МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«Национальный исследовательский ядерный университет «МИФИ»
Димитровградский инженерно-технологический институт –
филиал федерального государственного автономного образовательного учреждения высшего профессионального образования «Национальный исследовательский ядерный университет «МИФИ»
(ДИТИ НИЯУ МИФИ)

Информационно-технологический факультет

Кафедра информационных технологий

Дисциплина «Системы искусственного интеллекта»

Лабораторная работа №3

ОБУЧЕНИЕ ПЕРСЕПТРОНА ДЛЯ РЕШЕНИЯ ПРОБЛЕМЫ XOR

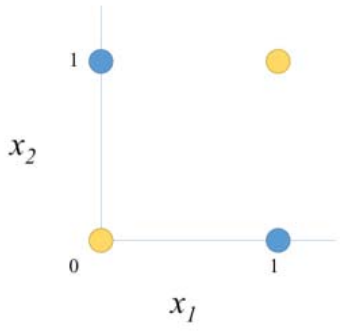
Цель работы

1. Научиться применять и использовать нейронные сети с использованием современных методов программирования пакета GNU Octave.

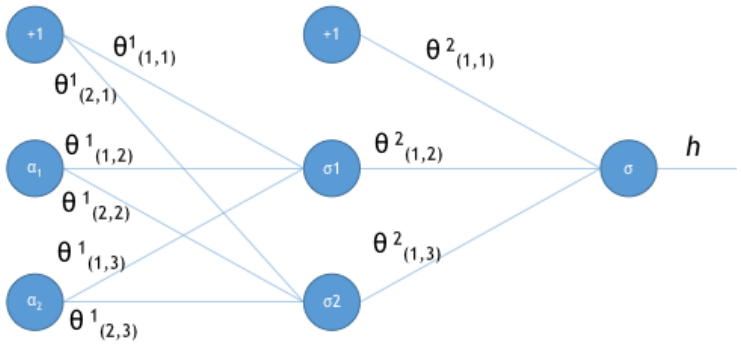
Методические указания

1. Проблема XOR. На графике показаны два входа x_1 и x_2 на соответствующих осях. Если x_1 и x_2 имеют одинаковое значение, на графике показаны желтые кружки, и, аналогично, где x_1 и x_2 различны, на графике показаны синие кружки. Есть важное ограничение, которое четко показывает график. Невозможно нарисовать одну прямую линию на графике, чтобы желтые круги были с одной стороны, а синие - с другой. Это называется «линейной отделимостью». Таким образом, проблема XOR не является линейно разделимой, что означает, что нам понадобится многослойная нейронная сеть для ее решения.

x_1	x_2	y	y
0	0	0	
0	1	1	
1	0	1	
1	1	0	



Построим нейронною сеть, которую можно обучить решению проблемы XOR.



На этой сети следует отметить: во-первых, входные данные из таблицы выше (x_1 и x_2) отображаются непосредственно на узлах, представленные как a_1 и a_2; во-вторых, узел смещения, выход которого всегда равен +1; в-третьих, скрытый слой содержит узел смещения, установленный на +1. Выходы узлов помечены маленькой греческой буквой сигма. Выход сети помечен как h.

Представим входные данные в виде вектора (одномерной матрицы) :

$$A1 = \begin{bmatrix} 1 \\ a_1 \\ a_2 \end{bmatrix},$$

где 1 представляет узел смещения, а два нуля представляют первую строку переменных из таблицы.

```
>> A1 = [1; 0; 0];
```

Связи между узлами первого уровня и узлами второго уровня имеют связанные с ними веса, обозначенные буквой тета (вместе с верхним индексом 1) на диаграмме нашей сети. Точно так же веса в слое 2 также показаны с верхним индексом 2.

Номера нижних индексов идентифицируют узлы на обоих концах канала в форме (i, j), где i - это узел, принимающий сигнал, а j - это узел, отправляющий сигнал. Здесь обратная логика, т.к. обычно сигнал передается с i на j. Представим все веса в одной матрице, которая выглядит следующим образом:

$$\Theta 1 = \begin{bmatrix} \theta_{1,1} & \theta_{1,2} & \theta_{1,3} \\ \theta_{2,1} & \theta_{2,2} & \theta_{2,3} \end{bmatrix}.$$

Как правило, начальные значения весов в сети устанавливаются на случайные значения от -1 до +1, так как мы понятия не имеем, какими они должны быть на самом деле. Мы можем сделать это в Octave следующим образом:

```
>> THETA1=2*rand(2,3)-1;
>> THETA2=2*rand(1,3)-1;
```

Так что теперь мы можем просто умножить эти матрицы вместе, чтобы выяснить, что представляет собой вход для второго слоя: $\mathrm{Z2} = \Theta 1 * \mathrm{A1}$, где Z представляет вход для второго слоя узлов. Это умножение приведет к другому вектору:

```
>> Z2 = THETA1 * A1;
```

Это намного лучше (и быстрее), чем вычислять каждый из этих входов отдельно. Фактически, большинство библиотек машинного обучения обеспечивают быстрое умножение матриц (и другие матричные операции) именно потому, что это эффективный способ моделирования стратегий машинного обучения. Чтобы рассчитать выход слоя 2, мы должны применить функцию к входу. Типичная используемая функция называется сигмоидальной функцией (см. Приложение Б):

```
function [result] = sigmoid(x)
   result = 1.0 ./ (1.0 + exp(-x));
end
```

Таким образом, выходной слой 2 является сигмоидом ввода, или $\mathrm{A2} = \sigma\langle \mathrm{Z2} \rangle$.

```
>> A2 = [1; sigmoid(Z2)];
```

Обратите внимание, что мы добавляем дополнительный 1 в качестве первого элемента в векторе, чтобы представить смещение, которое потребуется в качестве входа в слой 3. Повторим процесс для слоя 3, умножая выходные данные слоя 2 на матрицу весов для слоя 2, чтобы получить входные данные для слоя 3, а затем получая сигмоид результата:

$$\mathrm{Z3} = \Theta 2 * \mathrm{A2}$$
$$h = \sigma\langle \mathrm{Z3} \rangle$$

Выход из сети - это единственное значение, называемое нашей гипотезой (h). Это предположение сети на выходе, учитывая ее ввод:

```
>> Z3 = THETA2 * A2;
>> h = sigmoid(Z3);
```

Сеть полностью построена.

2. Алгоритм обратного распространения для обучения сети. В контексте нейронных сетей обучение означает корректировку этих весов, чтобы они все лучше и лучше давали правильные ответы.Первый шаг - выяснить, насколько неправильна сеть. Это называется стоимостью или потерей сети. Есть несколько способов сделать это от простого к

сложному. Общий подход называется «логистическая регрессия», который позволяет избежать проблем, связанных с тем, что нейронная сеть застревает в процессе обучения.

В таблице XOR два возможных выхода, 1 или 0, поэтому формулу стоимости в зависимости от выхода:

$$Cost(h_\theta(x), y) = \begin{cases} -\log\big(h_\theta(x)\big) \ if \ y = 1 \\ -\log\big(1 - h_\theta(x)\big) \ if \ y = 0 \end{cases},$$

где h_θ (x) - это гипотеза, которую производит сеть, учитывая входные данные x = (x_1, x_2) и веса θ; «Log» - это стандартная математическая функция, которая вычисляет натуральный логарифм заданного значения (отсюда и название «логистическая регрессия»); мы можем объединить в единую формулу:

$$Cost(h_\theta(x), y) = -y \log\big(h_\theta(x)\big) - (1 - y) \log\big(1 - h_\theta(x)\big).$$

Итак, наш код для представления этого:

```
>> y = 0;
>> J = ((y * log(h)) + ((1 - y) * log(1 - h))) * -1 ;
```

Лучшая сеть будет иметь меньшую стоимость, поэтому наша цель в обучении сети состоит в том, чтобы минимизировать стоимость. Первым шагом является оценка ошибки на выходном узле (уровень 3):

$$\Delta 3 = h - y$$

```
>> delta3 = h - y;
```

Ошибка сети на определенном уровне распространяет ее обратно по сети. В общем, формула для этого: $\Delta^{l-1} = \langle \Theta(l-1) \rangle^{T} \Delta^{l} . g'\big(z^{l-1}\big)$, где верхний индекс l в формуле относится к уровню в сети. Следующий уровень, который мы должны разработать, это уровень 2, поэтому l = 2. Далее, $\Theta(l-1)$ относится к матрице весов на слое l-1. Верхний индекс T означает транспонирование матрицы:

```
>> THETA2' * delta3
```

Обратите внимание на апостроф после THETA2. Это способ Octave транспонировать матрицу. Вторая часть формулы представляет собой производную функции активации. К счастью, есть простой способ рассчитать это, не вдаваясь в сложность исчисления: $g'(Z) = Z. \ * (1 - Z)$, где «. *» Означает поэлементное умножение матрицы. Таким образом, каждый член в первой матрице умножается на соответствующий член во второй матрице. В Octave это почти точно так же (для слоя 2): Z2. * (1 - Z2). Помните, что это матричные операции, поэтому delta2 тоже будет матрицей. Последняя часть строки кода (2: конец) означает перевод второго элемента матрицы в конец. Это сделано потому, что мы не передаем ошибку обратно из узла смещения. На этом этапе у нас есть ошибка в выходном сигнале на уровне 3 и уровне 2. Поскольку слой 1 является

входным слоем, там нет никакой ошибки, поэтому нам не нужно ничего вычислять.

Последний шаг - это корректировка весов, зная ошибки: $\Delta\langle\Theta l\rangle = \alpha\langle\Delta(l+1)A(l)\rangle$. Формула говорит, что изменение весов Θ(l) - это значения активации, умноженные на ошибки, скорректированные на значение α, которое представляет скорость обучения. Для решения проблемы XOR достаточно скорости обучения 0,01.

Итак:

```
THETA2 = THETA2 - (0,01 * (delta3 * A2 '));
THETA1 = THETA1 - (0,01 * (delta2 * A1 '));
```

Как только мы отрегулируем веса в матрицах, сеть даст нам немного другую гипотезу для каждого из наших входных данных. Мы можем пересчитать приведенную выше функцию стоимости, чтобы увидеть, насколько она улучшается. Если мы повторим расчеты, стоимость должна начать снижаться, и сеть будет становиться все лучше и лучше в достижении желаемого результата.

3. Скрипт запуска на обучение сети. После того, как настроили параметры сети (весовые коэффициенты между узлами), нужно подать пример обучающей выборки и заставить сеть прогнозировать некоторый результат, т.е. научить ее выдавать верные значения функции от двух переменных. Собирем все рассмотренные фрагменты кода вместе в одной функции:

function [THETA1_new, THETA2_new] = xor_nn(XOR, THETA1, THETA2, init_w=0, learn=0, alpha=0.01), где после слова «function» указывам выходные параметры, имя функции - «xor_nn», параметры функции:

- XOR – массив, в котором представлен учебный набор;
- THETA1 и THETA2 – текущие значения параметров в сети;
- init_w = 0[1] – означает, что необходимо инициализировать веса в сети;
- learn = 0 – означает, что необходимо обучение на предьявленном примере (см. ниже);
- альфа = 0,01 – скорость обучения (по умолчанию 0,01).

Первый шаг в нашей функции – проверка необходимости инициализации весов:

```
if (init_w == 1)
   THETA1 = 2*rand(2,3) - 1;
   THETA2 = 2*rand(1,3) - 1;
endif
J = 0.0; % инициализируем переменную стоимости
```

[1] Обратите внимание, что любой параметр, имеющий знак «=», является необязательным и будет отображать значение по умолчанию, если оно не указано явно.

Запишем количество обучающих примеров (количество наборов для двух переменных 4) и общую дельту по всем обучающим примерам

```
T1_DELTA = zeros(size(THETA1));
T2_DELTA = zeros(size(THETA2));
m = 0;
```

THETA1 и THETA2 являются матрицами, поэтому нужно инициализировать каждый элемент матрицы и сделать дельта-матрицы одинакового размера:

```
for i = 1:rows(XOR)
```

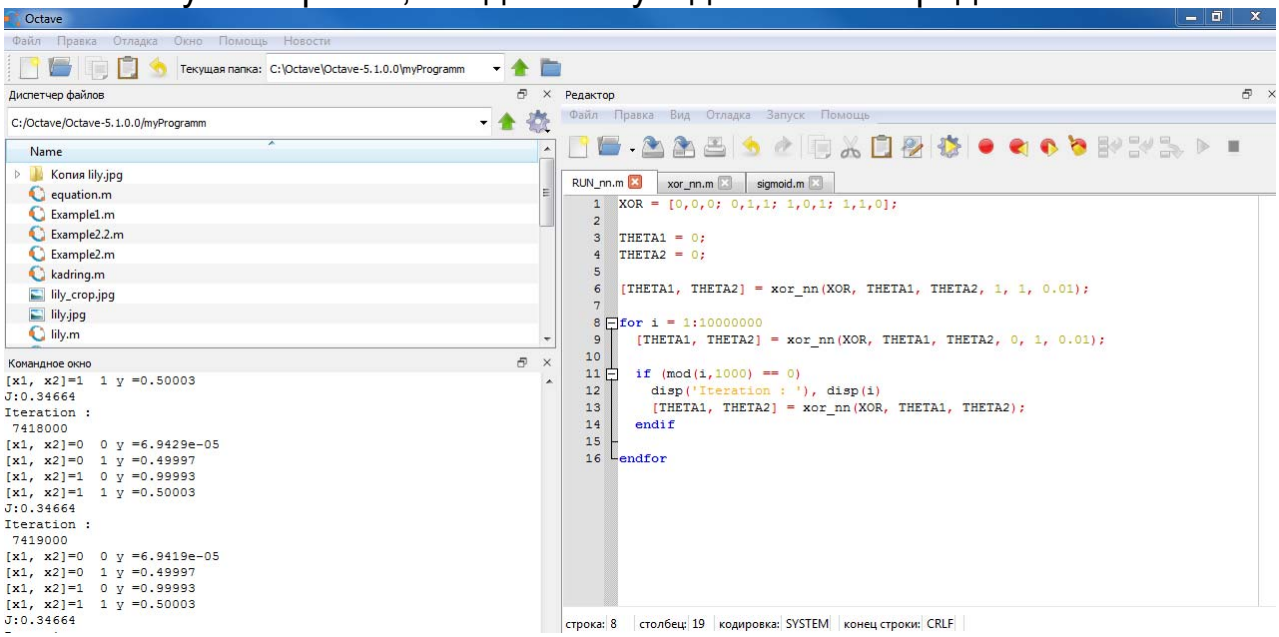
Вставим код, который обрабатывает ввод в пакетном режиме:

```
A1 = [1; XOR(i,1:2)'];
Z2 = THETA1 * A1;
A2 = [1; sigmoid(Z2)];
Z3 = THETA2 * A2;
h = sigmoid(Z3);
J = J + ( XOR(i,3) * log(h) ) + ( (1 - XOR(i,3)) * log(1 - h) );
m = m + 1;
if (learn == 1)
   delta3 = h - XOR(i,3);
   delta2 = ((THETA2' * delta3) .* (A2 .* (1 - A2)))(2:end);
   T2_DELTA = T2_DELTA + (delta3 * A2');
   T1_DELTA = T1_DELTA + (delta2 * A1');
else
   disp('Hypothesis for '), disp(XOR(i,1:2)), disp('is '), disp(h);
endif
endfor
J = J / -m;  % средняя стоимость по всем примерам
```

Обновление весов в сети в пакетном режиме:

```
if (learn==1)
   THETA1 = THETA1 - (alpha * (T1_DELTA / m));
   THETA2 = THETA2 - (alpha * (T2_DELTA / m));
else
   disp('J: '), disp(J);
endif
THETA1_new = THETA1;
THETA2_new = THETA2;
Endfunction
```

После запуска скрипта, мы должны увидеть что-то вроде этого:



От итерации к итерации мы увидим снижение стоимости сети – стоимость (J) уменьшается каждый раз, но не намного.

```
>> RUN_nn

Iteration :
 1000
[x1, x2]=0  0 y =0.51553
[x1, x2]=0  1 y =0.49919
[x1, x2]=1  0 y =0.49337
[x1, x2]=1  1 y =0.47858
J:0.69429
Iteration :
 2000
[x1, x2]=0  0 y =0.5182
[x1, x2]=0  1 y =0.5036
[x1, x2]=1  0 y =0.49827
[x1, x2]=1  1 y =0.48491
J:0.69406
Iteration :
 3000
[x1, x2]=0  0 y =0.51667
[x1, x2]=0  1 y =0.50339
[x1, x2]=1  0 y =0.49841
[x1, x2]=1  1 y =0.48617
J:0.69391
…
Iteration :
 9859000
[x1, x2]=0  0 y =5.1621e-05
[x1, x2]=0  1 y =0.49998
[x1, x2]=1  0 y =0.99995
```

```
[x1, x2]=1  1 y =0.50003
J:0.34662
Iteration :
 9860000
[x1, x2]=0  0 y =5.1616e-05
[x1, x2]=0  1 y =0.49998
[x1, x2]=1  0 y =0.99995
[x1, x2]=1  1 y =0.50003
J:0.34662
Iteration :
 9861000
[x1, x2]=0  0 y =5.161e-05
[x1, x2]=0  1 y =0.49998
[x1, x2]=1  0 y =0.99995
[x1, x2]=1  1 y =0.50003
J:0.34662
Iteration :
 9862000
[x1, x2]=0  0 y =5.1605e-05
[x1, x2]=0  1 y =0.49998
[x1, x2]=1  0 y =0.99995
[x1, x2]=1  1 y =0.50003
J:0.34662
…….
Iteration :
 9996000
[x1, x2]=0  0 y =5.0886e-05
[x1, x2]=0  1 y =0.49998
[x1, x2]=1  0 y =0.99995
[x1, x2]=1  1 y =0.50003
J:0.34662
Iteration :
 9997000
[x1, x2]=0  0 y =5.088e-05
[x1, x2]=0  1 y =0.49998
[x1, x2]=1  0 y =0.99995
[x1, x2]=1  1 y =0.50003
J:0.34662
Iteration :
 9998000
[x1, x2]=0  0 y =5.0875e-05
[x1, x2]=0  1 y =0.49998
[x1, x2]=1  0 y =0.99995
[x1, x2]=1  1 y =0.50003
J:0.34662
Iteration :
 9999000
[x1, x2]=0  0 y =5.087e-05
[x1, x2]=0  1 y =0.49998
```

[x1, x2]=1 0 y =0.99995
[x1, x2]=1 1 y =0.50003
J:0.34662
Iteration :
10000000
[x1, x2]=0 0 y =5.0865e-05
[x1, x2]=0 1 y =0.49998
[x1, x2]=1 0 y =0.99995
[x1, x2]=1 1 y =0.50003
J:0.34662

Средняя ошибка стала меньше **J:0.69429 < J:0.34662.** Обученная нейронная сеть выдает значения функции XOR ближе к ее истинным значениям, чем те значения, которые она выдавала на первых итерациях ее обучения, но не для всех примеров. В таблице ниже можно заметить, что нейронная сеть хорошо обучена для примеров [x1, x2, y] = (0,0,0) и (1,0,1).

Переменные функции XOR		**Значение функции Iteration :**		**Реальное значение функции**
x1	**x2**	**1000**	**10000000**	**y**
0	0	**[x1, x2]=0 0 y =0.51553**	**[x1, x2]=0 0 y =5.0865e-05 ≈ 0**	0
0	1	**[x1, x2]=0 1 y =0.49919**	**[x1, x2]=0 1 y =0.49998**	1
1	0	**[x1, x2]=1 0 y =0.49337**	**[x1, x2]=1 0 y =0.99995 ≈ 1**	1
1	1	**[x1, x2]=1 1 y =0.47858**	**[x1, x2]=1 1 y =0.50003**	0

Обучить нейронную сеть тому, чтобы она находила значения бинарной булевой функции XOR (исключающее или) сложнее, чем всех остальных перечисленных в таблице 1. В подтверждение данному заключению была предъявлена программе обучающая выборка в виде XOR = [0,0,0; 0,1,0; 1,0,0; 1,1,0], значения которой относятся к нулевой бинарной булевой функции $\mathrm{F}(x_1, x_2) \equiv 0$ (на всех парах значений для двух переменных значение функции тождественно равно нулю – в представленном массиве они подчеркнуты).

Переменные функции тождественного нуля		**Значение функции Iteration :**		**Реальное значение функции** $\mathrm{F}(x_1, x_2) \equiv 0$
x1	**x2**	**1000**	**1000000**	**y**
0	0	**[x1, x2]=0 0 y =0.060467**	**[x1, x2]=0 0 y = 5.8837e-05 ≈ 0**	0
0	1	**[x1, x2]=0 1 y =0.057833**	**[x1, x2]=0 1 y = 4.2769e-05 ≈ 0**	0
1	0	**[x1, x2]=1 0 y =0.055611**	**[x1, x2]=1 0 y = 5.5279e-05 ≈ 0**	0
1	1	**[x1, x2]=1 1 y =0.049437**	**[x1, x2]=1 1 y = 4.1605e-05 ≈ 0**	0

Обученная нейронная сеть для функции тождественного нуля выдает значения, которые практически совпадают с истинными значениями для всех примеров. Из таблицы выше можно заметить, что нейронная сеть $\mathrm{F}(x_1, x_2) \equiv 0$ хорошо обучена для всех примеров [x1, x2, y].

Варианты для выполнения задания

Построить простую нейронную сеть для обучения распознавания булевой функции соответствующего варианта. Номер варианта выбирается учащимися по номеру в журнале группы.

Таблица 1 – Таблица булевых функций от двух переменных

Вариант	**Булева функция двух переменных**	**Формула,** $\mathrm{F}(x_1, x_2) = y$
1	Стрелка Пирса	$x_1 \downarrow x_2$
2	штрих Шеффера	$x_1 \mid x_2$
3	дизъюнкция	$x_1 \vee x_2$
4	прямая импликация	$x_1 \rightarrow x_2$
5	эквивале́нтность	$x_1 \equiv x_2$
6	конъюнкция	$x_1 \wedge x_2$
7	обратная импликация	$x_1 \leftarrow x_2$
8	тождественная единица, тавтология	$\mathrm{F}(x_1, x_2) \equiv 1$
9	тождественный ноль	$\mathrm{F}(x_1, x_2) \equiv 0$

Порядок выполнения задания

1. Задание лабораторной работы выполняется индивидуально.
2. По результатам работы необходимо сформировать отчет, в котором отразить цель работы, последовательность выполненных действий, скрипт на языке GUI Octave с поясняющими комментариями, а также результаты выполнения работы, анализ того, насколько качественно обучена ваша нейронная сеть с выводами.
3. Отчёт сдается в электронном виде отправкой в электронный почтовый ящик преподавателя.

Ход работы

1. Изучить теоретический материал по теме лабораторной работы (методические указания, ссылки на литературные источники).
2. Согласно номеру своего варианта выбрать булеву функцию двух переменных.
3. Построить простую нейронную сеть для обучения распознавания булевой функции. В листинге скрипта RUN_nn.m замените обучающую выборку XOR = [0,0,0; 0,1,1; 1,0,1; 1,1,0]; на соответствующую для своей булевой функции.

4. Проект нейронной сети должен состоять из трех скриптов (см. Приложения А, Б), которые необходимо сохранить в одной текущей папке, перед тем как запускать на выполнение скрипт-запуска на обучение нейронной сети (Run_nn).
5. Проследить итеративность обучения нейронной сети в системе Octave, используя разобранный образец из приложения А.
6. При обучении нейронной сети использовать систему вычислений Octave.
7. Оформить отчет по лабораторной работе.
8. Защитить лабораторную работу.

Литература

1. https://www.gnu.org/software/octave/
2. Е. Р. Алексеев, О. В. Чеснокова «Введение в Octave для инженеров и математиков» М.: ALT Linux, 2012
3. https://github.com/StephenOman/Octave/tree/master/xor%20neural%20network
4. https://aimatters.wordpress.com/2015/12/19/

Листинг скрипта RUN_nn.m

Скрипт для запуска на обучение нейронной сети. В вызове функции xor_nn программа инициализирует весовые коэффициенты. Вызов этой функции производим в цикле 100 000 раз, распечатывая среднее значение отклонение по стоимости каждые 1000 итераций. При этом веса сети каждый раз корректируются на небольшую величину.

```
XOR = [0,0,0; 0,1,1; 1,0,1; 1,1,0];

THETA1 = 0;
THETA2 = 0;

[THETA1, THETA2] = xor_nn(XOR, THETA1, THETA2, 1, 1, 0.01);

for i = 1:100000
  [THETA1, THETA2] = xor_nn(XOR, THETA1, THETA2, 0, 1, 0.01);

  if (mod(i,1000) == 0)
    disp('Iteration : '), disp(i)
    [THETA1, THETA2] = xor_nn(XOR, THETA1, THETA2);
  endif

endfor
```

Приложение Б

Листинг функции xor_nn **сохранить в файле xor_nn.m**

```
function [THETA1_new, THETA2_new] = xor_nn(XOR, THETA1,
THETA2, init_w=0, learn=0, alpha=0.01)
  if (init_w == 1)
     THETA1 = 2*rand(2,3) - 1;
     THETA2 = 2*rand(1,3) - 1;
  endif
  T1_DELTA = zeros(size(THETA1));
  T2_DELTA = zeros(size(THETA2));
  m = 0;
  J = 0.0;
  for i = 1:rows(XOR)
    A1 = [1; XOR(i,1:2)'];
    Z2 = THETA1 * A1;
    A2 = [1; sigmoid(Z2)];
    Z3 = THETA2 * A2;
    h = sigmoid(Z3);
    J = J + ( XOR(i,3) * log(h) ) + ( (1 - XOR(i,3)) * log(1 - h) );
    m = m + 1;
    if (learn == 1)
      delta3 = h - XOR(i,3);
      delta2 = ((THETA2' * delta3) .* (A2 .* (1 - A2)))(2:end);
      T2_DELTA = T2_DELTA + (delta3 * A2');
      T1_DELTA = T1_DELTA + (delta2 * A1');
    else
      s=strcat('[x1, x2]=  ', num2str(XOR(i,1:2)), ' y = ', num2str(h));
      disp(s);
      %disp('Гипотеза для ', ), disp(XOR(i,1:2)), disp('is '), disp(h);
    endif
  endfor
  J = J / -m;
  if (learn==1)
    THETA1 = THETA1 - (alpha * (T1_DELTA / m));
    THETA2 = THETA2 - (alpha * (T2_DELTA / m));
  else
    s=strcat('J: ', num2str(J));
      disp(s);
    %disp('J: '), disp(J);
  endif
  THETA1_new = THETA1;
  THETA2_new = THETA2;
endfunction
```

Листинг функции sigmoid **сохранить в файле sigmoid.m**

```
function [result] = sigmoid(x)
  result = 1.0 ./ (1.0 + exp(-x));
 end
```