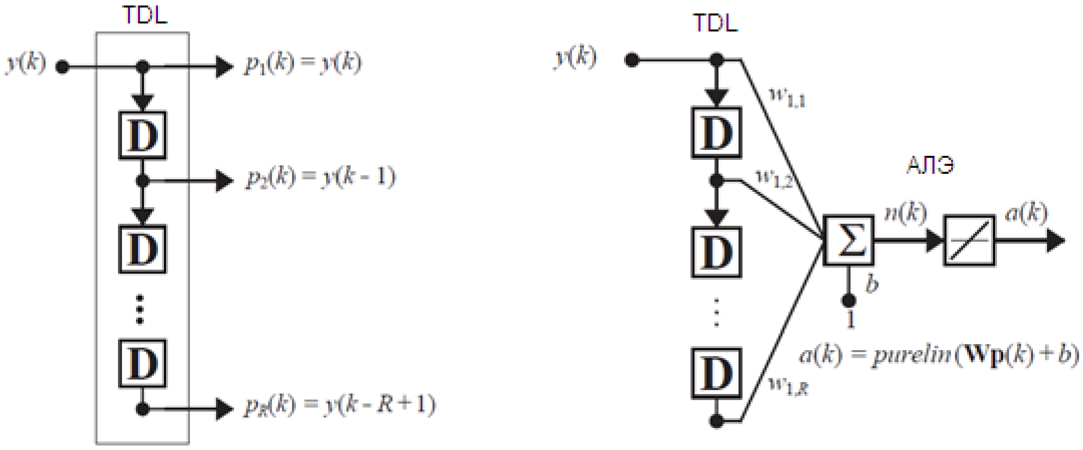

Рисунок 4.3 – Схема адаптивного фильтра на основе АЛЭ

С учетом принятых обозначений выход адаптивного фильтра запишется в виде:

$$a(k) = purelin(\mathbf{Wp} + b) = \sum_{i=1}^{R} w_{1,i}\, y(k-i+1) + b\,. \qquad \ldots\ldots(4.24)$$

Уравнение (4.24) соответствует фильтру с конечной импульсной характеристикой (КИХ), который также называют *трансверсальным фильтром.*

Адаптивные фильтры используются для решения задач моделирования, идентификации, подавления шумов, компенсации эхо-сигналов, построения эквалайзеров каналов связи, кодирования речи, управления адаптивными антенными решетками и др. Рассмотрим применение адаптивного фильтра для предсказания сигналов. На рисунке 4.4 изображена схема адаптивного предсказателя, построенного на основе АЛЭ, обеспечивающего предсказание текущего отсчета входного сигнала $y(k)$ по 2-м предыдущим отсчетам: $y(k\text{-}1)$ и $y(k\text{-}2)$.

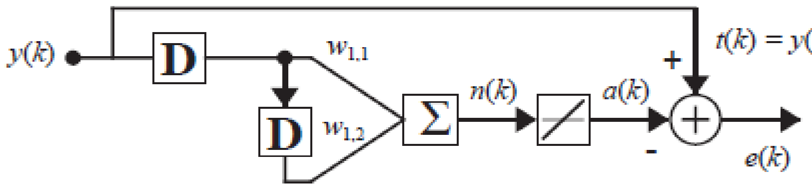

Рисунок 4.4 – Адаптивный предсказатель

Отметим, что адаптивный линейный предсказатель в ходе обучения обеспечивает поиск параметров, которые минимизируют СКО в виде целевой функции (4.12). Так как желаемое значение на выходе предсказателя должно совпадать с текущим входным значением, т.е. $t(k)=y(k)$ и входной вектор предсказателя равен

$$\mathbf{z}(k)=\mathbf{p}(\mathrm{k})=[y(k\text{-}1)\ y(k\text{-}2)]^{\mathrm{T}}\ , \tag{4.25}$$

то составляющие целевой функции (4.12) запишутся в виде:

$$\mathbf{R} = E[\,\mathbf{zz^T}\,] = E\begin{bmatrix} y^2(k-1) & y(k-1)y(k-2) \\ y(k-1)y(k-2) & y^2(k-2) \end{bmatrix}, \tag{4.26}$$

$$\mathbf{h} = E[\,t\mathbf{z}\,] = E\begin{bmatrix} y(k)y(k-1) \\ y(k)y(k-2) \end{bmatrix}, c = E[t^2(k)] = \ E[y^2(k)]. \tag{4.27}$$

При известных значениях матрицы $\mathbf{R}$ и вектора $\mathbf{h}$ оптимальные веса предсказателя определятся в виде решения минимума СКО (4.15). Например, пусть

$$\mathbf{R} = \begin{bmatrix} 0.54 & 0.45 \\ 0.54 & 0.45 \end{bmatrix}, \qquad \mathbf{h} = \begin{bmatrix} 0.54 \\ 0.45 \end{bmatrix}.$$

Тогда в соответствии с (4.15) оптимальный вектор параметров предсказателя будет равен $\mathbf{x}^*=[w_{1,1}\ w_{1,2}]=[1\ 0]$.

При исследовании свойств алгоритмов, базирующихся на СКО целевой функции полезно использовать графики *линий контуров постоянного уровня e=const*. Эти графики для поверхности квадратичной целевой функции, являющейся параболоидом, представляют собой гиперэллипсы (рисунок 4.5). Главные оси этих гиперэллипсов ориентированы в направлении собственных векторов матрицы $\mathbf{R}$, длина осей обратно пропорциональна собственным числам $\mathbf{R}$. Чем больше соответствующее собственное число матрицы $\mathbf{R}$, тем больше градиент $F(\mathbf{x})$ вдоль соответствующей оси эллипса и, соответственно, линии контуров равных уровней располагаются ближе друг к другу. Для рассматриваемого примера $\mathbf{R}$ собственные числа равны λ_1=0,17 и λ_2=1,97. Соответственно, эллипс вытянут вдоль собственного вектора с меньшим собственным значением, т.е λ_1=0,17.

Так как вычисление оптимальных параметров предсказателя в соответствии с (4.15) требует вычисления обратной матрицы, то на практике поиск оптимальных параметров выполняют на основе LMS-алгоритма. При этом скорость схождения, точность и устойчивость LMS-алгоритма сильно зависит от скорости обучения α. Максимальное устойчивое значение α определяется на основе соотношения (4.23). Для рисунка 4.5 α_{max}=1/ λ_2=0,507. На практике при использовании адаптивных фильтров определять собственные числа **R** не представляется возможным. Однако, учитывая свойство корреляционной матрицы, согласно которому след корреляционной матрицы равен сумме её собственных чисел, т.е. $tr(\mathbf{R}) = \sum_{i=1}^{R} r_{ii} = \sum_{i=1}^{R} \lambda_i$, можно заключить, что для СКО целевой функции $\lambda_{max} \leq tr(\mathbf{R})$. С учетом этого соотношения условие (4.23) можно представить в виде более жесткого условия, *гарантирующего* устойчивую работу LMS-алгоритма для трансверсального фильтра:

$$\alpha \leq \frac{1}{3\sum_{i=1}^{R} r_{ii}} = \frac{1}{3R\sigma_y^2} \qquad (4.28)$$

где σ_y^2 — дисперсия входного сигнала адаптивного фильтра. В соответствии с (4.28) для рассматриваемого примера должно быть $\alpha \leq 1/(3*2*0{,}54) = 0{,}31$.

Так как LMS-алгоритм аппроксимирует алгоритм наискорейшего спуска, то при малых скоростях обучения α траектория движения вектора параметров LMS-алгоритма будет менее хаотична и в среднем будет соответствовать движению вектора параметров алгоритма наискорейшего спуска, которая направлена перпендикулярно линиям равных контуров, как показано на рисунке 4.5. Однако при значениях α близких к α_{max} продвижение к некоторому малому уровню значений целевой функции происходит быстрее, несмотря на хаотичность. Кружок в центре рисунка 4.5. соответствует решению минимума СКО (4.15). Видно, что при выбранных значениях α решение, получаемое с помощью LMS-алгоритма, приближается к решению минимума СКО.

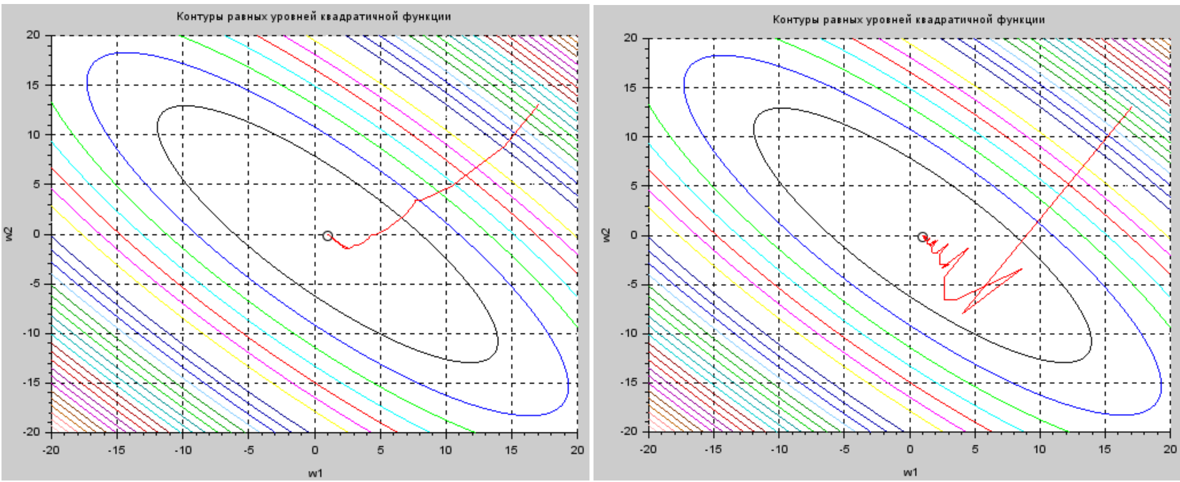

Рисунок 4.5 – Траектория движения вектора параметров предсказателя при использовании LMS-алгоритма: слева $\alpha = 0{,}1$; справа $\alpha = 0{,}507$

3. Варианты заданий и программа работы

3.1. Повторить теоретический материал, относящийся к архитектуре и правилам обучения адаптивного линейного элемента [1, 4, 5].

3.2. Даны четыре класса, каждый из которых представлен 2-мя точками (столбцами матрицы **P**), указанными в таблице 3.2 к лабораторной работе №3 (*используйте свой вариант*). Необходимо:

- разработать структурную схему классификатора на основе АЛЭ, распознающего эти 4 класса;
- выполнить предварительный анализ задачи, изобразив точки четырех классов и построив графически возможные границы решений;
- задать ту же целевую матрицу **T,** которая использовалась при выполнении задания 3.3 в лабораторной работе №3**,** заменив все нули на -1;
- полагая, что все входные векторы **p** равновероятны, написать программу, вычисляющую корреляционную матрицу **R** , собственные числа гессиана целевой функции **A=2R** и максимальное устойчивое значение параметра α_{max} LMS-алгоритма;
- изучить встроенные функции ann_ADALINE и ann_ADALINE_online пакета NeuralNetworks 2.0, реализующие блочный и последовательный варианты LMS-алгоритма;
- используя указанные функции, написать программу, которая:
  - отображает диаграмму размещения входных точек из **P** на плоскости с координатами (*p1, p2*);
  - обучает АЛЭ правильному распознаванию входных классов с использованием 2-х указанных функций при разных значениях параметра α;
  - строит кривые обучения - зависимости СКО от номера эпохи для 2-х указанных функций (для этого необходимо модифицировать встроенные функции ann_ADALINE и ann_ADALINE_online);
  - накладывает на диаграмму входных точек границы решения после обучения слоя АЛЭ;
  - выполняет тестирование полученного решения для всех заданных входных данных;
- сравнить получаемые границы решения слоя АЛЭ с границами решения персептрона, полученными в лабораторной работе №3, обратив внимание на равноудаленность границ от точек соседних классов для случая слоя АЛЭ;
- сравнить кривые обучения слоя АЛЭ двумя модифицированными функциями ann_ADALINE и ann_ADALINE_online для случая, когда параметр α LMS-алгоритма значительно меньше α_{max} и когда он близок к α_{max} .

3.3. Исследуйте адаптивный линейный предсказатель (рисунок 4.4):

- сгенерируйте входной $y(k)$ и желаемый $t(k)=y(k)$ сигналы адаптивного предсказателя. При этом параметры генератора выбирайте в соответствии с вариантом из таблицы 4.1. Для генерации используйте следующий программный код:

```
//Значения параметров генератора L, F, Fc выбирайте из таблицы 4.1;
//Параметры:
L=      ;
F=      ;
Fc=     ;
//Генератор полигармонического входного сигнала;
td=1/(20*F);
t=0:td:2/F;
fi=(2*%pi*F).*t;
Y=0;
for i=1:L
   Y=Y+(Fc)/(Fc+2*%pi*F*i)*sin(fi*i);
end
T=Y;
```

Таблица 4.1 – Параметры генератора входного сигнала

Вариант	Число составляющих L	Основная частота F	Частота среза Fc
1	5	0.01	F*5
2	10	0.01	F*6
3	15	0.01	F*7
4	20	0.01	F*8
5	25	0.01	F*9
6	5	0.02	F*5
7	10	0.02	F*6
8	15	0.02	F*7
9	20	0.02	F*8
10	25	0.02	F*9
11	5	0.04	F*5
12	10	0.04	F*6
13	15	0.04	F*7
14	20	0.04	F*8
15	25	0.04	F*9

- запишите развернутые выражения для всех элементов (**R**,**h**,c) целевой функции предсказателя, заданной в виде СКО (4.12);
- вычислите конкретные значения матрицы **R**, вектора **h** и константы c для сгенерированного входного сигнала $y(k)$;
- вычислите собственные значения и собственные векторы матрицы Гессе целевой функции предсказателя, точку минимума целевой функции, постройте линии контуров равных уровней целевой функции;
- вычислите максимальное устойчивое значение скорости обучения α_{max} для LMS-алгоритма;
- напишите программу, обучающую предсказатель с использованием встроенной функции ann_ADALINE_predict пакета NeuralNetworks 2.0;

- модифицируйте функцию ann_ADALINE_predict таким образом, чтобы по результатам её работы можно было построить кривую обучения предсказателя и траекторию движения вектора параметров предсказателя на диаграмме контуров равных уровней;
- постройте в одном графическом окне графики входного процесса и его предсказанных значений, кривую обучения, траекторию движения вектора параметров на диаграмме контуров;
- убедитесь, что алгоритм обучения сходится, если $\alpha<\alpha_{max}$ и нестабилен когда $\alpha>\alpha_{max}$;
- убедитесь, что при малых α траектория движения вектора параметров при использовании LMS алгоритма аппроксимирует в среднем траекторию движения вектора параметров алгоритма наискорейшего спуска.

3.4. Подготовьте и защитите отчет по работе.

4. Методические рекомендации по выполнению работы

4.1. Модуль NeuralNetwork 2.0 пакета Scilab содержит встроенные функции для обучения и моделирования АЛЭ:

[w,b]= ann_ADALINE(P, T, alpha, itermax, initfunc) – функция обучения АЛЭ, реализующая блочный (пакетный, batch) LMS-алгоритм, код которой с комментариями приведен в приложении А;
[w,b]= ann_ADALINE_online(P, T, alpha, itermax, initfunc) – функция обучения АЛЭ, реализующая последовательный (адаптивный) LMS-алгоритм, код которой с комментариями приведен в приложении Б;
y= ann_ ADALINE _run(P,w,b) – функция моделирования слоя АЛЭ, код которой с комментариями приведен в приложении Г.

4.2. При выполнении задания 3.2 необходимо сначала сформировать матрицу **T** целевых выходов слоя из АЛЭ. Используйте для этого матрицу целевых выходов, построенную в предыдущей лабораторной работе, заменив в ней все нули на -1. Например, для рисунка 3.5 матрица **T** целевых значений выходов классификатора, реализованного в виде слоя из АЛЭ, будет равна:

T= [-1 -1 -1 -1 1 1 1 1;
-1 -1 1 1 -1 -1 1 1].

Так как входные векторы **p** равновероятны, то для вычисления корреляционной матрицы **R** используйте простую формулу

$$\mathbf{R} = 1/Q \sum_{q=1}^{Q} \mathbf{z}_q \mathbf{z}_q^T ,$$

где $\boldsymbol{z}_q^T = [\mathbf{p}_q \ \ 1]$ – расширенный входной вектор слоя АЛЭ (здесь 1 соответствует входу с весом, равным смещению b), Q – число входных векторов **p.**

Для вычисления собственных значений гессиана **A=2R** целевой функции, заданной в виде СКО, используйте вызов встроенной функции Scilab:

```
evals=spec(2*R)
```

Для написания программы, которая выполняет обучение АЛЭ правильному распознаванию входных классов в двух режимах – блочном и последовательном – используйте модифицированные функции обучения ann_ADALINE и ann_ADALINE_online:

```
[w1,b1,mse1] = ann_ADALINE1(P,T,alpha,maxiter,'zeros');
[w2,b2,mse2] = ann_ADALINE1_online(P,T,alpha,maxiter,'zeros');
```

Функции ann_ADALINE1 и ann_ADALINE1_online модифицируется таким образом, чтобы можно было после обучения построить кривые обучения – зависимости ошибки СКО от номера эпохи обучения. Для этого указанные функции, кроме параметров обученного слоя АЛЭ, должны возвращать массивы значений средних квадратов ошибок, соответственно mse1 и mse2. Схема модификации кода функции ann_ADALINE:

```
function [w, b, mse1]=ann_ADALINE1(P, T, alpha, itermax, initfunc)
…
itercnt = 0;
mse1=zeros(1,itermax);
while itercnt < itermax
        …
          itercnt = itercnt + 1;
          mse1(itercnt) = mean(e.^2);
          //disp('Epoch:….
end
endfunction
```

Схема модификации кода функции ann_ADALINE1_online:

```
function [w, b, mse2]=ann_ADALINE1_online(P, T, alpha, itermax, initfunc)
…
itercnt = 0;
mse2=zeros(1,itermax);
while itercnt < itermax
   for cnt = 1:size(P,2)
        ….
       e_all(cnt) = e.^2
   end
   itercnt = itercnt + 1;
   mse2(itercnt)=mean(e_all);
  //disp('Epoch:…
end
endfunction
```

Добавленные операторы выделены красным цветом. Операторы отображения *disp* внутри функций рекомендуется исключить для уменьшения объемов данных, выводимых на экран.

Теперь, используя значения матриц **P** и **T**, вычисленное значение α_{max}, можно обучить слой АЛЭ, вызвав модифицированные функции обучения. По результатам обучения следует построить графики кривых обучения для 2-х модифицированных функций:

```
x_axis=1:maxiter;
plot(x_axis,mse1(x_axis),'r',x_axis,mse2(x_axis),'g');
xtitle('Средний квадрат ошибки', 'Эпоха','СКО');
```

В качестве примера на рисунке 4.6 изображены кривые обучения при разных значениях параметра скорости обучения α: когда α мало и когда близко к максимально допустимому значению (для рассматриваемого примера α_{max} =0,19) .

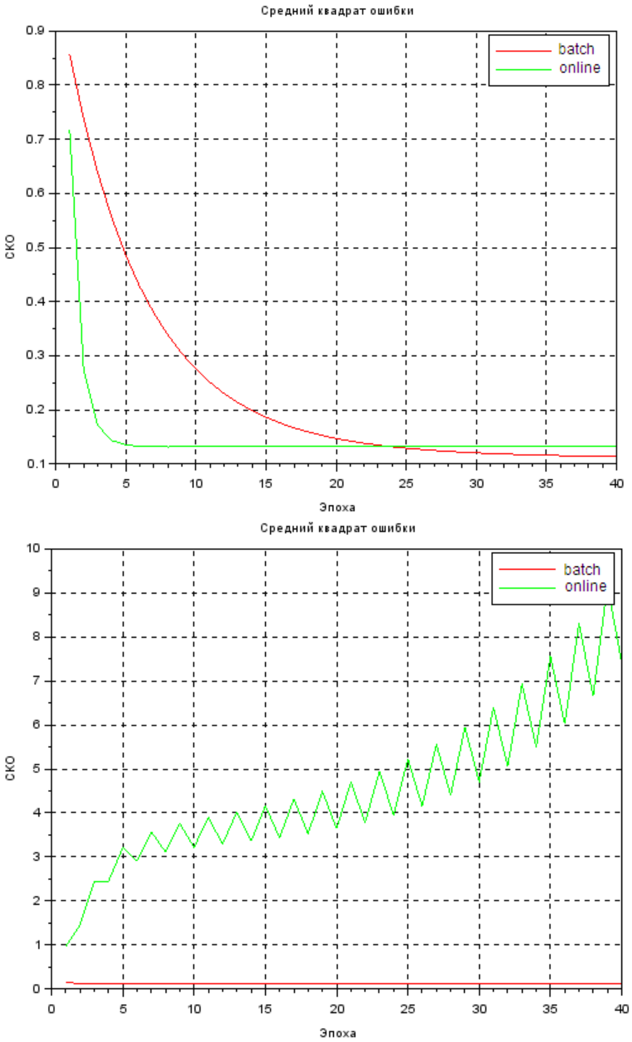

Рисунок 4.6 – Сравнение кривых обучения: слева α=0,02; справа α=0,19

Как следует из рисунка 4.6, блочный LMS-алгоритм при малых α требует для обучения большего числа эпох, чем последовательный алгоритм, но в итоге дает более точное решение. Последовательный LMS-алгоритм при малых α обучается быстрее, но при значении $\alpha = \alpha_{max}$ теряет устойчивость.

Для отображения диаграммы расположения точек классов и границ полученных решений (рисунок 4.7) используйте код, разработанный при выполнении лабораторной работы №3.

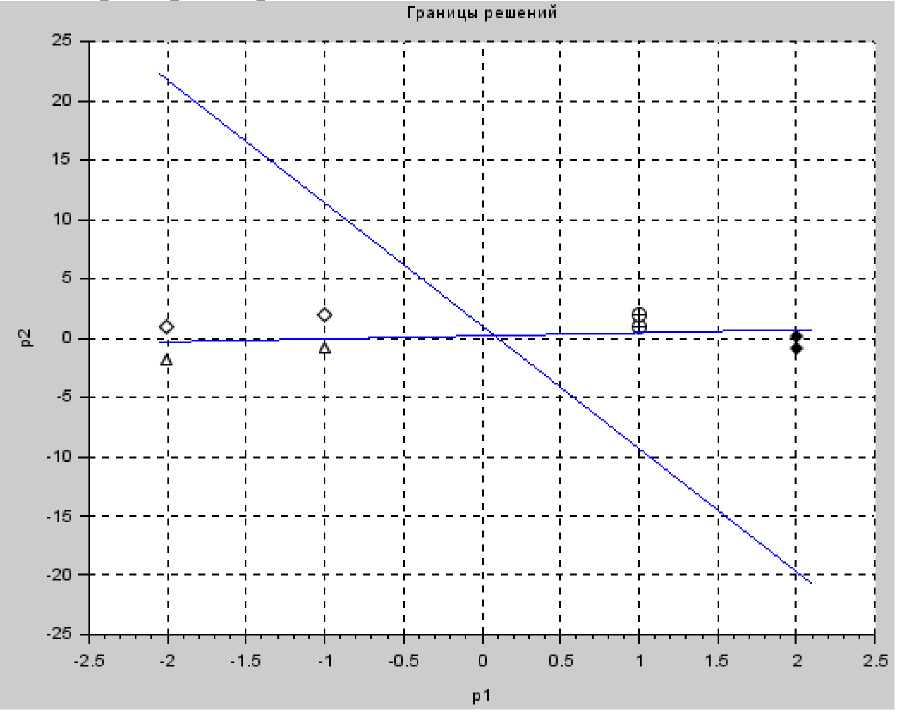

Рисунок 4.7 – Границы решений слоя АЛЭ (блочный LMS-алгоритм)

Сравните полученное решение с решением, изображенным на рисунке 3.5. Обратите внимание, что LMS-алгоритм находит границы решения, равноудаленные от центров соседних классов, и получаемое решение не зависит от способа инициализации весов слоя АЛЭ. В то же время слой персептронов формировал различные границы решения при разных начальных значениях весов.

4.3. В задании 3.3 требуется обучить линейный предсказатель значений стационарного сигнала $y(k)$ по двум его предыдущим значениям. Соответствующая схема предсказателя с двумя элементами задержки D=2 изображена на рисунке 4.4. В соответствии с заданием вектор отсчетов входного сигнала **Y** и вектор целевых значений **T** предсказателя генерируются с помощью программного кода, приведенного в п.3.3. Отметим, что **T**=**Y**.

На первом этапе необходимо вычислить все элементы (**R**,**h**,c) целевой функции предсказателя, заданной в виде СКО (4.12). Для этого следует воспользоваться общими формулами (4.26)-(4.27). Чтобы применить эти формулы, нужно сначала получить значения вектора **p** , элементы которого соответствуют выходам линии задержки (см. рисунок 4.3). Для формирования матрицы **P**, каждый столбец которой равен очередному входному вектору **p** АЛЭ, можно использовать начальную часть кода встроенной функции ann_ADALINE_predict (приложение B), предназначенной для обучения адаптивного линейного предсказателя в последовательном режиме на основе LMS-алгоритма:

```
T=Y;
D=2;
P = [];
for cnt = 1:D // для каждого выхода линии задержки
   //формируем строки  матрицы P из отсчетов Y
   // очередная строка P – сдвинутая на один отсчет копия предыдущей строки
   P = [P; Y(cnt:$-D+cnt-1)];
end
//формируем вектор целевых значений с длиной, равной длине строки из P
T1 = T(1:$-D+1);
```

Теперь, используя формулы (4.26)-(4.27), можно вычислить значения **R**,**h** и **c.** Для вычисления собственных векторов и собственных значений матрицы Гессе (**A**=**2R**) целевой функции предсказателя следует использовать вызов встроенной функции Scilab в форме:

```
[evals, diagevals]=spec(2*R).
```

Здесь переменная evals – вектор-столбец из собственных значений матрицы **2R**, а diagevals – матрица, столбцы которой представляют собственные векторы матрицы **2R.** При известных значениях матрицы **R** и вектора **h** *точка минимума целевой функции* (**x***, xstar) находится в виде решения минимума среднего квадрата ошибки (4.15).

Для построения контуров равных уровней целевой функции следует получить уравнение её поверхности (4.12) в виде функции матрицы **R**, вектора **h** и элементов вектора **x.** Пусть

$$\mathbf{R} = \mathbf{Cor}\ ,\ \ \mathbf{x} = \begin{bmatrix} \mathrm{w}(1) \\ \mathrm{w}(2) \end{bmatrix}.$$

Тогда для вычисления значений целевой функции (4.12) в зависимости от её параметров можно определить в Scilab следующую функцию:

```
function z=f(w1, w2, c, h, Cor)
   x=[w1;w2];
   z=c-2*x'*h+x'*Cor*x;
endfunction
```

Для построения 3D поверхности и линий контуров равных уровней используйте функции feval, surf и contour2d:

```
x=linspace(-20,20,100);
y=linspace(-20,20,100);
z=feval(x,y,f); //вычисляем значения высот целевой функции f на сетке x,y

clf(1);
figure(1);
subplot(1,2,1);
surf(x,y,z'); //строим 3D поверхность
```

```
subplot(1,2,2);
contour2d(x,y,z,30); //отображаем линии контуров равных уровней
xset("fpf"," "); //подавляем отображение значений на линиях контуров
xtitle("Контуры равных уровней квадратичной функции","w1","w2");
xgrid;
plot(xstar(1),xstar(2),'*b'); //отображаем точку решения минимума СКО (4.15)
```

В результате работы приведенного фрагмента кода будут построена 3D поверхность и линии контуров равных уровней (рисунок 4.8., см. ниже).

Для обучения адаптивного предсказателя используйте встроенную функцию ann_ADALINE_predict пакета NeuralNetworks 2.0. Чтобы по результатам её работы можно было построить кривую обучения предсказателя и траекторию движения вектора параметров предсказателя модифицируйте функцию по схеме:

```
function [w,b,y,ee,mse,W] = ann_ADALINE1_predict(Y,T,alpha,itermax,D,initfunc)
…
mse=zeros(1,itermax); // создаем вектор СКО
W=[w]; //матрица, каждая строка которой – вектор весов на очередном шаге
itercnt = 0; // Счетчик итераций - эпох
while itercnt < itermax
   for cnt = 1:size(P,2)  // Цикл по всем обучающим примерам из P (1 эпоха)
      …
      w = (w + 2*alpha*e*P(:,cnt)');
      // b = b + 2*alpha*e;
      // Вычисляем и запоминаем квадраты текущих  ошибок
      e_all(cnt) = e.^2;
      W=[W;w];
   end
   itercnt = itercnt + 1;
   mse(itercnt)=mean(e_all); // вычисляем СКО на шаге и запоминаем
   //disp('Epoch: …
   …
end
endfunction
```

Теперь функция обучения будет возвращать вектор СКО ошибок **mse** и матрицу **W** из векторов-строк весовых коэффициентов предсказателя на каждой итерации. Вектор **mse** позволяет построить кривые обучения (см. пример на рисунке 4.6), а матрица **W** – траекторию движения вектора параметров предсказателя, наложив её (траекторию) на линии равных контуров (рисунок 4.8):

```
plot2d(W(:,1),W(:,2),5);
```

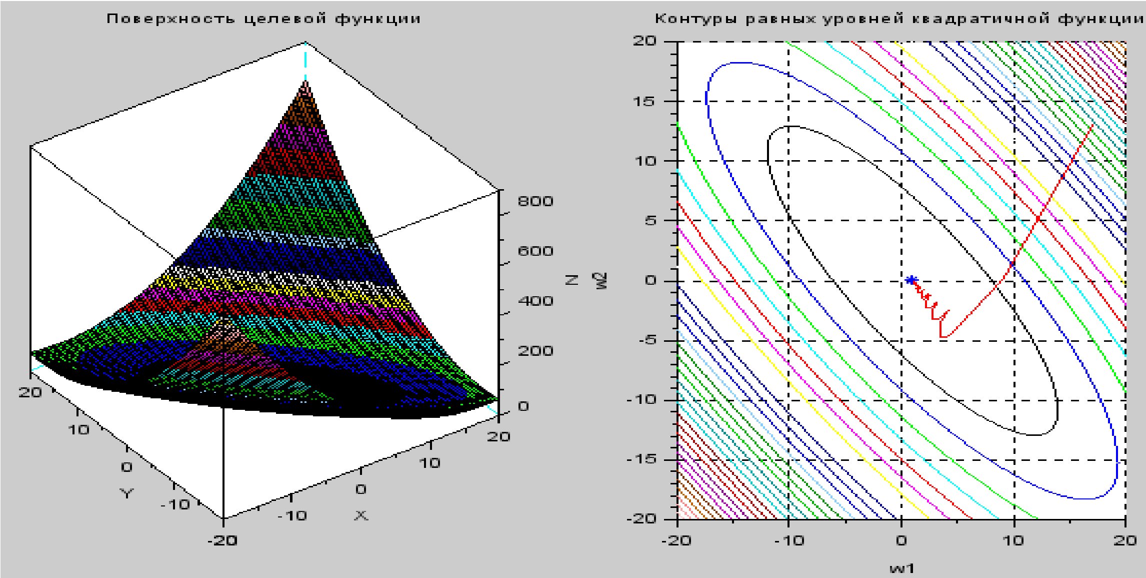

Рисунок 4.8 – Поверхность целевой функции, линии контуров
и траектория движения вектора параметров для LMS-алгоритма.

5. Содержание отчета

5.1. Цель работы.
5.2. Вариант задания.
5.3. Схема слоя АЛЭ, решающего задачу классификации 4-х классов; вычисленное значение матрицы **R** и её собственных чисел, значения весов и смещений, максимальное значения скорости обучения; диаграмма расположения классов с границами решений, соответствующими обученному слою АЛЭ; графики с кривыми обучения; листинг программы с комментариями.
5.4. Схема адаптивного линейного предсказателя; правила обучения; результаты вычислений: значение матрицы **R** и её собственных чисел и векторов, значение максимальной устойчивой скорости обучения; решение минимума СКО; кривые обучения и результирующие значения параметров предсказателя; графики 3D поверхности целевой функции и линий контуров равных уровней, траектории движения вектора параметров; листинг программы с комментариями.
5.5. Выводы по результатам исследований.

6. Контрольные вопросы

6.1 Нарисуйте схему однослойной сети из адаптивных линейных элементов, объясните все обозначения, запишите формулу вычисления вектора выходных значений и объясните её.
6.2 Сколько линейно-разделимых классов способна распознать сеть, содержащий S адаптивных линейных нейронов?
6.3. Изобразите АЛЭ с 2-мя входами. Запишите уравнение границы решения.
6.6. Как связана ориентация вектора весов с границей решения АЛЭ?
6.7. Что понимают под целевой функцией сети?

6.8. Сформулируйте условие минимума первого порядка для целевой функции.
6.9. Сформулируйте условие минимума второго порядка для целевой функции.
6.10. Что такое матрица Гессе целевой функции? Каким условиям должна удовлетворять эта матрица, чтобы целевая функция имела строгий минимум?
6.11. Запишите выражение для квадратичной целевой функции в общей форме, найдите градиент и гессиан этой функции.
6.12. Сформулируйте алгоритм наискорейшего спуска. Запишите условие схождения алгоритма.
6.13. Запишите выражение для целевой функции в виде среднего квадрата ошибки. Приведите его к квадратичной форме.
6.14. Запишите условие минимума СКО целевой функции и получите решение минимума СКО.
6.15. Что такое стохастический градиент? Сформулируйте LMS-алгоритм.
6.16. Запишите выражения LMS-алгоритма в матричной форме, сформулируйте условие схождения алгоритма.
6.17 Изобразите схему адаптивного фильтра на основе АЛЭ. Укажите области применения адаптивной фильтрации.
6.18. Изобразите схему адаптивного линейного предсказателя. Запишите выражения для вычисления корреляционной матрицы **R** и вектора кросс-корреляции **h**.
6.19. Сформулируйте гарантирующие условие схождения LMS-алгоритма для трансверсального фильтра.
6.20. Что такое кривая обучения? Как её построить? Какие выводы можно сделать, анализируя кривую обучения?
6.21. Как ориентирована траектория движения вектора параметров на графике линий контуров равных уровней? Чем объясняется хаотичность траектории для LMS-алгоритма?
6.22. Объясните алгоритмы, реализуемые функциями ann_ADALINE, ann_ADALINE_online, ann_ADALINE_predict.

Приложение А. Код функции ann_ADALINE

```
function [w, b]=ann_ADALINE(P, T, alpha, itermax, initfunc)
// Обучение слоя адаптивных линейных нейронов  в блочном режиме
// Длина блока Q  равна числу обучающих примеров
// Примеры вызовов
//     [w,b] = ann_ADALINE(P,T)
//     [w,b] = ann_ADALINE(P,T,alpha,itermax,initfunc)
//Параметры:
//     P : Матрица обучающих примеров (RxQ)
//     T : Матрица целевых выходных значений (SxQ)
//     alpha : Скорость обучения (по умолчанию =0.01)
//     itermaxs : Максимальное число итераций – эпох (по умолчанию – 100)
//     initfunc : Функция инициализации значений w и b: 'rand', 'zeros', или 'ones'
//                 По умолчанию используется ‘rand’.
//     w : веса сети (SxR))
```

```
//     b : смещения (Sx1)

// 1.===== Обработка списка входных аргументов функции ======
rhs=argn(2); // argn(2) – возвращает число аргументов  функции
if rhs < 2; error("Должно быть минимум 2 аргумента: P и T"); end
if rhs < 3; alpha = 0.01; end
if rhs < 4; itermax = 100; end
if rhs < 5; w = rand(size(T,1),size(P,1)); b = rand(size(T,1),1); end
//Инициализация весов и смещений
if  rhs == 5 then
    select initfunc
    case 'rand' then
        w = rand(size(T,1),size(P,1));
        b = rand(size(T,1),1);
    case 'zeros' then
        w = zeros(size(T,1),size(P,1));
        b = zeros(size(T,1),1);
    case 'ones' then
        w = ones(size(T,1),size(P,1));
        b = ones(size(T,1),1);
    else
        error("Неверное значение входного аргумента 5");
    end
end

if itermax == []; itermax = 100; end
if alpha == []; alpha = 0.01; end

//2. ====== Реализация правил обучения ==========================
itercnt = 0; // Счетчик итераций - эпох
while itercnt < itermax
    // Вычисляем вектор выхода сети a и вектор ошибки сети e
    // сразу для всего блока данных P (1 эпоха)
    n = w*P + repmat(b,1,size(P,2));
    a = ann_purelin_activ(n);
    e = T - a;
    //Реализуем блочное  правило обучения Уидроу-Хоффа
    // вычисляется среднее обновление для w и b  в пределах блока
    w = w + (2*alpha*e*P')./size(P,2);
    b = b + 2*alpha*mean(e,2);
    // Вычисляем вектор ошибки при обновленных w и b
    n = w*P +  repmat(b,1,size(P,2));
    a = ann_purelin_activ(n);
    e = T - a;
    // Вычисляем значения СКО
    mse = mean(e.^2);
    itercnt = itercnt + 1;
    disp('Эпоха: ' + string(itercnt) + ' СКО: ' + string(mean(mse)));
end
endfunction
```

Приложение Б. Код функции ann_ADALINE_online

```
function [w, b]=ann_ADALINE_online(P, T, alpha, itermax, initfunc)
// Обучение слоя адаптивных линейных нейронов  в последовательном режиме
// Примеры вызовов
//     [w,b] = ann_ADALINE_online(P,T)
//     [w,b] = ann_ADALINE_online (P,T,alpha,itermax,initfunc)
//Параметры:
//     P : Матрица обучающих примеров (RxQ)
//     T : Матрица целевых выходных значений (SxQ)
//     alpha : Скорость обучения (по умолчанию =0.01)
```

```
//     itermaxs : Максимальное число итераций – эпох (по умолчанию – 100)
//     initfunc : Функция инициализации значений w и b: 'rand', 'zeros', или 'ones'
//                   По умолчанию используется ‘rand’.
//     w : веса сети (SxR))
//     b : смещения (Sx1)

// 1.===== Обработка списка входных аргументов функции ======
rhs=argn(2); // argn(2) – возвращает число аргументов  функции
if rhs < 2; error("Должно быть минимум 2 аргумента: P и T"); end
if rhs < 3; alpha = 0.01; end
if rhs < 4; itermax = 100; end
if rhs < 5; w = rand(size(T,1),size(P,1)); b = rand(size(T,1),1); end
//Инициализация весов и смещений
if  rhs == 5 then
   select initfunc
   case 'rand' then
      w = rand(size(T,1),size(P,1));
      b = rand(size(T,1),1);
   case 'zeros' then
      w = zeros(size(T,1),size(P,1));
      b = zeros(size(T,1),1);
   case 'ones' then
      w = ones(size(T,1),size(P,1));
      b = ones(size(T,1),1);
   else
      error("Неверное значение входного аргумента 5”);
   end
end

if itermax == []; itermax = 100; end
if alpha == []; alpha = 0.01; end

//2. ====== Реализация правил обучения ==========================
itercnt = 0; // Счетчик итераций - эпох
while itercnt < itermax
   for cnt = 1:size(P,2) // Цикл по всем обучающим примерам  - 1 эпоха
      // Вычисляем вектор текущей ошибки сети e для одного примера P(:,cnt)
      e = T(:,cnt) - ann_purelin_activ(w*P(:,cnt)+b);
     //Реализуем правило обучения Уидроу-Хоффа
      //для каждого примера  P(:,cnt)  вычисляем обновление  w и b
      w = (w + 2*alpha*e*P(:,cnt)');
      b = b + 2*alpha*e;
      // Вычисляем и запоминаем квадраты текущих  ошибок
      e_all(cnt) = e.^2;
   end
   itercnt = itercnt + 1;
   disp('Epoch: ' + string(itercnt) + '  MSE: ' + string(mean(e_all)));
end
endfunction
```

Приложение B. Код функции ann_ADALINE_predict

```
function [w, b, y, ee]=ann_ADALINE_predict(X, T, alpha, itermax, D, initfunc)
// Обучение адаптивного линейного предсказателя с линией задержки
 // в последовательном режиме
// Примеры вызовов
//     [w,b] = ann_ADALINE__predict (P,T, alpha)
//     [w,b] = ann_ADALINE__predict (P,T,alpha,itermax, D, initfunc)
//Параметры:
//     X: входные значения  предсказателя
//     P : Матрица обучающих примеров (RxQ)
```

```
//     T : Матрица целевых выходных значений (SxQ)
//     alpha: Скорость обучения (по умолчанию – 0.01)
//     itermaxs : Максимальное число итераций – эпох (по умолчанию – 100)
//     D – число элементов задержки (по умолчанию – 1)
//     initfunc : Функция инициализации значений w и b: 'rand', 'zeros', или 'ones'
//                  По умолчанию используется ‘rand’.
//     w : веса сети (SxR))
//     b : смещения (Sx1)
//      y : Выход предсказаният
//      ee : Ошибка между  T и  y

// 1.===== Обработка списка входных аргументов функции ======
rhs=argn(2); // argn(2) – возвращает число аргументов  функции
if rhs < 2; error("Должно быть минимум 2 аргумента: P и T"); end
if rhs < 3; alpha = 0.01; end
if rhs < 4; itermax = 100; end
if rhs < 5; D = 1; end
if rhs < 6; w = rand(size(T,1),D); b = rand(size(T,1),1); end

//Инициализация весов и смещений
if  rhs == 6 then
    select initfunc
    case 'rand' then
        w = rand(size(T,1),D);
        b = rand(size(T,1),1);
    case 'zeros' then
        w = zeros(size(T,1),D);
        b = zeros(size(T,1),1);
    case 'ones' then
        w = ones(size(T,1),D);
        b = ones(size(T,1),1);
    else
        error("Неверное значение входного аргумента 5”);
    end
end

if itermax == []; itermax = 100; end
if alpha == []; alpha = 0.01; end
if D == []; D = 1; end

// Формирование выходов линии задержки: реформатирование X в матрицу P
P = [];
for cnt = 1:D // для каждого выхода линии задержки
    //формируем строки  матрицы P из отсчетов X
    // очередная строка P – сдвинутая на один отсчет копия предыдущей строки
    P = [P; X(cnt:$-D+cnt-1)];
end
//формируем вектор целевых значений
T = T(1:$-D+1);

//2. ====== Реализация правил обучения ==========================
itercnt = 0; // Счетчик итераций - эпох
while itercnt < itermax
    for cnt = 1:size(P,2)  // Цикл по всем обучающим примерам из P (1 эпоха)
        n = w*P(:,cnt)+b;  // Сетевая функция АЛЭ
        a = ann_purelin_activ(n);
        e = T(:,cnt) - a;    // Ошибка
        y(cnt) = a;          // Запоминаем выход АЛЭ
        ee(cnt) = e;         // Запоминаем ошибку
        //Реализуем правило обучения Уидроу-Хоффа
        //для каждого примера  P(:,cnt)  вычисляем обновление  w и b
        w = (w + 2*alpha*e*P(:,cnt)');
        b = b + 2*alpha*e;
```

```
    // Вычисляем и запоминаем квадраты текущих  ошибок
    e_all(cnt) = e.^2;
  end
  itercnt = itercnt + 1;
  disp('Epoch: ' + string(itercnt) + '  MSE: ' + string(mean(e_all)));

end

endfunction
```

Приложение Г. Код функции ann_ADALINE_run

```
function y=ann_ADALINE_run(P, w, b)
// Функция моделирования адаптивного линейного элемента.
// Вызов:
//     y = ann_ADALINE_run(P,w,b)
//
// Параметры:
//     P : Матрица  входных тестовых примеров (RxQ)
//     w : веса слоя (SxR)
//     b : смещения слоя(Sx1)
//     y : выход слоя (SxQ)
y = ann_purelin_activ(w*P+repmat(b,1,size(P,2)));

endfunction
```

Список рекомендованной литературы

1. Бондарев В.Н. Искусственный интеллект: Учеб. пособие для студентов вузов / В. Н. Бондарев, Ф. Г. Аде. — Севастополь: Изд-во СевНТУ, 2002. — 613 с.
2. Ерин С.В. Scilab – примеры и задачи: практическое пособие / С.В. Ерин – М.: Лаборатория «Знания будущего», 2017. – 154 с.
3. Медведев, В.С. Нейронные сети. MATLAB 6 / В.С. Медведев, В.Г. Потемкин; под общ. ред. В.Г. Потемкина. — М.: ДИАЛОГ-МИФИ, 2002. — 496 с.
4. Хайкин С. Нейронные сети: Полный курс. Пер. С англ. / С. Хайкин. — М.: Изд. «Вильямс», 2006. — 1104 с.
5. Hagan M.T. Neural Network Design. The 2nd edition [Электронный ресурс] /M.T.Hagan, H.B.Demuth, M.H.Beale, O.D. Jesus. . — Frisco, Texas, 2014 . — 1012 p. Режим доступа: https://www.hagan.okstate.edu/NNDesign.pdf. —Последний доступ: 14.01.2019. —Название с экрана.