Chapter 5

Building the Memory of the Neuro Immune Network

Researchers have only begun to understand the acquired, adaptive, immunity recently. Although the specificity of the acquired immunity makes it the core of more research than the innate immunity, it worth mentioning that both of them constitute the main arms of the immune system. By using the specificity feature, the immune system is able to distinguish non-self cells from self-cells, and can distinguish one foreign antigen from another. While a macrophage will engulf any foreign cell, acquired immune cells select selecting, only a precisely defined target. For this reason, the *acquired immune system* is also called the *specific immune system*.

Memory cells enable immunization to re-infection from the same pathogen. When the acquired immune cells encounter a particular organism, they persist and convey resistance to that organism, for an extended period of time. So, if the system has fought an infection once, it will rapidly be able to do so again [10].

The cells that are responsible for specificity are lymphocytes, of which there are two sub-populations: B-Cells and T-Cells. B-Cells are responsible for the rapid response to extra cellular and mucosal microorganisms. They produce soluble factors known as Antibodies against these microorganisms. T-Cells have two roles: the first role as coordinator of the acquired immune responses, which is accomplished by their production of a wide variety of cytokines and surface cell signals. The second is as the Primary Responder, to long-term Intracellular infections. Simply T-cells help both arms of the immune system: The cell mediated response, and the humoral response [8].

Our main focus in this chapter is to build an Artificial Immune Network (AIM) that is capable of mimicking the role of the biological memory. The AIM design was simply introduced in chapter three. In this chapter we will discuss design and simulation details of the AIM and evaluate its performance as memory subsystem in the overall immune system [44], [45], [46], and [47].

1. Using Artificial Neural Networks To Solve The IKS Problem

In robot position control, mapping from Cartesian space to joint space is a fundamental problem. This inverse kinematics problem is nonlinear and the solution is not easy to find or does not even always exist in closed form. Recalling the previously discussed advantages of artificial neural networks, and with the aid of literature review, we found that ANNs can be used to solve the inverse kinematics problem. Multi-layer networks are capable of approximating any function with a finite number of discontinuities. In order to learn the inverse kinematics the neural network needs information about coordinates, joint angles, and actuator positions.

Figure5.1 illustrates the structure of the designed AIM network. As the figure shows, it is a mixture of expert network, which means that the output decision will be a collective output resulting from several expert ANNS. Originally, the set of inputs fed to the network were the initial position of the end-effector of the robot arm, and the target position. We modified this input to serve the generality feature of the overall network, since it is somehow difficult to have a training set that covers most of the manipulator workspace.

The new set of inputs that we use is the set of delta differences between the indices of the initial position of the end-effector, and the target position. As a matter of fact, you can take advantage of this feature by choosing a part of the workspace, build the training set of various examples, and then chose an offset to generate new training examples. So, the first problem is to build a training set with examples that represent most of the working space of the arm manipulator, then use the offset to generate more training examples if necessary.

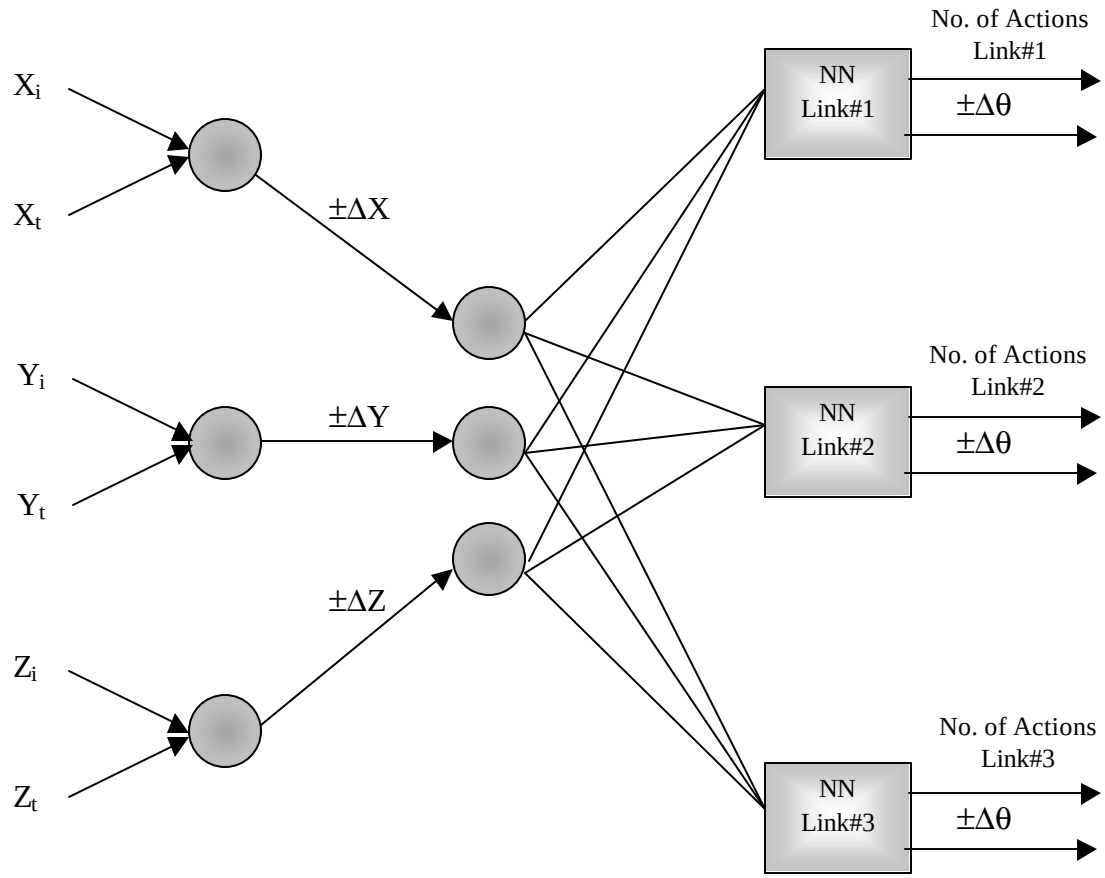


Figure5.1 The AIM Structure

2. Generating Training Examples

The training file consists of several training examples. Each example consists of three delta inputs differences paired with two desired outputs, which are the desired angle theta and the desired number of actions for the robot arm (CW or CCW). To get the desired number of actions for these inputs, we must feed the AIN with the target position as well as the joints' angles. The output after the AIN learning will be a set of actions needs to be processed later and involved in the training process.

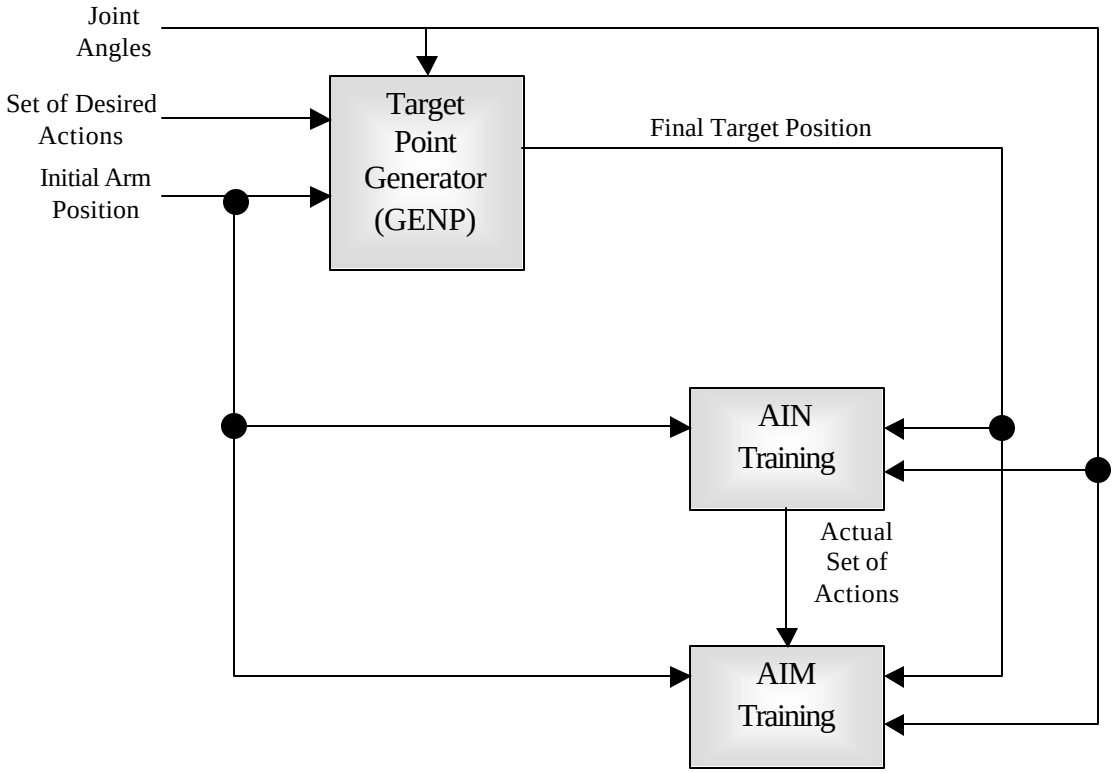


Figure5.2 Target Point Generation

To get an output from AIN, we need to feed it with the target position. We have to be sure that this target position is situated within the working space of the robot arm. Giving the AIN an unreachable point will result in false learning, and error in the training set. We solved this problem by writing a Matlab code that takes an initial position of the robot arm, and a set of actions, with a specific joint angle, as in input.

The output of this program is the final position of the target; see Figure5.2. This program has the following advantages:

1-The desired output of AIN is specifically defined.

2-The target point is guaranteed to fall within the robot arm working space.

3. Preparing Data for the Training File

The first step is to get the output set of actions resulted from AIN simulation, and prepare it to be processed, and passed to the AIM network. Some examples will be shown to clarify the generation of data to the training file. The simulation result will be shown in a table formatted in a way that facilitates the result-analysis, see Figure5.3.

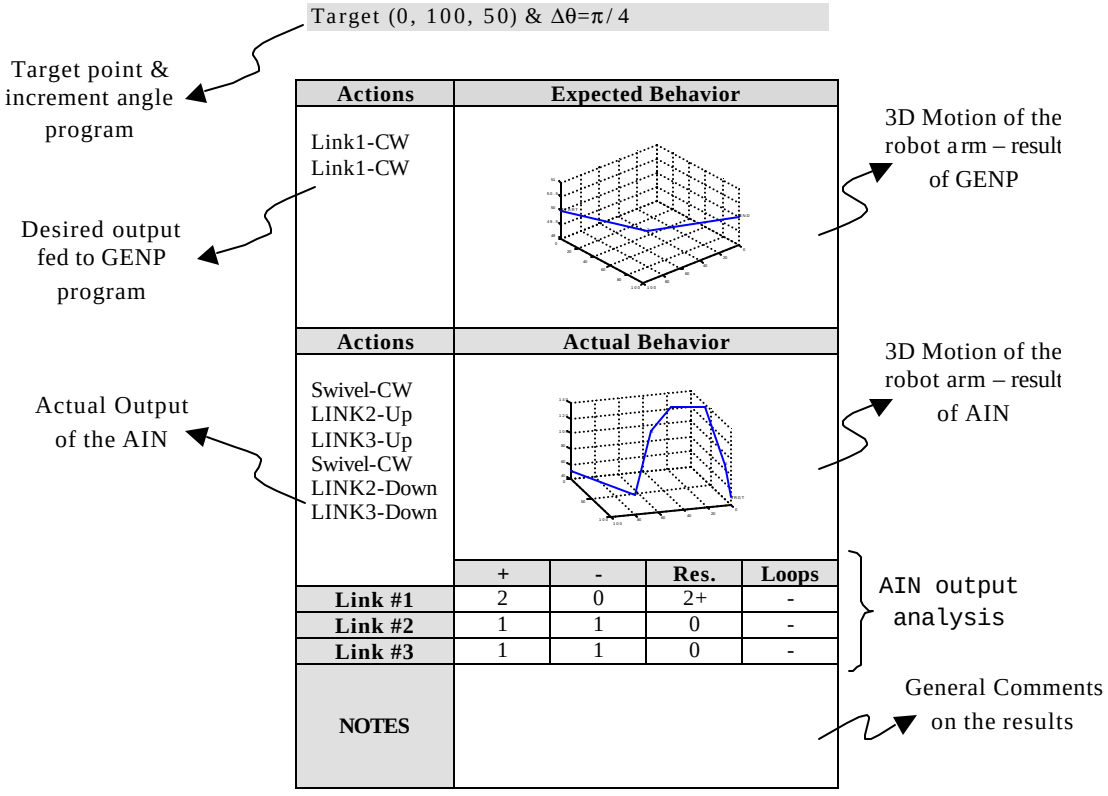
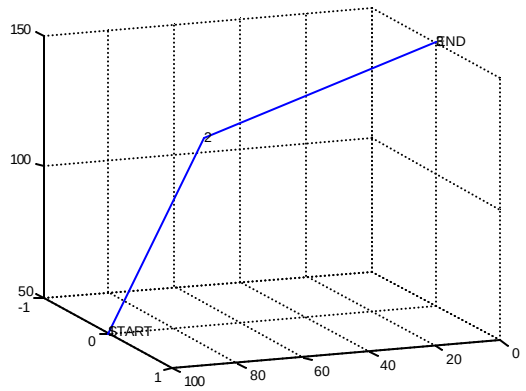
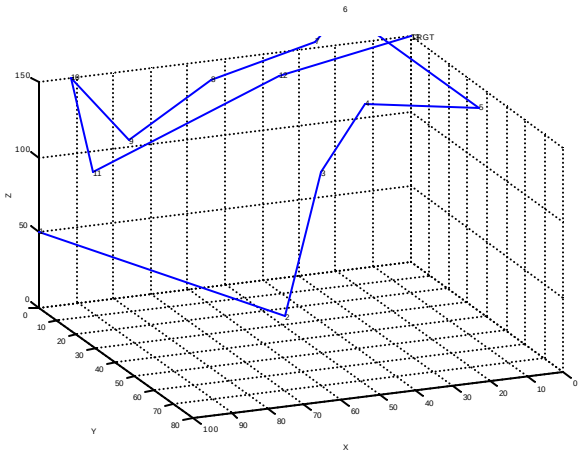


Figure5.3. Result-Table Contents

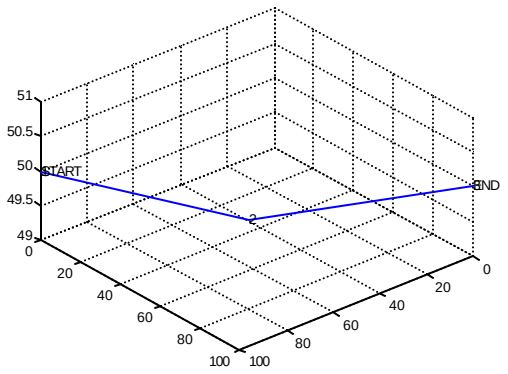
Target (0, 0, 150) & $\Delta\theta=\pi/4$

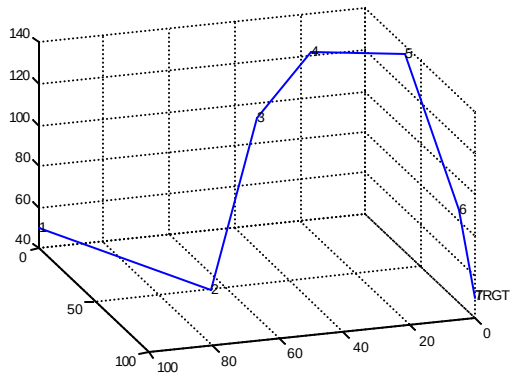
Actions	Expected Behavior
Link2-UP Link2-UP	
Actions	**Actual Behavior**
Swivel-CW LINK2-Up LINK3-Up Swivel-CW LINK2-Up LINK3-Up Swivel-CW LINK2-Up LINK3-Down Swivel-CW LINK2-Down LINK3-Down	

	+	-	Res.	Loops
Link #1	4	-	4+	-
Link #2	3	1	2+	-
Link #3	2	2	0	-
NOTES	Going 4x Swivel-CW is a waste of time, but it did not affect reaching the desired target position.			

Table5.1 Reaching a Target at (0, 0, 150)

Target (0, 100, 50) & $\Delta\theta=\pi/4$

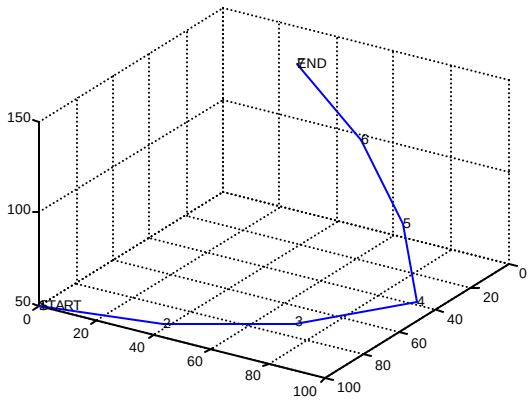
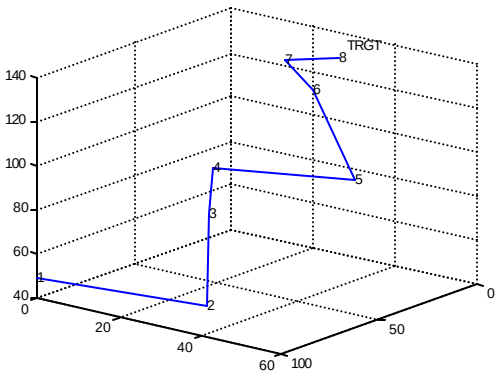
Actions	Expected Behavior
Link1-CW Link1-CW	

Actions	Actual Behavior
Swivel-CW LINK2-Up LINK3-Up Swivel-CW LINK2-Down LINK3-Down	

	+	-	Res.	Loops
Link #1	2	1	2+	-
Link #2	1	1	0	-
Link #3	1	1	0	-
NOTES	The resultant actions of the AIN are equivalent to the desired Actions made by GENP (Ideal Case).			

Table5.2 Reaching a Target at (0, 100, 50)

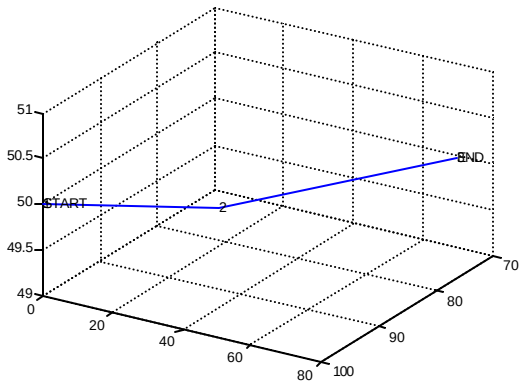
Target (14.6447 35.3553 142.3880) & $\Delta\theta=\pi/8$

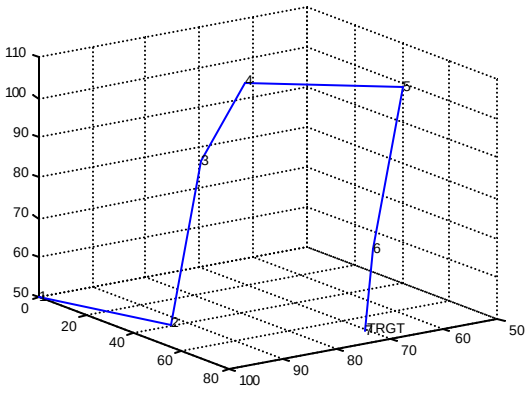
Actions	Expected Behavior
Swivel-CW Swivel-CW Swivel-CW LINK2-Up LINK2-Up LINK2-Up	
Actions	**Actual Behavior**
Swivel-CW LINK2-Up LINK3-Up Swivel-CW LINK2-Up LINK3-Up Swivel-CW LINK2-Up	

	+	-	Res.	Loops
Link #1	3	-	3+	-
Link #2	3	-	3+	-
Link #3	2	-	2+	-
NOTES	This case is different from reaching the target at (0, 0, 150), because we need at least one Swivel-CW to reach (x=14.64).			

Table5.3 Reaching a Target at (14.64, 35.35, 142.38)

Target (70.7107 70.7107 50.0000) & $\Delta\theta=\pi/8$

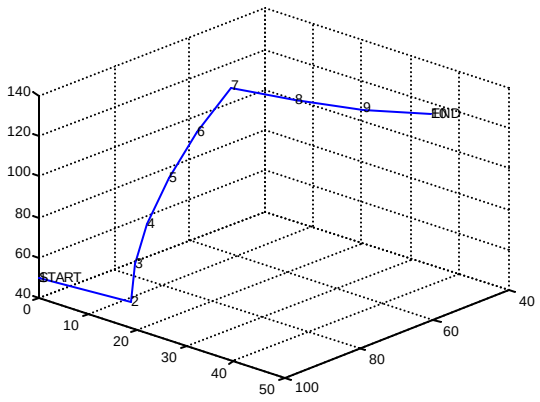
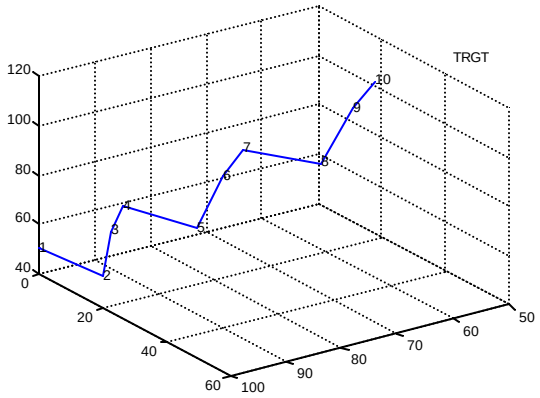
Actions	Expected Behavior
Swivel-CW Swivel-CW	

Actions	Actual Behavior
Swivel-CW LINK2-Up LINK3-Up Swivel-CW LINK2-Down LINK3-Down	

	+	-	Res.	Loops
Link #1	2	-	2+	-
Link #2	1	1	-	-
Link #3	1	1	-	-
NOTES	The resultant actions of the AIN are equivalent to the desired Actions made by GENP (Ideal Case).			

Table5.4 Reaching a Target at (70.71, 70.71, 50.00)

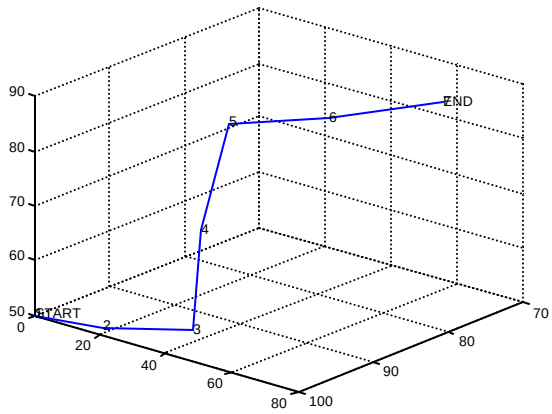
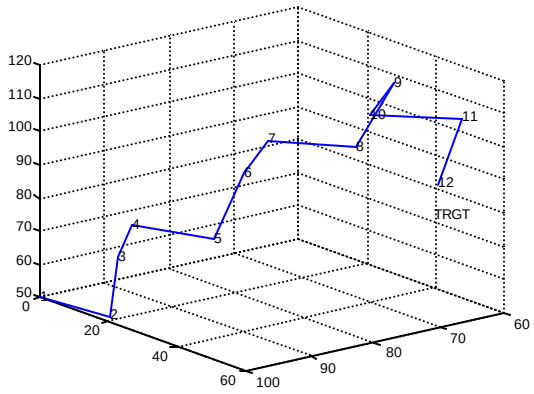
Target (49.2404 41.3176 126.6044) & $\Delta\theta=\pi/18$

Actions	Expected Behavior
Swivel-CW LINK2-Up LINK2-Up LINK2-Up LINK2-Up LINK2-Up Swivel-CW Swivel-CW Swivel-CW	
Actions	**Actual Behavior**
Swivel-CW LINK2-Up LINK3-Up Swivel-CW LINK2-Up LINK3-Up Swivel-CW LINK2-Up LINK3-Up	

	+	-	Res.	Loops
Link #1	3	-	3+	-
Link #2	3	-	3+	-
Link #3	3	-	3+	-
NOTES	The AIN produced 2 more extra movements to approach as close as possible to the target position.			

Table5.5 Reaching a Target at (49.24, 41.31,126.60)

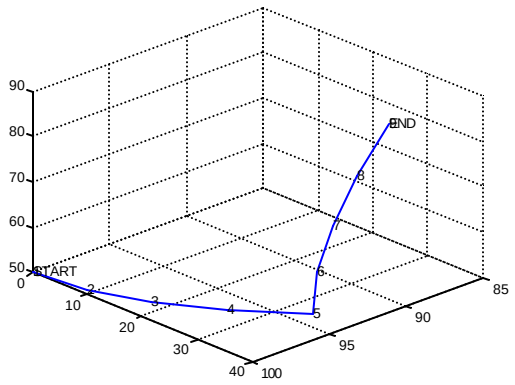
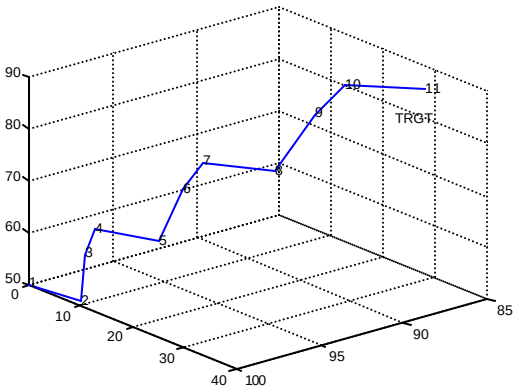
Target (71.9846 60.4023 84.2020) & Δθ=π/18

Actions	Expected Behavior
Swivel-CW Swivel-CW LINK2-Up LINK2-Up Swivel-CW Swivel-CW	
Actions	**Actual Behavior**
Swivel-CW LINK2-Up LINK3-Up Swivel-CW LINK2-Up LINK3-Up Swivel-CW LINK2-Up LINK3-Down Swivel-CW LINK2-Down	

	+	-	Res.	Loops
Link #1	4	-	4+	-
Link #2	3	1	2+	-
Link #3	2	2	0	-
NOTES	The resultant actions of the AIN are equivalent to the desired Actions made by GENP (Ideal Case).			

Table5.6 Reaching a Target at (71.98, 60.40, 84.20)

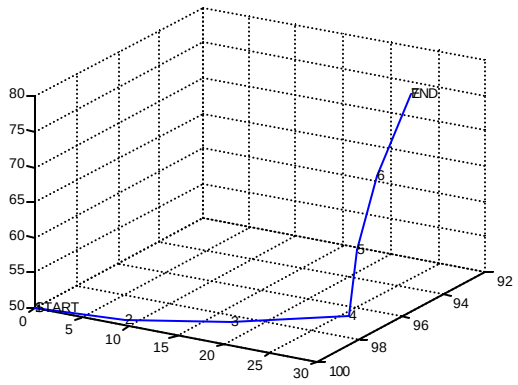
Target (88.3022 32.1394 84.2020) & $\Delta\theta=\pi/36$

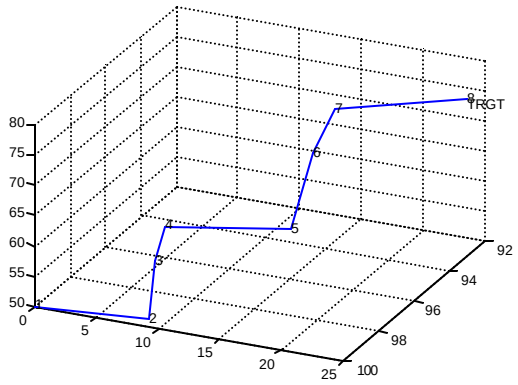
Actions	Expected Behavior
Swivel-CW Swivel-CW Swivel-CW Swivel-CW LINK2-Up LINK2-Up LINK2-Up LINK2-Up	
Actions	**Actual Behavior**
Swivel-CW LINK2-Up LINK3-Up Swivel-CW LINK2-Up LINK3-Up Swivel-CW LINK2-Up LINK3-Up Swivel-CW	

	+	-	Res.	Loops
Link #1	4	-	4+	-
Link #2	3	-	3+	-
Link #3	3	-	3+	-
NOTES	The AIN produced 2 more extra movements to approach as close as possible to the target position.			

Table5.7 Reaching a Target at (88.30, 32.13, 84.2020)

Target (93.3013 25.0000 75.8819) & $\Delta\theta=\pi/36$

Actions	Expected Behavior
Swivel-CW Swivel-CW Swivel-CW LINK2-Up LINK2-Up LINK2-Up	

Actions	Actual Behavior
Swivel-CW LINK2-Up LINK3-Up Swivel-CW LINK2-Up LINK3-Up Swivel-CW	

	+	-	Res.	Loops
Link #1	3	-	3+	-
Link #2	2	-	2+	-
Link #3	2	-	2+	-
NOTES	The AIN produced one more extra movement to approach as close as possible to the target position.			

Table5.8 Reaching a Target at (93.30, 25.00, 75.88)

All the previous tables are part of the training file, which contains more training examples. We notice on the "*AIN analysis part of the table* " that we compute the resultant actions for each link. This resultant set of actions represents the way by which the AIN learns to reach its target, or in another word how B-cells regulate Ab performance to defeat the Ag. Loop analysis is another method, which is used to analyze the output of the AIN. By loop analysis we mean that the occurrence of a loop in any action should be eliminated. For example, if link number one moves 8 movements with step $\pi/4$ while learning, this means that it returned to the initial starting position. The resultant action method, and loop analysis are just some ways to prepare the training file for the ANN in a way that insures consistent learning.

4. Training The Net

Recalling Figure5.1, we will focus on the ANN training part. First, we will train NN#1, which is responsible for link#1 actions; see the network structure at Figure5.4. Backpropagation NN is used to represent NN#1, and we discussed its advantage previously in chapter one.

The structure of NN#1 is as follows:

1- Three inputs: $\pm\Delta X$, $\pm\Delta Y$, and $\pm\Delta Z$.
2- One hidden layer with three neurons.
3- Output layer with 2 neurons: number of actions, and the angle $\Delta\theta$.
4- The input training file has the previously three mentioned inputs paired with the desired output of the number of actions, and the value of $\Delta\theta$.
5- The transfer function of the first hidden layer is the *tan-sigmoid* transfer function, and the transfer function for the output layer is a *linear* transfer function.

Used functions:

1- AdaptFcn: 'adaptwb'
2- InitFcn: 'initlay'
3- PerformFcn: 'mse'

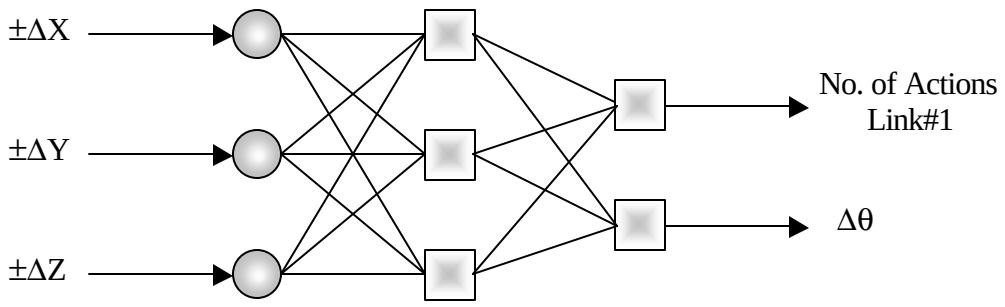


Figure5.4 NN#1 Structure

There are different backpropagation training algorithms differ on the way they treat the training date. We will list some of them, and also display the results of using them as training algorithms for our AIM. It worth noting that normalizing the data has a great effect on the performance of the network. We trained NN#1 with the same previously mentioned network configuration, and the performance was very low before normalizing the data, and preprocessing it, see Figure5.5.

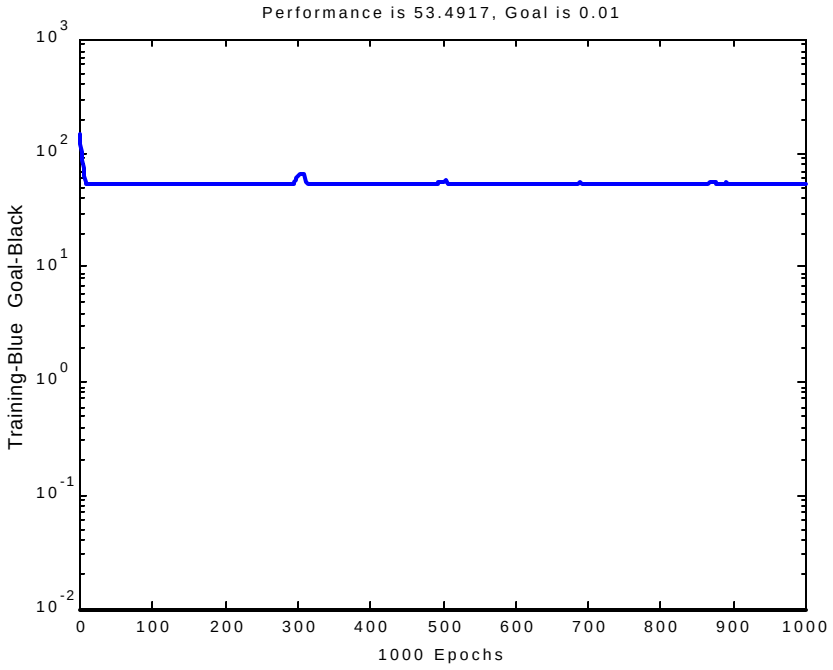


Figure5.5 NN#1 Performance Using *traingdx* Training Algorithm

Matlab program output:

```
» aim1
TRAINGDX, Epoch 0/300, MSE 173.274/0.01, Gradient 831.224/1e-006
TRAINGDX, Epoch 50/300, MSE 53.5015/0.01, Gradient 0.280493/1e-006
TRAINGDX, Epoch 100/300, MSE 53.4919/0.01, Gradient 0.0348032/1e-006
TRAINGDX, Epoch 150/300, MSE 53.4917/0.01, Gradient 0.00191441/1e-006
TRAINGDX, Epoch 200/300, MSE 53.4917/0.01, Gradient 0.00021685/1e-006
TRAINGDX, Epoch 250/300, MSE 57.3346/0.01, Gradient 5.54464/1e-006
TRAINGDX, Epoch 300/300, MSE 53.4924/0.01, Gradient 0.0759401/1e-006
TRAINGDX, Maximum epoch reached, performance goal was not met.
```

We decided to perform pre-processing analysis steps for the training data before feeding it into the ANN. One of the effective pre-processing algorithms is the Min-Max algorithm in which inputs and targets are scaled so that they always fall within a specified range. We tried training the AIM same data file after pre-processing it, and the error level decreased, see Figure5.6. We have done some regression analysis between the network response and the corresponding targets, see Fig5.7 (a, b).

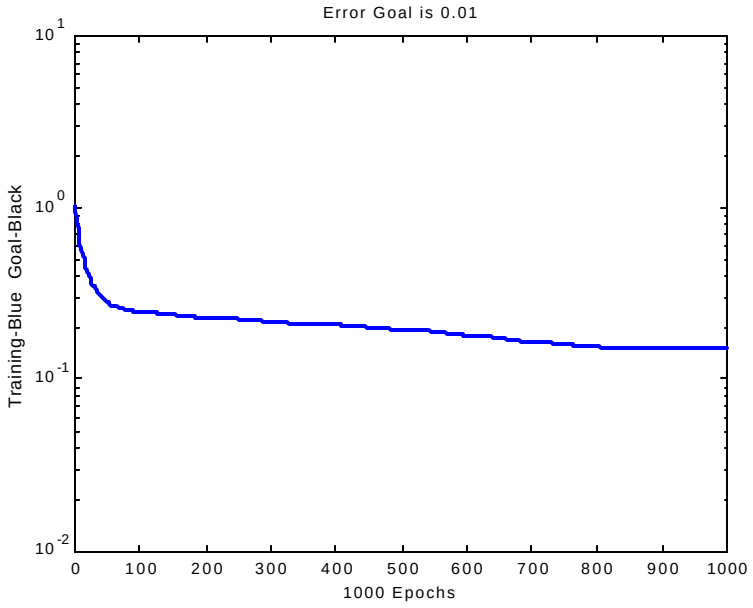
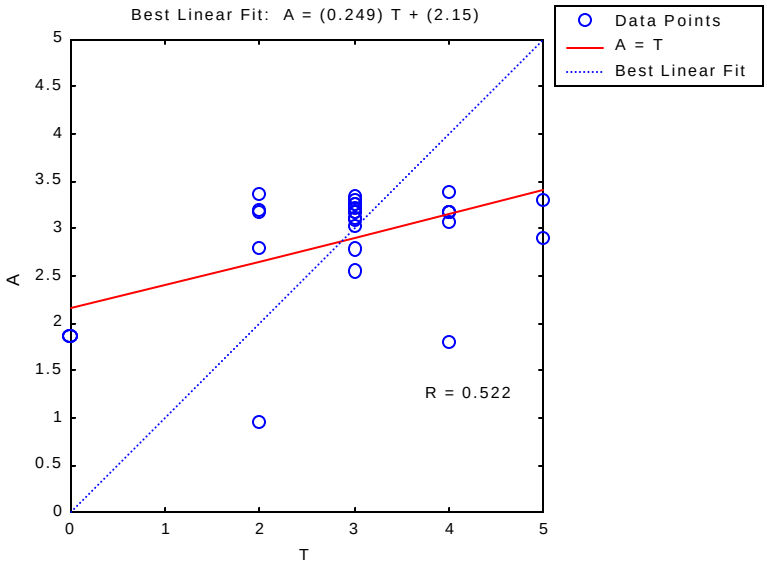
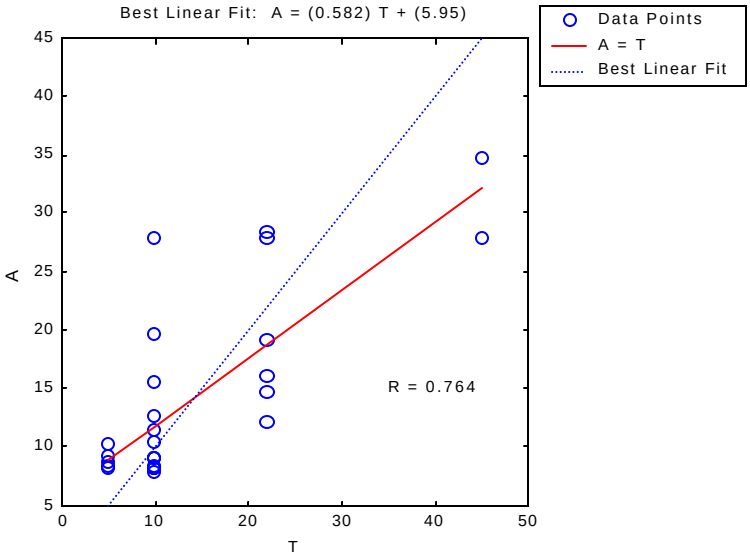


Figure5.6 NN#1 Performance Using *traingd* Training Algorithm, and *Min-Max* Preprocessing



(a)



(b)

Figure5.7 Regression Analysis for NN#1 Training
(a) No of Actions (b) $\Delta\theta$

There are several backpropagation training algorithms that differ in the way they achieve convergence, the way they increase network generalization, and the way they manage memory during training. In Figure5.6 we tried an algorithm called *traingd*, which is a basic gradient descent algorithm. We notice that the regression analysis shows small R Values, which means that the network needs better training, see Figure5.7 (a, b). Getting use of previous weight step increment is useful in future weight adjustments; *traingdm* is a gradient descent algorithm with momentum, and generally it converges faster than *traingd*, see Figure5.8, and Figure5.9 (a, b). We used another backpropagation algorithm to train the network; *traingdx* is another algorithm that uses adaptive learning rate, and it converges faster than *traingd*, see Figure5.10, and Figure8.11 (a, b).

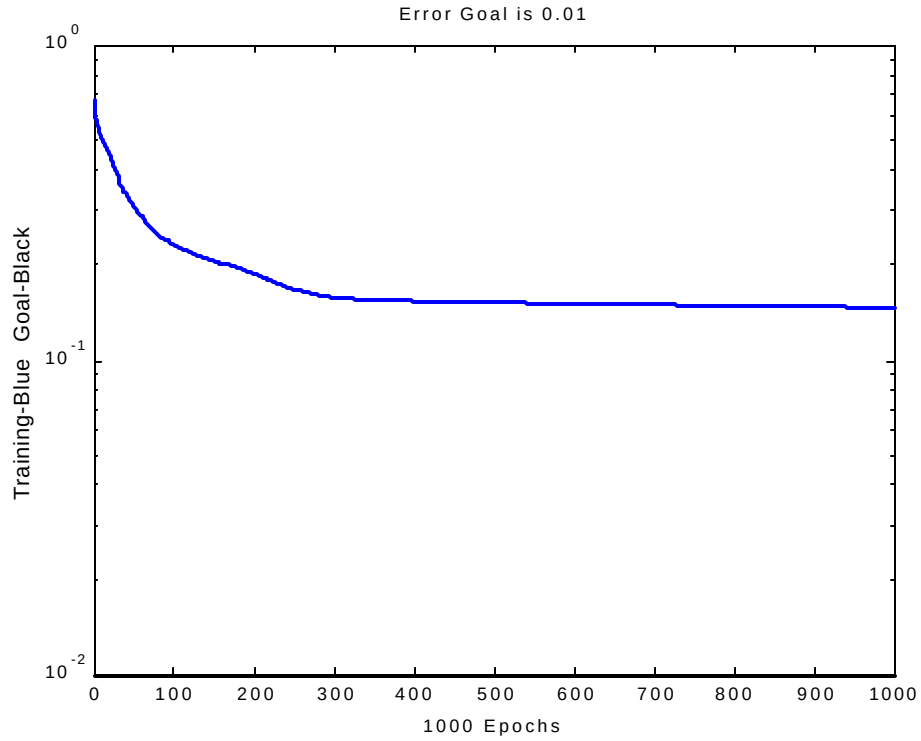
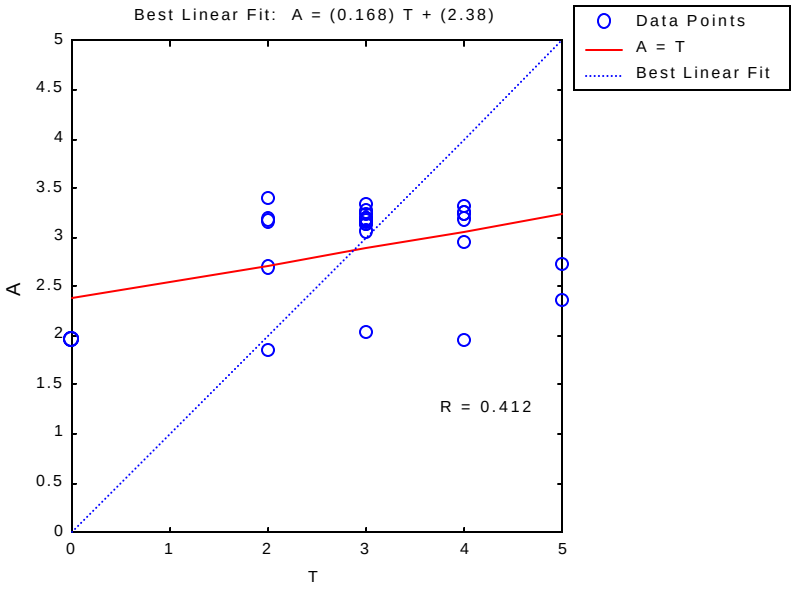
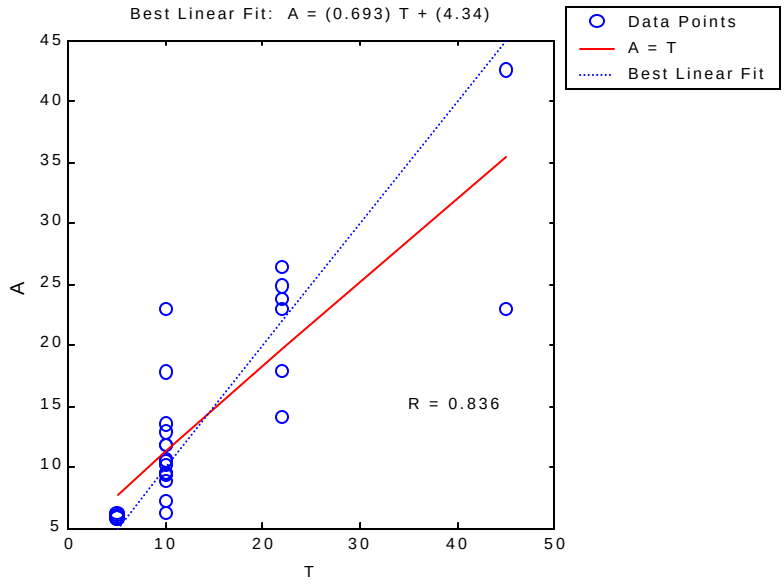


Figure5.8 NN#1 Performance Using *traingdm* Training Algorithm, and *Min-Max* Preprocessing



(a)



(b)

Figure5.9 Regression Analysis for NN#1 Training
(a) No. of Actions (b) $\Delta\theta$

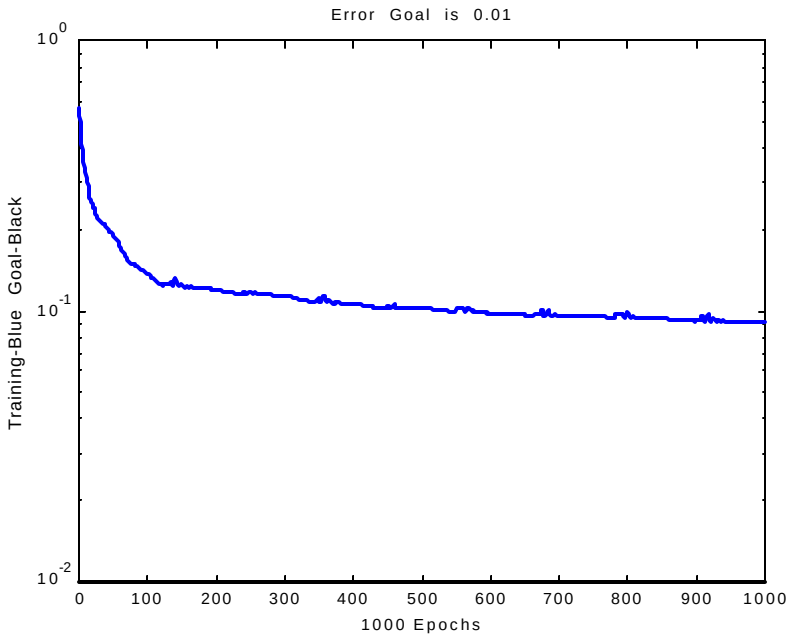
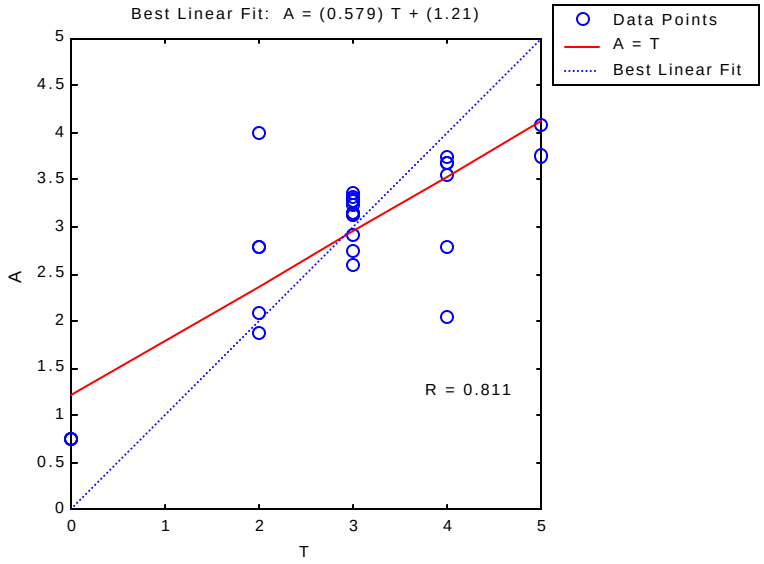
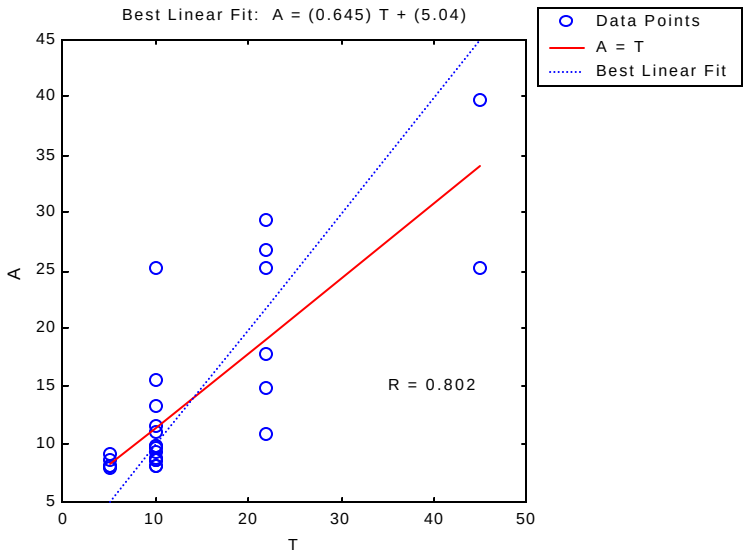


Figure5.10 NN#1 Performance Using *traingdx* Training Algorithm, and *Min-Max* Preprocessing



(a)



(b)

Figure5.11 Regression Analysis for NN#1 Training
(a) No of Actions (b) $\Delta\theta$

We have tried several back propagation training algorithms in the previous examples. When we analyze the regression analysis results, we find out that the best combination will be using the Mini-Max pre-possessing method, and using the traingdx backpropagation training algorithm. The efficiency value that is achieved after modifying the training for NN# can be increased by using a post-processing method that manipulates the output of the AIM. This remark comes from actual output evaluation by Matlab built-in routines, and their strict numeric manipulation when evaluating the correlation between the actual and the desired output. For example, two values the output variable "number of actions" can have a value 4.7 for the actual output, and have a value of 5 for the desired output. The result of this difference on the R-value might decrease its value, but practically if we use approximation in post-processing of the output data, this will be rounded to 5, which is equal to the desired value. Another important point is being aware of the over-fitting problem, which will happen if the R-value is approximately one after training. In the following figures we will show the output results of training NN#2, and NN#3 using the same principles that are used with NN#1; see Figure 5.12, 5.13(a,b), 5.14, 5.15(a,b).

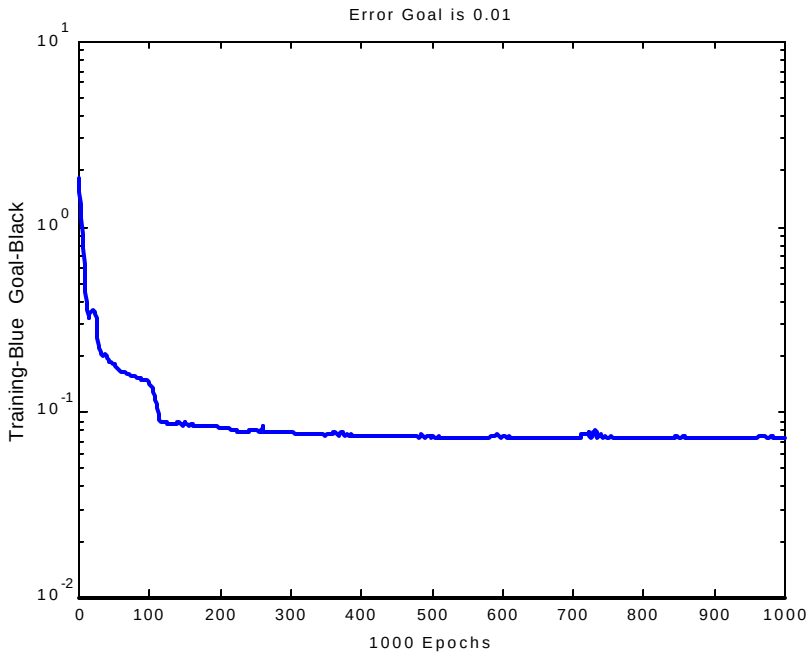
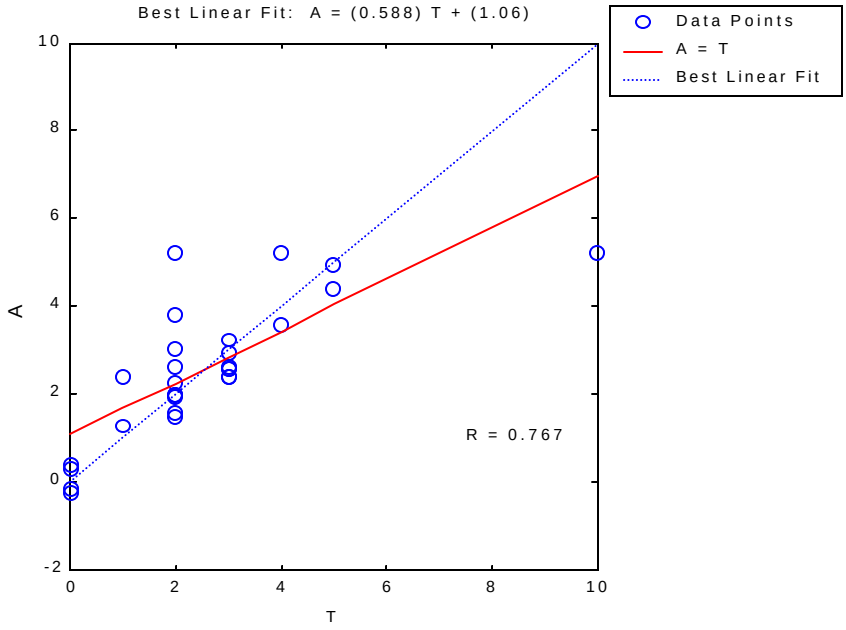
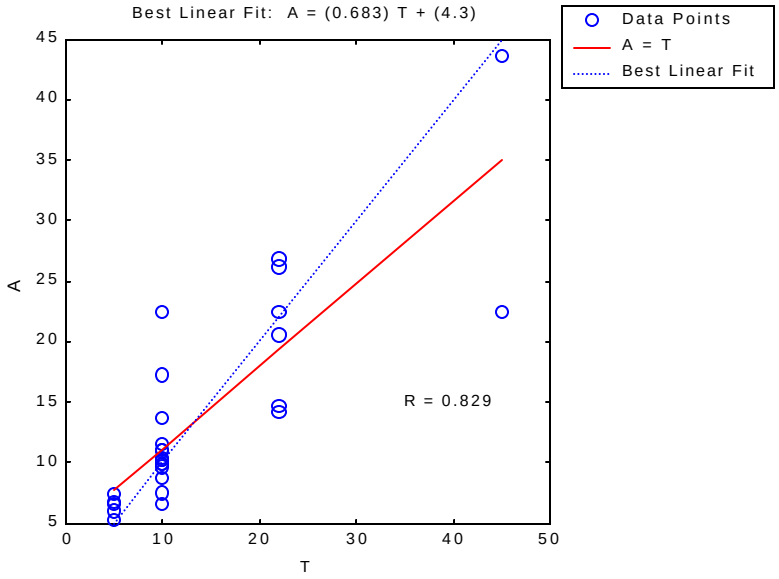


Figure5.12 NN#2 Performance Using *traingdx* Training Algorithm, and Min-Max Preprocessing



(a)



(b)

Figure5.13 Regression Analysis for NN#2 Training
(a) No. of Actions (b) Δθ Value

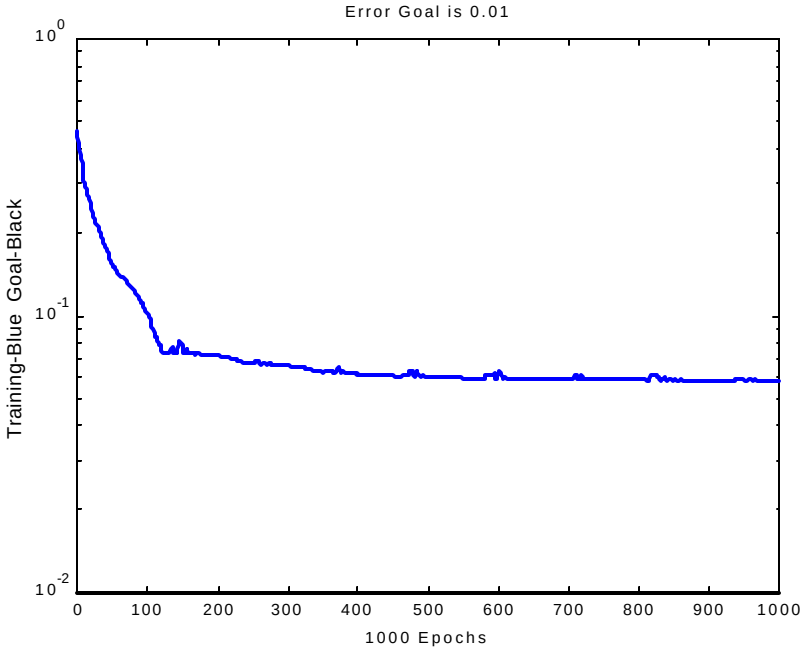
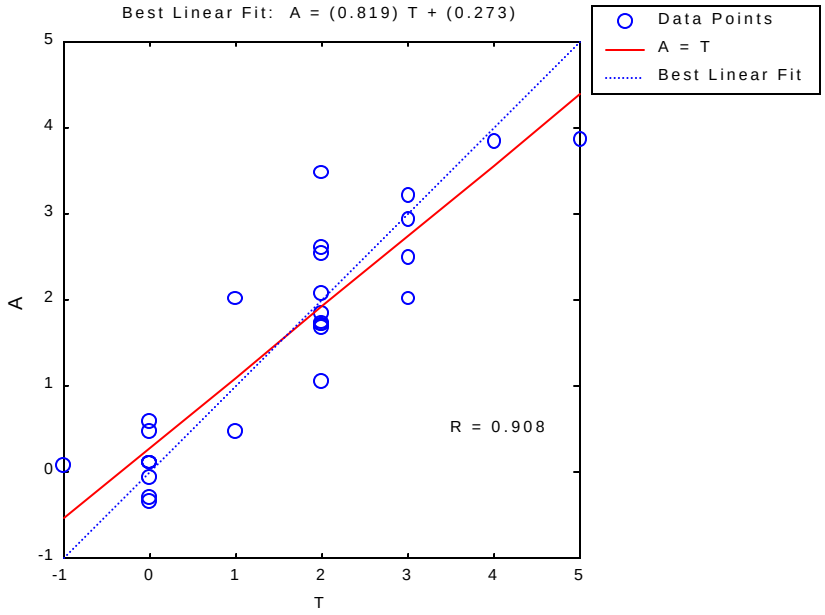
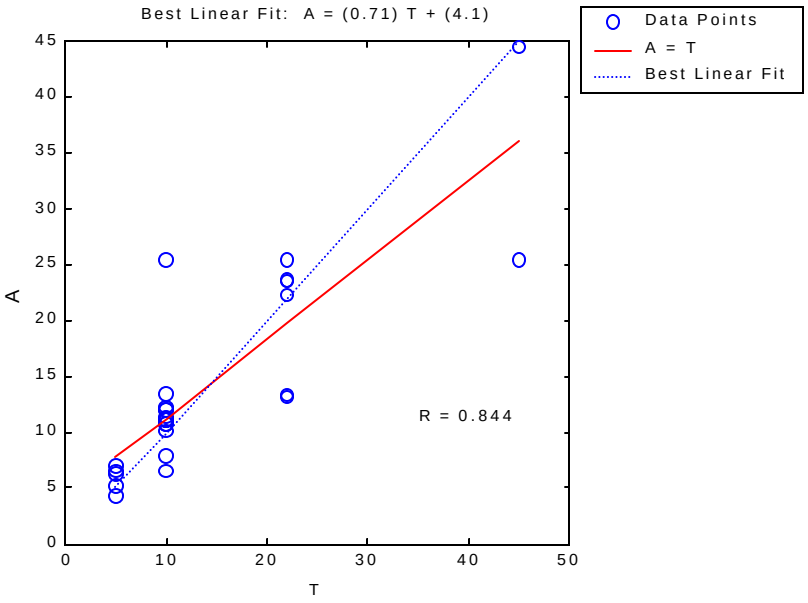


Figure5.14 NN#3 Performance Using *traingdx* Training Algorithm, and Min-Max Preprocessing



(a)



(b)

Figure5.15 Regression Analysis for NN#3 Training
(a) No. of Actions (b) Δθ Value