Лабораторная работа №5

Исследование многослойного персептрона: алгоритмы обратного распространения с адаптивной скоростью обучения и моментом

1 Цель работы

Углубление теоретических знаний в области архитектуры многослойных нейронных сетей прямого распространения, исследование свойств алгоритмов обучения многослойных нейронных сетей, приобретение практических навыков обучения многослойного персептрона при решении задач классификации и аппроксимации функций.

2 Основные теоретические положения

2.1 Структура многослойного персептрона и его возможности

Многослойный персептрон (MLP – multi-layer perceptron) состоит из нескольких слоёв. В качестве примера на рисунке 5.1 изображена структурная схема 3-х слойного персептрона. Структуру сети можно описывать сокращенно: R-S^1-S^2-S^3, где R – число входов персептрона, S^1-S^2-S^3 -число нейронов в каждом слое.

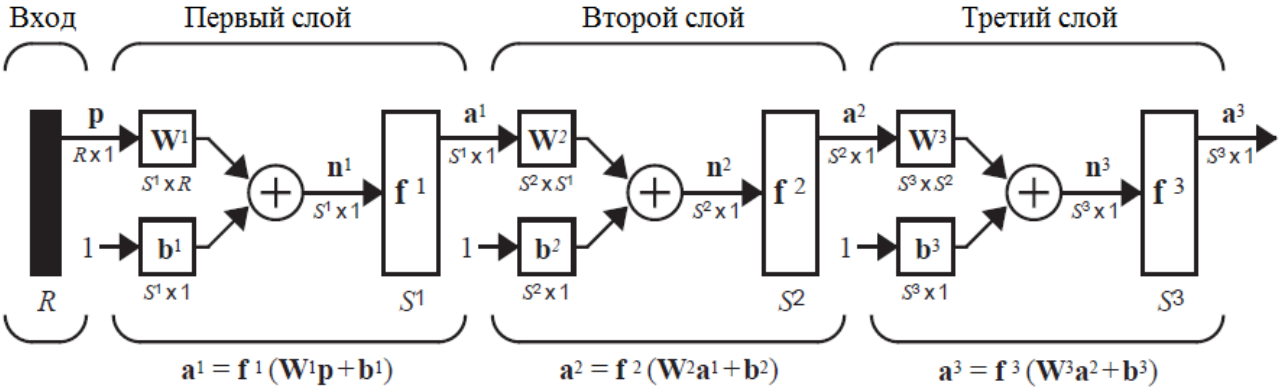

Рисунок 5.1 – Структура 3-х слойного персептрона

В MLP выход одного слоя является входом следующего слоя:

$$\mathbf{a}^{m+1} = \mathbf{f}^{m+1}(\mathbf{W}^{m+1}\mathbf{a}^{m} + \mathbf{b}^{m+1}) \text{ for } m = 0, 1, \ldots, M-1, \qquad (5.1)$$

где m – номер слоя, M – число слоев MLP. На вход нейронов первого слоя подается внешний вектор **p**:

$$\mathbf{a}^{0} = \mathbf{p}.$$

Выход последнего слоя является выходом MLP:

$$\mathbf{a} = \mathbf{a}^M .$$

Для моделирования работы MLP в модуле NeuralNetworks 2.0 пакета Scilab имеется функция ann_FFBP_run, код которой с комментариями приведен в приложении А. Функция ann_FFBP_run(P,W,af) принимает на вход матрицу P, столбцы которой представляют собой входные векторы **p**, расширенную матрицу весов и смещений всех слоев W, список af имен активационных функций каждого слоя MLP. Функция реализует вычисления выходного вектора **a** в полном соответствии с формулой (5.1).

В отличие от однослойного персептрона MLP способен решать задачи классификации, которые не ограничиваются случаем линейной сепарабельности. Например, MLP успешно решает задачу исключающего ИЛИ (рисунок 5.2). Одно из решений – использовать 2 нейрона в первом слое, которые создают две границы. Первая отделяет p_1=(0,0) от остальных образов, а вторая отделяет образ p_4=(1,1) . Второй слой комбинирует эти две границы, используя AND операцию

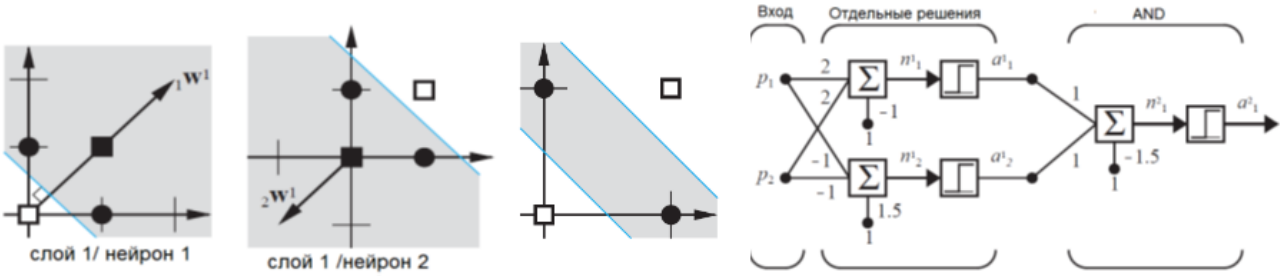

Рисунок 5.2 – Структура MLP для решения задачи исключающего ИЛИ

MLP также широко используют для аппроксимации функций. Можно показать, что *2-х слойная сеть со скрытым сигмоидальным слоем и линейным выходным слоем способна аппроксимировать любую непрерывную функцию с заданной точностью при надлежащем числе нейронов скрытого слоя* (теорема универсальной аппроксимации [3, 4]). Отметим, что теорема универсальной аппроксимации ничего не говорит об оптимальности MLP с одним скрытым слоем для решения задачи аппроксимации в отношении времени обучения, простоты применения, обобщающих способностей обученного MLP.

2.2 Целевая функция и BP-алгоритм обучения MLP

Обучение MLP осуществляется на основе **алгоритма обратного распространения (BP – back propagation)**. Алгоритм BP является развитием алгоритма LMS. Оба алгоритма минимизируют целевую функцию в виде среднего квадрата ошибки (СКО, MSE). Для MLP со многими выходами СКО будет равен:

$$F(\mathbf{x}) = E[\mathbf{e}^T\mathbf{e}] = E[(\mathbf{t}-\mathbf{a})^T(\mathbf{t}-\mathbf{a})]. \tag{5.2}$$

Как и LMS, **алгоритм BP аппроксимирует СКО квадратами ошибки** на каждом шаге k

$$\hat{F}(\mathbf{x}) = (\mathbf{t}(k)-\mathbf{a}(k))^T(\mathbf{t}(k)-\mathbf{a}(k)) = \mathbf{e}^T(k)\mathbf{e}(k). \tag{5.3}$$

Тогда в соответствии с алгоритмом наискорейшего спуска SDA (4.9**)** **алгоритм обучения MLP** запишется в виде

$$w^m_{i,j}(k+1) = w^m_{i,j}(k) - \alpha\frac{\partial\hat{F}}{\partial w^m_{i,j}},$$
$$b^m_i(k+1) = b^m_i(k) - \alpha\frac{\partial\hat{F}}{\partial b^m_i} \tag{5.4}$$

С учетом того, что целевая функция F в этих формулах является стохастической, то они подобны формулам алгоритма LMS. **Сложность заключается в вычислении частных производных.** Если ввести обозначение

$$s^m_i \equiv \frac{\partial\hat{F}}{\partial n^m_i}, \tag{5.5}$$

где s^m_i — **чувствительность** целевой функции F к изменению n^m_i — i-го сетевого входа в слое m, то можно показать, что выражения для частных производных в формуле (5.4) запишутся в виде

$$\frac{\partial\hat{F}}{\partial w^m_{i,j}} = s^m_i a^{m-1}_j, \quad \frac{\partial\hat{F}}{\partial b^m_i} = s^m_i. \tag{5.6}$$

Соответственно стохастическая аппроксимация SDA (5.4) для MLP запишется в виде

$$w^m_{i,j}(k+1) = w^m_{i,j}(k) - \alpha s^m_i a^{m-1}_j, \quad b^m_i(k+1) = b^m_i(k) - \alpha s^m_i \tag{5.7}$$

Или векторно-матричной форме

$$\mathbf{W}^m(k+1) = \mathbf{W}^m(k) - \alpha\mathbf{s}^m(\mathbf{a}^{m-1})^T,$$
$$\mathbf{b}^m(k+1) = \mathbf{b}^m(k) - \alpha\mathbf{s}^m. \tag{5.8}$$

Чувствительности $\mathbf{s}^m$ также вычисляются с помощью рекуррентных соотношений. При этом чувствительность слоя m определяется через чувствительность слоя m+1, т.е. в обратном направлении **(backpropagation).** Чтобы показать это, запишем рекуррентное соотношение для вычисления чувствительностей, используя цепочное правило дифференцирования сложной функции

$$\mathbf{s}^m = \frac{\partial \hat{F}}{\partial \mathbf{n}^m} = \left(\frac{\partial \mathbf{n}^{m+1}}{\partial \mathbf{n}^m}\right)^T \frac{\partial \hat{F}}{\partial \mathbf{n}^{m+1}} = \left(\frac{\partial \mathbf{n}^{m+1}}{\partial \mathbf{n}^m}\right)^T \mathbf{s}^{m+1} \quad (5.9).$$

Здесь матрица частных производных от векторной функции $\mathbf{n}^{m+1}$ по $\mathbf{n}^m$, называемая *якобианом* (производная от векторной функции), равна

$$\frac{\partial \mathbf{n}^{m+1}}{\partial \mathbf{n}^m} = \mathbf{W}^{m+1}\dot{\mathbf{F}}^m(\mathbf{n}^m), \quad (5.10)$$

где $\dot{\mathbf{F}}^m(\mathbf{n}^m)$ — диагональная матрица с производными $\dot{f}^m(n_j^m)$ на главной диагонали.

Таким образом, **значения чувствительностей слоев распространяются в обратном направлении – от последнего слоя к первому.** Значение чувствительности выходного слоя

$$s_i^M = -2(t_i - a_i)\dot{f}^M(n_i^M) \quad \text{или} \quad \mathbf{s}^M = -2\dot{\mathbf{F}}^M(\mathbf{n}^M)(\mathbf{t} - \mathbf{a}). \quad (5.11)$$

Объединяя формулы, приведенные выше, получим алгоритм обучения MLP, называемый *алгоритмом обратного распространения* или алгоритмом BP (рисунок 5.3).

Алгоритм BP

1. Инициализировать веса и смещения небольшими случайными значениями, задать допустимое значение ошибки $E_{\text{доп}}$.
2. Распространить очередной входной вектор по сети в прямом направлении: формула (5.1).
3. Распространить чувствительности в обратном направлении, выполнив вычисления по формулам:

$$\mathbf{s}^M = -2\dot{\mathbf{F}}^M(\mathbf{n}^M)(\mathbf{t} - \mathbf{a}),$$

$$\mathbf{s}^m = \dot{\mathbf{F}}^m(\mathbf{n}^m)(\mathbf{W}^{m+1})^T\mathbf{s}^{m+1}, \qquad m = M-1, \ldots, 2, 1. \quad (5.12)$$

4. Обновить веса и смещения в соответствии с формулой (5.8).
5. Проверить значение ошибки, если ошибка на выходе сети больше $E_{\text{доп}}$, то перейти к п.2., иначе “стоп”.

Рисунок 5.3 – Алгоритм обратного распространения

2.3 Последовательный и блочный алгоритмы BP

Рассмотренный алгоритм BP является последовательным, т.к. предполагает обновление параметров сети после предъявления каждого очередного входного вектора. Возможен альтернативный блочный вариант алгоритма, когда полный градиент вычисляется до обновления весов и

смещений (т.е. после подачи на вход сети всех входных векторов **p**). Например, если все входные векторы равновероятны, то СКО будет равен

$$F(\mathbf{x}) = E[\mathbf{e}^T\mathbf{e}] = E[(\mathbf{t}-\mathbf{a})^T(\mathbf{t}-\mathbf{a})] = \frac{1}{Q}\sum_{q=1}^{Q}(\mathbf{t}_q-\mathbf{a}_q)^T(\mathbf{t}_q-\mathbf{a}_q). \quad (5.13)$$

Полный градиент этой целевой функции будет равен среднему градиенту отдельных квадратов ошибок:

$$\nabla F(\mathbf{x}) = \nabla\left\{\frac{1}{Q}\sum_{q=1}^{Q}(\mathbf{t}_q-\mathbf{a}_q)^T(\mathbf{t}_q-\mathbf{a}_q)\right\} = \frac{1}{Q}\sum_{q=1}^{Q}\nabla\{(\mathbf{t}_q-\mathbf{a}_q)^T(\mathbf{t}_q-\mathbf{a}_q)\}. \quad (5.14)$$

Для реализации блочного алгоритма ВР выполняются шаги 1-3 алгоритма ВР для всех входных векторов. Затем индивидуальные градиенты усредняются, чтобы получить полный градиент, и параметры сети обновляются на основе формул, содержащих средние обновления:

$$\mathbf{W}^m(k+1) = \mathbf{W}^m(k) - \frac{\alpha}{Q}\sum_{q=1}^{Q}\mathbf{s}_q^m(\mathbf{a}_q^{m-1})^T, \quad \mathbf{b}^m(k+1) = \mathbf{b}^m(k) - \frac{\alpha}{Q}\sum_{q=1}^{Q}\mathbf{s}_q^m. \quad (5.15)$$

Так как рассмотренные алгоритмы ВР представляют собой аппроксимацию алгоритма градиентного спуска (gradient descent), то будем обозначать их как BP_GD. В модуле NeuralNetworks 2.0 пакета Scilab имеется функция ann_FFBP_gd, реализующая блочный вариант алгоритма BP_GD. Текст функции с комментариями приведен в приложении Б. В общем случае формат вызова функции требует задания следующих параметров:

```
W = ann_FFBP_gd(P,T,N,af,lr,itermax,mse_min,gd_min)
```

где

P- матрица обучающих примеров входных данных;

T – матрица обучающих целевых значений;

N – вектор, задающий число нейронов каждого слоя, включая входной и выходной слои. Например, [2 30 1] – для сети с 2-мя входами, скрытым слоем из 30 нейронов и выходным слоем с 1-м нейроном;

af – список имен активационных функций слоев, например ['ann_tansig_activ','ann_purelin_activ'] для сети со структурой N=[2 30 1]. Число элементов списка функций должно быть равно числу слоёв без учета входного слоя;

lr – скорость обучения (параметр α в формуле 5.8), по умолчанию 0,01;

itermax – максимальное число эпох обучения, по умолчанию 1000;

mse_min – минимальное допустимое значение ошибки $E_{доп}$ (значение СКО-целевой функции), по умолчанию 1e-5;

gd_min – минимальное значение градиента целевой функции, по умолчанию 1e-5.

В отличие от алгоритма BP_GD, описанного выше, функция ann_FFBP_gd прекращает процесс обучения при выполнении одного из 3-условий:

$$mse < mse\_min,\ itercnt > itermax,\ gd < gd\_min,$$

где mse, itercnt, gd – соответственно значения СКО, счетчика эпох и значения градиента целевой функции на каждой итерации алгоритма. Обратим внимание на то, что по мере приближения к минимуму целевой функции значения mse должны снижаться, а значения gd должны быть близкими к нулю.

2.4 Эвристические разновидности алгоритмов BP

Основной недостаток алгоритмов BP_GD – медленное схождение. Методы ускорения этих алгоритмов могут быть разделены на 2 группы:

1) эвристические методы:
   - методы, основанные на изменении скорости обучения;
   - методы, основанные на использовании моментов инерции;
2) методы, использующие алгоритмы оптимизации.

Рассмотрим эвристические методы. Алгоритмы, относящиеся к этой группе, используют процедуру BP_GD в качестве базовой. Как отмечалось выше, BP_GD эквивалентен LMS, если сеть однослойная. *Однако BP_GD, применяемый для обучения MLP, обладает совсем другими свойствами.* Это связано с отличием поверхностей целевых функций для однослойной и многослойной сетей. Если поверхность целевой функции однослойной сети имеет одну стационарную точку и постоянную кривизну, то поверхность целевой функции для MLP может иметь **много локальных минимумов** и **кривизна её может существенно изменяться** в различных областях пространства параметров.

Это говорит о том, что необходимо изменять скорость обучения в процессе обучения. Например, на участках, где целевая функция уменьшается можно скорость обучения повышать, на плоских участках – оставлять неизменной, а на участках с растущими значениями целевой функции скорость обучения следует снижать.

Другой путь улучшения сходимости – сглаживание траектории изменения параметров сети. Когда алгоритм расходится, он начинает осциллировать (обычно поперек некоторой длинной узкой ложбины целевой функции). Если выполнить фильтрацию (сглаживание) траектории спуска путем усреднения обновления параметров, то получим более устойчивую траекторию спуска.

Для преодоления проблемы нескольких минимумов, следует запускать алгоритм с разными начальными условиями. При этом необходимо контролировать минимум достигнутой ошибки и выбирать те параметры сети, которые обеспечивают минимальный уровень СКО.

В соответствии с указанными подходами сформулируем две эвристические разновидности алгоритма ВР для многослойной сети прямого распространения.

2.4.1 Алгоритм ВР с моментом инерции

Как указывалось, схождение можно улучшить, если сглаживать траекторию спуска. Это можно сделать с помощью фильтра нижних частот. Рассмотрим усредняющий фильтр 1-го порядка:

$$y(k) = \gamma y(k-1) + (1-\gamma)w(k), \tag{5.16}$$

где $w(k)$ – вход фильтра, $y(k)$ – выход фильтра, γ – коэффициент момента ($0 \le \gamma < 1$). Поскольку фильтр, в общем, суммирует очередные входные значения $w(k)$ с предыдущими своими выходами $y(k\text{-}1)$, то на выходе фильтра будут формироваться значения $y(k)$, соответствующие некоторому среднему значению $w(k)$. Иными словами, фильтр добавляет в динамику обновления $w(k)$ некоторый момент инерции. Чем больше γ, тем более инерционен фильтр. При $\gamma=0$ выход фильтра $y(k)= w(k)$ и фильтр не будет вносить никакой инерции в динамику $w(k)$.

В соответствии с алгоритмом ВР обновление параметров определяется выражениями

$$\Delta \mathbf{W}^m(k) = -\alpha \mathbf{s}^m (\mathbf{a}^{m-1})^T, \quad \Delta \mathbf{b}^m(k) = -\alpha \mathbf{s}^m.$$

При введении в эти уравнения момента инерции в соответствии с (5.16) получим *алгоритм BP_GD с моментом инерции*, который будем обозначать как BP_GDM

$$\begin{aligned} \Delta \mathbf{W}^m(k) &= \gamma \Delta \mathbf{W}^m(k-1) - (1-\gamma)\alpha \mathbf{s}^m (\mathbf{a}^{m-1})^T, \\ \Delta \mathbf{b}^m(k) &= \gamma \Delta \mathbf{b}^m(k-1) - (1-\gamma)\alpha \mathbf{s}^m. \end{aligned} \tag{5.17}$$

BP_GDM делает траекторию спуска более гладкой при большей скорости обучения по сравнению с алгоритмом BP_GD.

В модуле NeuralNetworks 2.0 пакета Scilab имеется функция ann_FFBP_gdm, реализующая этот алгоритм. В общем случае формат вызова функции требует задания следующих параметров:

```
W = ann_FFBP_gdm(P,T,N,af,lr,Mr,itermax,mse_min,gd_min),
```

где Mr – коэффициент момента (γ), по умолчанию 0,9. Остальные параметры функции аналогичны параметрам функции ann_FFBP_gd.

2.4.2 Алгоритм ВР с адаптивной скоростью обучения

Как отмечалось, скорость обучения из-за изменения кривизны поверхности целевой функции MLP следует изменять в процессе обучения. Сформулируем правила управления скоростью обучения $\alpha(k)$ для блочного алгоритма BP:

1. Если в ходе обучения квадрат ошибки (по всему обучающему множеству) увеличивается более, чем на ξ процентов (1-5%) после обновления весов, то обновление весов следует отменить, скорость обучения уменьшить, умножив её на значение $0<\rho<1$, а коэффициент момента γ (если используется) обнулить.

2. Если квадрат ошибки уменьшается после обновления весов, то обновление весов принимается, скорость обучения умножается $\eta>1$; если γ было обнулено, то его значение восстанавливается.

3. Если квадрат ошибки увеличивается менее чем на ξ процентов, то обновление весов принимается, скорость обучения не изменяется; если γ было обнулено, то его значение восстанавливается.

В модуле NeuralNetworks 2.0 пакета Scilab имеется функция ann_FFBP_gda, реализующая этот алгоритм. В общем случае формат вызова функции требует задания следующих параметров:

```
W = ann_FFBP_gda(P,T,N,af,lr,lr_inc,lr_dec,itermax,mse_min,gd_min,mse_diff_max),
```

где lr_inc – коэффициент увеличения скорости обучения (η), по умолчанию 1,05; lr_dec – коэффициент уменьшения скорости обучения (ρ), по умолчанию 0,75; mse_diff_max – допустимое относительное увеличение СКО (ξ), по умолчанию 0,01. Остальные параметры функции аналогичны параметрам функции ann_FFBP_gd.

Также в составе модуля NeuralNetworks 2.0 пакета Scilab имеется функция ann_FFBP_gdx, которая комбинирует алгоритм BP с моментом и алгоритм BP с адаптивной скоростью обучения. Функция имеет расширенный список параметров:

```
(P,T,N,af,lr,lr_inc,lr_dec,Mr,itermax,mse_min,gd_min,mse_diff_max).
```

Интерпретация этих параметров была рассмотрена выше.

Эвристические алгоритмы обеспечивают ускорение сходимости для некоторых задач. Однако такие алгоритмы требуют установки значений дополнительных параметров, тогда как BP_GD требует задания только скорости обучения. Часто эвристические алгоритмы оказываются чувствительными к изменениям этих параметров. Выбор значений дополнительных параметров также зависит от задачи.

3. Варианты заданий и программа работы

3.1. Повторить теоретический материал, относящийся к архитектуре и алгоритмам обучения многослойного персептрона [1, 4, 5].

3.2. Для варианта из п. 4.3 лабораторной работы №1, где задана одномерная функция $y=f(x)$ на некотором интервале определения:

3.2.1. Сформировать два множества (класса) точек данных (не менее 200 точек в каждом множестве), которые располагаются выше и ниже кривой $y=f(x)$, и отстоят от неё на расстояние d= $0{,}03|(y_{max} - y_{min})|$;

3.2.2. Отобразить классы и кривую $y=f(x)$ на двумерной плоскости;

3.2.3. Разработать программу обучения многослойного персептрона с архитектурой *R*-*S*-1 для классификации этих классов, в качестве активационных функций скрытого слоя использовать функции ann_tansig_active или ann_logsig_active, а в качестве активационной функции выходного слоя — ann_purelin_active, обучение персептрона выполнять с использованием функции ann_FFBP_gd;

3.2.4. Выполнить предварительное обучение MLP с архитектурой [*R*-10-1] на небольшом числе эпох itermax =200 при разных значениях параметра скорости обучения lr с целью определения её квазиоптимального устойчивого значения.

3.2.5. Используя полученное значение скорости обучения lr, выполнить обучение MLP с архитектурой [*R*-*S*-1] при разных *S* (10,30,40,50) на большом числе эпох itermax =1000.

3.2.6. Для различных MLP, обученных в соответствии с п.3.2.5, выполнить моделирование MLP и проверить качество классификации на **тестовом** множестве данных. Для этого сформировать тестовое множество данных аналогично п. 3.2.1, провести классификацию данных с помощью обученного MLP, отобразить точки полученных выходных классов и кривую $y=f(x)$ на двумерной плоскости, вычислить вероятности правильной классификации при разных *S*.

3.3. Используя вариант из п. 4.5 лабораторной работы №1, где задана двумерная функция $z=f(x,y)$ на некотором интервале определения:

3.3.1. Сформировать подмножества обучающих и тестовых данных. Для этого выбрать на плоскости (x,y) 300 случайных точек и определить в этих точках значение функции $z=f(x,y)$. В качестве входного вектора использовать вектор **p**=[x;y], в качестве целевого значения – $t=z$. Полученные данные разделить на 2 подмножества: обучающее (80%) и тестовое (20%).

3.3.2. Разработать программу обучения многослойного персептрона с архитектурой *R*-*S*-1для аппроксимации этой функции, в качестве активационных функций скрытого слоя использовать функции ann_tansig_active или ann_logsig_active, а в качестве активационной функции выходного слоя -- ann_purelin_active, обучение персептрона выполнять с использованием функции ann_FFBP_gd;

3.3.3. Построить кривые обучения MLP при разных значениях *S* (5 10 15 20) и фиксированной скорости обучения lr (например, lr =0.005); выбрать квазиоптимальное значение *S* для дальнейшего использования. Построение кривых выполнить при значении параметра itermax=300;

3.3.4. Выполнить обучение MLP (при квазиоптимальном *S*) с помощью функций ann_FFBP_gdm, ann_FFBP_gda, ann_FFBP_gdx, сравнить получаемые кривые обучения с кривыми, полученными в п. 3.3.3;

3.3.5. Используя тестовое подмножество данных, выполнить моделирование 4-х вариантов MLP, обученных с помощью 4-х разных функций, указанных выше. Построить сопоставительные графики значений функции $z=f(x,y)$ и соответствующих значений на выходе MLP, вычислить СКО аппроксимации функции на тестовом подмножестве, сравнить со значениями СКО, полученными при обучении MLP.

3.4. Подготовьте и защитите отчет по работе.

4. Методические рекомендации по выполнению работы

4.1. В задании 3.2. исследуется задача бинарной классификации. Цель задания – убедиться в том, что MLP способен строить нелинейные границы решения между классами.

Для этого в соответствии с п.3.2.1 задания необходимо сначала сформировать два множества (класса) случайных точек, разделяемых заданной по варианту нелинейной функцией. Ниже приведен пример кода функции для формирования случайных точек, принадлежащих 2-м классам, разделяемых кривой y=f(x). Точки классов отстоят от разделяющей кривой на расстояние d=kd*abs(ymax-ymin), где ymax,ymin – максимальное и минимальное значения функции y=f(x) на заданном интервале Фрагменты кода, отмеченные красным цветом, требуют уточнения. Модернизируйте этот код для формирования множества точек 2-х классов в соответствии с заданным вариантом.

```
function [P, T, y, x]=gendata(Q, kd)
//формирование множеств случайных точек для 2-классов,
// разделяемых кривой y=f(x)
//Q - число точек в каждом множестве
//kd - коэффициент, задающий величину отступа от границы

//задание области определения функции - координата x
xmin=0.7;
xmax=4;
//поиск ymax и ymin
stepx=abs(xmax-xmin)/Q;
x=xmin:stepx:xmax;
y=f(x);  // функция, заданная вариантом. Определяется отдельно.!
```

```
ymin=min(y);
ymax=max(y);
//диапазон изменения значений функции
range=abs(ymax-ymin);
//задание расстояния между классами: d=kd*|(ymax-ymin)|
d=kd*range;

//формирование случ. мн-ва точек datax вдоль x
datax=grand(1,Q,'unf',xmin,xmax);

// формирование обучающего множества {P,T}
  P=[];
  T=[];
  for j=1:1:Q
      //значение границы border для случайной точки datax(1,j)
      border=f(datax(1,j));
      //вычисление ординаты "y" 2-х случайных точек, отстоящих от border на
расстояние d
      y_class1=grand(1,1,'unf',border+d,border+range);
      y_class2=grand(1,1,'unf',border-range,border-d);
      //формирование x,y координат 2-х j-тых точек классов 1 и 2 в виде вектора
      j_class1=[datax(1,j); y_class1];
      j_class2=[datax(1,j); y_class2];
      //добавление полученных j-тых точек во множество входных примеров
      P=[P j_class1 j_class2];
      // формирование соответсвующих целевых выходных значений MLP
      t_class1=1;
      t_class2=0;
      T=[T t_class1  t_class2];
  end
endfunction
```

Для отображения точек 2-х классов (задание п.3.2.2) и разделяющей границы (рисунок 5.4) можно использовать следующий код:

```
  //отображение множества точек 2-х классов и разделяющей границы
  drawlater(); // прорисовка отображения будет позже
  plot(P(1,T==1),P(2,T==1),'o'); //отображать точки 1-го класса знаком 'o'
  plot(P(1,T==0),P(2,T==0),'g*'); //отображать точки 2-го класса знаком '*'
  plot(x,y,'r'); // отображать границу красным цветом
  drawnow(); // включаем прорисовку
  xtitle('Классы и граница');
```

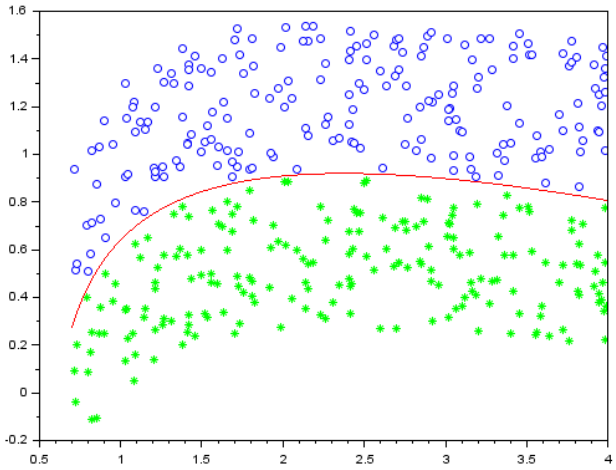

Рисунок 5.4 – Сгенерированные классы и нелинейная граница между классами

Выполнение п. 3.2.4 заключается в предварительном обучении MLP с использованием функции ann_FFBP_gd. При каждом запуске функция будет отображать прогресс обучения в отдельном графическом окне (рисунок 5.5)

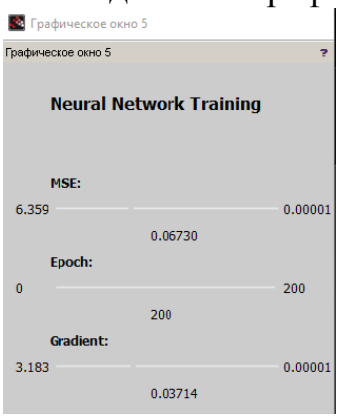

Рисунок 5.5 Окно прогресса, отображаемое функцией ann_FFBP_gd

Для сравнения результатов обучения при разных значениях скорости обучения lr контролируйте значение среднего квадрата ошибки MSE. Рекомендуется значение скорости обучения выбирать из списка lr=[0.05, 0.025, 0.01, 0.005] . В качестве квазиоптимального значения lr следует выбрать то значение, которое обеспечивает минимальное конечное значение MSE в окне прогресса (рисунок 5.5).

При выполнении п.п. 3.2.5-3.2.6 следует обучать MLP с разным числом нейронов в скрытом слое и тестировать результаты обучения. Обучение необходимо выполнять при itermax =1000 и квазиоптимальной скорости обучения, полученной ранее. Для тестирования результатов обучения следует использовать функцию ann_FFBP_run и **множество данных, на котором MLP не обучался**. Поэтому необходимо тестовое множество входных данных сформировать с помощью функции gendata(**Q**, **kd**):

```
[P_test,T_test,y_test,x_test]=gendata(Q,kd);
```

Чтобы вычислить вероятность правильной классификации необходимо сначала выходные значения MLP привести к бинарным уровням 0 и 1. Для этого следует их округлить, а затем посчитать вероятность правильной классификации, как отношение числа случаев правильной классификации к числу точек данных:

```
a_test = ann_FFBP_run(P_test,W,af);
y_bin_test=round(a_test); //делаем выход бинарным
//вычисляем вероятность правильной классификации
n_correct=sum(T_test == y_bin_test);
rate=n_correct/length(T_test);
```

Для отображения результатов классификации обученным MLP используйте код, аналогичный коду, который применялся для построения рисунка 5.4, заменив целевой вектор T, на вектор y_bin_test с результатами бинарной классификации:

```
plot(P(1, y_bin_test ==1),P(2, y_bin_test ==1),'o'); //отображать точки 1-го класса'
plot(P(1, y_bin_test ==0),P(2, y_bin_test ==0),'g*'); //отображать точки 2-го класса'
plot(x_test,y_test,'m'); // отображать границу малиновым цветом
```

Пример отображения выходных классов, формируемых обученным MLP, приведен на рисунке 5.6. Обратите внимание, что ряд точек MLP классифицировал неправильно. Вероятность правильной классификации при S=10 для рассматриваемого примера равна 0,945. Необходимо оценить вероятности правильной классификации и при других S, указанных в задании.

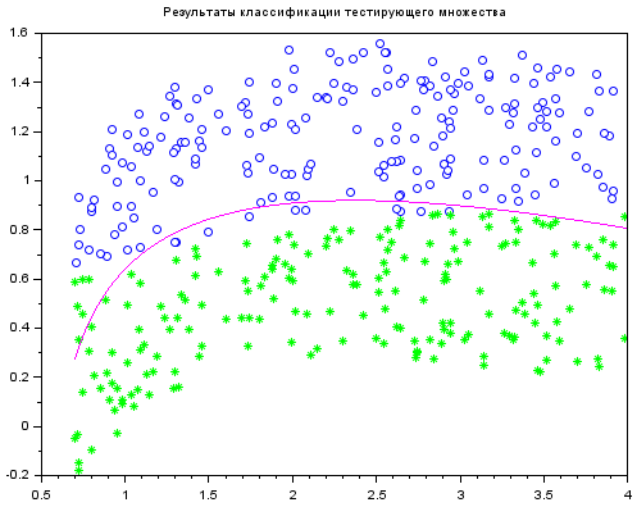

Рисунок 5.6 – Результаты классификации тестирующего множества, вероятность правильной классификации 0,945 (S=10)

4.2. В задании 3.3. исследуется задача аппроксимации двумерной функции с помощью MLP. Цель задания – убедиться в том, что MLP способен обучаться аппроксимации нелинейных функций на основе множества обучающих данных.

Для обучения MLP в соответствии с п.3.3.1 задания требуется сформировать подмножества обучающих и тестовых данных. Тестовые данные используются только в ходе тестирования обученной нейросети с целью выявления качества её функционирования и обобщающих способностей. Ниже приведен фрагмент кода для генерации обучающих P_train, T_train и тестовых P_test, T_test подмножеств данных.

```
Q=300; //Общее число элементов множества данных
Q_train=floor(0.8*Q); //число эл-тов обучающего мно-ва 80% от Q

x=grand(1,Q,'unf',minx,maxx); //формирование случайных координат x,y
y=grand(1,Q,'unf',miny,maxy);

x_train=x(1:Q_train);  //формирование обучающего подмно-ва
y_train=y(1:Q_train);
P_train=[x_train;y_train];
T_train=f(x_train,y_train);

x_test=x(Q_train+1:$);//формирование тестового подмно-ва
y_test=y(Q_train+1:$);
P_test=[x_test;y_test];
T_test=f(x_test,y_test);
```

В задании п. 3.3.3 требуется обучить MLP на основе алгоритма BP_gd при разных значениях *S* и построить кривые обучения. Построение кривых обучения потребует модификации кода встроенной функции ann_FFBP_gd. Необходимо, чтобы функция возвращала вектор значений СКО (в программе обозначен out_mse). Выполните модернизацию функции по следующей схеме:

```
function [W, out_mse]=ann_FFBP_gd1(P, T, N, af, lr, itermax, mse_min, gd_min)
        …
// Stopping Criteria
      mse = mean(e.^2);
      itercnt = itercnt + 1;
      out_mse(itercnt)=mse; // добавленная строка программы
      gd = mean(s(1).^2);
        …
endfunction
```

Изменения, внесенные в функцию, отмечены курсивом. Для запоминания значений СКО при разных *S* организуйте циклический вызов модифицированной функции ann_FFBP_gd1:

```
af=['ann_tansig_activ','ann_purelin_activ'];
lr=0.005
```

```
itermax=300
S=[5, 10,15,20];
for i=1:length(S)
    N=[2 S(i) 1];
    [W_gd, out_mse_gd] = ann_FFBP_gd1(P_train,T_train,N,af,lr,itermax);
    mse_gd_hist(i,:)=out_mse_gd;
end;
```

Векторы значений СКО будут запоминаться в строках матрицы mse_gd_hist. Это позволит после завершения цикла построить кривые обучения для каждого *S* (рисунок 5.7). Из анализа кривых обучения (для примера на рисунке 5.7) следует, что квазиоптимальным будет значение *S*=15, т.к обеспечивает наиболее быстрое снижение СКО и наименьшее конечное значение СКО в ходе обучения (контролируется по значениям матрицы mse_gd_hist).

Для сравнения рассмотренных в п.2.4 эвристических алгоритмов обучения с алгоритмом BP_gd (п. 3.3.4 задания) следует обучить MLP одной и той же структуры, используя функции обучения, указанные ниже:

```
[W_gd, out_mse_gd] = ann_FFBP_gd1(P_train,T_train,N,af,lr,itermax,mse_min,gd_min);
[W_gda,out_mse_gda] =
  ann_FFBP_gda1(P_train,T_train,N,af,lr,lr_inc,lr_dec,itermax,mse_min,gd_min,mse_diff_max);
[W_gdm, out_mse_gdm]=ann_FFBP_gdm1(P_train,T_train,N,af,lr,Mr,itermax,mse_min,gd_min);
[W_gdx, out_mse_gdx] =
ann_FFBP_gdx1(P_train,T_train,N,af,lr,lr_inc,lr_dec,Mr,itermax,mse_min,gd_min);
```

Код всех этих функций модифицируется аналогично функции ann_FFBP_gd1. Результаты обучения MLP разными алгоритмами запоминаются в матрицах весов и векторах ошибок (хранят значения СКО), возвращаемых функциями. Запоминание весов обученных MLP необходимо для тестирования результатов обучения, а запоминание векторов значений СКО – для построения кривых обучения (рисунок 5.7).

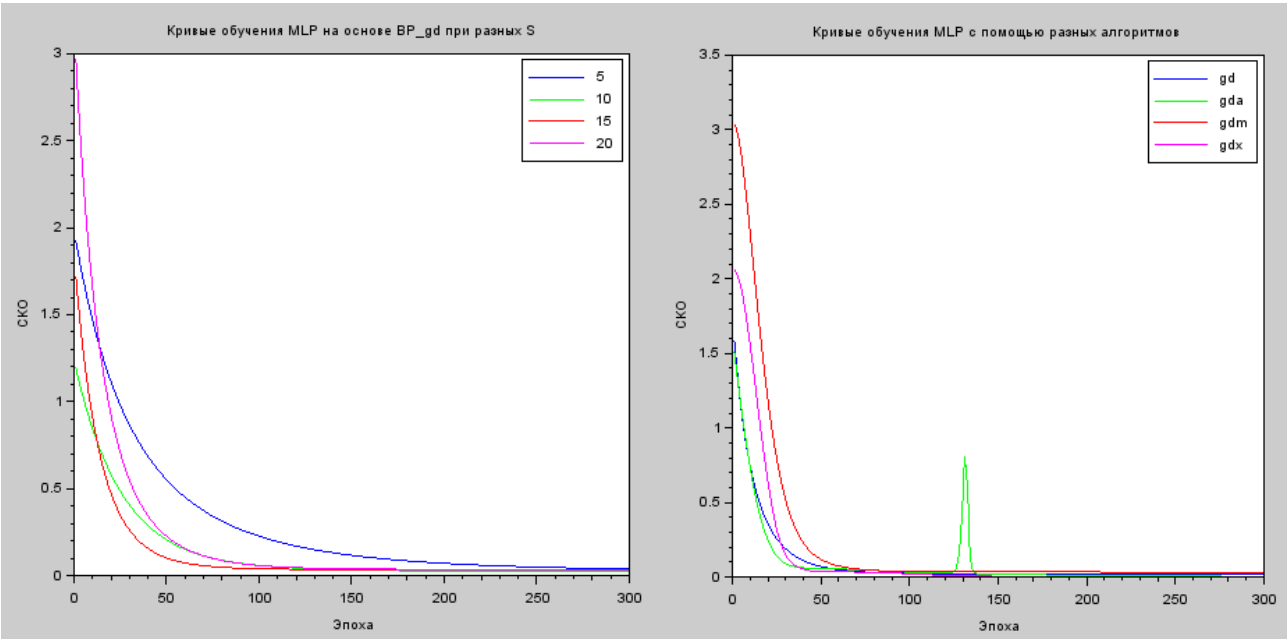

Рисунок 5.7. Кривые обучения MLP

Тестирование результатов обучения (п. 3.3.5 задания) выполняется исключительно на данных, которые не использовались для обучения. Это позволяет оценить обобщающие свойства полученных моделей нейросетей. Для тестирования набора полученных нейросетей используйте функцию ann_FFBP_run:

```
y_test_gd = ann_FFBP_run(P_test,W_gd,af);    // сеть, обученная алг-м BP_gd
y_test_gda = ann_FFBP_run(P_test,W_gda,af); // сеть, обученная алг-м BP_gda
y_test_gdm = ann_FFBP_run(P_test,W_gdm,af); // сеть, обученная алг-м BP_gdm
y_test_gdx = ann_FFBP_run(P_test,W_gdx,af); // сеть, обученная алг-м BP_gdx

// вычисление СКО на тестовых данных
mse_gd=((T_test-y_test_gd)*(T_test-y_test_gd)')/Q_test
mse_gda=((T_test-y_test_gda)*(T_test-y_test_gda)')/Q_test
mse_gdm=((T_test-y_test_gdm)*(T_test-y_test_gdm)')/Q_test
mse_gdx=((T_test-y_test_gdx)*(T_test-y_test_gdx)')/Q_test
```

По результатам тестирования необходимо построить сопоставительные графики функции $z=f(x,y)$, определенной на тестовом множестве входных данных, и графики выходных сигналов MLP, обученных разными алгоритмами (рисунок 5.8), а также сравнить вычисленные выше СКО аппроксимации функции на тестовом подмножестве со значениями СКО, полученными на этапе обучения соответствующих MLP.

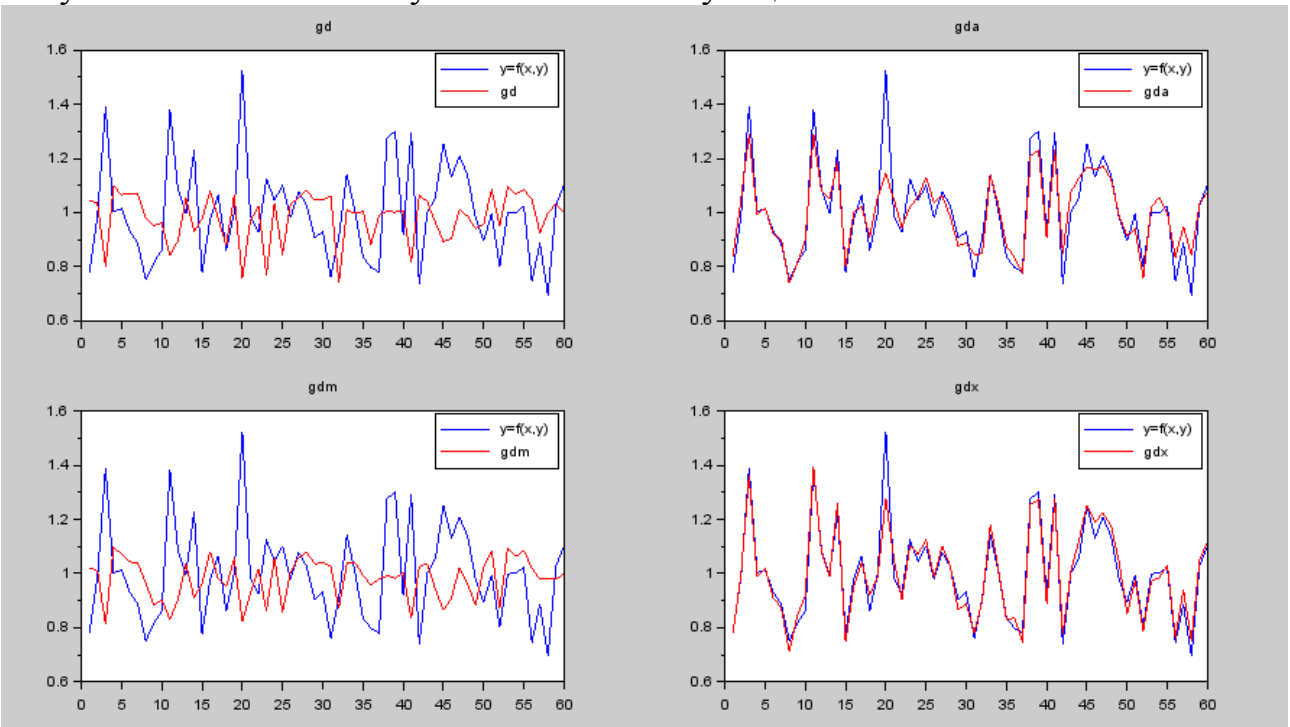

Рисунок 5.8 – Сопоставительные графики тестовых значений функции $z=f(x,y)$ и значений на выходе MLP, обученных разными функциями

Из рисунка 5.8 следует, что алгоритмы BP_gd и BP_gdm дают неудовлетворительные результаты на выбранном числе эпох обучения. Эти алгоритмы медленно сходятся и требуют значительно большего числа эпох обучения по сравнению с алгоритмами, основанными на управлении скоростью обучения. Наименьшее значение СКО получается при обучении MLP с помощью алгоритма BP_gdx.

5. Содержание отчета

5.1. Цель работы.
5.2. Вариант задания.
5.3. Схема MLP, решающего задачу классификации 2-х классов; программа обучения MLP бинарной классификации с использованием функции ann_BP_gd; результаты предварительного обучения MLP при разных значениях параметра lr и выбор квазиоптимального lr; результаты обучения MLP при разных *S* и большом числе эпох; результаты тестировании обученного бинарного классификатора на основе MLP и вероятности правильной классификации при разных *S*.
5.4. Схема MLP для решения задачи аппроксимации функций; программа обучения MLP; кривые обучения MLP при разных значениях *S* и выбор квазиоптимального *S*; кривые обучения MLP (при квазиоптимальном *S*) с помощью функций ann_FFBP_gd, ann_FFBP_gdm, ann_FFBP_gda, ann_FFBP_gdx; результаты тестирования MLP, обученных указанными функциями (сравнительные графики), значения СКО на этапе обучения и на этапе тестирования.
5.5. Выводы по результатам исследований.

6. Контрольные вопросы

6.1. Нарисуйте схему многослойного персептрона и запишите выражения для вычисления его выходных значений в векторно-матричной форме.
6.2. Объясните алгоритм, положенный в основу функции ann_FFBP_run(P,W,af), и назначение её параметров.
6.3. Нарисуйте структуру MLP для решения задачи исключающего ИЛИ.
6.4. Сформулируйте общий смысл теоремы об универсальной аппроксимации в отношении MLP.
6.5. Запишите выражение целевой функции, используемой в алгоритме BP.
6.6. Запишите общие выражения алгоритма наискорейшего спуска применительно к MLP.
6.7. Что понимают под чувствительностью целевой функции MLP?
6.8. Запишите выражения для производных целевой функции MLP с использованием чувствительности.
6.9. Запишите векторно-матричные выражения стохастической аппроксимации SDA для MLP.

6.10. Запишите рекуррентное соотношение для вычисления чувствительностей, используя цепочное правило дифференцирования.
6.11. Чему равен якобиан применительно к MLP?
6.12. Сформулируйте последовательный алгоритм BP.
6.13 Сформулируйте и запишите выражения для блочного алгоритма BP.
6.14. Объясните параметры функции ann_FFBP_gd, реализующей блочный вариант алгоритма BP_GD.
6.15. При каких условиях функция ann_FFBP_gd прекращает процесс обучения?
6.16. Какой основной недостаток алгоритма BP_GD?
6.17. На какие группы подразделяются эвристические методы улучшения BP_GD? Дайте их краткую характеристику.
6.18. Сформулируйте алгоритм BP с моментом инерции.
6.19. Объясните параметры функции ann_FFBP_gdm.
6.20. Сформулируйте алгоритм BP с адаптивной скоростью обучения.
6.21. Объясните параметры функции ann_FFBP_gda.
6.22. Объясните назначение и параметры функции ann_FFBP_gdx.
6.23. Как формируются обучающее и тестовые подмножества данных? Зачем нужно тестовое подмножество?
6.24. Почему отличаются СКО обучения и тестирования? Какая из ошибок обычно меньше?

Приложение А. Функция ann_FFBP_run

```
function y=ann_FFBP_run(P, W, af)
// Функция моделирования сети прямого распространения
// Пример вызова:
//    [y] = ann_FFBP_run(W,P,af)
// Параметры:
//    P : Тестовые входные значеният
//    W : Веса и смещения
//    af : Список активационных функций
//    y : Выходные значения

//1. ================Обработка списка аргументов функции=========================
rhs=argn(2);
if rhs < 3; af = ['ann_tansig_activ','ann_purelin_activ']; end

if size(W) ~= size(af,2) then
   error('Numbers of activation functions must match numbers of layers (N-1)');
end

//2.================= Фаза моделирования====================================
// Определяем число слоев сети по размерам W
layers = size(W);
//Создаем списки для хранения сетевых значений и активностей слоев
n = list(0);
a = list(0);
//Вычисляем сетевые значения 1-го слоя: n=Wp+b
n(1) = W(1)(:,1:$-1)*P + repmat(W(1)(:,$),1,size(P,2)); // Можно временно хранить в n для экономии памяти
// Оцениваем значения на выходе 1-го слоя: a=af(n), где af – нелинейность слоя
a(1) = evstr(af(1)+'(n('+string(1)+'))');
// Повторяем вычисления для каждого следующего слоя
for cnt = 2:layers
```

```
    n(cnt) = W(cnt)(:,1:$-1)*a(cnt-1) + repmat(W(cnt)(:,$),1,size(P,2)); // Можно временно хранить в n для экономии памяти
    a(cnt) = evstr(af(cnt)+'(n('+string(cnt)+'))');
end
// Результирующие значения соответствуют выходам последнего слоя
y = a($);
endfunction
```

Приложение Б. Функция ann_FFBP_gd

```
function W=ann_FFBP_gd(P, T, N, af, lr, itermax, mse_min, gd_min)
// Обучение на основе блочного  алгоритма нисходящего градиента с обратным распространением.
//
// Примеры вызова:
//     W = ann_FFBP_gd(P,T,N)
//     W = ann_FFBP_gd(P,T,N,af,lr,itermax,mse_min,gd_min)
//
// Параметры
//     P : Обучающий вход
//     T : Обучающие целевые значения
//     N : Вектор, задающий число нейронов каждого слоя, включая входной и выходной слои
//     af : Список имен активационных функций от 1-го до выходного слоев
//     lr : скорость обучения
//     itermax : Максимальное число эпох обучения
//     mse_min : Минимальная ошибка  (Значение целевой функции)
//     gd_min : Минимальное значение градиента
//     W : Выходные веса и смещения

//1. ================Обработка списка аргументов функции=======================
rhs=argn(2);

// Проверка ошибки списка аргументов
if rhs < 3; error("Expect at least 3 arguments, P, T and N");end
// Выбор значений аргументов по умолчанию
if rhs < 4; af = ['ann_tansig_activ','ann_purelin_activ']; end
if rhs < 5; lr = 0.01; end
if rhs < 6; itermax = 1000; end
if rhs < 7; mse_min = 1e-5; end
if rhs < 8; gd_min =  1e-5; end

if af == []; af = ['ann_tansig_activ','ann_purelin_activ']; end
if lr == []; lr = 0.01; end
if itermax == []; itermax = 1000; end
if mse_min == []; mse_min = 1e-5; end
if gd_min == []; gd_min =  1e-5; end

// Проверка ошибки списка активационных функций
if size(N,2)-1~= size(af,2) then
    error('Numbers of activation functions must match numbers of layers (N-1)');
end

//2.======================Инициализация====================

// Инициализация сети и формирование списка имен производных активационных функций
format(8);
W = ann_ffbp_init(N);
itercnt = 0;
af_d = strsubst(af,'ann_','ann_d_');
mse = %inf;
gd = %inf;

//Инициализация GUI отображения прогресса обучения
handles = ann_training_process();
handles.itermax.string = string(itermax);
handles.msemin.string = string(mse_min);
handles.gdmax.string = 'inf';
handles.gdmin.string = string(gd_min);

// Задание числа слоев и списков промежуточных переменных
layers = size(N,2)-1; // слои считаются от 1-го скрытого слоя до выходного слоя
```

```
n = list(0);
a = list(0);
m = list(0);
s = list(0);

// 3. ====================Реализация цикла эпох обучения============================
while mse > mse_min & itercnt < itermax & gd > gd_min
      // Прямое распространение – моделирование (аналогично функции ann_FFBP_run (формула 5.1))
      n(1) = W(1)(:,1:$-1)*P + repmat(W(1)(:,$),1,size(T,2));
      a(1) = evstr(af(1)+'(n('+string(1)+'))');
      for cnt = 2:layers
         n(cnt) = W(cnt)(:,1:$-1)*a(cnt-1) + repmat(W(cnt)(:,$),1,size(T,2));
         a(cnt) = evstr(af(cnt)+'(n('+string(cnt)+'))');

      end
      // Вычисление ошибки
      e = T - a($);
      // Обратное распространение значений чувствительности
      m(layers) = evstr(af_d(layers)+'(a('+string(layers)+'))'); // Вычисление производных актив. функций последнего слоя
      s(layers) = (-2*m(layers).*e); // Вычисление чувствительности выходного слоя  (формула 5.12)
      for cnt = layers-1:-1:1  //Вычисляем производные актив. функций слоев и распространяем обратно
чувствительности.
         m(cnt) = evstr(af_d(cnt)+'(a('+string(cnt)+'))');
         s(cnt) = m(cnt).*(W(cnt+1)(:,1:$-1)'*s(cnt+1));   (формула 5.12)
      end

      // Обновление весов сети (формула 5.15)
      W(1)(:,1:$-1) = W(1)(:,1:$-1) - (lr*s(1)*P')./size(P,2); // Обновление весов 1-го скрытого слоя
      W(1)(:,$) = W(1)(:,$) - lr*mean(s(1),2); // Обновление смещений 1-го скрытого слоя
      for cnt = 2:layers // Обновление весов и смещений следующих слоев
         W(cnt)(:,1:$-1) = W(cnt)(:,1:$-1) - (lr*s(cnt)*a(cnt-1)')./size(P,2);
         W(cnt)(:,$) = W(cnt)(:,$) - lr*mean(s(cnt),2);
      end

     // Вычисление критериев остановки алгоритма
      mse = mean(e.^2);  // СКО
      itercnt = itercnt + 1; //Число эпох
      gd = mean(s(1).^2); // Средний квадрат значения градиента целевой функции

     // программирование GUI отображения прогресса обучения
     if  itercnt == 1 then
        mse_max = mse;
        handles.msemax.string = string(mse_max);
        gd_max = gd;
        handles.gdmax.string = string(gd_max);
        mse_span = log(mse) - log(mse_min);
        iter_span = itermax;
        gd_span =  log(gd) - log(gd_min);
     end

      // для версии выше Scilab 5.5
      handles.iter.value = round((itercnt/iter_span)*100);
      handles.mse.value = -(log(mse)-log(mse_max))/mse_span * 100;// round(((log(mse) - log(mse_min))/mse_span)*100);
      handles.gd.value = -(log(gd)-log(gd_max))/gd_span * 100; //round(((log(gd) - log(gd_min))/gd_span)*100);

      handles.itercurrent.string = string(itercnt);
      handles.msecurrent.string = string(mse);
      handles.gdcurrent.string = string(gd);

end
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.