Neural Network Toolbox – пакет расширения MATLAB, содержащий средства для проектирования, разработки и визуализации нейронных сетей. Для создания и обучения нейронной сети в рабочей области MatLAB (рис. 1) необходимо выполнить ряд последовательных операций, позволяющих выбрать структуру нейронной сети, задать обучающую выборку, определить алгоритм обучения и обучить нейронную сеть.

1. Создание и инициализация сети

Формирование архитектуры сети

Первый шаг при работе с нейронными сетями - это создание модели сети. Для созда­ния сетей с прямой передачей сигнала в NNT предназначена функция newff.

Создадим сеть с прямого распространения, имеющую три нейрона на входе, один на выходе; 1 скрытый слой с 4-мя нейронами:

t = [0.3 0.5 ….. 0.9]; *// Выход*

a = [0.1 0.2 ….. 0.5];

b = [0.2 0.3 ….. 0.6];

c = [0.4 0.5 ….. 0.7];

p = [a; b; c]; *// Вход*

net = newff (minmax(p), [3,4,1], {'tansig', 'tansig', 'purelin'}, 'traingd');

Один вход/выход задается в виде вектора значений, несколько – в виде матрицы значений.

Функция newff имеет 4 входных аргумента и 1 выходной аргумент - объект класса network. Первый входной аргумент minmax(p) - это массив размера Rx2, содержащий допустимые границы значений (минимальное и максимальное) для каждого из R элементов вектора входа; второй [3,4,1] - мас­сив для задания количества нейронов каждого слоя; третий - массив ячеек, содержащий имена функций активации для каждого слоя {'tansig', 'tansig', 'purelin'}; четвертый 'traingd' - имя функции обучения.

В данном случае были использованы следующие функции активации:

**tansig(X)** - возвращает матрицу значений сигмоидальной (гиперболический тангенс) функции

**purelin(X)** – возвращает матрицу значений линейной функции активации.

Также в Neural Network Toolbox можно использовать следующие функции активации:

**compet(X)** –функция конкуренции - в качестве аргумента использует матрицу **X**, столбцы которой ассоциируются с векторами входов. Возвращает разреженную матрицу с единичными элементами, индексы которых соответствуют индексам наибольших элементов каждого столбца.

**hardlim(X)** – пороговая функция активации с порогом = 0; аргумент имеет тот же смысл, что и для предыдущей команды. Возвращает матрицу, размер которой равен размеру матрицы **X**, а элементы имеют значения 0 или 1 - в зависимости от знака соответствующего элемента **X**.

**hardlims(X)** – знаковая или сигнатурная функция активации; действует так же, как функция **hardlim(X)**, но возвращает значения -1 или +1

**logsig(X)** – сигмоидальная логистическая функция. Возвращает матрицу, элементы которой являются значениями логистической функции от аргументов - элементов матрицы **X**.

**poslin(X)** – возвращает матрицу значений полулинейной функции

**radbas(X)** – возвращает матрицу значений радиальной базисной функции

**satlin(X)** – возвращает матрицу значений полулинейной функции с насыщением.

**satlins(X)** – возвращает матрицу значений линейной функции с насыщением.

**softmax(X)** – возвращает матрицу, элементы которой вычисляются по формуле

, где N - число строк матрицы-аргумента **X**

**tribas(X)** – возвращает матрицу значений треугольной функции принадлежности.

В данном примере в качестве алгоритма обучения был использован алгоритм обратного распространения ошибки – **traingd**. Также в Neural Network Toolbox можно использовать следующие алгоритмы обучения:

**trainbfg** – функция обучения, реализующая разновидность квазиньютоновского алгоритма обратного распространения ошибки (BFGS).

**trainbr** – функция, реализующая так называемый Байесовский метод обучения, сущность которого заключается в подстройке весов и смещений сети на основе алгоритма Левенберга-Марквардта.

**traincgb** – функция обучения НС, реализующая разновидность алгоритма сопряженных градиентов (так называемый метод Powell-Beale).

**traincgf** – функция обучения НС, реализующая разновидность алгоритма обратного распространения ошибки в сочетании с методом оптимизации Флетчера-Поуэлла.

**traincgp –** то же, что в предыдущем случае, но с использованием метода Polak-Ribiere.

**traingd** – функция, реализующая "классический" алгоритм обратного распространения ошибки.

**traingda** – алгоритм обратного распространения ошибки с адаптацией коэффициента скорости обучения.

**traingdm** – функция, реализующая модифицированный алгоритм обратного распространения ошибки с введенной "инерционностью" коррекции весов и смещений.

**traingdx** – функция, реализующая комбинированный алгоритм обучения, объединяющий особенности двух вышеприведенных.

**trainlm** – даннаяфункция возвращает веса и смещения НС, используя алгоритм оптимизации Левенберга-Марквардта.

**trainoss**  – функция, реализующая разновидность алгоритма обратного распространения ошибки с использованием метода секущих.

**trainrp** – функция, реализующая разновидность алгоритма обратного распространения ошибки, так называемый упругий алгоритм обратного распространения (resilient backpropagation algorithm, RPROP).

**trainscg** – даннаяфункция возвращает веса и смещения НС, используя алгоритм масштабируемых сопряженных градиентов.

**trainwb** – данная функция корректирует веса и смещения сети в соответствии с заданной функцией обучения нейронов.

**trainwb1** – то же, что и предыдущая функция, но одновременно на вход сети предъявляется только один вектор входа.

М-функция newff не только создает архитектуру сети, но и инициализирует ее веса и смещения, подготавливая нейронную сеть к обучению. Однако существуют ситуации, когда требуется специальная процедура инициализации сети.

Инициализация сети

После того как сформирована архитектура сети, должны быть заданы начальные значения весов и смещений, или, иными словами, сеть должна быть инициализирована. Такая процедура выполняется с помощью метода init для объектов класса network. Опера­тор вызова этого метода имеет вид:

net = init(net).

1. Обучение нейронной сети

Задание параметров остановки обучения

Задание количества эпох обучения:

net.trainParam.epochs = 500;

*Задание среднеквадратичной ошибки обучения:*

net.trainParam.goal = 0.05;

*Обучение нейронной сети*

Заканчивается, если достигнута среднеквадратичная ошибка или по прошествии заданного количества эпох обучения.

net = train (net, p, t);

где

net – название сети;

p – вход сети;

t – выход сети.

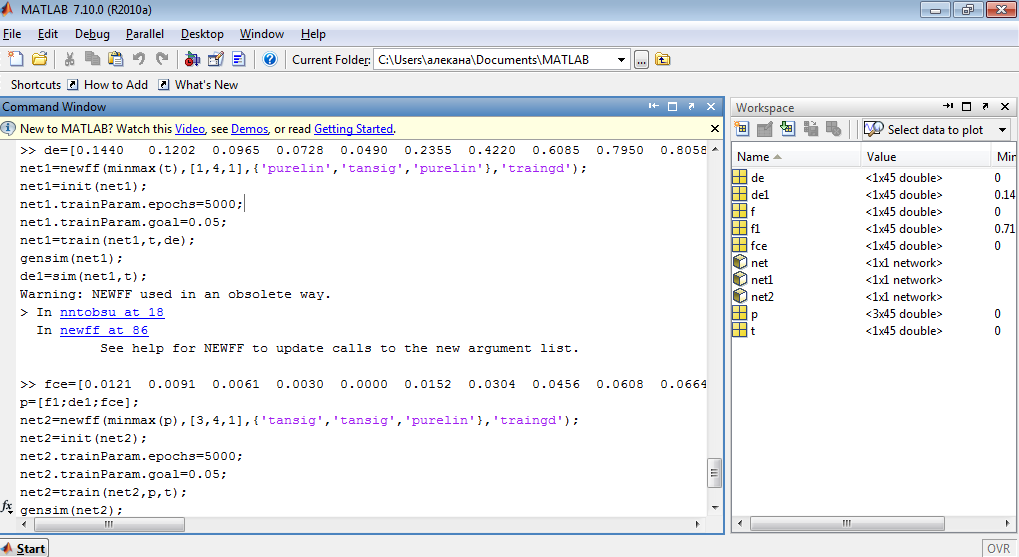


Рис. 1. Рабочее окно MatLab

Открывается окно, в котором виден процесс обучения нейронной сети (рис. 2):

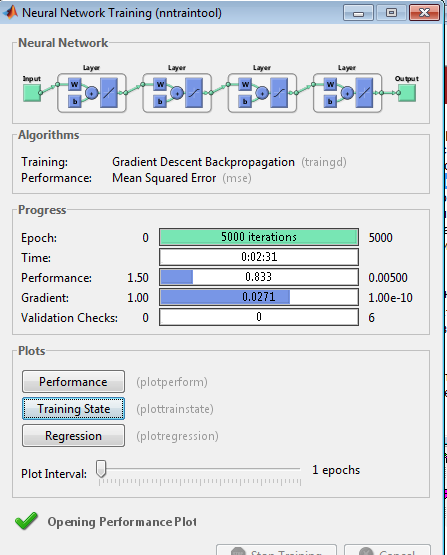


Рис. 2. Обучение нейронной сети

Где

Epoch – число итераций обучения;

Times – время обучения;

Performance – значение среднеквадратичной ошибки обучения;

Gradient – величина градиента;

Validation Checks – число проверок достоверности.

При нажатии на кнопку Performance открывается окно с графиком обучения нейронной сети (зависимость значения среднеквадратичной ошибки обучения от количества эпох) (рис. 3):

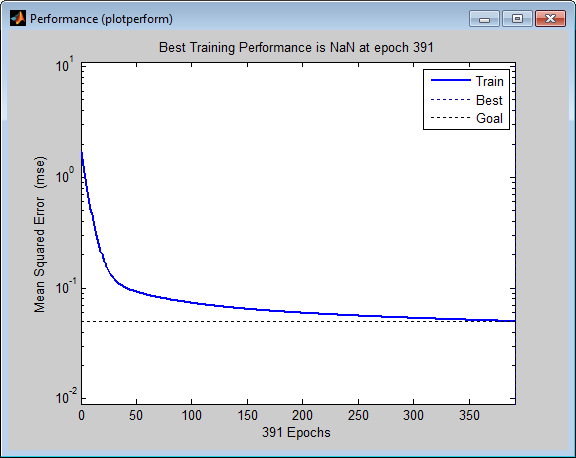


Рис. 3. График обучения нейронной сети

Моделирование сети

Для выводов результатов моделирования нейронной сети используют следующую функцию (рис. 4):

sim (net, p)

ans

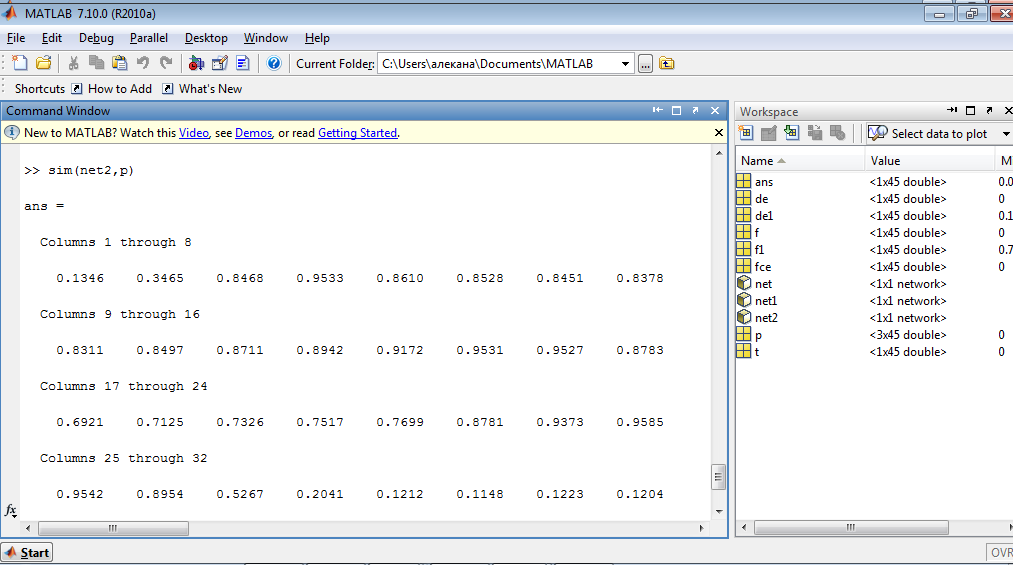


Рис. 4. Результаты моделирования сети

1. **Генерация НС в Simulink**

Для генерации нейросетевой модели в среде Simulink необходимо в рабочем окне MatLab вызвать следующую функцию:

gensim (net);

Это приведет к открытию блок-диаграммы на рис. 5.

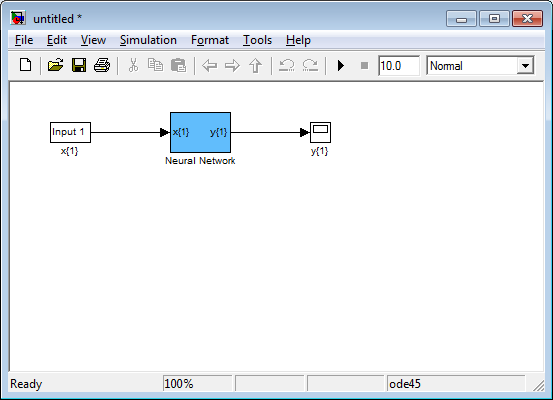


Рис. 5. Модель нейросетевого регулятора в Simulink

Чтобы посмотреть структуру нейронной сети, двойной щелчок по Neural Network (рис. 6).

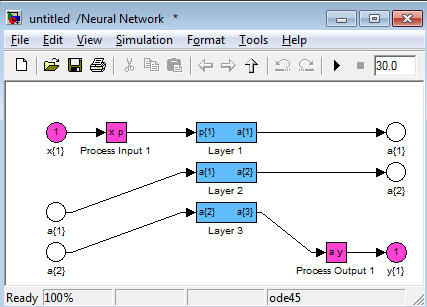


Рис. 6. Структура нейронной сети в Neural Network Toolbox

// Создание НС для моделирования регулятора, на выходе которого получаем значения изменения расхода пара на распыл сырья

t=[0.3262 0.4572 0.5882 0.7192 0.8502 0.8618 0.8733 0.8848 0.8963 0.8675 0.8387 0.8100 0.7812 0.7226 0.6640 0.6054 0.5468 0.5519 0.5571 0.5622 0.5673 0.6055 0.6437 0.6819 0.7202 0.6137 0.5073 0.4008 0.2944 0.2208 0.1472 0.0736 0.0000 0.2500 0.5000 0.7500 1.0000 0.8131 0.6263 0.4394 0.2525 0.3317 0.4109 0.4901 0.5692];

>> f=[0.8317 0.8502 0.8686 0.8871 0.9056 0.9292 0.9528 0.9764 1.0000 0.9359 0.8718 0.8078 0.7437 0.5578 0.3718 0.1859 0.0000 0.1411 0.2822 0.4233 0.5644 0.6441 0.7238 0.8035 0.8833 0.8894 0.8954 0.9015 0.9076 0.8619 0.8162 0.7705 0.7248 0.7325 0.7402 0.7479 0.7556 0.7355 0.7153 0.6952 0.6751 0.6895 0.7038 0.7182 0.7325];

>> net=newff(minmax(t),[1,4,1],{'purelin','tansig','purelin'},'traingd');

>> net=init(net);

>> net.trainParam.epochs=5000;

>> net.trainParam.goal=0.05;

>> net=train(net,t,f);

>> gensim(net);

>> f1=sim(net,t);

// Создание НС для моделирования регулятора, на выходе которого получаем значения изменения кратности циркуляции катализатора

de=[0.1440 0.1202 0.0965 0.0728 0.0490 0.2355 0.4220 0.6085 0.7950 0.8058 0.8166 0.8274 0.8382 0.8786 0.9191 0.9595 1.0000 0.9511 0.9023 0.8534 0.8045 0.8037 0.8028 0.8019 0.8010 0.7629 0.7248 0.6867 0.6487 0.4991 0.3495 0.1999 0.0503 0.1121 0.1740 0.2358 0.2976 0.2607 0.2238 0.1870 0.1501 0.1126 0.0750 0.0375 0.0000];

>> net1=newff(minmax(t),[1,4,1],{'purelin','tansig','purelin'},'traingd');

>> net1=init(net1);

>> net1.trainParam.epochs=5000;

>> net1.trainParam.goal=0.05;

>> net1=train(net1,t,de);

>> gensim(net1);

>> de1=sim(net1,t);

// Создание МНС прямого распространения для моделирования объекта управления

fce=[0.0121 0.0091 0.0061 0.0030 0.0000 0.0152 0.0304 0.0456 0.0608 0.0664 0.0720 0.0776 0.0831 0.0931 0.1031 0.1132 0.1232 0.1224 0.1216 0.1208 0.1200 0.1182 0.1163 0.1145 0.1126 0.1083 0.1040 0.0996 0.0953 0.2503 0.4052 0.5602 0.7151 0.7863 0.8575 0.9288 1.0000 0.9948 0.9896 0.9844 0.9792 0.9808 0.9824 0.9840 0.9856];

p=[f1;de1;fce];

net2=newff(minmax(p),[3,4,1],{'tansig','tansig','purelin'},'traingd');

net2=init(net2);

>> net2.trainParam.epochs=5000;

>> net2.trainParam.goal=0.05;

>> net2=train(net2,p,t);

>> gensim(net2);

// Создание НС Элмана для моделирования объекта управления

net3=newelm(minmax(p),[3,4,1],{'tansig','tansig','purelin'},'traingd');

net3=init(net3);

>> net3.trainParam.epochs=5000;

>> net3.trainParam.goal=0.05;

>> net3=train(net3,p,t);

>> gensim(net3);