САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ПЕТРА ВЕЛИКОГО

Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Отчет

Лабораторная 2

Исследование нейронных сетей прямого распространения

Дисциплина
«Нейроинформатика»

выполнил:
Косенков М.А.
группа: 33531/2
преподаватель:
Никитин К.В.

Санкт-Петербург
2018

Оглавление

Изучение вспомогательных функций

1.1. Изучение фукций инициализации

Задание

1. С помощью команды network создайте НС с n входами, 1 выходным слоем с m нейронами.

Задайте функцию инициализации НС initlay.

2. Задайте функцию инициализации слоев initnw. Произведите инициализацию НС с помощью функции init. Проанализируйте матрицы смещений и весов нейронной сети.

3. Задайте функцию инициализации слоев initwb.

Задавайте для весов и смещений различные функции инициализации (**initzero**, **rands**, **randsmall** – и для весов, и для смещений, **midpoint**, **randnc**, **randnr**, **initlvq**, **initsompc** – только для весов, **initcon** – только для смещений). Произведите инициализацию НС с помощью функции init. Проанализируйте матрицы смещений и весов нейронной сети.

Напишем функцию ***task1_1*** для создания НС, которая принимает параметры, позволяющие устанавливать количество входов, нейронов в слое и функции инициализации. Напишем еще функцию ***print***, которая будет печатать веса и смещения.

*Программа **task1_1***

```
function task1_1(n, m, layer_fun, weight_fun, bias_fun)
  rng('default');
  lfuns = {'initnw', 'initwb'};
  ifuns = {'initzero', 'rands', 'randsmall', ... % 1-3
        'midpoint', 'randnc', 'randnr', 'initlvq', 'initsompc', ... % 4-8          'initcon' % 9 };
  net = network(n, 1, 1, ones(1, n), 0, 1);
  net.layers{1}.size = m;
  net.layers{1}.initFcn = lfuns{layer_fun};
  net.biases{1}.initFcn = ifuns{bias_fun};
  for i = 1 : n
    net.inputs{i}.size = 1;
    net.inputWeights{i}.initFcn = ifuns{weight_fun};
  end
  net = init(net);
  print(net, n);
end
```

*Программа **print***

```
function print(net, n)
  disp('Biases for neyrons');
  disp(net.b{1});
  for i = 1 : n
    disp(['Weights for the ', num2str(i), ' input ']);
    disp(net.IW{1,i});
  end
end
```

Выберем количество входов $n=4$, и количество нейронов $m=3$.

Будем вызывать функцию ***task1_1*** через терминал *Matlab*, задавая различные функции инициализации.

Функция *initlay* используется по умолчанию, поэтому нет необходимости задавать ее вручную.

Зададим функцию инициализации слоев *initnw*, а остальные функции можно задавать любыми, так как они будут проигнорированы. Результат показан в Таблице 1.1.1.

Таблица 1.1.1. Функция *initnw*.

Для какого нейрона	Веса			Смещения
	1-го входа	2-го входа	3-го входа	
1-го	0.6294	0.2647	0.9150	0.3110
2-го	0.8116	-0.8049	0.9298	-0.6576
3-го	-0.7460	-0.4430	-0.6848	0.4121
4-го	0.8268	0.0938	0.9412	-0.9363

Зададим теперь функцию инициализации весов *initwb*. Теперь нам нужно задавать функции инициализации весов и смещений. Начнем с функций, которые применимы и к весам, и к смещениям: *initzero*, *rands*, *randsmall*. Результаты представлены в Табл. 1.1.2, 1.1.3 и 1.1.4.

Таблица 1.1.2. Функция *initzero*.

Для какого нейрона	Веса			Смещения
	1-го входа	2-го входа	3-го входа	
1-го	0	0	0	0
2-го	0	0	0	0
3-го	0	0	0	0
4-го	0	0	0	0

Таблица 1.1.3. Функция *rands*.

Для какого нейрона	Веса			Смещения
	1-го входа	2-го входа	3-го входа	
1-го	0.2647	0.9150	0.9143	0.6294
2-го	-0.8049	0.9298	-0.0292	0.8116
3-го	-0.4430	-0.6848	0.6006	-0.7460
4-го	0.0938	0.9412	-0.7162	0.8268

Таблица 1.1.3. Функция *randsmall*.

Для какого нейрона	Веса			Смещения
	1-го входа	2-го входа	3-го входа	
1-го	0.0026	0.0092	0.0091	0.0063
2-го	-0.0080	0.0093	-0.0003	0.0081
3-го	-0.0044	-0.0068	0.0060	-0.0075
4-го	0.0009	0.0094	-0.0072	0.0083

Функция *initzero* дает нам нули, *rands* – генерирует случайные значения в диапазоне от -1 до 1, а *randsmall* – генерирует такие же случайные значения, какие и *rands* только меньше на 2 порядка.

Теперь посмотрим на функции для инициализации весов: *midpoint*, *randnc*, *randnr*, *initlvq*, *initsompc*. Для смещений зададим initzero, однако они в таблицах представлены не будут.

Начнем с функции *midpoint*. Она устанавливает веса как средние значения диапазона возможных значений. Необходимо установить *net.inputs{i}.range*. Напишем файл *task1_1_1* и укажем диапазоны допустимых на входе значений. Результаты представлены в Табл. 1.1.4.

Программа ***task1_1_1***

```
rng('default');
n = 3;
m = 4;
net = network(n, 1, 1, ones(1, n), 0, 1);
net.layers{1}.size = m;
net.layers{1}.initFcn = 'initwb';
for i = 1 : n
   net.inputs{i}.size = 1;
   net.inputWeights{i}.initFcn = 'midpoint';
end
net.inputs{1}.range = [2 8];
net.inputs{2}.range = [-10 27];
net.inputs{3}.range = [0.4 1];
net = init(net);
print(net, n);
```

Таблица 1.1.4. Функция *midpoint*.

Для какого нейрона	Веса		
	1-го входа	1-го входа	1-го входа
1-го	5	8.5000	0.7000
2-го	5	8.5000	0.7000
3-го	5	8.5000	0.7000
4-го	5	8.5000	0.7000

Теперь посмотрим функции *randnc* и *randnr*, которые формируют нормализованные значения по столбцам или строкам, снова запустив функцию *task1_1*. Результаты представлены в Табл. 1.1.5 и в Табл. 1.1.6.

Таблица 1.1.5. Функция *randnc*.

Для какого нейрона	Веса		
	1-го входа	1-го входа	1-го входа
1-го	0.4155	0.2755	0.5234
2-го	0.5357	-0.8378	0.5318
3-го	-0.4925	-0.4611	-0.3917
4-го	0.5457	0.0976	0.5383

Таблица 1.1.6. Функция *randnr*.

Для какого нейрона	Веса		
	1-го входа	1-го входа	1-го входа
1-го	1	1	1
2-го	1	-1	1
3-го	-1	-1	-1
4-го	1	1	1

1.2. Изучение функций предобработки входов/выходов

Задание

Задавайте в качестве функций предобработки различные встроенные функции (**fixunknowns**, **mapminmax**, **mapstd**, **processpca**, **removeconstantrows**, **removerows**) – по очереди по одной.

Для каждой из функций предобработки подберите актуальные входные примеры, на которые эти функции должны срабатывать (для fixunknowns, например, задайте в качестве одного из входных сигналов NaN). Подайте эти примеры на вход этих функций и проанализируйте обработанные функциями примеры.

Посмотрим на то, как работают функции пред/пост-обработки данных. Для демонстрации работы функций была написана программа *task1_2*.

*Программа **task1_2***

```
data1 = [1 2 1 3 NaN; NaN 2 3 1 5];
data1fixed = fixunknowns(data1)
data2 = [1 5 10 -1];
data2fixed = mapminmax(data2)
data3 = [2 3 4 109; .1 .3 .12 .07];
data3fixed = mapstd(data3)
data4 = [1 2 1 2 3; 3 5 3 5 7; 2 4 2 4 6];
data4fixed = processpca(data4, 0.02)
data5 = [1 1 1 1; 2 3 1 2; 2 2 2 2; 0 8 0 0];
data5fixed = removeconstantrows(data5)
data6 = [1 2 3; 2 3 5; 9 8 0];
data6fixed = removerows(data6, [1 3])
```

Функция *fixunknowns* заменяет неизвестное значение на адекватное. При непосредственном вызове данной функции мы получим под каждой строкой, в которой присутствовали неизвестные значения, новую строку, показывающую в каком столбце предыдущей строки, были известные значения – это 1 и неизвестные – это 0. Результат обработки data1, показан в Табл. 1.2.1.

Таблица 1.2.1. Матрица *data1fixed*.

1.0000	2.0000	1.0000	3.0000	1.7500
1.0000	1.0000	1.0000	1.0000	0
2.7500	2.0000	3.0000	1.0000	5.0000
0	1.0000	1.0000	1.0000	1.0000
1.0000	2.0000	1.0000	3.0000	1.7500

Функция *mapminmax* нормализует входные данные так, чтобы они лежали в интервале [-1, 1]. Максимальному значению присваивается 1, минимальному – -1, промежуточные пересчитываются. Результат обработки входного вектора *data2* представлен в Табл. 1.2.2.

Таблица 1.2.2. Вектор *data2fixed*.

-0.6364	0.0909	1.0000	-1.0000

Функция *mapstd* нормализует входные данные так, чтобы их математическое ожидание было равно нулю, а среднеквадратичное отклонение было единицей. Результат обработки входного матрицы *data3* показан в Табл. 1.2.3.

Таблица 1.2.3. Вектор *data3fixed*.

-0.5188	-0.4999	-0.4811	1.4998
-0.4580	1.4703	-0.2651	-0.7472

Функция *processpca* – метод главных компонент, метод уменьшения размерности входных данных при наименьшей потере информации. Для теста этого метода была написана матрица *data4* с линейно-зависимыми строками. Для функции *processpca* можно указать степень коррелированности данных при которой можно удалять строки. Делается это следующим образом:

```
net.inputs{1}.processFcns = {'processpca'};
net.inputs{1}.processParams{1}.ind = 0.02;
```

Результаты работы данной функции на матрице *data4* показаны в Табл. 1.2.4.

Таблица 1.2.4. Вектор *data4fixed*.

-3.7256	-6.7082	-3.7256	-6.7082	-9.6907

Функция *removeconstantrows* просто удаляет строки, в которых все значения одинаковые. Результат обработки матрицы *data5* показан в Табл. 1.2.5.

Таблица 1.2.5. Вектор *data5fixed*.

2	3	1	2
0	8	0	0

Функция *removerows* удаляет заданные строки. Для задания строк, которые необходимо удалить нужно задать параметр для функции предобработки. Например, так:

```
net.inputs{1}.processFcns = {'removerows'};
net.inputs{1}.processParams{1}.ind = [1 3];
```

Результат обработки входной матрицы data6 показан в Табл. 1.2.6.

Таблица 1.2.6. Вектор *data6fixed*.

2	3	5

1.3. Изучение функции расчета производительности (усреднения ошибки)

Задание

Сформируйте различные вектора (матрицы) ошибок E (интерпретируйте их как вектора ошибок НС на тестовой выборке). Размерность E = число выходов * число векторов. Последовательно примените к ним функции mae, mse, sae, sse, crossentropy, mseparse.

Проанализируйте каждый из способов усреднения ошибки (можно ли ему доверять и когда – при каком числе выходов НС и при каком объеме тестовой

выборки). Рассмотрите случаи, когда ошибки сильно рознятся по входам/примерам.

Рассмотрим формулы некоторых из функций усреднения ошибки.

Функция *mse*

$$mse = \frac{1}{Q}\sum_{i=1}^{Q}(t_i - x_i)^2$$

Функция *mae*

$$mae = \frac{1}{Q}\sum_{i=1}^{Q} ¿\, t_i - x_i \vee ¿¿$$

Функция *sse*

$$sse = \sum_{i=1}^{Q}(t_i - x_i)^2$$

Функция *sae*

$$sae = \sum_{i=1}^{Q} ¿\, t_i - x_i \vee ¿¿$$

Для тестирования работы функций была написана программа *task1_3*. В качестве целевого вектора написан массив *E1_X*, а в качестве ответа нейронной сети взят вектор *X1_T*. Ответ нейронной сети отличается от цели двумя значениями, которые больше на 0.3 каждое. Так же есть вектора *E2_X* и *E2_T*, которые равны векторам *E1_X* и *E1_T* разделенным на 100. Для функции кросс-энтропии написаны вектора *E и E1*. Ожидается, что распознанный объект принадлежит первому классу, а НС с вероятностью 0.7 распознала входные данный, как принадлежащие классу 1.

Программа ***task1_3***

```
function task1_3
E1_X = [1 2 1 3 1 2 1];
E1_T = [1 2.3 1 3 1 2 1.3];
disp('E1');
disp('mae');
mae(E1_X, E1_T)
disp('mse');
mse(E1_X, E1_T)
disp('sae');
sae(E1_X, E1_T)
disp('sse');
sse(E1_X, E1_T)
disp('E2');
E2_X = E1_X / 100;
E2_T = E1_T / 100;
disp('mae');
mae(E2_X, E2_T)
disp('mse');
mse(E2_X, E2_T)
disp('sae');
sae(E2_X, E2_T)
disp('sse');
sse(E2_X, E2_T)

E = [1 0 0 0 0 0 0];
```

```
E1 = [0.7 0.1 0.05 0.05 0.05 0.03 0.02];
crossentropy(E, E1)
```

Результаты программы представлены в Табл. 1.3.1.

Функция	Результат	
	E1	E2
mae	0.0857	8.5714e-04
mse	0.0257	2.5714e-06
sae	0.6000	0.0060
sse	0.1800	1.8000e-05
crossentropy		0.0952

Видим, что для значений больше 1 подходят все функции: *mae*, *mse*, *sae* и *sse*, но для значений меньше единицы лучше использовать функцию *sae*. Для сетей, занимающихся классификацией, нужно использовать *crossentropy*. Ее можно регулировать, изменяя параметры *performParam regularization* и *normalization*.

1.4. Изучение функции деления выборки

Задание

Для объема выборки Q = 100 примените функции деления выборки divideblock, divideind, divideint, dividerand, dividetrain. Ознакомьтесь с входными параметрами каждой из функций и при исследовании функций меняйте эти параметры.

Проанализируйте, как формируются обучающая, тестовая и контрольная выборки для каждой из функций.

Установить каждую из функций можно через *net.divideFcn*, параметры – через *net.divideParam*.

Функция *divideblock* делит выборку по умолчанию на три части: обучающую – 70%, контрольную – 15% и тестовую – 15%. Можно самостоятельно задать процентное соотношение этих величин через *trainRatio*, *valRatio* и *testRatio* для обучающей, контрольной и тестовой частей соответственно. При вызове отдельной функции, она принимает параметры в следующем порядке:

```
divideblock(data, trainRatio, valRatio, testRatio)
```

Функция *divideind* делит выборку на три части по заданным индексам. Индексы задаются через *trainInd*, *valInd* и *testInd* для обучающей, контрольной и тестовой частей соответственно. При вызове отдельно данной функции, она принимает параметры в следующем виде:

```
divideind(data, trainInd, valInd, testInd)
```

Функция *divideint* делит выборку на три части по заданным соотношениям, на выходе дает три массива индексов. Параметры задаются такие же, какие и в функции *divideblock*.

Функция *dividerand* работает аналогично *divideblock*, однако разбиение происходит случайным образом, то есть индексы входного массива, которые попадут в тот или иной подмассив случайны. В остальном функция аналогична *divideblock*.

Функция *dividetrain* выделяет всю выборку как тренировочные данные. Если подать в функцию данные, то она вернет данные, если подать число, то вернет индексы от 1 до заданного числа.

2. Линейные нейронные сети

2.1. Аппроксимация нелинейной зависимости

Задание

1. Сформируйте обучающую и тестовую выборки для задачи аппроксимации 1- мерной нелинейной зависимости (варианты с номером 5).
2. Создайте линейную НС и обучите ее на обучающей выборке. Постройте графики зависимостей, реализуемых НС на различных этапах обучения на одном графике с исходной зависимостью. Оцените ошибку аппроксимации (абсолютную и относительную) на обучающей и тестовой выборках.
3. Создайте НС и одновременно обучите ее в соответствии с МНК с помощью функции newlind. Сравните результаты (результирующую функцию и ошибки) с предыдущим вариантом.

Для выполнения данного задания была написана программа *task2_1*. В ней создаются 2 НС: одна с помощью функции *network*, другая с помощью функции *newlind*. Первая обучается 4 раза с установленным количеством эпох 1, чтобы можно было смотреть промежуточные результаты. Ошибки проверяются с помощью функции *mae*. В качестве функции обучения была выбрана функция "*trainscg*" – метод сопряженных градиентов с масштабированием.

Программа ***task2_1***

```
rng('default')
Ntrain = 100; % Size of train data
Ntest = 2000; % Size of test data

raw_train = rand(1, Ntrain); % Generate random values for train data
raw_train = cat(2, -raw_train(1:Ntrain/2), raw_train(Ntrain/2 + 1:end) );
raw_test = rand(1, Ntest);
raw_test = cat(2, -raw_test(1:Ntest/2), raw_test(Ntest/2 + 1:end) );
target_train = my_fun(raw_train);
target_test = my_fun(raw_test);

net = network(1, 1, 1, 1, 0, 1);
net.inputs{1}.size = 1;
net.layers{1}.initFcn = 'initnw';
net.layers{1}.size = 1;

net = init(net);
net.trainFcn = 'trainscg';
net.trainParam.epochs = 1;

net = train(net, raw_train, target_train);
net_answer_1 = sim(net, raw_test);
netwb_1 = getwb(net); % ax + b, a and b
net = train(net, raw_train, target_train);
net_answer_2 = sim(net, raw_test);
netwb_2 = getwb(net);
net = train(net, raw_train, target_train);
net_answer_3 = sim(net, raw_test);
netwb_3 = getwb(net);

netl = newlind(raw_train, target_train);
netl_answer = sim(netl, raw_test);
```

```
netlwb = getwb(netl);
figure
hold on
scatter(raw_train, target_train, 'r');
plot(raw_test, netlwb(2) * raw_test + netlwb(1), 'b');
plot(raw_test, netwb_1(2) * raw_test + netwb_1(1), 'm');
plot(raw_test, netwb_2(2) * raw_test + netwb_2(1), 'c');
plot(raw_test, netwb_3(2) * raw_test + netwb_3(1), 'g');
legend(["real" "By newlind" "After 1 epoch" "After 2 epochs" "After 3 epochs"]);
axis([-1 1 -1 1])
axis square
hold off
mae(target_test, net_answer_1)
mae(target_test, net_answer_2)
mae(target_test, net_answer_3)
mae(target_test, netl_answer)
```

Функции, полученные двумя НС представлены на Рис. 2.1.1. Ошибки представлены в Табл. 2.1.1.

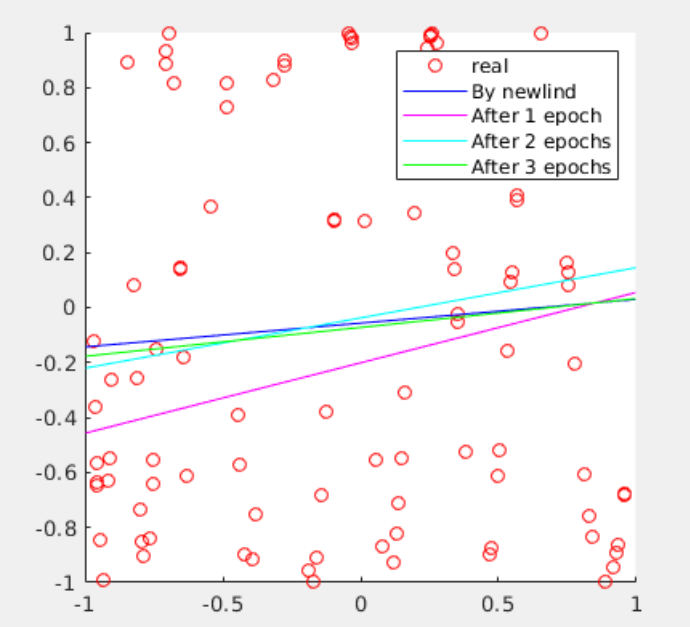


Рис. 2.1.1. Графики функций прямых.

Таблица 2.1.1. Ошибки НС.

Функция		Ошибка
network	1 epoch	0.6564
	2 epoch	0.6478
	3 epoch	0.6401
newlind		0.6391

Видим, что после 3 эпох НС, созданая функцией network дает почти такую же ошибку, какие и НС, созданная с помощью newlind.

2.2. Выбор скорости обучения

Задание

Сформируйте обучающую выборку для функции (варианты с номером 5). Определите максимальную скорость обучения с помощью функции maxlinlr (разберитесь с принципом определения этой скорости). Проверьте, является ли эта скорость максимальной и если нет, то экспериментальным путем определите максимально допустимую скорость обучения. Затем проведите последовательность экспериментов для трех вариантов (скорость обучения а) больше (1 значение), б) равна (1 значение), в) меньше максимально допустимой (3 значения):

1. Задайте фиксированные начальные значениями [W,b].
2. Проведите обучение на достаточном числе циклов обучения. Постройте график изменения вектора [W,b] в течение обучения на графике поверхности (линий равного уровня) ошибок.

Сравните результаты. Определите значение скорости обучения, при которой обучение проходит быстрее всего.

Функция *maxlinlr* определяет мксимальную скорость обучения для линейной сети. Функция принимает в качестве своего аргумента массив, который подается на вход нейронной сети. Для слоя без смещений используется следующий алгоритм.

Шаг 1. Умножение входного массива на себя транспонированного.

$$X^{i} = X * X^{T}$$

Шаг 2. Нахождение собственных чисел массива X^{i}.

$$V = eig(X^{i})$$

Шаг 3. Нахождение максимального собственного числа.

$$maxv = max(V)$$

Шаг 4. Деление 0.999 на максимальное собственное число.

$$lr = \frac{0.999}{maxv}$$

Для слоев со смещениями на первом шаге перед умножением к матрице X добавляется строка, состоящая из единиц.

Напишем программу *task2_2* (за основу взята программа *neyro_lin_example1*) и посмотрим, как выбор разной скорости обучения влияет на ошибку.

Программа ***task2_2***

```
function task2_2 (mult)
rng('default')
```

```
close all;
Ntrain = 500; % Size of train data
Ntest = 500; % Size of test data
raw_train = rand(1, Ntrain); % Generate random values for train data
raw_train = sort(cat(2, -raw_train(1:Ntrain/2), raw_train(Ntrain/2 + 1:end)));
raw_test = rand(1, Ntest);
raw_test = sort(cat(2, -raw_test(1:Ntest/2), raw_test(Ntest/2 + 1:end)));
Ttrain = my_fun(raw_train);
Ttest = my_fun(raw_test);
maxlr = mult * maxlinlr(raw_train,'bias');
delta_epochs = 1;
n_delta = 40;
net = newlin(raw_train,1,0, maxlr);
net.trainParam.showWindow = 0;
net.iw{1} = -.9;
net.b{1} = -.32;
net.trainparam.epochs = delta_epochs;
w = zeros(n_delta +1,2);
w(1,:) = [net.iw{1} net.b{1}];
for i = 1: n_delta
    net = train(net,raw_train,Ttrain);
    w(i+1,1) = net.iw{1};
    w(i+1,2) = net.b{1};
end
subplot(2,1,1);
w_range = -1:0.1:1;
b_range = -1:0.1:1;
ES = errsurf(raw_train,Ttrain,w_range,b_range,'purelin');
contour(w_range, b_range, ES);
hold on;
grid on;
plot(w(:,1), w(:,2), '-ro');
xlabel('w_{1}'); ylabel('b');
subplot(2,1,2);
plot(raw_train,Ttrain, '-ro');
hold on; grid on;
Y = sim(net,raw_test);
plot(raw_test,Y, '-bx');
legend({'Desire', 'Real'});
xlabel('x'); ylabel('y');
fprintf('mse: %f; \tmae: %f\n', mse(Ttest, Y), mae(Ttest, Y));
fprintf('sse: %f; \tsae: %f\n', sse(Ttest, Y), sae(Ttest, Y));
w(:,1)
w(:,2)
```

Будем запускать программу указывая различные значения *mult*.

Графики изменения весов и смещений, а также фукнции, реализуемые в итоге НС для каждого из 5 значений *mult* представлены на Рис. 2.2.1 – 2.2.5.

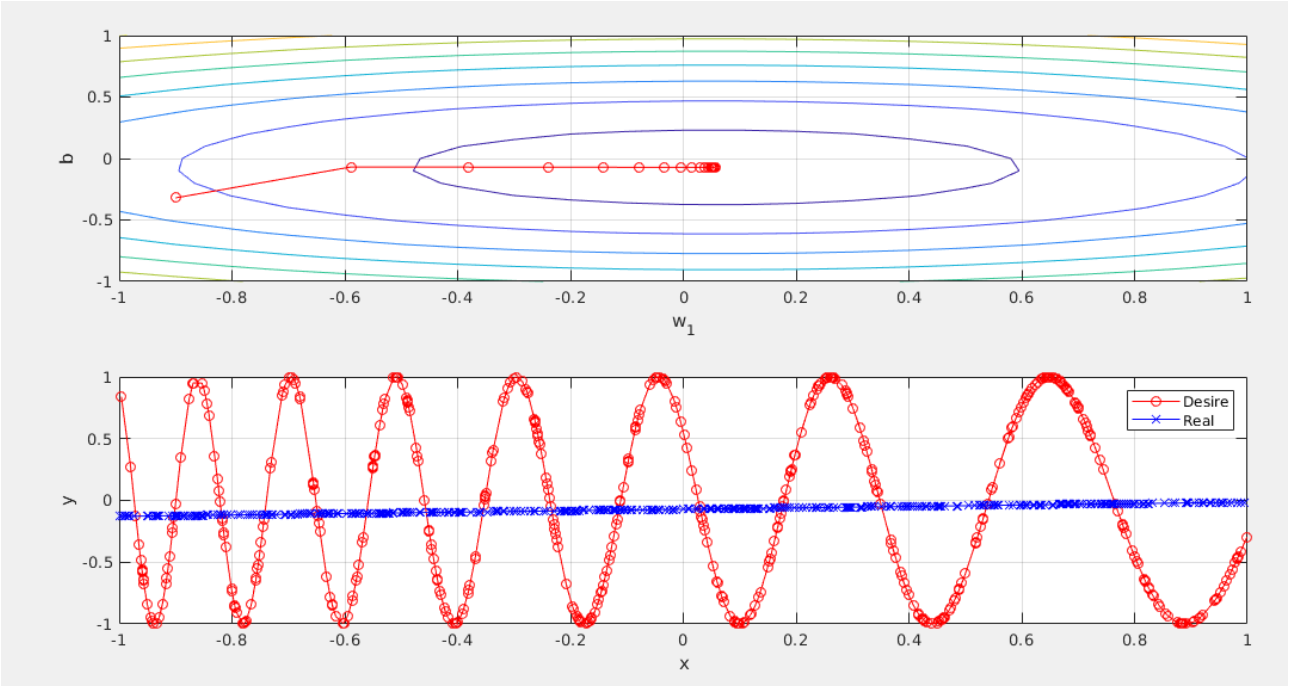


Рисунок 2.2.1. Значение *mult* = 1.

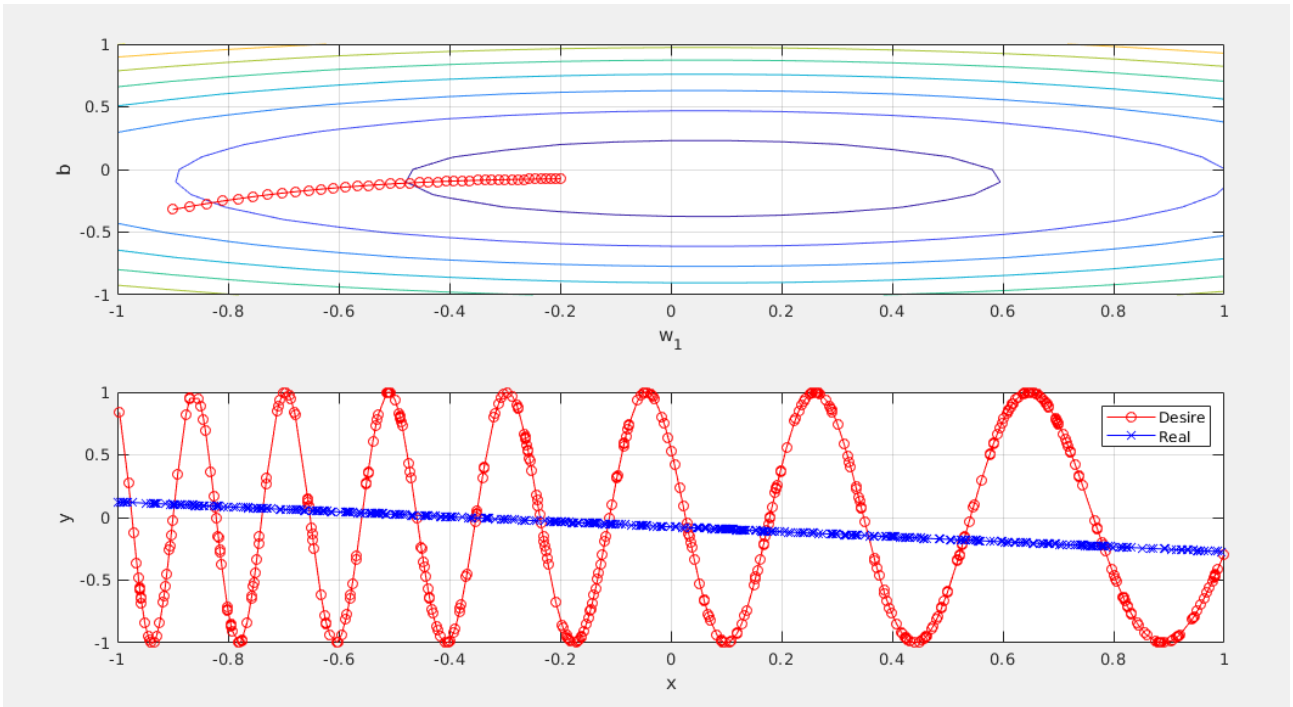


Рисунок 2.2.2. Значение *mult* = 0.1.

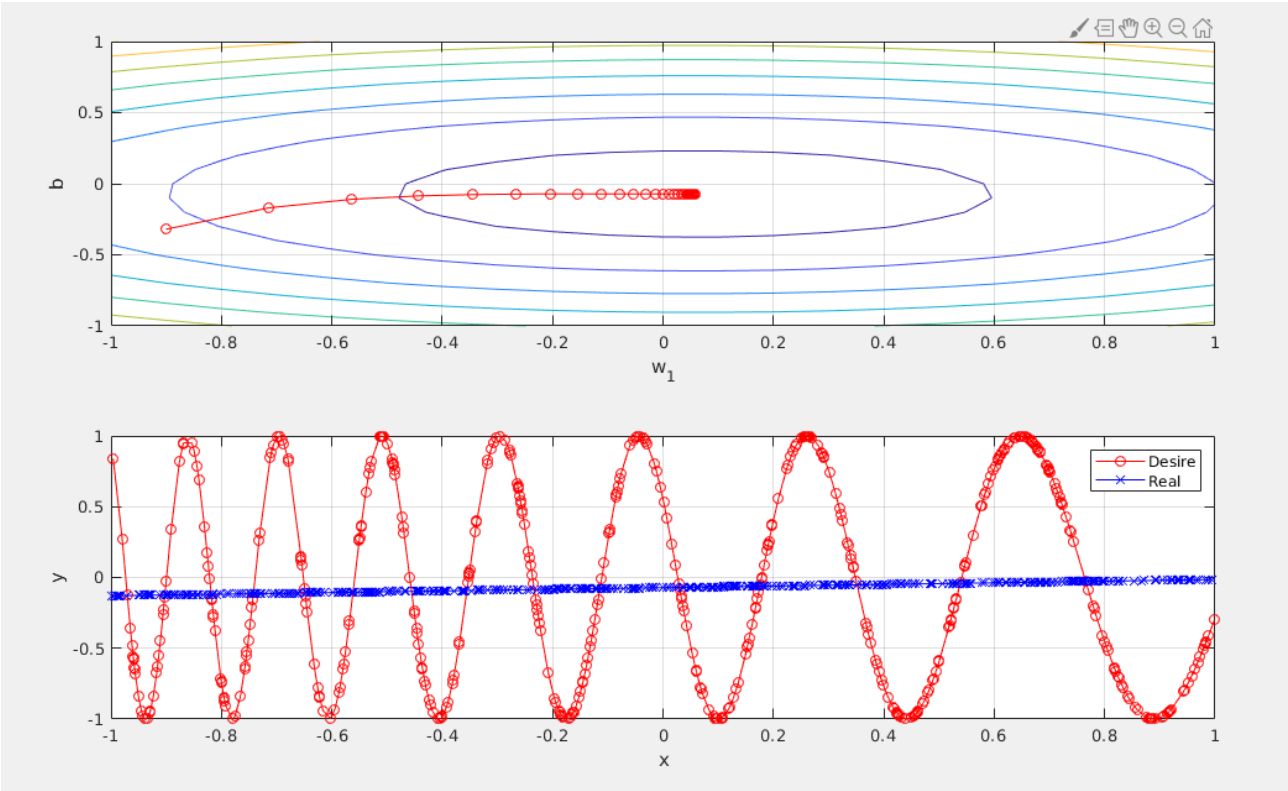


Рисунок 2.2.3. Значение *mult* = 0.6.

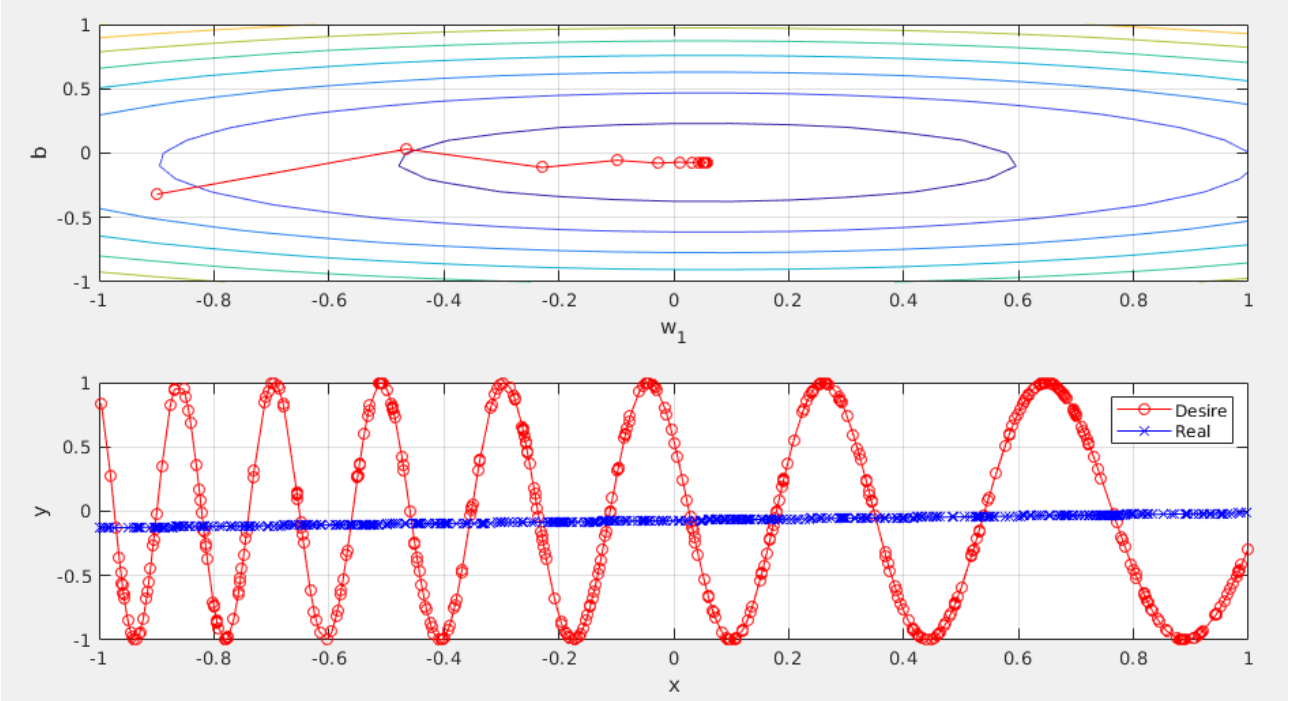


Рисунок 2.2.4. Значение *mult* = 1.4.

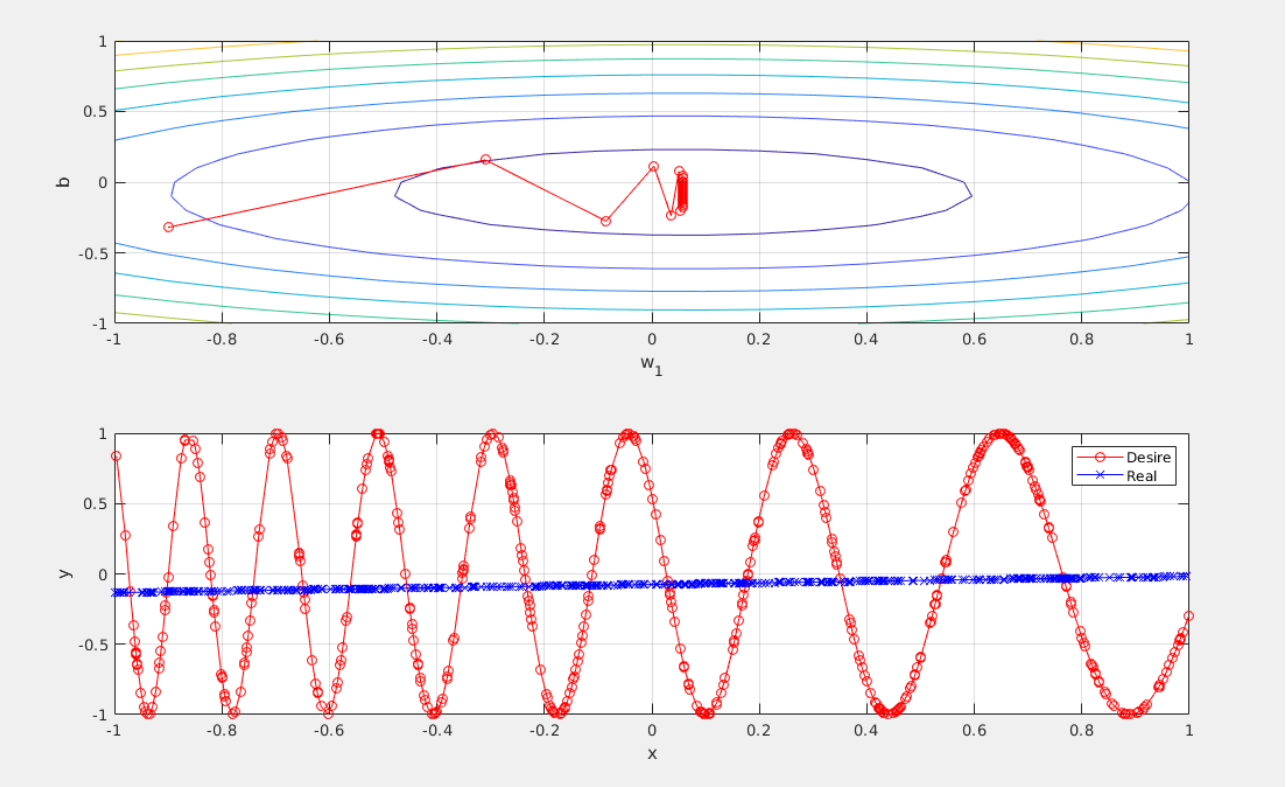


Рисунок 2.2.5. Значение *mult* = 1.9.

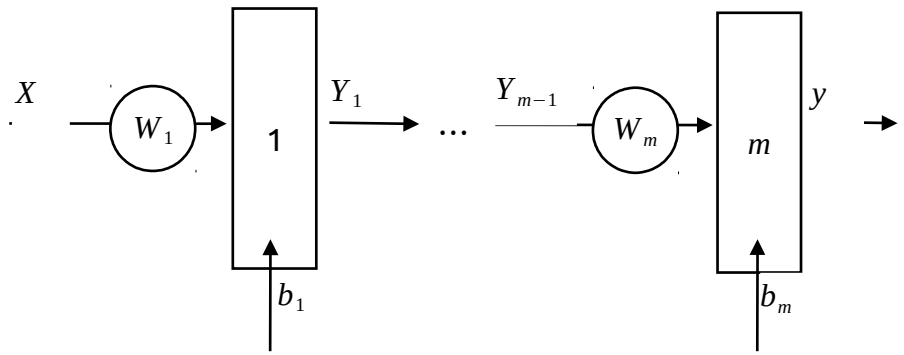
2.3 Многослойная линейная НС

Задание

Покажите математически, что многослойная линейная сеть эквивалентна однослойной линейной НС.

Разработайте соответствующий алгоритм перехода от многослойной к эквивалентной однослойной НС для некоторой формализованной многослойной линейной НС (n0 входов, s слоев, в каждом слое i по ni нейронов, весовой коэффициент от i нейрона k-го слоя к j нейрону (k+1)-го слоя равен $w_{ij}^{(k)}$.

Докажем, что многослойная линейная НС эквивалента однослойной. Рассмотрим сеть, состоящую из $m-1$ скрытого слоя. Каждый i – ый слой имеет n_i нейронов, веса W_i и смещения b_i. Размерность входа n_0. Размерность выхода 1.



Напишем матричные уравнения для каждого слоя.

$Y_1 = W_1 X + b_1$
$Y_2 = W_2 Y_1 + b_2$
$Y_3 = W_3 Y_2 + b_3$
…
$Y_{m-1} = W_{m-1} Y_{m-2} + b_{m-1}$
$y = W_m Y_{m-1} + b_m$

В каждом уравнении следующего слоя используется выход предыдущего слоя. Подставим уравнение для Y_1 в уравнение для Y_2.

$$Y_2 = W_2 \left(W_1 X + b_1 \right) + b_2 = W_2 W_1 X + W_2 b_1 + b_2$$

Размерности векторов:

$W_1 - n_1 \times n_0$
$W_2 - n_2 \times n_1$
$X - n_0 \times 1$
$b_1 - n_1 \times 1$
$b_2 - n_2 \times 1$

Умножение матриц $W_2 W_1$ дает матрицу размерности $n_2 \times n_0$, которую можно умножить на вектор X, размер которого $n_0 \times 1$. В итоге будет получен вектор, размерность которого $n_2 \times 1$.

Умножение вектора и столбца $W_2 b_1$ дает вектор, размерность которого $n_2 \times 1$.

Получаем три слагаемых, размеры которых совпадают.

Перейдем к следующему слою и подставим полученное выражение в уранение 3-го слоя.

$$Y_3 = W_3 \left(W_2 W_1 X + W_2 b_1 + b_2 \right) + b_3 = W_3 W_2 W_1 X + W_3 W_2 b_1 + W_3 b_2 + b_3$$

Теперь каждое из слагаемых, полученных на предыдущем шаге, умножается слева на матрицу W_3, размер которой $n_3 \times n_2$. После всех перемножений мы получим вектора-столбцы размерностью $n_3 \times 1$ и в конце еще слогаемое b_3 с такой же размерностью. Продолжая выполнять эти действия, мы будем каждый раз получать вектора-столбцы размерностью $n_i \times 1$, где i – номер слоя.

На $m-1$ слое на выходе будет вектор размерностью $n_{m-1} \times 1$. Рассмотрим уравнение для выходного слоя, в котором только 1 нейрон, следовательно, размерность W_m будет $n_m \times 1$, и размерность $b_n - 1 \times 1$.
Запишем уравнение.

$$y = W_m W_{m-1} \dots W_1 X + W_m \dots W_2 b_1 + W_m \dots W_3 b_2 + \dots + W_m b_{m-1} + b_m$$

Посчитаем все слагаемые перед X. Их рамерность в итоге будет $1 \times n_0$.

$$W^{¿} = W_m W_{m-1} \dots W_1$$

Посчитаем теперь веса, умноженные на смещения.

$W_m \dots W_2 b_1 = b_1^{¿}$ $W_m \dots W_3 b_2 = b_2^{¿}$
$W_m \dots W_4 b_3 = b_3^{¿}$

…

$W_m b_{m-1} = b'_{m-1}$

Все смещения имеют размерность 1×1

$b' = b'_1 + b'_2 + b'_3 + \ldots + b'_{m-1} + b_m$

В итоге мы получаем следующую формулу.

$y = W' X + b'$

Таким образом любую линейную нейронную сеть, состоящую из $m-1$ слоя в каждом из которых по n_i нейронов можно свести к однослойной сети, состоящей всего из одного нейрона.

3. Аппроксимация статических зависимостей

Задание

1. Инициализация.

Создайте НС ПР с 1 скрытым слоем и 20 скрытыми нейронами с помощью функции newff, feedforwardnet или fitnet. Инициализируйте НС с помощью функции init случайными значениями. Приведите график структуры НС с помощью команды view. В первом скрытом слое должна быть сигмоидальная передаточная функция, а в выходном – линейная.

2. Формирование обучающей и тестовой выборки.

В качестве аппроксимируемой функции используйте собственную функцию из заданий с No 5. Сформируйте обучающую и тестовую выборки. Объем обучающей выборки должен быть достаточным, а сама выборка – представительной.

Для проверки можно построить обучающую выборку и визуально проанализировать, достаточно ли точно по этой выборке можно воссоздать исходную зависимость.

Тестовая и обучающая выборка не должны содержать одинаковых элементов. Самый простой и надежный вариант для генерации входных значений – случайные значения, равномерно распределенные в диапазоне возможных значений аппроксимируемой функции. Т.е. если функция задана в диапазоне [-10, 10], можно сформировать $Ntrain$ и $Ntest$ случайных значений с равномерным распределением $R(-10,10)$.

!!! При недостаточном объеме обучающей выборки все дальнейшие исследования могут оказаться бесполезными, поэтому следует уделить этому процессу повышенное внимание.

3. Задание критериев останова обучения.

Научитесь задавать разными способами критерий останова обучения НС (с помощью полей структуры net.trainParam – time, goal, min_grad, max_fail, epochs). В дальнейшем необходимо будет подстраивать эти значения в зависимости от алгоритма, параметров обучения, а также параметров НС и требуемых показателей качества. Посмотрите, какая функция производительности (расчета ошибки) используется по умолчанию.

4. Обучение НС.

Задайте у НС $n\mathrm{h}idden$ скрытых нейронов и алгоритм обучения trainlm. Обучите НС на обучающей выборке. Проанализируйте результат обучения. Постройте зависимость ошибки аппроксимации на тестовой выборке от номера эпохи.

Постройте график аппроксимации функции, реализуемый НС. Для этого здесь и в дальнейшем всегда используйте данные тестовой выборки.

Зам. $n\mathrm{h}idden$ следует подобрать таким, чтобы после обучения графики аппроксимации и исходной функции визуально мало отличались.

3.1 Инициализация

Найдем оптимальные параметры обучения для алгоритма обучения ***trainlm***. Количество эпох оставим как есть – 1000 эпох. Параметр *max_fail* поставим 20. Для подбора параметров была написана программа *task3*. Количество нейронов в скрытом слое было выбрано 32, что соответсвует количеству бугорков на графике аппроксимируемой функции. Веса и смещения инициализируются случайными величинами. Функция производительности выбрана *sse*. По умолчанию использовалась функция *mse*, но она не подходит для нашего случая.

*Программа **task3***

```
net = feedforwardnet(32, 'trainlm');
net.inputs{1}.range = [0 1];
net.inputs{1}.size = 1;
net.layers{2}.size = 1;
net.layers{1}.initFcn = 'initwb';
net.layers{2}.initFcn = 'initwb';
net.layerWeights{1}.initFcn = 'rands';
net.layerWeights{2}.initFcn = 'rands';
net.biases{1}.initFcn = 'rands';
net.biases{2}.initFcn = 'rands';
net.inputWeights{1}.initFcn = 'rands';
net = init(net);
view(net)

Ntrain = 3000;
Ntest = 9000;
raw_train = rand(1, Ntrain);
raw_test = rand(1, Ntest);
raw_train = cat(2, -raw_train(1:Ntrain/2), raw_train(Ntrain/2 + 1:end));
raw_test = sort(cat(2, -raw_test(1:Ntest/2), raw_test(Ntest/2 + 1:end)));
Ttrain = my_fun(raw_train);
Ttest = my_fun(raw_test);

net.trainParam.goal = 0.9;
net.trainParam.max_fail = 50;
net.performFcn = 'sse';

net.trainParam.mu = 0.002;
net.trainParam.mu_dec = 0.5;
net.trainParam.mu_inc = 1.22;
net.trainParam.mu_max = 1e10;
net = train(net, raw_train, Ttrain);
y = sim(net, raw_test);

hold on
scatter(raw_test, y, 'r');
scatter(raw_test, Ttest, 'b');
legend(["NN", "Real"]);
hold off
```

```
fprintf('mae: %f \t mse : %f \nsae : %f \t sse : %f', ...
mae(y, Ttest), mse(y, Ttest), sae(y, Ttest), sse(y, Ttest));
```

Были подобраны следующие параметры обучения:

$mu = 0.002$
$mu_{dec} = 0.5$
$mu_{inc} = 1.22$
$mu_{max} = 1e10$

Изображение, полученное командой *view* показано на Рис 3.11.

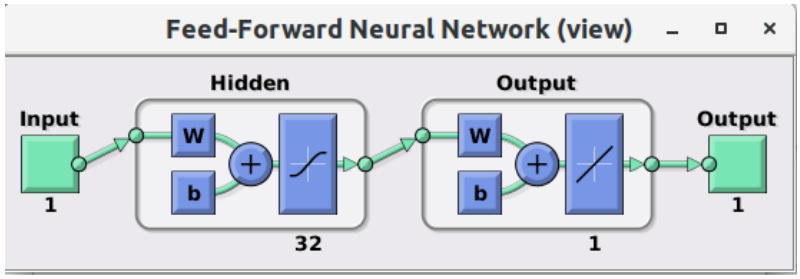


Рис 3.1.1. Структура нейронной сети.

График зависимости ошибки от номера эпохи представлен на Рис. 3.1.2.

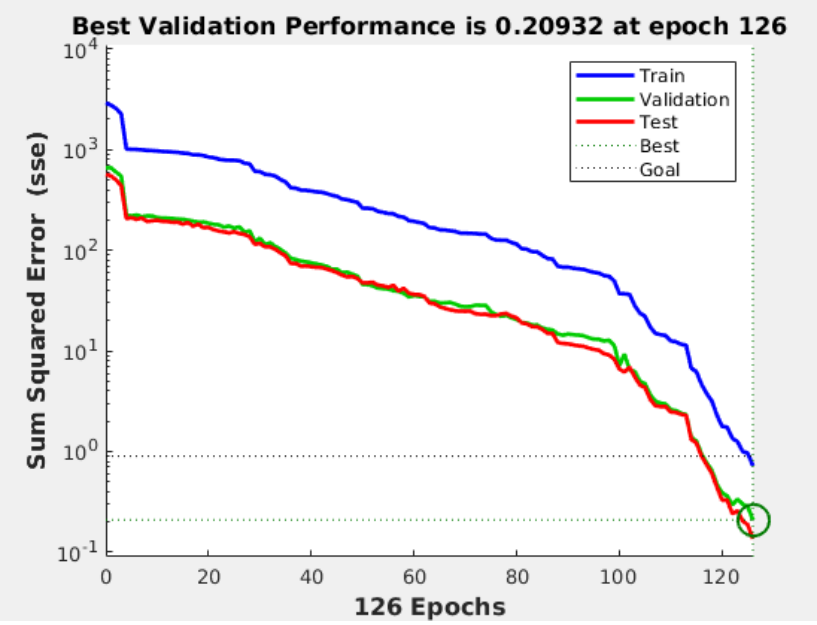


Рис 3.1.2. График зависимости ошибки от номера эпохи.

Полученная функция аппроксимации и реальная функция представлены на Рис. 3.1.3.

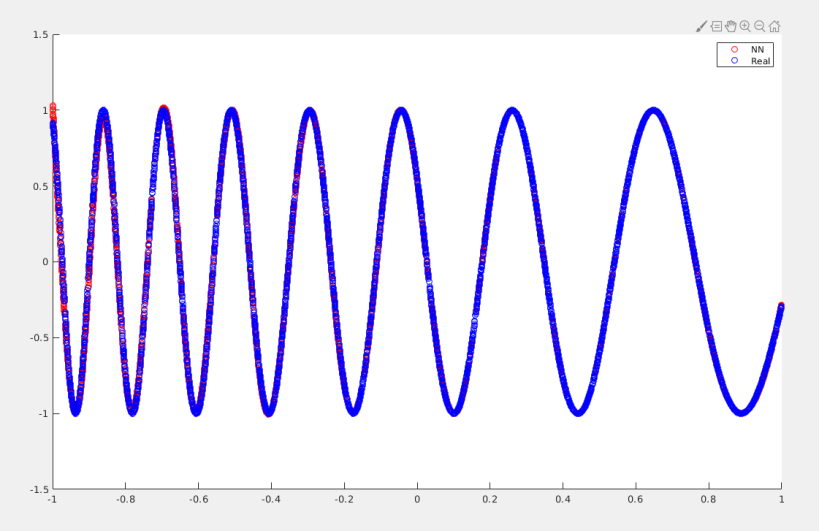


Рис 3.1.3. Реальная функция и ее аппроксимация.

3.2 Зависимость ошибки от числа нейронов в скрытом слое

Задание

5. Зависимость ошибки от числа нейронов в скрытом слое.

Изменяя число нейронов в скрытом слое в разумном диапазоне, рассчитайте зависимость ошибки (или функции производительности) как функцию от числа нейронов в скрытом слое. Определите наиболее подходящий диапазон значений числа скрытых нейронов. Постройте на одном графике аппроксимации для наилучших, удовлетворительных и наихудших случаев. На этом же графике отобразите исходную аппроксимируемую функцию.

Зам. Максимальное число нейронов в скрытом слое в эксперименте можно взять равным $k \cdot nbase$, где $nbase$ – минимальное число нейронов, при котором задача h$idden$ h$idden$ решается с приемлемой точностью, а k – некоторая константа (в диапазоне 5-10).

Зам. Поскольку при каждом новом запуске обучения выполняется случайная инициализация НС, для более точного вычисления ошибки следует проводить эксперименты многократно и затем усреднять полученные показатели. Например, для числа скрытых нейронов nh$idden$ следует провести хотя бы $nrun$ = 10 экспериментов, в каждом из них посчитать ошибку и затем вычислить среднее значение.

Программа *task3_trainlm* запускает обучение заданное количество раз и возвращает массив ошибок и количества эпох. Вторая программа – *task3_trainlm_plot* запускает первую с разным количеством нейронов, считает среднюю ошибку и рисует график зависимости ошибки от количества нейронов.

*Программа **task3_trainlm***

```
function errmatrix = task3_trainlm (nrun, num_of_neurons, perf)
res = zeros(4, nrun);
for i = 1:nrun
   net = feedforwardnet(num_of_neurons, 'trainlm');
```

```
    net.trainParam.showWindow = 0;
    net.inputs{1}.range = [0 1];
    net.inputs{1}.size = 1;
    net.layers{2}.size = 1;
    net.layers{1}.initFcn = 'initwb';
    net.layers{2}.initFcn = 'initwb';
    net.layerWeights{1}.initFcn = 'rands';
    net.layerWeights{2}.initFcn = 'rands';
    net.biases{1}.initFcn = 'rands';
    net.biases{2}.initFcn = 'rands';
    net.inputWeights{1}.initFcn = 'rands';
    net = init(net);

    Ntrain = 4000;
    Ntest = 4000;

    raw_train = rand(1, Ntrain);
    raw_test = rand(1, Ntest);
    raw_train = cat(2, -raw_train(1:Ntrain/2), raw_train(Ntrain/2 + 1:end));
    raw_test = sort(cat(2, -raw_test(1:Ntest/2), raw_test(Ntest/2 + 1:end)));
    Ttrain = my_fun(raw_train);
    Ttest = my_fun(raw_test);

    net.trainParam.max_fail = 20;
    net.performFcn = perf;
    net.trainParam.mu = 0.003;
    net.trainParam.mu_dec = 0.4;
    net.trainParam.mu_inc = 1.25;
    net.trainParam.mu_max = 1e10;
    [net, tr] = train(net, raw_train, Ttrain);

    y = sim(net, raw_test);
    stop_code = 0;
    if (strcmp(tr.stop, 'Validation stop.')); stop_code = 1;
    elseif (strcmp(tr.stop, 'Goal reached.')); stop_code = 2; end
    res(:, i) = [i tr.num_epochs sse(Ttest, y) stop_code]';
end
errmatrix = res;
```

Программа ***task3_trainlm_plot***

```
nmin = 2; ndelta = 5; nmax = 157; nrun = 10;
perf = 'sse';
means = zeros(1, (nmax - nmin) / ndelta + 1);
for i = nmin:ndelta:nmax
    x = task3_trainlm (nrun, i, perf);
    % mean of sse
    mean = sum(x(3,:))/nrun;
    means(fix((i - nmin) / ndelta) + 1) = mean;
end
plot(nmin:ndelta:nmax, means, 'r');
xlabel('neurons'); ylabel('perf(sse)');
```

Полученные результаты представлены на Табл 3.2.1 и Рис. 3.2.1.

№	Кол-во нейронов	Ошибка (sse)
1	25	4.36146408365598e-05
2	30	2.46091660425635e-06
3	35	1.52308888535509e-06
4	40	0.000261469206363395
5	45	5.93244194829806e-06

6	50	1.51140181100289e-06
7	55	7.70491793668664e-07
8	60	0.00164163424144235
9	65	1.05914002303103e-06
10	70	1.68600504250627e-06
11	75	2.56132518468363e-06
12	80	1.54481460806261e-06
13	85	2.38403303214801e-06

Таблица 3.2.1

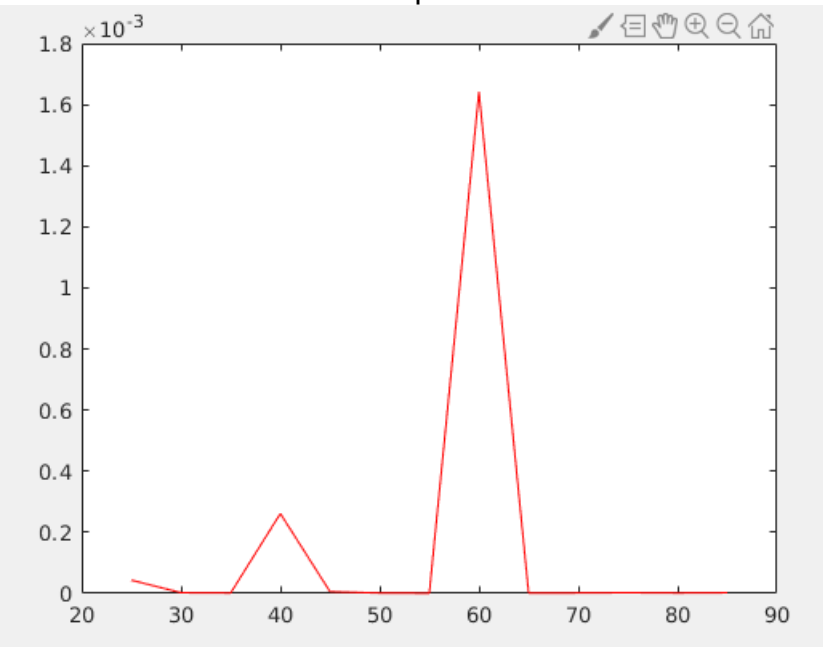


Рис 3.2.1. График зависимости ошибки от количества нейронов.

4. Классификация с помощью НСПР

Задание

Решите задачу классификации линейно неразделимых образов (из заданий с номером 3). Для этого используйте НС ПР с 1, 2 и 3 скрытыми слоями. Рассмотрите обычную и каскадную архитектуры. Выходной слой может быть сформирован двумя способами:

-с сигмоидальной передаточной функцией logsig, функция производительности mse, mae, sae, sse.

- с передаточной функцией softmax, функция производительности crossentropy.

Используйте в дальнейшем для обозначения сигнатуру при описании конкретной модели НС. Например, 2-logsig-K или 3-softmax обозначают соответственно каскадную НС с 2 скрытыми слоям с выходной ПФ logsig и НС с 3 скрытыми слоями с выходной ПФ

softmax. Для обозначения количества скрытых нейронов можно указывать их в скобках после числа скрытых слоев. Например, 2(15-5)-logsig-K или 3(15-5-3)-softmax.

Зам. Количество нейронов в выходном слое совпадает с количеством классов, т.е. равно двум. Количество входов совпадает с количеством признаков, т.е. тоже равно двум.

Напишем программу, которая будет создавать сеть необходимого типа, обучать ее и выводить результаты аппроксимации на тестовой выборке.

*Программа **task4***

```
layers = 1;
train_alg = 1;
arch_type = 1;
perf_fcn = 1;
to_fun = 1;
% Preparing
Ntrain = 10000;
Ntest = 4000;
raw_train = rand(2, Ntrain);
raw_test = rand(2, Ntest);
% is_in_area from lab0
Ttrain = is_in_area(raw_train');
Ttest = is_in_area(raw_test');

net = createnn(train_alg, layers, arch_type, to_fun);
net = setParams(net, train_alg, perf_fcn);

view(net)

t = [Ttrain; ~Ttrain];
net = train(net, raw_train, t);
y = sim(net, raw_test);
y2 = zeros(1, Ntest);

for i = 1:Ntest
  if (y(1,i) > y(2,i)); y2(i) = 1;
  else; y2(i) = 0; end
end
% plot2classe from lab0
plot2classes(raw_test', y2);
crossentropy([Ttest; ~Ttest], y)
```

```
function net = createnn(ftype, hidden_neurons, atype, tofun)
  tfuns = {'traingdx', 'traincgf', 'trainbfg', 'trainlm'};
  trans_out_funs = {'logsig', 'softmax'};
  if atype == 1
    net = feedforwardnet(hidden_neurons, tfuns{ftype});
  elseif atype == 2
    net = cascadeforwardnet(hidden_neurons, tfuns{ftype});
  end
  net.layers{length(hidden_neurons) + 1}.size = 2;
  net.layers{length(hidden_neurons) + 1}.transferFcn = trans_out_funs{tofun};
  net.numInputs = 2;
%     Create connection from each input
  net.inputConnect = [1 1; net.inputConnect(2:end, :)];
  net.inputs{1}.range = [-1 1];
  net.inputs{2}.range = [-1 1];
  net = init(net);
end

function neto = setParams(net, ftype, pfcn)
  pfcns = {'mse', 'mae', 'sse', 'sae', 'crossentropy'};
  net.performFcn = pfcns{pfcn};
  trainParam = net.trainParam;
  trainParam.epochs = 6000;
  trainParam.time = Inf;
  trainParam.goal = 0.005;
  trainParam.min_grad = 1e-05;
  trainParam.max_fail = 40;
  switch ftype
    case 1 %traingdx
      trainParam.lr = 0.01;
      trainParam.mc = 0.9;
      trainParam.lr_inc = 1.4;
      trainParam.lr_dec = 0.3;
      trainParam.max_perf_inc  = 1.04;
    case {2, 3} % traincgf, trainbfg
      if ftype == 2
        trainParam.searchFcn = 'srchcha';
      else
        trainParam.searchFcn = 'srchbac';
      end
      trainParam.scale_tol = 15;
      trainParam.alpha = 0.001;
      trainParam.beta = 0.1;
      trainParam.delta = 0.1;
      trainParam.gama = 0.001;
      trainParam.low_lim = 0.1;
      trainParam.up_lim = 0.5;
      trainParam.max_step = 100;
      trainParam.min_step = 1.0e-6;
      trainParam.bmax = 26;
      if ftype == 3
        trainParam.batch_frag = 10;
      end
    case 4 % trainlm
      trainParam.mu = 0.001;
      trainParam.mu_dec = 0.6;
      trainParam.mu_inc = 1.25;
      trainParam.mu_max = 1e10;
  end
  net.trainParam = trainParam;
  neto = net;
end
```

4.1 Модель с logsig-выходным слоем

Запустим нашу программу с параметром, задающим выходную функцию *logsig* и другими необходимымы параметрами. Результаты для различных алгоритмов обучения представлены в Табл. 4.1.1, визуализация аппроксимации на Рис. 4.1.1 – 4.1.3.

Таблица 4.1.1. Результаты для однослойной сети с logsig-выходным слоем

Функция	Нейроны	mse	time	epochs	perf	validation stop
trainlm	40	0.0175	0:00:12	137	0.00468	no
trainbfg	-	-	-	-	-	-
traingdx	80	0.0187	0:01:05	3052	0.0150	no
traincgf	60	0.00455	0:00:45	653	0.130	yes

Обучение с алгоритмом trainbfg не дало никаких результатов, как ни менялись параметры. Как видно, наиболее предпочтительным алгоритмом является *trainlm*.

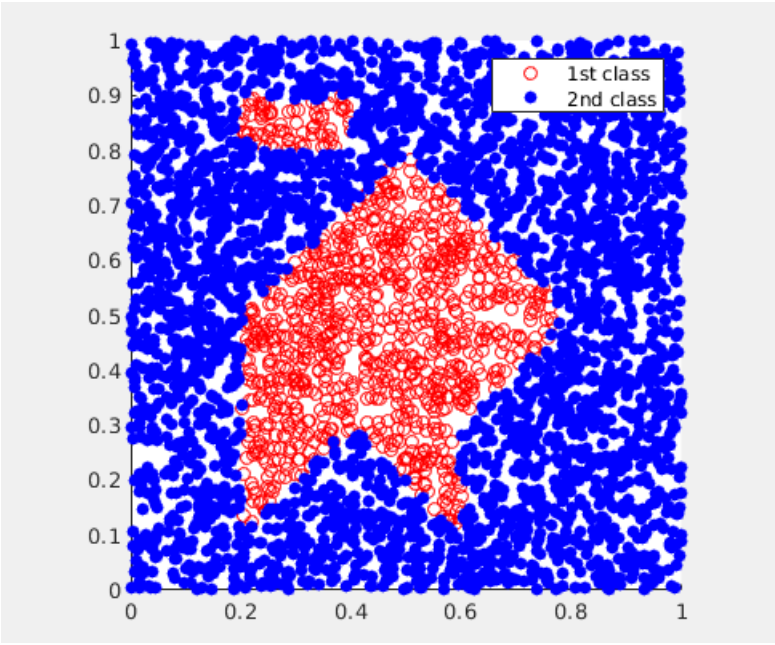


Рис. 4.1.1. Аппроксимация НС 1(40)-logsig-lm.

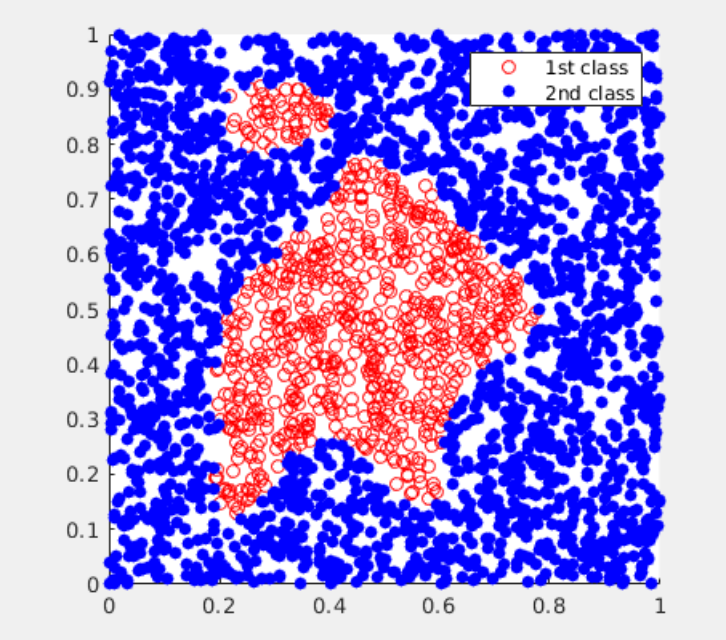


Рис. 4.1.2. Аппроксимация НС 1(60)-logsig-gdx.

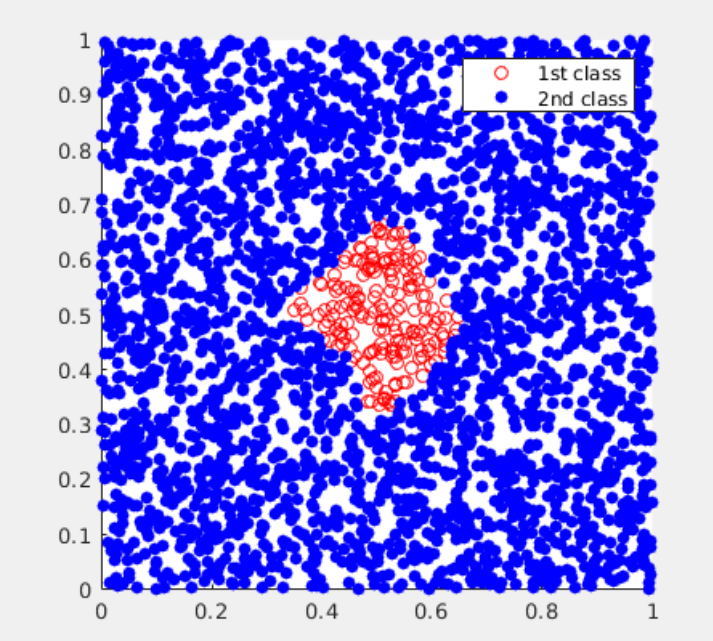


Рис. 4.1.3. Аппроксимация НС 1(60)-logsig-cgf.

Алгоритмы обучения *trainlm* и *traingdx* дают неплохие результаты, однако второй не всегда хорошо обучается.

4.2 Модель с softmax-выходным слоем

Вызовем нашу функцию, задав функцию выходного слоя softmax, функцию подсчета ошибки crossentropy. Алгоритм обучения по очереди задаем *trainbfg*, *traingdx* и *traincgf*. Также запустим с *trainlm*, однако у него функция ошибки будет *mse*.

Результаты обучения представлены в Табл. 4.2.1. Аппроксимация НС с различными алгоритмами обучения представлены на Рис. 4.2.1 – 4.2.4.

Таблица 4.2.1. Результаты для однослойной сети с softmax-выходным слоем

Функция	Нейроны	crossentropy	time	epochs	perf	validation stop
trainlm	40	0.0119	0:00:04	54	0.00792	no
trainbfg	40	0.00774	0:00:18	211	0.0150	no
traingdx	25	0.120	0:00:58	6000	0.0559	no
traincgf	40	0.00689	0:00:34	410	0.0150	no

НС с алгоритмом обучения *traingdx* не смогла корректно обучиться вне зависимости от попыток подобрать параметры. С функциями *softmax* и *crossentropy* снова лучше всего показал себя алгоритм *trainlm*.

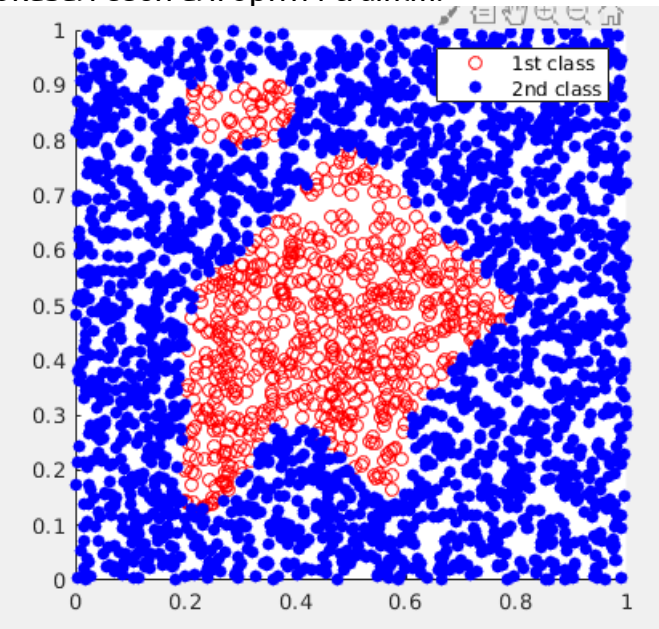


Рис. 4.2.1. Аппроксимация НС 1(40)-softmax-lm.

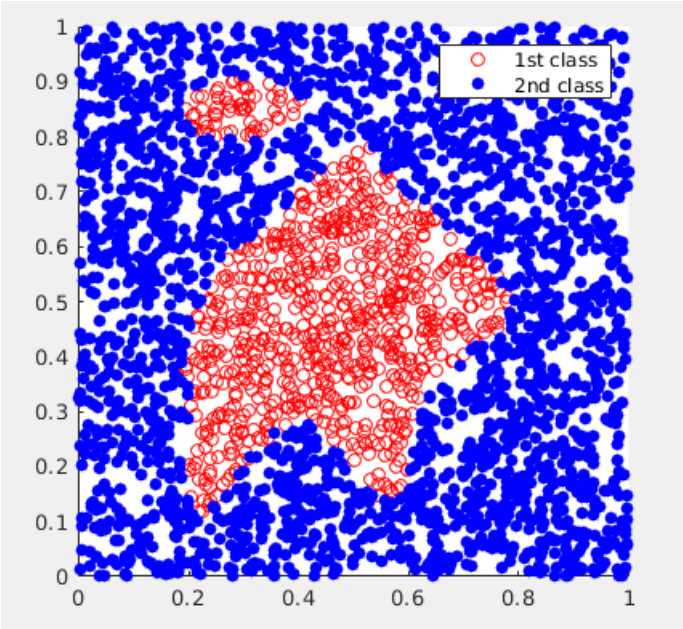


Рис. 4.2.2. Аппроксимация НС 1(40)-softmax-bfg.

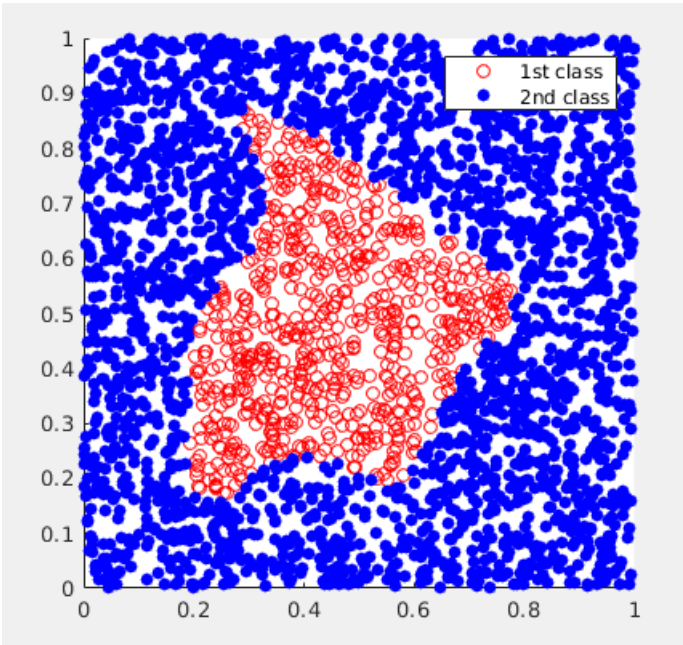


Рис. 4.2.3. Аппроксимация НС 1(40)-softmax-gdx.

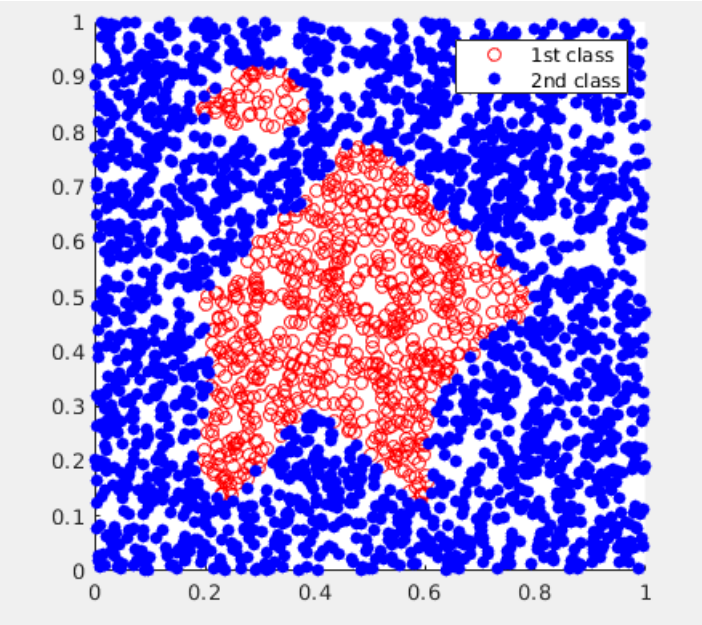


Рис. 4.2.4. Аппроксимация НС 1(40)-softmax-cgf.

И по визуализации аппроксимации видно, что наболее подходящим алгоритмом обучения для пары *softmax* и *crossentropy* является *trainlm*.

4.3 Каскадная НСПР

Протестируем сети с каскадной архитектурой двух типов: первая – сеть с *logsig*-выходным слоем, вторая – сеть с *softmax*-выходным слоем. Наилучшие полученные результаты представлены в Табл. 4.2.1. Результаты аппроксимации тестовой выборки данными НС представлены на Рис. 4.3.1 – 4.3.2. Для НС с *logsig*-выходным слоем функция производительности выбрана mse, для НС с *softmax*-выходным слоем – *crossentropy*. Функции обучения в обоих случаях -trainlm.

Таблица 4.2.1. Результаты для однослойной сети с logsig и softmax-выходными слоями

	Функция	Нейроны	time	epochs	perf	validation stop
logsig	trainlm	40	0:00:17	107	0.00798	no
softmax	trainlm	40	0:00:13	78	0.00788	no

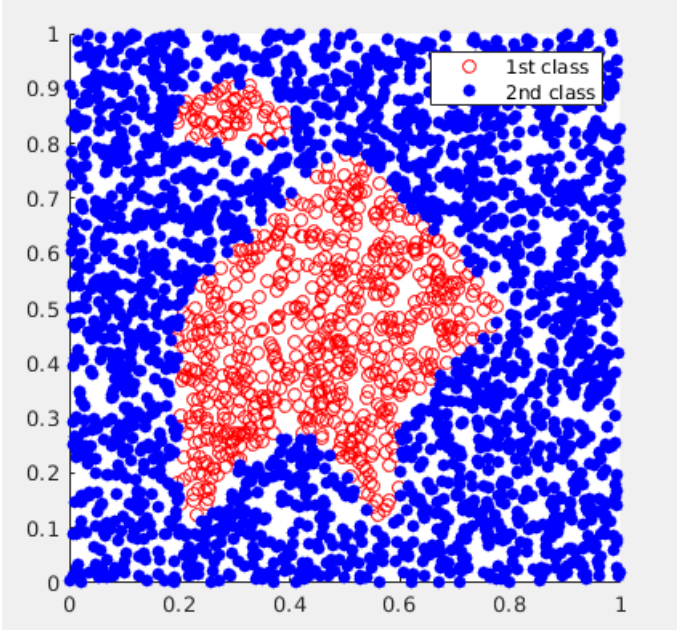


Рис. 4.3.1. Аппроксимация НС 1(40)-logsig-lm-K.

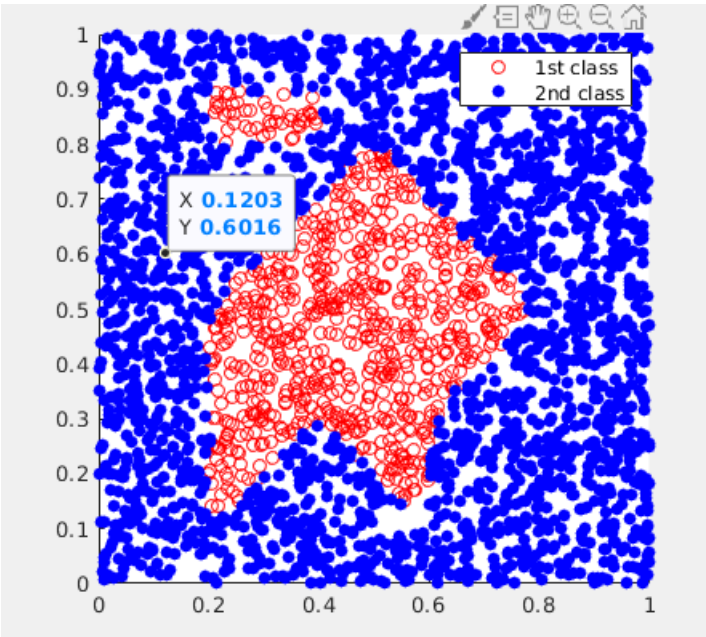


Рис. 4.3.2. Аппроксимация НС 1(40)-softmax-lm-K.

Как видим, более эффективной оказалась сеть с softmax-выходным слоем.

4.4 НСПР с 2 и 3 слоями

Сначала протестируем НСПР с 2 слоями. Результаты тестирования представлены в Табл. 4.4.1. Результаты аппроксимации тестовой выборки на Рис. 4.4.1 – 4.4.2. В обоих случаях в качестве алгоритма обучения был выбран trainlm.

Таблица 4.4.1. Результаты для моделей с logsig и softmax выходными слоями.

	Функция	Нейроны	time	epochs	perf	validation stop
logsig	trainlm	15 3	0:00:02	32	0.00788	no
softmax	trainlm	15 3	0:00:06	68	0.00786	no

Добавление второго слоя положительно сказалось на результатах. Однако здесь модель с softmax-выходным слоем показала себя хуже.

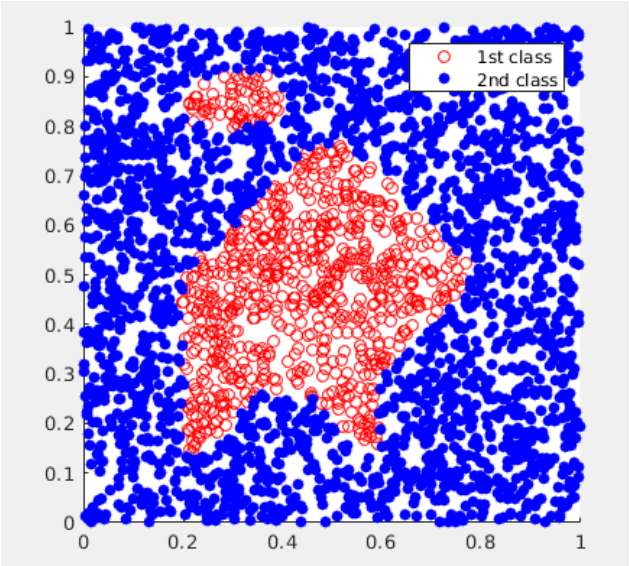


Рис. 4.4.1. Аппроксимация НС 2(15 3)-logsig-lm.

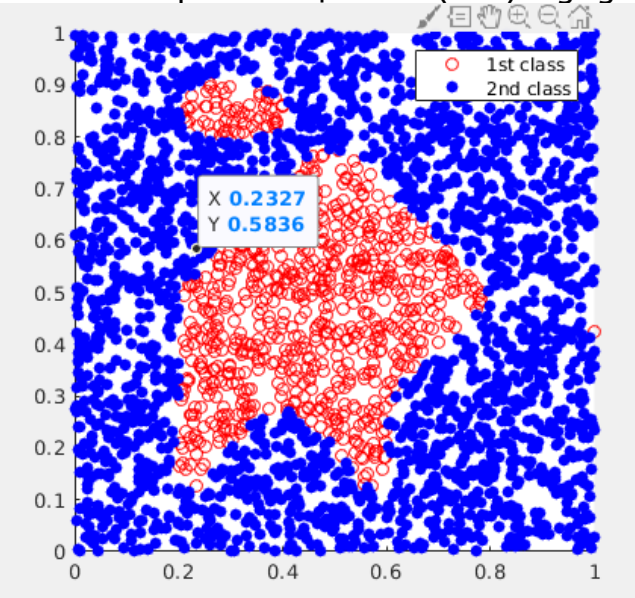


Рис. 4.4.2. Аппроксимация НС 2(15 3)-softmax-lm.

Теперь протестируем НСПР с 3 слоями. Результаты тестирования представлены в Табл. 4.4.2. Результаты аппроксимации тестовой выборки на Рис. 4.4.1 – 4.4.2. В обоих случаях в качестве алгоритма обучения был выбран trainlm.

Таблица 4.4.2. Результаты для моделей с logsig и softmax выходными слоями.

	Функция	Нейроны	time	epochs	perf	validation stop
logsig	trainlm	15 5 3	0:00:03	35	0.00733	no
softmax	trainlm	15 5 3	0:00:10	68	0.00779	no

Добавление третьего слоя не принесло видимых положительных результатов.

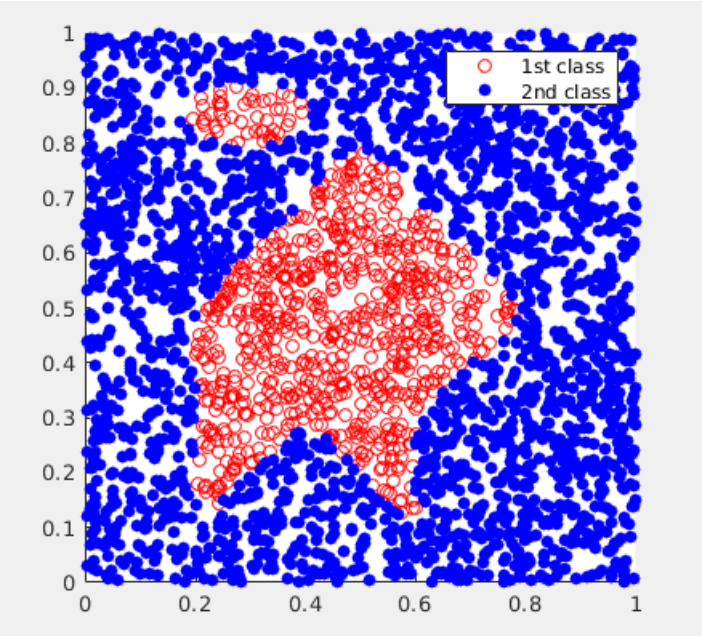


Рис. 4.4.1. Аппроксимация НС 3(15 5 3)-logsig-lm.

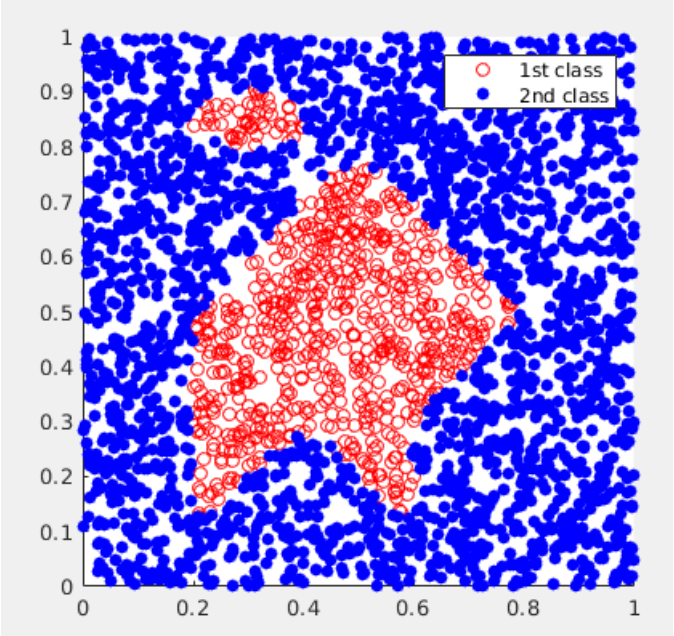


Рис. 4.4.2. Аппроксимация НС 3(15 5 3)-softmax-lm.

4.5 Сравнение результатов

Для сравнения все результаты из предыдущих пунктов представлены в Табл. 4.6.1.

Таблица 4.6.1. Сводная таблица результатов.

Однослойная НСПР							
	Функция	Нейроны	crossentropy	time	epochs	perf	validation stop
logsig	trainlm	40	0.0175	0:00:12	137	0.00468	no
	trainbfg	-	-	-	-	-	-
	traingdx	80	0.0187	0:01:05	3052	0.0150	no
	traincgf	60	0.00455	0:00:45	653	0.130	yes
softmax	trainlm	40	0.0119	0:00:04	54	0.00792	no
	trainbfg	40	0.00774	0:00:18	211	0.0150	no
	traingdx	25	0.120	0:00:58	6000	0.0559	no
	traincgf	40	0.00689	0:00:34	410	0.0150	no

Каскадная НСПР						
	Функция	Нейроны	time	epochs	perf	validation stop
logsig	trainlm	40	0:00:17	107	0.00798	no
softmax	trainlm	40	0:00:13	78	0.00788	no

Двухслойная НСПР						
	Функция	Нейроны	time	epochs	perf	validation stop
logsig	trainlm	15 3	0:00:02	32	0.00788	no
softmax	trainlm	15 3	0:00:06	68	0.00786	no

Трехслойная НСПР						
	Функция	Нейроны	time	epochs	perf	validation stop
logsig	trainlm	15 5 3	0:00:03	35	0.00733	no
softmax	trainlm	15 5 3	0:00:10	68	0.00779	no

Наилучшие результаты были получены при трехслойной архитектуре с *logsig* выходным слоем и алгоритмov обучения *trainlm.*

5. Классификация для случая нескольких классов

Задание

1. В качестве исходных данных используйте задания с номером 4. Сформируйте обучающую и тестовую выборки.

Зам. Количество нейронов в выходном слое совпадает с количеством классов. Количество входов совпадает с количеством признаков, т.е. равно двум.

2. Выполните исследование по аналогии с заданием 4.

Напишем программу, аналогичную программе *task4* , модифицированную для работы с 5 классами.

*Программа **task5***

```
function task5(layers, train_alg, arch_type, perf_fcn, to_fun)
  % Preparing
  Ntrain = 10000;
  Ntest = 1000;
  raw_train = rand(2, Ntrain);
  raw_test = rand(2, Ntest);
  Ttrain = is_in_area2(raw_train', 2);
  Ttest = is_in_area2(raw_test', 2);

  net = createnn(train_alg, layers, arch_type, to_fun);
  net = setParams(net, train_alg, perf_fcn);

  view(net)

  net = train(net, raw_train, Ttrain);
  y = sim(net, raw_test);
  y2 = zeros(1, Ntest);

  for i = 1:Ntest
    [~, ind] = max(y(:,i));
    y2(ind, i) = 1;
  end

  plot5classes(raw_test', y2);
  crossentropy(Ttest, y)

  function net = createnn(ftype, hidden_neurons, atype, tofun)
    tfuns = {'traingdx', 'traincgf', 'trainbfg', 'trainlm'};
    trans_out_funs = {'logsig', 'softmax'};

    if atype == 1
      net = feedforwardnet(hidden_neurons, tfuns{ftype});
    elseif atype == 2
      net = cascadeforwardnet(hidden_neurons, tfuns{ftype});
    end
    net.layers{length(hidden_neurons) + 1}.size = 7;
    net.layers{length(hidden_neurons) + 1}.transferFcn = trans_out_funs{tofun};
    net.numInputs = 2;
%     Create connection from each input
    net.inputConnect = [1 1; net.inputConnect(2:end, :)];
    net.inputs{1}.range = [0 1];
    net.inputs{2}.range = [0 1];
    net = init(net);
  end
```

```
  function neto = setParams(net, ftype, pfcn)
    pfcns = {'mse', 'mae', 'sse', 'sae', 'crossentropy'};
    net.performFcn = pfcns{pfcn};
    trainParam = net.trainParam;
    trainParam.epochs = 6000;
    trainParam.time = Inf;
    trainParam.goal = 0.001;
    trainParam.min_grad = 1e-05;
    trainParam.max_fail = 40;

    switch ftype
      case 1 %traingdx
        trainParam.lr = 0.01;
        trainParam.mc = 0.9;
        trainParam.lr_inc = 1.4;
        trainParam.lr_dec = 0.3;
        trainParam.max_perf_inc  = 1.04;
      case {2, 3} % traincgf, trainbfg
        if ftype == 2
          trainParam.searchFcn = 'srchcha';
        else
          trainParam.searchFcn = 'srchbac';
        end
        trainParam.scale_tol = 15;
        trainParam.alpha = 0.001;
        trainParam.beta = 0.1;
        trainParam.delta = 0.1;
        trainParam.gama = 0.001;
        trainParam.low_lim = 0.1;
        trainParam.up_lim = 0.5;
        trainParam.max_step = 100;
        trainParam.min_step = 1.0e-6;
        trainParam.bmax = 26;
        if ftype == 3
          trainParam.batch_frag = 10;
        end
      case 4 % trainlm
        trainParam.mu = 0.001;          % Initial mu
        trainParam.mu_dec = 0.6;        % mu decrease factor
        trainParam.mu_inc = 1.25;       % mu increase factor
        trainParam.mu_max = 1e10;       % Maximum mu
    end
    net.trainParam = trainParam;
    neto = net;
  end
end
```

Изображение с разделением классов показано на Рис. 5.1.

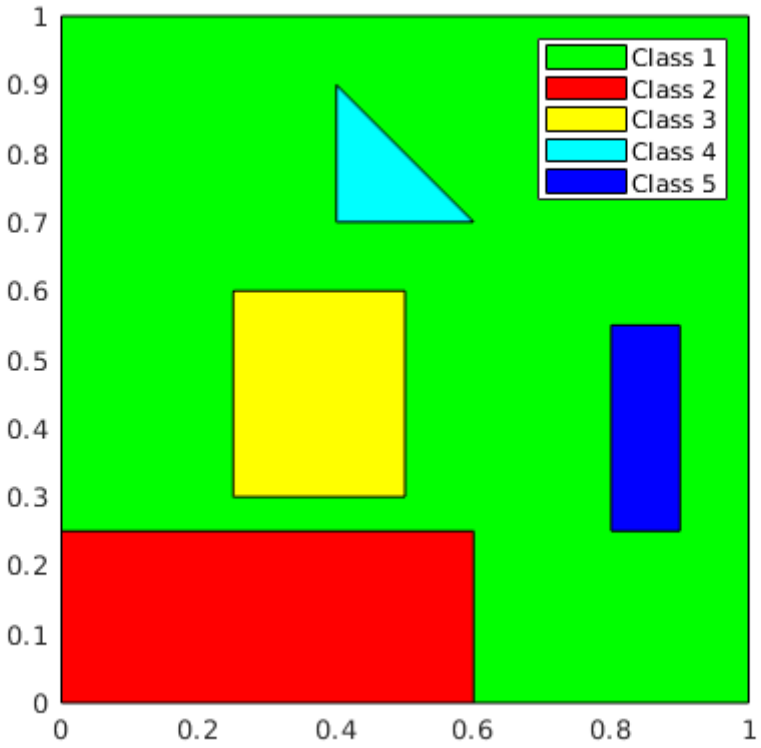


Рис. 5.1. Разделение плоскости на 5 классов.

5.1 Модель с logsig-выходным слоем

Протестируем модель с logsig-выходным слоем. Результаты тестирование представлены в Табл. 5.1.1. Аппроксимация тестовой НС с различными функкциями обучения представлены на Рис. 5.1.1 – 5.1.2.

Таблица 5.1.1. Результаты для НС с logsig-выходным слоем.

Функция	Нейроны	crossentropy	time	epochs	perf	validation stop
trainlm	20	0.0060	0:00:14	146	0.000982	no
traingdx	25	0.0132	0:00:38	3958	0.006050	yes

Лучше всего показала себя НС с функцией обучения *trainlm*. Также более или менее неплохой результат дала НС с функцией обучения *traincgf*. Остальные алгоритмы не удалось эффективно использовать для обучения НС.

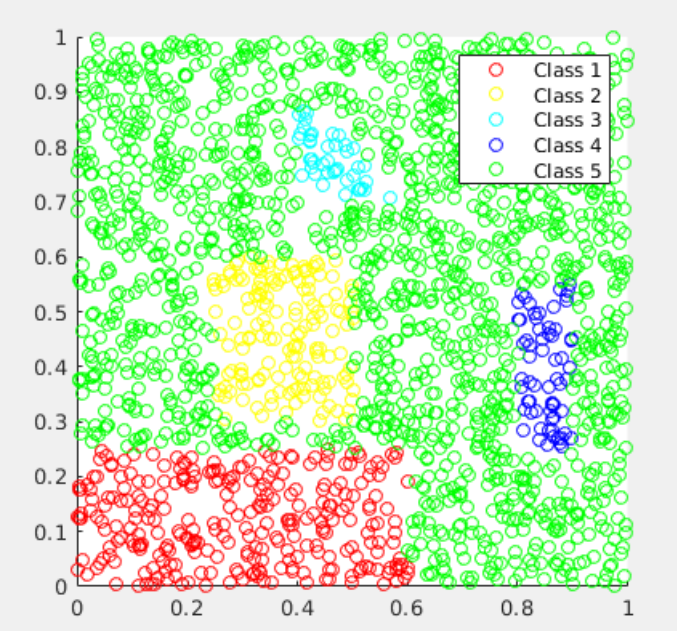


Рис. 5.1.1. Аппроксимация НС 1(20)-logsig-lm.

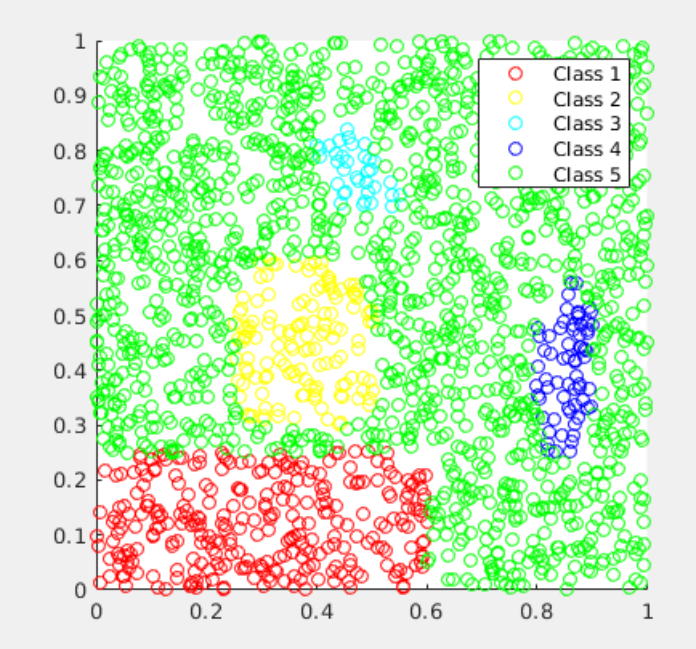


Рис. 5.1.2. Аппроксимация НС 1(25)-logsig-gdx.

По графикам тоже видно, что НС с функцией обучения trainlm аппроксимирует лучше.

5.2 Модель с softmax-выходным слоем

Протестируем НС с *softmax*-выходным слоем. Результаты тестирования представлены в Табл. 5.2.1. Аппроксимация НС с различными функциями обучения представлены на Рис. 5.2.1 – 5.2.2.

Таблица 5.2.1. Результаты тестирования НС с softmax-выходным слоем.

Функция	Нейроны	crossentropy	time	epochs	perf	validation stop
trainlm	20	0.0075	0:00:09	102	0.000178	yes
traingdx	28	0.0633	0:00:45	4315	0.009250	yes

Как и с НС с *logsig*-выходным слоем, лучший результат показала НС с функцией обучения *trainlm*, однако все другие НС, кроме traingdx, не смогли обучиться корректно.

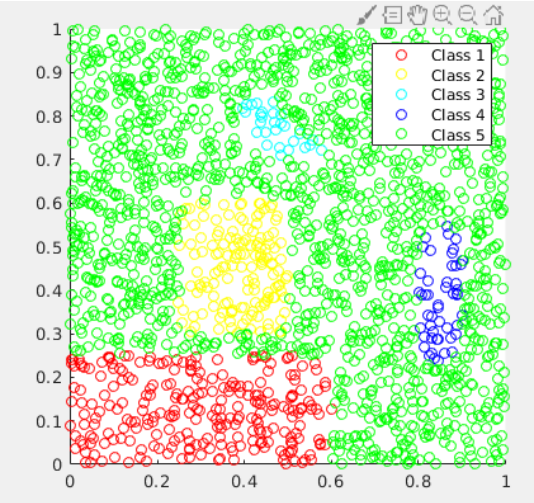


Рис. 5.2.1. Аппроксимация НС 1(20)-softmax-lm.

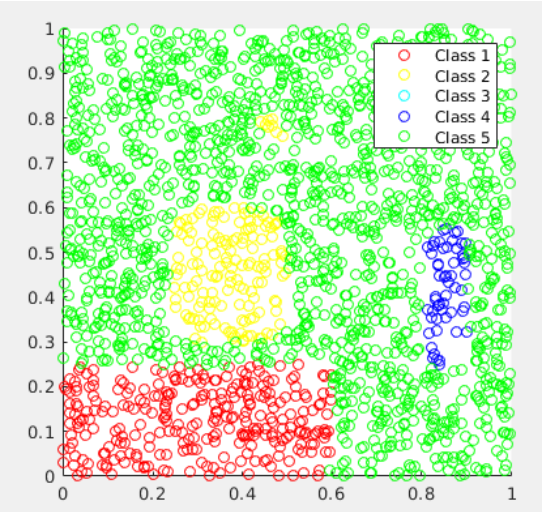


Рис. 5.2.2. Аппроксимация НС 1(28)-softmax-gdx.

5.3 Каскадная НСПР

Протестируем каскадную НС. Сначала с logsig-выходным слоем, затем с softmax-выходным слоем. Результаты тестирование представлены в Табл. 5.3.1. Аппроксимация различными сетями с разными функциями в выходном слое представлены на Рис. 5.3.1 – 5.3.8.

Табл. 5.3.1. Результаты тестирования различных сетей.

	Функция	Нейроны	crossentropy	time	epochs	perf	validation stop
logsig	trainlm	20	0.0078	0:00:04	52	0.000913	no
	traingdx	28	0.0080	0:01:04	6000	0.004840	no
softmax	trainlm	20	0.0065	0:00:04	50	0.000984	no
	traingdx	25	0.0060	0:01:00	6000	0.005660	no

Каскадная сеть показала неплохие результаты, все были хороши кроме, *trainbfg* и *traingdx* с *logsig*.

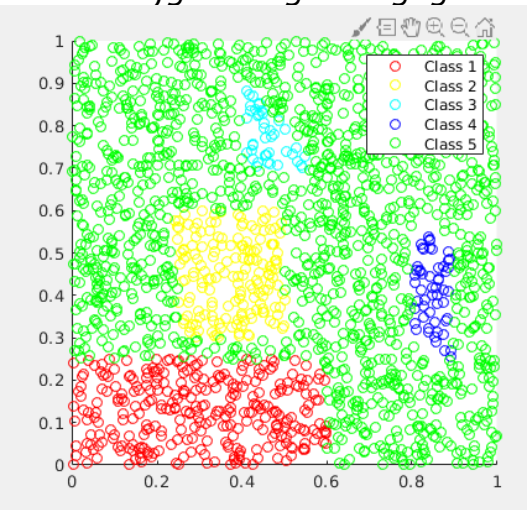


Рис. 5.3.1. Аппроксимация НС 1(20)-logsig-lm-K.

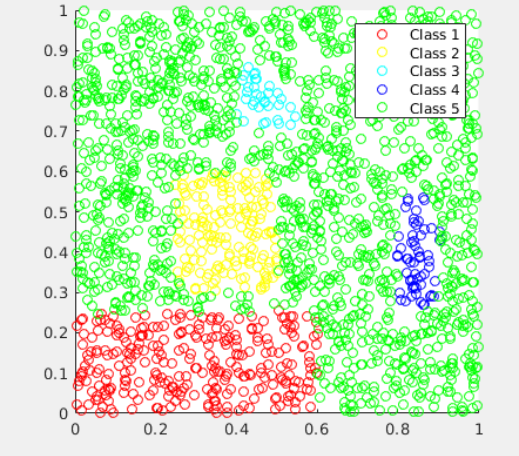


Рис. 5.3.2. Аппроксимация НС 1(28)-logsig-gdx-K.

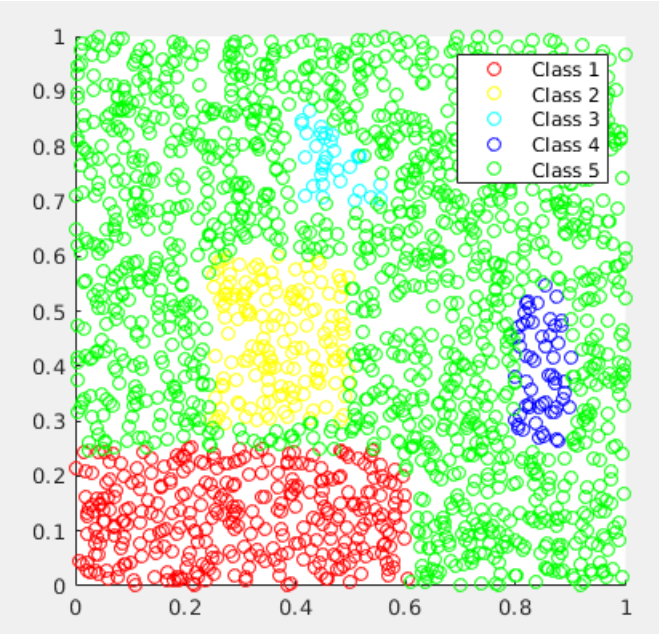


Рис. 5.3.1. Аппроксимация НС 1(20)-softmax-lm-K.

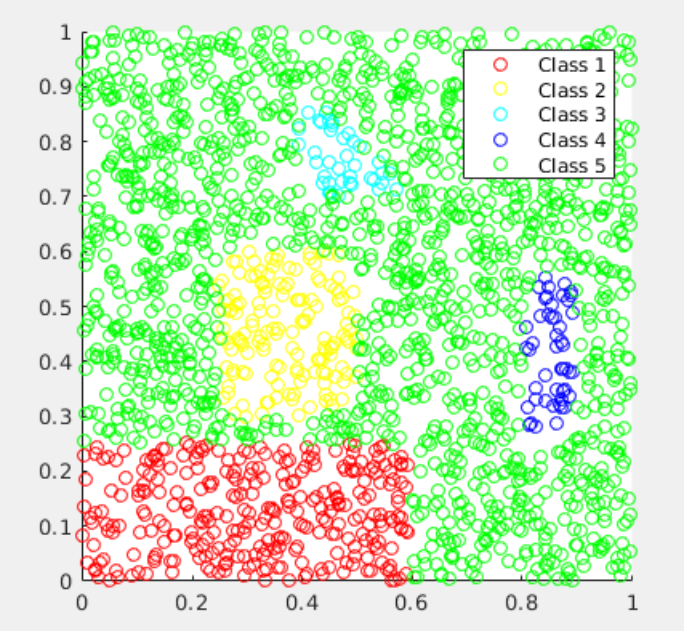


Рис. 5.3.2. Аппроксимация НС 1(25)-softmax-gdx-K.

5.4 НСПР с 2 и 3 слоями

Протестируем сеть с двумя слоями. В качестве алгоритма обучения будем использовать trainlm как самый эффективный, судя по предыдущим тестам. Результаты тестирования представлены в Табл. 5.4.1 и Табл. 5.4.2. Аппроксимации НС представлены на Рис. 5.4.1 – 5.4.4.

Табл. 5.4.1. Результаты тестирования двухслойной сети.

	Функция	Нейроны	crossentropy	time	epochs	perf	validation stop
logsig	trainlm	15 5	0.0081	0:00:03	36	0.000986	no
softmax	trainlm	15 5	0.0038	0:00:04	95	0.000972	no

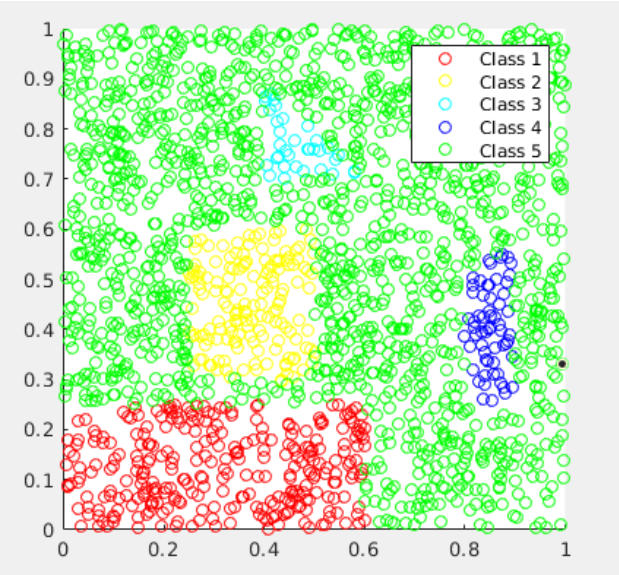


Рис. 5.4.1. Аппроксимация НС 2(15 5)-logsig-lm.

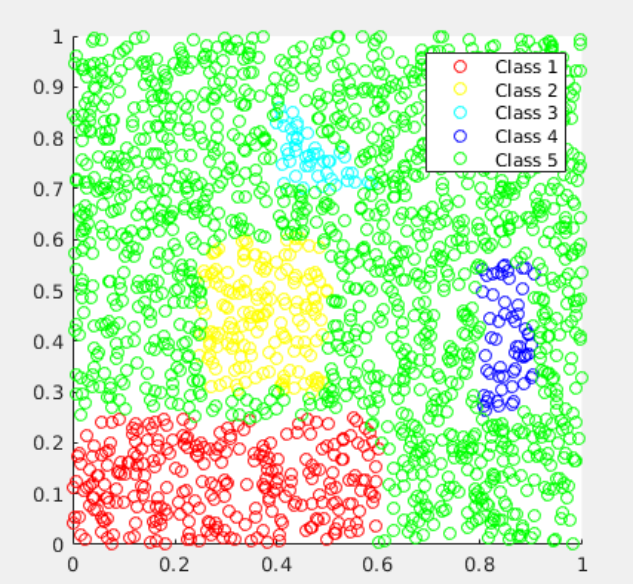


Рис. 5.4.2. Аппроксимация НС 2(15 5)-softmax-lm.

Табл. 5.4.2. Результаты тестирования трехслойной сети.

	Функция	Нейроны	crossentropy	time	epochs	perf	validation stop
logsig	trainlm	17 5 5	0.0035	0:00:06	52	0.000448	no
softmax	trainlm	17 5 5	0.0051	0:00:05	41	0.000780	no

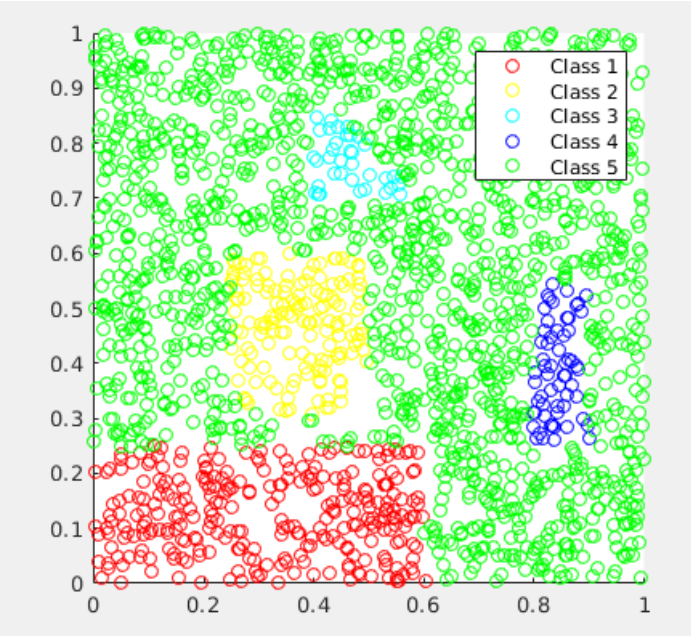


Рис. 5.4.3. Аппроксимация НС 3(17 5 5)-logsig-lm.

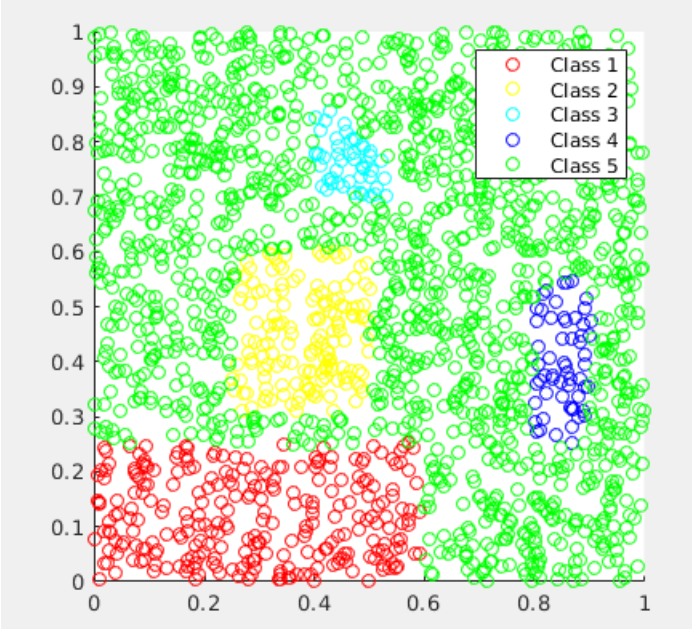


Рис. 5.4.4. Аппроксимация НС 3(17 5 5)-softmax-lm.

В случае с двух и трехслойными сетями и алгоритмом обучения trainlm модель с softmax-выходным слоем показывает себя наравне с logsig, а иногда даже выигрывает (НС 2(15 5)-softmax-lm).

5.5 Сравнение результатов

Для сравнения все результаты из предыдущих пунктов представлены в Табл. 5.5.1.

Табл 5.5.1

Вых. слой	Функция	Нейроны	crossentropy	time	epochs	perf	validation stop
Однослойная НС							
logsig	trainlm	20	0.0060	0:00:14	146	0.000982	no
	traingdx	25	0.0132	0:00:38	3958	0.006050	yes
softmax	trainlm	20	0.0075	0:00:09	102	0.000178	yes
	traingdx	28	0.0633	0:00:45	4315	0.009250	yes
Каскадная НС							
logsig	trainlm	20	0.0078	0:00:04	52	0.000913	no
	traingdx	28	0.0080	0:01:04	6000	0.004840	no
softmax	trainlm	20	0.0065	0:00:04	50	0.000984	no
	traingdx	25	0.0060	0:01:00	6000	0.005660	no
Двухслойная НС							
logsig	trainlm	15 5	0.0081	0:00:03	36	0.000986	no
softmax	trainlm	15 5	0.0038	0:00:04	95	0.000972	no
Трехслойная НС							
logsig	trainlm	17 5 5	0.0035	0:00:06	52	0.000448	no
softmax	trainlm	17 5 5	0.0051	0:00:05	41	0.000780	no

Из сравнительной таблицы видно, что лучше всего себя показали сети (в порядке убывания):

- НС 3(17 5 5)-logsig-lm
- НС 2(15 5)-softmax-lm
- НС 3(17 5 5)-softmax-lm
- НС 1(25)-softmax-gdx-K
- НС 1(20)-logsig-lm.