The Pennsylvania State University

The Graduate School

College of Engineering

A NEURAL NETWORK BASED APPROACH FOR PREDICTING A PATIENT'S CONVERSION TO ALZHEIMER'S DISEASE BASED ON BRAIN SCAN DATA

A Thesis in

Electrical Engineering

by

Mahadevan Srinivasan

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

August 2011

The thesis of Mahadevan Srinivasan was reviewed and approved* by the following:

David J. Miller
Professor of Electrical Engineering
Thesis Adviser

George Kesidis
Professor of Electrical Engineering

Kultegin Aydin
Professor of Electrical Engineering
Graduate Program Coordinator

* Signatures are on file in the Graduate School

ABSTRACT

We studied several neural network architectures for predicting whether a patient will convert to Alzheimer's disease after being initially diagnosed with Mild Cognitive Impairment. The first architecture to be studied was what we call a Localized Neuron Architecture which tries to learn the non-linear relationship between the brain atrophy and the age of the patient. Next, we studied how good the performance is when using a standard multilayer perceptron architecture. We used the brain scan data of the first visit only since the prediction is prognostic. Furthermore, we observed how including the age of the patient when the base line scan was taken would impact the performance. On a previous study based on this data, a support vector machine (SVM) was used to predict conversion. Here, we are using a neural network in place of an SVM. Also, we are trying to predict when the conversion will happen. The challenge is to train a neural network of the correct size and correct structure such that the error in predicting conversion and the standard deviation in the prediction of time at which conversion takes place are minimal.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

First and foremost, I would like to express my gratitude to my advisor Dr. David Miller for his continued support during the course of my study. Without his subject expertise, I would not have completed the thesis. I would like to thank Dr. George Kesidis for taking time out of his busy schedule to be my co-advisor. I also want to thank all my friends here at Penn State who made my stay here a memorable one. Lastly and most importantly, I thank my parents and my entire family for the love and encouragement they have given me throughout my life.

CHAPTER 1

INTRODUCTION

Alzheimer's disease (AD) is the most common form of dementia which is usually seen in people whose age is 65 and above. The disease has no cure and is almost always terminal. The course of the disease follows a unique path for every individual and hence it is very difficult to predict. In most cases the symptoms can be mistaken as problems due to old age. The transitional state between normal aging and Alzheimer's is commonly referred to as Mild Cognitive Impairment (MCI). But, there is no unanimous opinion on whether MCI patients truly convert to AD. Even though there is no consensus on the conversion, many of the past works have used such a definition (Davatzikos et al., 2010; Chou et al., 2009; Misra et al., 2009). The reason for using such a definition is to design models which will help in predicting how the disease progresses and to identify early disease biomarkers.

Alzheimer's disease Neuro-imaging Initiative (ADNI) [Ref. 1] is established to define the progress rate of MCI and AD. Their main aim is to develop better methods for clinical trials and to provide a bigger database for aiding design of treatment trials. MRI scans of the volunteers are taken every six to twelve months for a period of approximately 3 years. A database is formed from their brain scan information, age and clinical scores such as CDR and MMSE (explained below).

1.1 Past Approaches

Prior works in this area have used either the Clinical Dementia Rating (CDR) or the Mini-Mental State Examination (MMSE) scores (Misra et al., 2009; Davatzikos et al., 2010; Wang et al., 2010). The problem with such an approach can be understood by looking at how the

scores are obtained. For example, CDR[14] scores are obtained by evaluating the performance of the patients in six areas: memory, orientation, judgment & problem solving, community affairs, home & hobbies, and personal care. Scores in each of these are combined to obtain a composite score ranging from 0 to 3. The higher the score the more pronounced the dementia, with 0 being no dementia and 3 being severe dementia. Similarly, the MMSE scores (Folstein et al., 1975) are also derived from interviews with the patients.

The main problem with the past approaches is that they use these CDR / MMSE scores to define the ground truth ignoring the information present in the MRI scan of the patient. There are other concerns with using CDR/ MMSE also. For example, the CDR scores usually do not change very much from the baseline score for most patients (whether they have AD or MCI). This makes the data set biased towards non-converters and eventually makes it difficult to design an accurate classifier. The problem of predicting whether a patient converts from MCI to AD using the brain scan information is not extensively studied. Even though (Wang et al., 2010) worked on predicting future MMSE scores from the baseline scans that was not the main focus of the paper – they focused on predicting the current score.

1.1.1 Trajectory based Approach

This thesis extends the work by (Aksu et al., 2011) which used basically two things available from the ADNI database.

1) Brain scan image

2) Labeled examples saying whether a patient is AD or Control.

It is important to note that the problem of identifying converters from non-converters falls neither under supervised learning nor unsupervised learning. This is because we do not have

labeled examples for converters and non-converters. Rather, we have the labeled examples only for AD or Control.

Given this data, (Aksu et al., 2011) built an accurate image-based linear kernel based SVM classifier to predict whether a patient is AD or control. The classification accuracy was approximately 91%. The examples were labeled using the CDR scores which ranged from 0 to 3. But the authors made sure that they only used patients whose scores equaled 0 (Control) and those whose scores are more than 1 (AD) for training and test purposes. This classifier was then applied to a population of MCI patients and they observed which side of the classifier the patients fell into for each visit. The classifier, apart from the binary decision of AD or Control, gives a numerical output called the "score" for each visit, which tells how far the patient is from the classifier boundary. Based on these scores, they defined converters as those who went to the AD side from the control side or those who stayed on the AD side throughout all the visits and non-converters as those who stayed on the control side during all the visits. The ground truth for a patient being a converter or a non-converter is thus established.

Next, the authors built another linear-kernel based SVM classifier which took as its input the baseline image (first visit MRI scan) and tries to predict if the patient is classified as a Converter or Non-converter by the first classifier. The second SVM classifier was trained and tested using the data and provided cross-validation (CV) generalization accuracy as high as 83% when they did a voxel-based (nearly 11,000 features) classification. For the region based classification (100 features), the accuracy obtained was as only high as 77% (D.J. Miller, 2011).

(Schuff et al., 2010) conducted experiments on how the brain-atrophy is related to age of the patients. Their results show that the brain region volumes are non-linear functions of age with each region volume having a different non-linear independence. So, the authors (Aksu et al.,

2011) also built nonlinear-kernel (Gaussian-kernel) SVMs to consider the non-linear dependence on age. The AD-Control classifier had an accuracy of 90% and the second classifier accuracy was as high as 71%.

One could also identify converters and non-converters by their CDR scores (by a suitably defined SVM). In fact, the authors (Aksu et al., 2011) studied how such a definition would compare against their by-trajectory definition. The overall accuracy in prediction when using by-CDR definition was only 56% for the best case. They noticed that nearly 66% of the patients classified as non-converters by the by-CDR definition were classified as converters when using the by-trajectory definition.

In this thesis, we are trying to extend the work to see if the prediction accuracy could be improved by using a neural network instead of the second SVM used previously. Apart from that, we are also trying to predict when the actual conversion will happen. Instead of trying to predict whether a patient is converter or non-converter directly, here we trained the network to predict the scores given out by the first SVM classifier. This was done by giving the 100 features (region-based) and the age of the patient during the baseline scan and the delta-age at which the score was obtained. Section 3.2 explains how this delta-age input is obtained from the age information present in the ADNI Database.

1.2 DATA

We have the data for a total of 303 patients. For each patient, we may have the data for a maximum of six visits. Each data point, if available, will have the volume information for 100 brain regions along with the age of the patient for that visit and the AD-control SVM's output for each visit which we call the score. The score will indicate whether a patient has Alzheimer's or

not. Also, the duration between visits for different patients is different. Some patients will have more number of visits than others. On an average, we have 3 sets of data for each patient. We have not used all the data present. That is, even though we have the brain scan data for all the visits, we have used the brain scan information of the first visit data only (at least in our initial experiments) to train the neural network along with the age information of that visit and other visits of the patient. Only with such a network will we be able to do a prognosis on the conversion. We call a patient a converter if he/she was on the control side at the first visit and in one of the later visits moves to the AD side or if they are on the AD side during all the visits.

Out of the 303 patients, small amounts (~5%) of the patients are considered outliers. This is because their scores go from the AD side to the control side. Removing the outliers, we had a total of 287 patients. Based on the by-trajectory definition used in (Aksu et al., 2011), we have in total 176 converters and 111 Non-converters.

1.3 THE NETWORK

As mentioned already, we use the age of the patient during the first visit and delta-age information as additional network inputs. With these inputs, we can use the scores at all the visits to train the network. The next step is identifying the structure of the network and size of the network.

Due to (Schuff et al., 2010), we believed that a neural network which has 100 neurons (one for each feature) in the input layer, possibly followed by a hidden layer whose size should be determined, followed by an output layer with one neuron should be able to model the data well. Each neuron in the input layer will be fed with a feature (volume of one of the brain regions), age of the patient during the first visit and delta-age for the target output. Having a non-

linear transfer function for the neurons in the first layer will help the network learn the relationship between the brain volume and age. Furthermore, the age input is given to make the network aware of “when” the MRI scan was taken. We studied how the network behaved with and without the age input.

CHAPTER 2

REVIEW OF MULTI-LAYER PERCEPTRONS

2.1 Introduction to (Artificial) Neural Networks

Artificial Neural Networks (ANNs) are composed of series of interconnected neurons arranged in layers. They are primarily used in two applications:

1. Classification
2. Regression

We are concerned about regression in this thesis. The network parameters are adapted based on the input given to it and the associated output.

2.2 Neuron

An Artificial Neuron (hereafter called simply a neuron) is a mathematical model of a biological neuron. They are the basic elements of any Artificial Neural Network. The block diagram of Figure 1.1 shows the model of a neuron. As can be seen from the figure, there are three main elements in the model.

1. Synapses
2. An Adder
3. An Activation Function

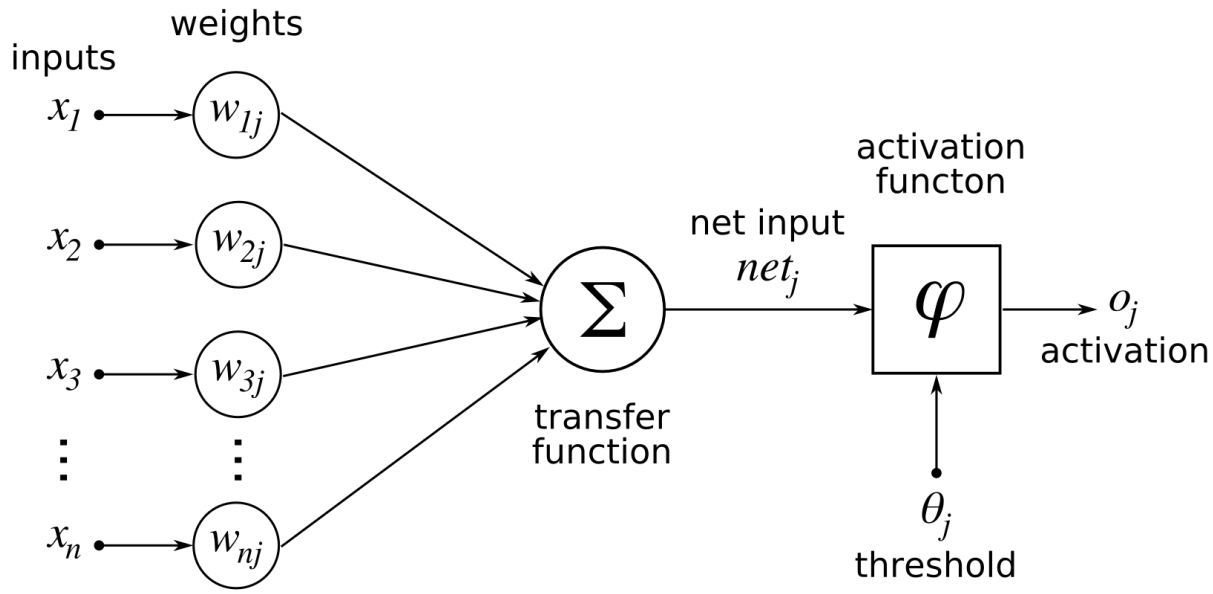

Figure 2.1: An Artificial Neuron

2.2.1 Synapses

In the brain, synapses are the connections between biological neurons. In ANNs, we model synapses using weights. The weights along with the choice of the activation function decide when a neuron will be activated. Apart from the weights, we may also have biases (not shown in figure) for each neuron. The bias term helps in making affine transformations to the data. The weights and biases can take positive as well as negative values.

2.2.2 Adder

An adder simply computes the weighted sum of the all the inputs where the weights are the values of different synapses.

2.2.3 Activation Function

Activation function of the neurons decides how expressive the network can be in learning the data. It helps in limiting the magnitude of the output of a neuron. There are three basic types of Activation Functions:

1. Threshold Function
2. Linear Activation Function
3. Non-Linear Activation Function.

2.2.3.1 Threshold Function

A threshold function is an ordinary step function whose output is 1 if the input is more than 0 and 0 otherwise. A neuron with such an activation function is commonly known as the McCulloch-Pitts model.

2.2.3.2 Linear Activation Function

In this case, the output of a neuron is just a linear combination of the inputs plus the bias term. Usually, this type of activation function is used in the output layer where we need a wide range. With a linear transfer function, the output of the neuron can take any value.

2.2.3.3 Non-linear Activation Function

In this case, the output of a neuron is a non-linear function of the inputs. A commonly used non-linear activation function is a sigmoid whose shape is shown in the figure below. The main reason for using a non-linear function is because such a function has a clearly defined derivative, which will be useful when computing the weights of the network.

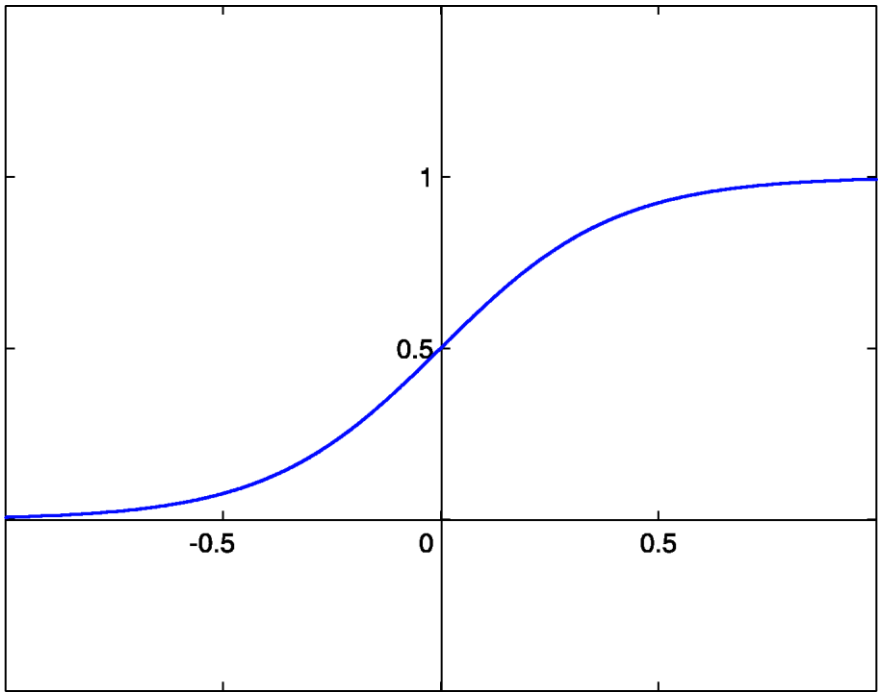

Figure 2.2: Sigmoid Function

2.3 Learning

Generally artificial neural networks are basic input and output devices, with the neurons, as mentioned already, arranged in layers. For example, perceptrons consist of an input layer followed by a layer of output neurons with a layer of weights between them.

The data is given to the input layers which are then weighted and are in turn passed to the output layers. Based on the type of transfer function and the input presented to the output layer, it decides whether to fire a neuron or not.

The challenge now is to find the weights which minimize the error between the network output and the target output (Supervised Learning). This type of learning is also called as "learning with a teacher". Conceptually, the teacher has the knowledge of the problem where knowledge is the set of input-output examples. The network has to be fed with the knowledge.

By using the target values for each output, the teacher is able tell the network how to react to such an input. This is called “training of the network”. First, the network weights are initialized to random values and then the inputs are passed through the network. The weights of the network are adjusted based on the error between the network output and the target value. Thus, the network learns about the data iteratively. Once the entire knowledge is transferred, we no longer need the teacher. The network would have learned enough to make its own decisions on inputs it has not seen before. The most commonly used method for minimizing the error is Gradient Descent. Gradient Descent is an optimization algorithm used to find the local minimum of a function. It is not guaranteed that we will find the global minimum unless the function has only one minimum. By running the algorithms several times with random starting points, we can try to ensure that we reach a minimum close to the global minimum.

2.4 Multilayer Perceptrons

The problem with perceptrons is its expressive power. It can only be used to solve problems which are linearly separable. And most of the real-life problems are in fact not linearly separable. This is where Multi-layer Perceptrons (MLPs) come in. MLPs consist of an input layer, one or more hidden layer of neurons followed by an output layer of neurons. It can be easily proven that an MLP with a single hidden layer can approximate any function provided it has enough number of hidden neurons (Ref. 4).

2.5 Back propagation Algorithm

The back propagation algorithm is one of the most commonly used methods for training an ANN. It requires that we know the target output value for any given input (Supervised Learning). Also, it requires that the activation function used is differentiable. So, we cannot have

a step function as activation function. We could still have either a linear or a non-linear transfer function.

There are two steps in training a network using back propagation algorithm:

1. Forward Pass
2. Backward Pass

2.5.1 Forward Pass

In this step, the input is given to the network and based on the weights and biases of the network, the output is calculated. Here the input is allowed to propagate through the network in the forward direction.

2.5.2 Backward Pass

In this step, the errors are computed starting from the output layer and then going back layer by layer and the weights are updated in each layer to minimize the errors. Since the errors are propagated starting from the output and going back, this step is called backward pass. A detailed explanation of how the algorithm works can be found in [6].

2.6 Model Order Selection

One of the biggest problems in designing a neural network is in choosing the size of the hidden layer. The hidden layer is a layer of neurons which are neither the input nor the output layer and are called so because they are hidden from the user's view point. Having a large number of neurons in the hidden layer may result in over-fitting of the data. The effect is that the network memorizes the data and may be useless when presented with an unseen data. On the other hand, having less number of neurons may mean that the network will not have the

complexity to properly learn the data. In most cases, the ideal model order is selected heuristically.

CHAPTER 3

EXPERIMENTAL METHODS

3.1 Data pre-processing

As mentioned already, we had the data for 287 patients. If the score starts out positive and then goes negative or if it stays negative throughout, we say that the patient is a converter, meaning he/she has gone from having Mild Cognitive Impairment (MCI) to AD. The patients whose score never goes below the threshold, we call them Non-converters. There is also a third category, patients who start in the AD side and then go to the control side. We have a very few of those (5%) and they are considered outliers and we leave those patients from the data set. There are two ways of classifying patients as converters or non-converters. One is based on the raw scores and other is by fitting a line to the data and then using that line to classify. The raw scores as such are quite noisy as seen from the diagram below (Figure 2.1). Figure 2.2 shows how the same data look like when fitted using a line. Once the line fitting is done, we use the line to predict the score for the sixth visit by extrapolating. Similarly, one could fit a line to the output of the network before making a decision on whether the patient is a converter or not. The effects of using fitted scores vs. raw scores are summarized in section 4.3. All the results shown before that section are based on training the network using the fitted scores and testing it using raw scores. The reason for using this method is also explained in section 4.3.

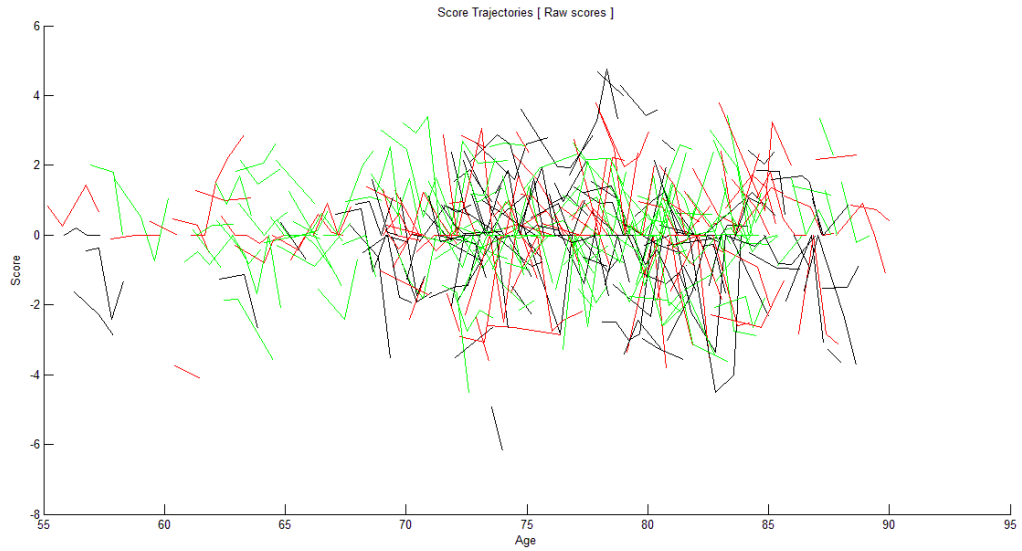

Figure 3.1: Raw Score Trajectories vs. Age

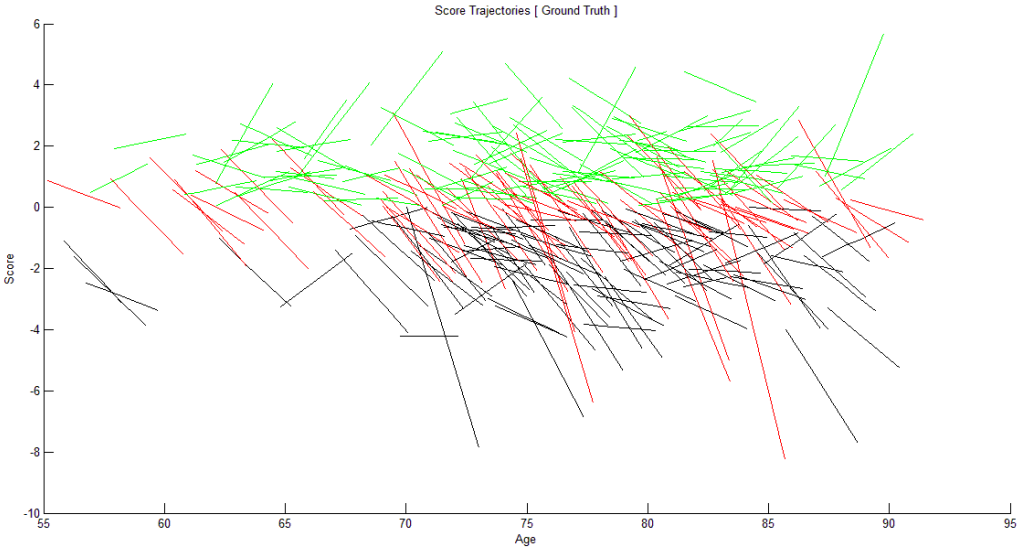

Figure 3.2: Fitted Score Trajectories vs. Age (Ground Truth)

3.2 Delta age input

As mentioned already, for each visit of a patient, we have the brain volume data for 100 regions along with the patient's age during that visit. For example, a sample data may look like what is present in Table 3.1. The delta-age values are calculated from the age value by making the first-visit delta-age equal to zero and by subtracting the first visit age value from the future visit values. In this example, visit 1 is the first visit for which we have the data. So, the delta-age for visit 1 is made 0 and the rest is obtained by subtracting 69.42 (first visit age) from the age during that visit.

	Age	**Features**	**Delta-Age**
Visit 1	69.42	100 x 1 Vector	0
Visit 2	N/A	N/A	N/A
Visit 3	70.44	100 x 1 Vector	1.02
Visit 4	71.08	100 x 1 Vector	1.66
Visit 5	N/A	N/A	0
Visit 6	72.42	100 x 1 Vector	3

Table 3.1: Sample Data for a Patient

3.3 Network Architectures

We started with neural network architecture where we have 100 neurons in the Input Layer with each neuron supplied with 3 Inputs:

1. A unique feature from the brain scan data
2. Age
3. Delta-age

Here, the features are extracted from the 1st visit data's brain scan. Even though we have the brain volume information for the subsequent visits, we are not using them since the prediction is intended to be prognostic. We expected to see a non-linear relation between the

brain features and the age data. But, we did also try out a linear transfer function in the input layer. The second layer of the network had only one neuron and it is the output layer. We also experimented with the input structure. We left out age input and studied the performance.

The next architecture we studied is similar to the above architecture but with an extra hidden layer. We experimented with various values for number of neurons in the hidden layer.

After studying these architectures, we checked to see how well a standard MLP neural network model would work. We had three layers, an input layer, a hidden layer and an output layer. The hidden layer had a non-linear transfer function and the output layer had a linear transfer function. The reason for not trying a linear transfer function in the hidden layer is explained in chapter 4. The output layer had only one neuron. We gradually increased the number of neurons in the hidden layer and checked the performance. The input to the neural network is the 100 features, age and the delta age information with all the 102 inputs fed to all the neurons in the input layer. As before, we evaluated the performance by leaving out age.

3.4 Training the Network

After removing the outliers, we had the data for 287 patients. Of these 287, we kept 143 for training and validation purposes and the remaining 144 for the test set. Since the training may end early due to a local minimum, we had to train the network several times with random weight initialization each time. We trained the network 100 times for each architecture. The data split (training and test split) was same for each cycle but the training and validation split was random for each cycle. Also, the weight and bias initializations were random for each run. The 143 patients in the training set is split into two sets again, one for training and the other is for the validation. The performance measure used for training the network is the mean squared error

(MSE). The weights and biases are initialized in two different ways. One is using Matlab's default *initnw* function which uses the Nguyen-Widrow initialization algorithm (Ref. 8). The other method is initializing them using values randomly from the interval $[-10^{-5}, 10^{-5}]$. The second initialization method is chosen to make sure that we are not saturating the network by initializing the weights to a large value.

3.4 Testing the Network

After the training is complete, the test inputs are passed through the network to obtain the network output. Here, we do not use MSE as performance measure since our goal is predicting the converters and non-converters correctly. We have two performance measures for the test set:

1. Total accuracy in predicting converters and non-converters

    a. Converter Prediction Accuracy (A Measure of Sensitivity)

    b. Non-converter Prediction Accuracy (A Measure of Specificity)

2. Standard deviation in predicting when conversion will happen.

Here again, we could use either the raw scores or the fitted line for checking whether a patient is a converter or not. For the standard deviation, we have to use a fitted line to see where the line crosses the threshold. For simplicity, we will hereafter call the Converter Prediction Accuracy as Sensitivity and Non-Converter Prediction Accuracy as Specificity.

CHAPTER 4

MODEL DEVELOPMENT AND RESULTS

As mentioned already, we first trained the network with a linear transfer function in the hidden layer. The problem with such a network can be understood by looking at the trajectories the network was predicting for the test set (Figure 4.1) and comparing it with Figure 3.2, the ground truth. Clearly, the linear transfer function makes all the trajectories follow a same whereas a non-linear transfer function tries to better estimate the ground truth behavior as can be seen from Figure 4.2. Even though it is not obvious, it is quite trivial to prove mathematically why the trajectories for all the patients follow the same slope in the linear transfer function case.

$$Output_{linear} = \sum_{i=1}^{100} (w_{1i} v_i + w_{2i} a + w_{3i} d + B)$$

$$= \sum_{i=1}^{100} w_{1i} v_i + a \sum_{i=1}^{100} w_{2i} + d \sum_{i=1}^{100} w_{3i} + B = \sum_{i=1}^{100} w_{1i} v_i + W_2 a + W_3 d + B$$

$$where\ v_i\ is\ the\ volume\ of\ region\ i; W_2 = \sum_{i=1}^{100} w_{2i}; W_3 = \sum_{i=1}^{100} w_{3i}$$

$a\ is\ the\ age\ of\ the\ patient\ when\ baseline\ scan\ was\ taken;$

$d\ is\ the\ delta - age\ of\ the\ patient\ for\ the\ particular\ target\ output;$

$B\ is\ the\ sum\ of\ all\ the\ biases$

Above equation clearly shows that in case of the linear transfer function in the hidden layer, the localized architecture and the standard MLP architecture are the same. This is the reason why we did not try a linear transfer function in the hidden layer with the standard MLP architecture.

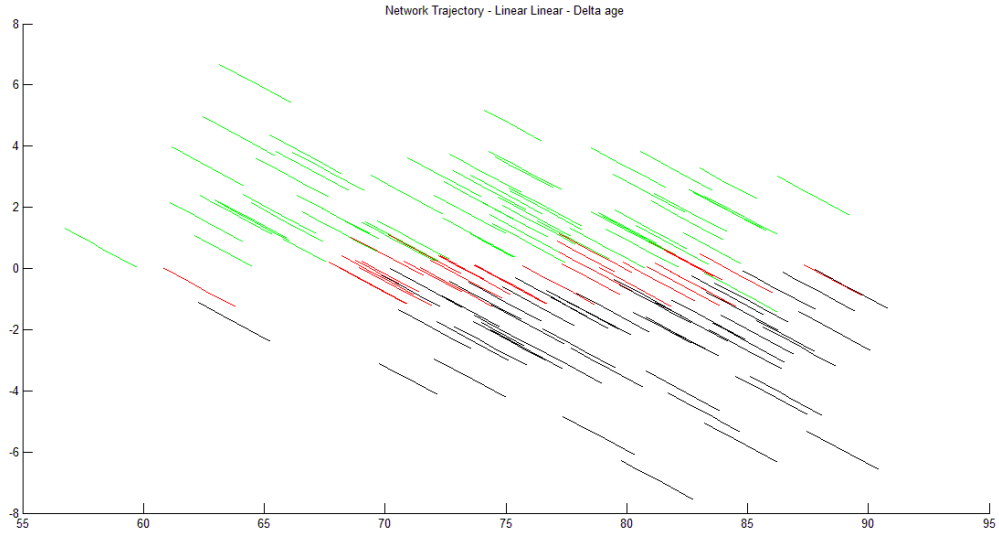

Figure 4.1 Trajectories while using a Linear Transfer Function in the Hidden Layer

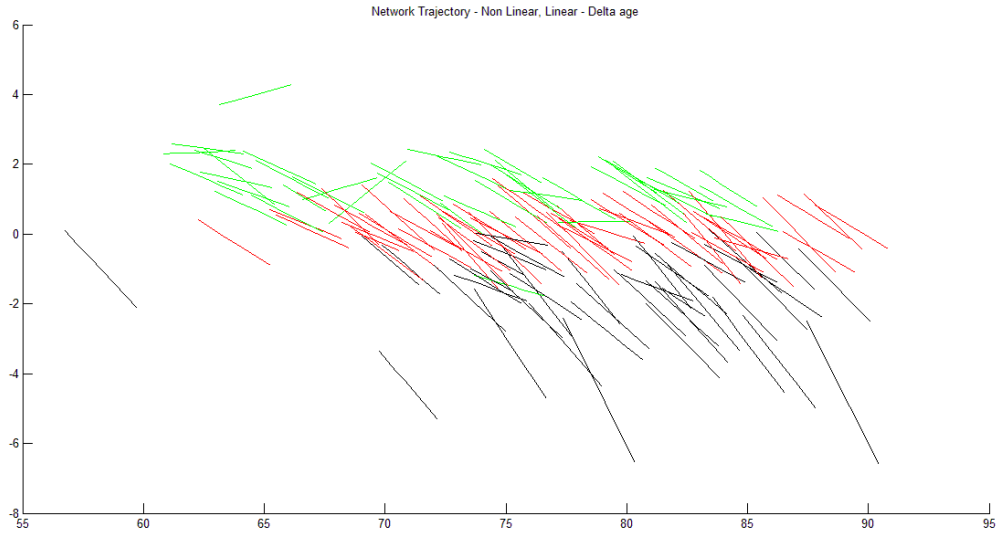

Figure 4.2 Trajectories while using a Non-Linear Transfer Function in the Hidden Layer

4.1 Localized Neuron Architecture

We started with localized neuron architecture with 1 or 2 hidden layers to see what transfer functions will work well in different layers. Also, we wanted to study how good the performance was when including age and/or delta age. With this, we experimented with all

possible combinations of transfer functions. Figure 4.3 shows a pictorial representation of how the first hidden layer looks like.

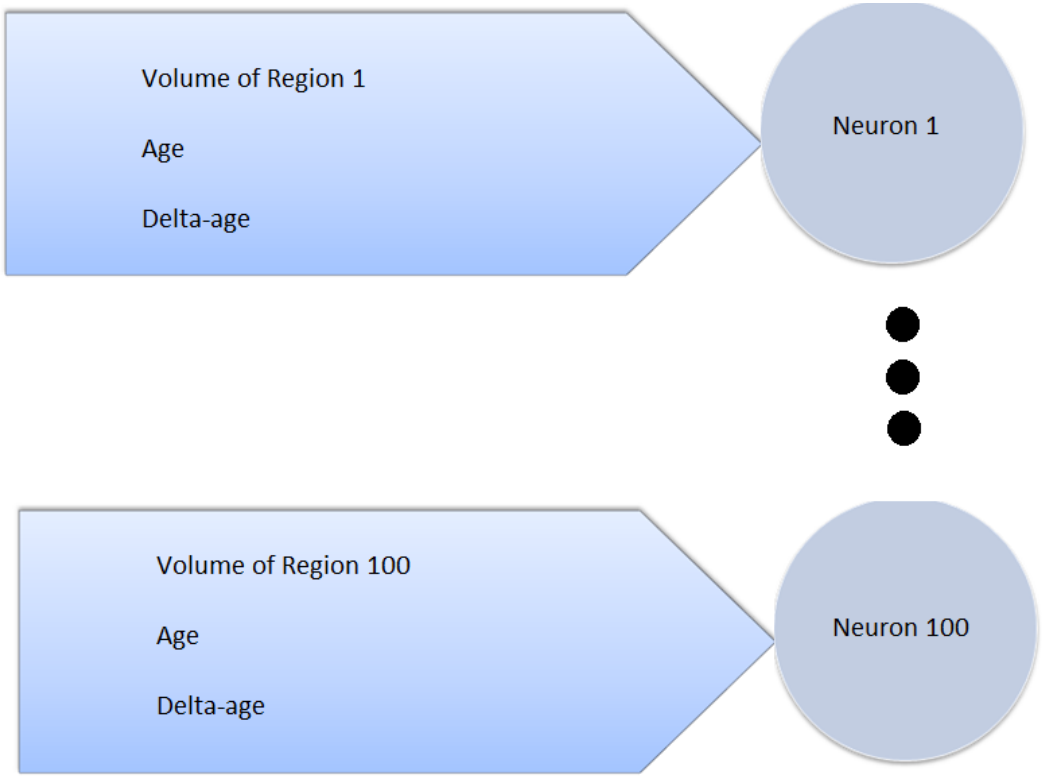

Figure 4.3: Localized Architecture

4.1.1 Localized Neuron Architecture (with one hidden layer)

Tables 4.1 and 4.2 show the (average) performance of the best architectures when having just a single hidden layer. It was clear that having a non-linear transfer function in the hidden layer and a linear transfer function in the output layer worked well in terms of prediction accuracy as well as in terms of predicting the score trajectories. Surprisingly, having a linear transfer function in the input layer gave better prediction accuracy. But, the trajectories show that the linear network treats all patients as equals which we know is not true.

In the following table, a 3 in the type field indicates that the inputs age and delta-age were used in the network along with the features.

Model Type	Linear, Linear	Non-Linear, Linear	Linear, Linear,3
Training MSE	1.1717	2.2982	1.7031
Validation MSE	1.2552	2.2477	1.6468
Test MSE	2.0309	2.3538	1.9634
Total Accuracy (%)	74.1259	67.1329	67.8322
Non-Converter Accuracy (%)	57.4815	40.2469	47.8458
Converter Accuracy (%)	80.7865	84.1448	83.8340

Table 4.1: Performance with a Validation Set (One Hidden Layer)

4.1.2 Localized Neuron Architecture (With two hidden layers)

Having evaluated the performances for the single hidden layer architecture, we tried a two hidden layer architecture taking into consideration the best network architectures in the previous step. Since, the output of the network can take any value, we settled for a linear transfer function in the output layer. The performances of the networks are shown in Tables 4.3 and 4.4. We randomly initialized the number of neurons in the 2nd hidden layer to 18 and noted the performance. These experiments were done with only delta-age and volume information as inputs.

Model	Linear, Linear, Linear	Non-Linear, Linear, Linear	Non-Linear, Non-Linear, Linear
Training MSE	1.184	2.275	2.983
Validation MSE	1.284	2.58	2.866
Total Accuracy (%)	73.008	66.293	61.818
Non-Converter Accuracy (%)	58.547	31.947	11.017
Converter Accuracy (%)	82.299	88.779	95.364

Table 4.2: Performance with a Validation set (2nd Hidden Layer of 18 neurons)

Model	Linear, Linear, Linear	Non-Linear, Linear, Linear	Non-Linear, Non-Linear, Linear
Training MSE	0.614	0.576	0.094
Total Accuracy (%)	70.56	70.141	62.866
Non-Converter Accuracy (%)	61.298	60.682	54.827
Converter Accuracy (%)	76.719	76.304	67.945

Table 4.3: Performance without Validation set (2nd Hidden Layer of 18 neurons)

Next, we wanted to study the performance as a function of the number of neurons in the second hidden layer.

4.1.2.1 Localized Architecture (Non-Linear, Non-Linear, Linear)

From the plots for the training performance and test performance (Figures 4.3 and 4.4), we could make no inference on which order is the best. By looking at the prediction accuracies for various orders, we could see that the best performance (with only delta-age) in terms of total accuracy was obtained for order 4, in terms of sensitivity for order 20, and in terms of specificity for order 4. When using age also, the best performance in terms of total accuracy was obtained for order 3, in terms of sensitivity for order 11, and in terms of specificity for order 20.

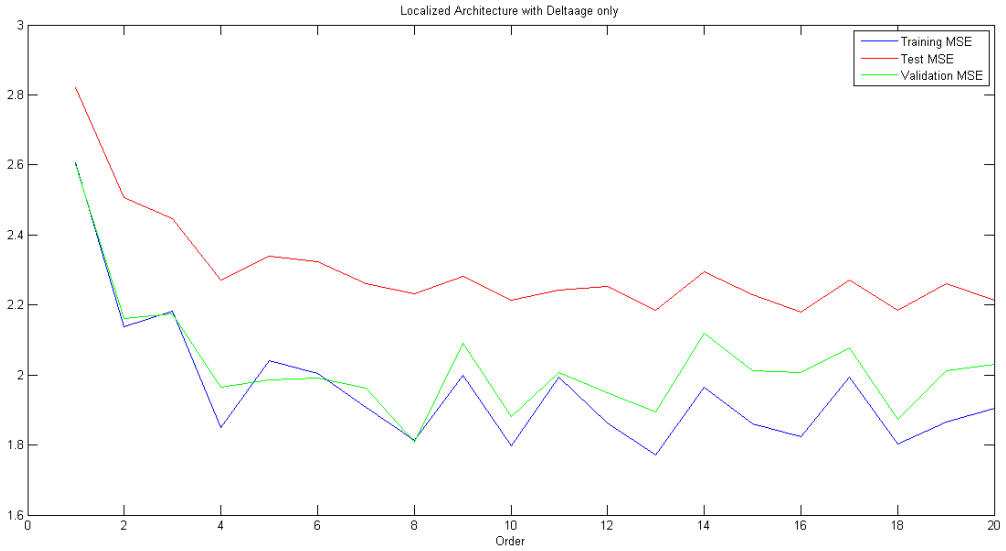

Figure 4.4: Performance vs. Order for Non-Linear, Non-Linear, Linear Arch. (Delta Age only)

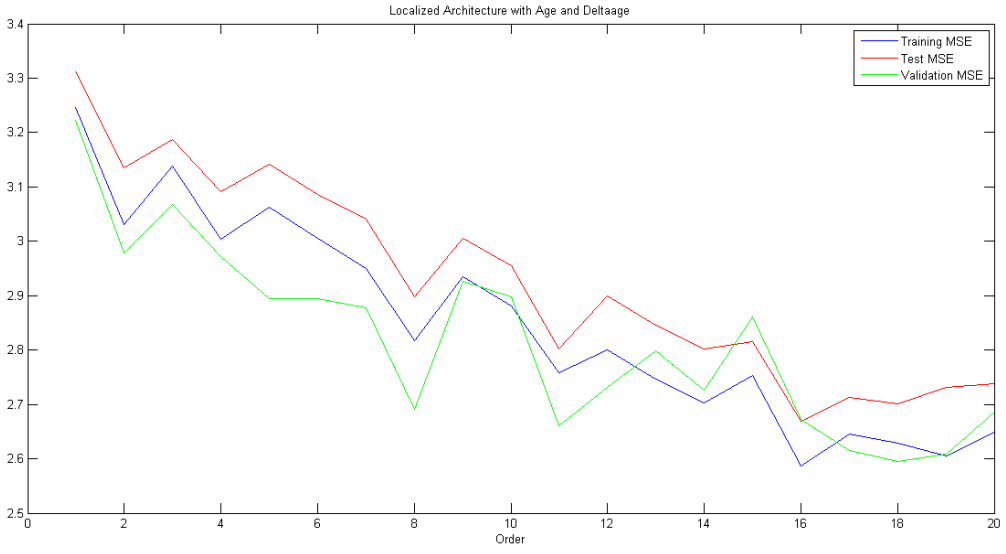

Figure 4.5: Performance vs. Order for Non-Linear, Non-Linear, Linear Arch. (Age and Delta Age)

Model Order / Performance Measure	4 / Total Accuracy	20 / Converter Accuracy	4 / Non-Converter Accuracy
Training MSE	1.8486	1.9036	1.8486
Test MSE	2.2703	2.2126	2.2703
Validation MSE	1.964	2.0301	1.964
Total Accuracy (%)	69.3244	68.6661	69.3244
Non-Converter Accuracy (%)	49.9141	45.3588	49.9141
Converter Accuracy (%)	82.1843	84.1109	82.1843

Table 4.4: Best Model Performances for Localized Arch. with Delta age only

Model Order / Performance Measure	3 / Total Accuracy	11 / Converter Accuracy	20 / Non-Converter Accuracy
Training MSE	3.0301	2.7581	2.6496
Test MSE	3.1348	2.8019	2.7372
Validation MSE	2.9784	2.6611	2.6855
Total Accuracy (%)	66.345	64.9383	65.9263
Non-Converter Accuracy (%)	39.0162	29.8729	39.5386
Converter Accuracy (%)	84.2437	88.0214	83.1958

Table 4.5: Best Model Performances for Localized Arch. with Age & Delta age

4.2 Standard MLP Neural Network Model

After studying all these localized architectures, we wanted to know how well they fared when compared to a standard MLP neural network with two layers and with all the inputs given to all the neurons. As always, we tried both combinations in the inputs by having delta-age only and having them both.

Seeing the performance curves of the two models (Delta Age only and Age & Delta Age), we see that the training MSE and validation MSE decrease continuously as a function of model order for the first model. But the test MSE is not in the same order as the training and validation MSEs. Clearly, the network was over-fitting the data in spite of the validation dataset. The over-fitting phenomenon was confirmed by looking at the training MSEs for those runs whose test MSEs were high. Training continued for 1000 epochs and the training MSEs were

nearly zero. For both the models, it is interesting and gratifying (looking at figures 4.7, 4.9 & tables 4.6, 4.7) to note that the best performances (in terms of classification accuracy) were obtained when the test set MSEs were small.

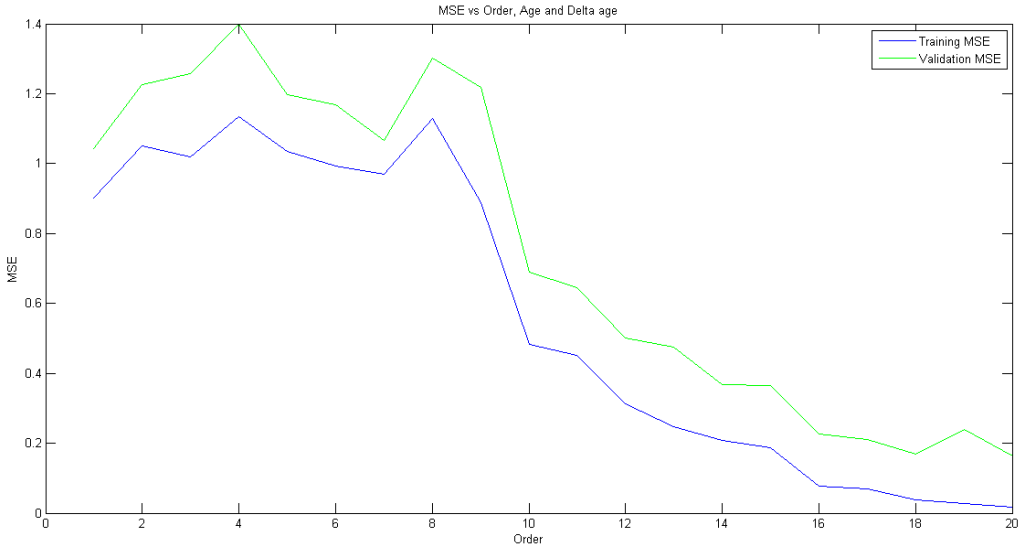

Figure 4.6: Training & Validation Performance vs. Order for the Standard MLP model (Delta Age only)

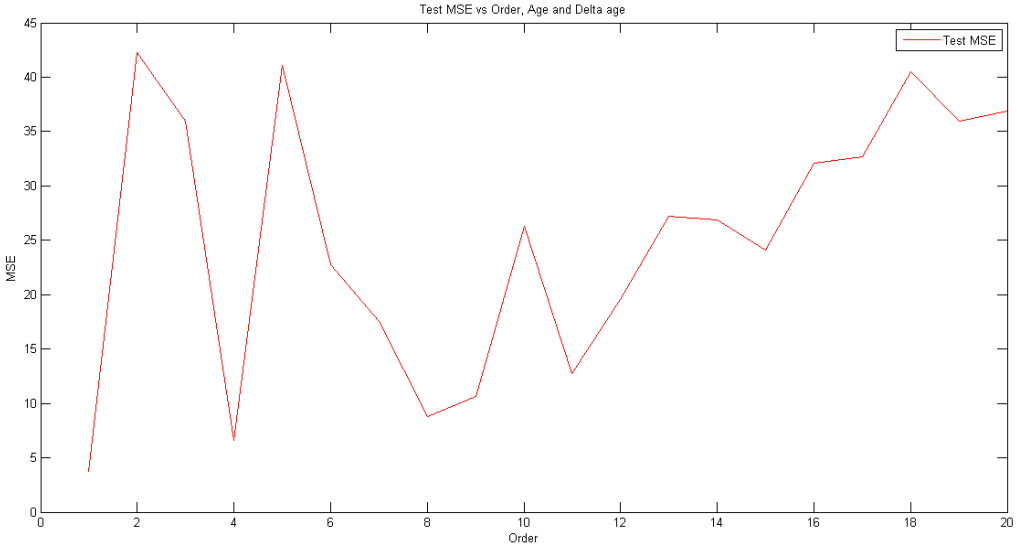

Figure 4.7: Test Performance vs. Order for the Standard MLP model (Delta Age Only)

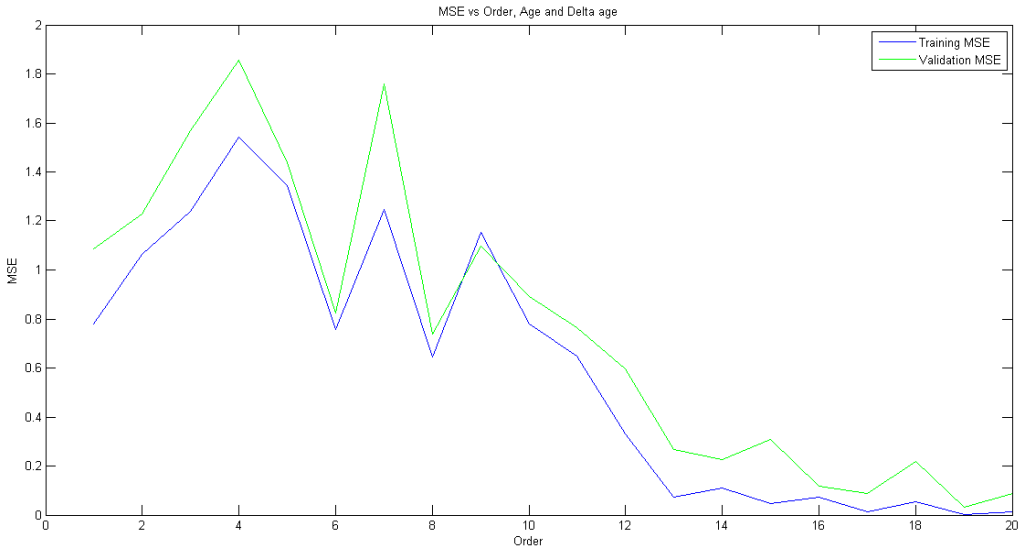

Figure 4.8: Training & Validation Performance vs. Order for the Standard MLP model (Age and Delta Age)

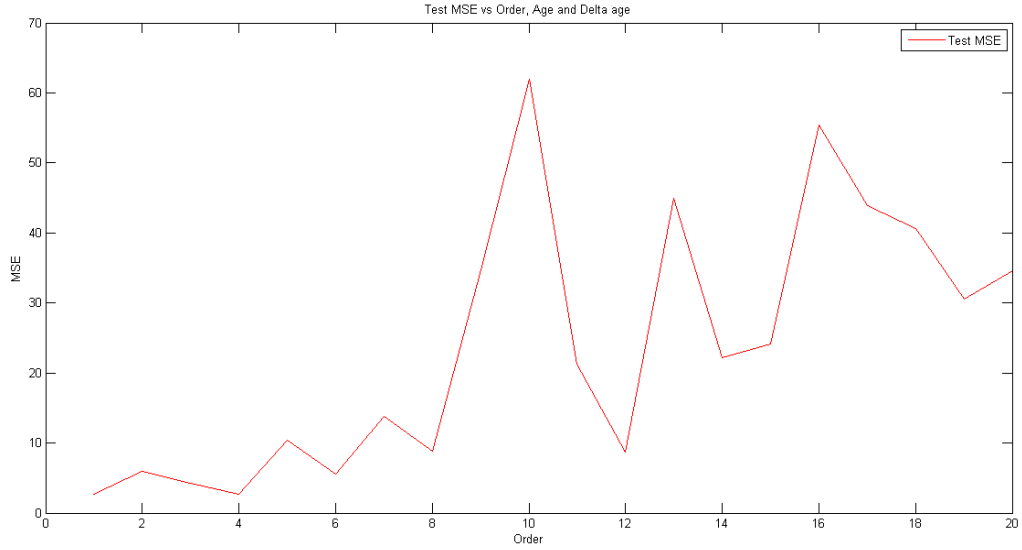

Figure 4.9: Test Performance vs. Order for the Standard MLP model (Age and Delta Age)

Model Order / Performance Measure	8 / Total Accuracy	8 / Converter Accuracy	4 / Non-Converter Accuracy
Training MSE	1.1303	1.1303	1.1338
Test MSE	8.7531	8.7531	6.6029
Validation MSE	1.3022	1.3022	1.3994
Total Accuracy (%)	67.2797	67.2797	65.9301
Non-Converter Accuracy (%)	60.1547	60.1547	63.8411
Converter Accuracy (%)	71.847	71.847	67.258

Table 4.6: Best Model Performances for Standard MLP Arch. Delta age only

Model Order / Performance Measure	12 / Total Accuracy	7 / Converter Accuracy	12 / Non-Converter Accuracy
Training MSE	0.3309	1.2485	0.3309
Test MSE	8.6384	13.8257	8.6384
Validation MSE	0.5951	1.7592	0.5951
Total Accuracy (%)	71.1888	68.4755	71.1888
Non-Converter Accuracy (%)	67.5926	59.6111	67.5926
Converter Accuracy (%)	73.3708	73.8539	73.3708

Table 4.7: Best Model Performances for Standard MLP Arch. with Age & Delta age

4.3 Standard deviation in predicting the conversion

The second performance measure we used is the standard deviation in predicting the conversion measured in years. After we have computed the accuracy in predicting the converters and non-converters, we take the subset of converters predicted correctly by the neural network. We then fitted a line to their predicted scores and saw where the line crossed the threshold of zero and compared it with actual conversion time based on their actual scores. With these two data, we computed the standard deviation in predicting the conversion. For those converters whose scores are all below the threshold, we set their first visit age as their conversion time. The following are the results of the standard deviation values for the best models in various architectures.

Model Order	4	20	4
Total Accuracy (%)	69.3244	68.6661	69.3244
Non-Converter Accuracy (%)	49.9141	45.3588	49.9141
Converter Accuracy (%)	82.1843	84.1109	82.1843
Deviation in Prediction Accuracy	0.6476	0.5467	0.6476

Table 4.8: Deviation in Prediction Accuracy for Localized Arch. with Delta age only

Model Order	3	11	20
Total Accuracy (%)	66.345	64.9383	65.9263
Non-Converter Accuracy (%)	39.0162	29.8729	39.5386
Converter Accuracy (%)	84.2437	88.0214	83.1958
Deviation in Prediction Accuracy	0.75	0.6944	0.7848

Table 4.9: Deviation in Prediction Accuracy for Localized Arch. with Age and Delta age

Model Order	8	8	4
Total Accuracy (%)	67.2797	67.2797	65.9301
Non-Converter Accuracy (%)	60.1547	60.1547	63.8411
Converter Accuracy (%)	71.847	71.847	67.258
Deviation in Prediction Accuracy	0.6565	0.6565	0.4814

Table 4.10: Deviation in Prediction Accuracy for Standard MLP Arch. with Delta age only

Model Order	12	7	12
Total Accuracy (%)	71.1888	68.4755	71.1888
Non-Converter Accuracy (%)	67.5926	59.6111	67.5926
Converter Accuracy (%)	73.3708	73.8539	73.3708
Deviation in Prediction Accuracy	0.6473	0.9707	0.6473

Table 4.11: Deviation in Prediction Accuracy for Standard MLP Arch. With Age and Delta age

4.4 Raw scores Vs. Fitted scores

Up till this point, we have used fitted scores for training the network and raw scores for evaluating the network. In this section, we study the performance of the networks when the other combinations were used.

4.4.1 Raw scores for training the neural network

Using raw scores for training the network means that we cannot extrapolate the scores to get the sixth visit score. In this case, the network performance was far better (in terms of prediction accuracy) when compared to using the fitted scores. Best accuracy was ~79% when compared to the 71% obtained when using fitted scores. The standard deviations in predicting when conversion will happen were also smaller. Table 4.10 gives the performance values when

using delta-age only and 4.11 gives it for the case when age was also used. Raw scores were used at the output of the network also. When fitted scores were used at the output instead of raw scores, no significant changes in the prediction accuracy were seen.

Model Order / Performance Measure	19 / Total Accuracy	8 / Converter Accuracy	7 / Non-Converter Accuracy
Training MSE	1.2945	1.2719	1.4702
Test MSE	1.6982	1.6853	1.9580
Validation MSE	1.4807	1.3643	1.4446
Total Accuracy (%)	78.87	77.06	77.34
Non-Converter Accuracy (%)	80.93	73.33	84.50
Converter Accuracy (%)	77.38	79.76	72.17
Standard Deviation in Prediction Accuracy	0.2607	0.2314	0.3196

Table 4.12: Best Model Performances for Localized Arch. with Delta age only (Raw score Training & Testing)

Model Order / Performance Measure	19 / Total Accuracy	19 / Converter Accuracy	15 / Non-Converter Accuracy
Training MSE	1.6392	1.6392	1.9920
Test MSE	2.1919	2.1919	2.7036
Validation MSE	1.6252	1.6252	1.8411
Total Accuracy (%)	75.61	75.61	67.13
Non-Converter Accuracy (%)	78.96	78.96	80.67
Converter Accuracy (%)	73.19	73.19	57.35
Standard Deviation in Prediction Accuracy	0.3437	0.3437	0.5689

Table 4.13: Best Model Performances for Localized Arch. with Age & Delta age (Raw score Training & Testing)

4.4.2 Raw scores for evaluating the performance of the network

For the test data, when we compute the prediction accuracy, we have a choice of using the raw score or the fitted score trajectories. We have plotted the network output trajectories for raw scores as well as fitted scores along with ground truth for a few patients whose conversion the network predicted correctly. As can be seen, the network actually tries to fit a line by itself. This behavior can be explained due to the nature of training data we used. The training was done based on a fitted line for the patients' scores. So, the network is trying to imitate the behavior.

Also, we saw a slight improvement in the prediction accuracy while using raw scores instead of fitted scores at the network output. This again can be explained by viewing the line fitting at the network's output as an addition of noise.

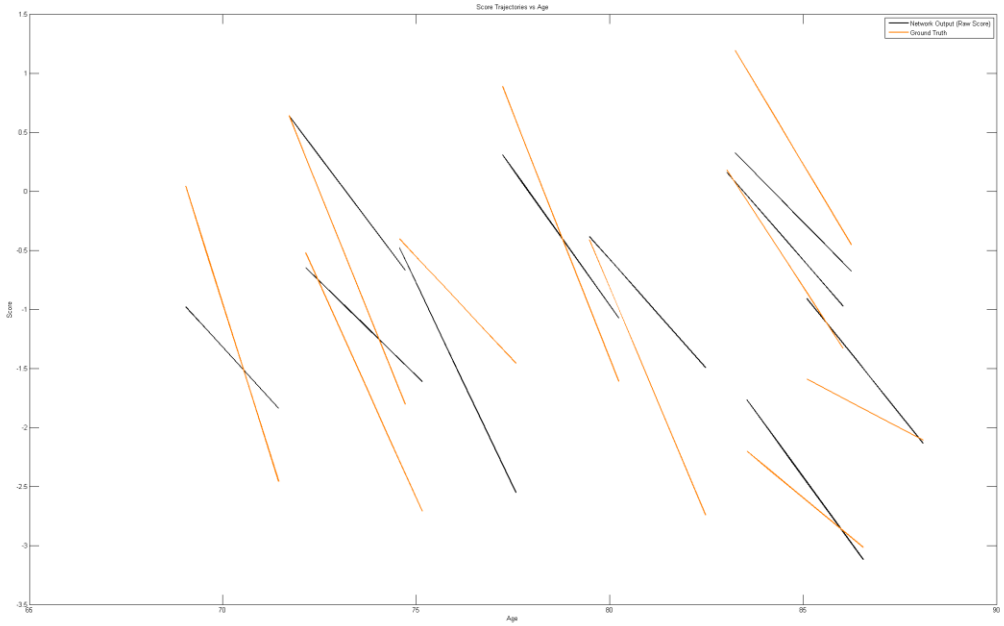

Figure 4.10: Fitted Score Trajectories vs. Age compared to Ground Truth

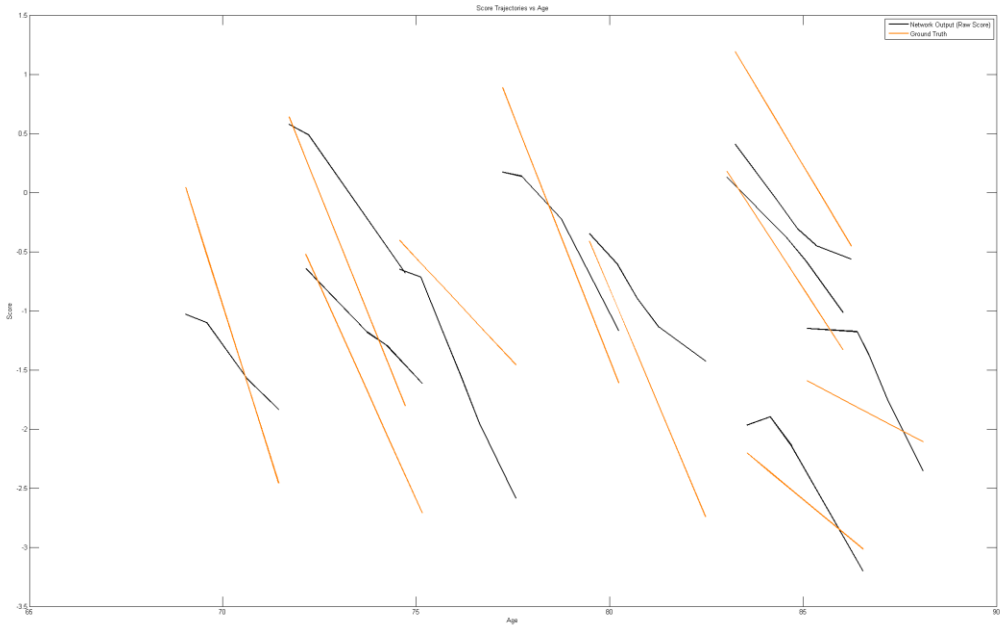

Figure 4.11: Raw score trajectories vs. Age compared to Ground Truth

4.5 Linear Mean Squares Approach

Since the accuracy for the linear architecture was high, we wanted to see how the performance would be if we were to use a Linear Mean Squares (LMS) Approach for this problem. For the fitted scores approach, we extrapolated the scores to the sixth visit and for the raw scores approach we just used what was available. The impact of having the age parameter was also studied. For each combination, we split the data into two halves randomly and repeated the experiment 20 times to get the average performance. The following table summarizes the results. Even though the standard deviation in predicting the conversion is slightly higher, the accuracy is far better than any other neural network architecture studied. The main difference between training a linear neural network and this approach is that here we directly jump to the global minimum instead of settling for some local minima.

Score Type / Age	Raw / Age included	Raw / Age not included	Fitted / Age included	Fitted / Age not included
Total Accuracy	80.88	77.78	73.99	69.55
Converter Accuracy	79.46	77.17	77.29	76.21
Non-converter Accuracy	82.69	78.36	68.40	59.57
Standard Deviation	1.53	1.31	0.48	0.59
Test Set MSE	1.06	1.49	2.20	2.78

Table 4.14: Accuracy obtained when using the LMS Approach

4.6 Diagnosis Vs. Prognosis

Once the prediction accuracies were studied for prognostic prediction of conversion to AD from MCI, we wanted to see how that compared to a diagnosis. By studying this, we could see how difficult it is to do a prognosis for this problem. We used standard MLP neural network architecture for this purpose and we did three types of experiments based on how we defined a patient as a converter or a non-converter. Note that, for diagnosis we used the MRI scans from all the visits for training and testing of the neural network.

4.5.1 Approach 1: Visit Accuracy

In the first approach, we disregarded the idea of a patient and computed the visit accuracy. To clarify, we used each visit by a patient as a data point. We trained the network using 50% of the data and tested it using the remaining 50%. Here, a visit is termed as a conversion visit if the score is below the threshold (0) and a non-conversion visit otherwise. As before, we computed the accuracy in predicting the converters, non-converters and also the total accuracy. Here, we cannot have the concept of standard deviation in predicting the conversion since the testing is visit-based and not patient based. Obviously, we expected to see a much better performance when compared to the prognosis performance. The results of the testing are shown in the following table (Table 4.15).

Model Order, Score Usage	1 / Raw Scores	2 / Fitted Scores
Training MSE	0.0834	0.1799
Validation MSE	0.1457	0.3058
Test MSE	0.1593	0.4386
Total Accuracy	92.35 %	87.29 %
Converter Accuracy	92.46 %	87.58 %
Non-converter Accuracy	92.29 %	87.15 %

Table 4.15: Best Performances for Visit-Accuracy Approach

4.5.2 Approach 2: Patient Conversion Accuracy

This approach is similar to the previous approach but instead of defining conversion visits or non-conversion visits, we defined converters or non-converters. Here, the definition of converter and non-converter is slightly changed when compared to the prognosis experiments. If a patient's score goes below the threshold for at least one visit, then the patient is classified as a converter. The results of this approach are shown in table 4.16.

Model Order / Score Usage	1 / Raw Scores	1 / Fitted Scores
Training MSE	0.2398	0.4582
Validation MSE	0.3174	0.5534
Test MSE	0.2791	0.6222
Total Accuracy	88.52 %	86.91 %
Converter Accuracy	83.33 %	81.85 %
Non-converter Accuracy	95.81 %	95.09 %

Table 4.16: Best Performances for Patient-Conversion-Accuracy Approach

4.5.3 Approach 3: Patient Conversion Accuracy while predicting the score trajectory

In this final approach, we gave the network brain volume information from all the six visits of a patient as a single input. That is, the network took in 600 inputs. For patients who had some of the visits missing, we did interpolation as well as extrapolation depending on which visit data was missing. The network outputted the scores for all six visits together. As before, we used the by-trajectory definition for classifying a patient as a converter or a non-converter. Note that, in this method we cannot use raw scores for training and testing the network since we need 600

inputs and none of the patients had the scan information for all 6 visits. The results of this approach are shown in table 4.17.

Model Order / Performance Measure	8 / Total Accuracy	20 / Non-Converter Accuracy	6 / Converter Accuracy
Training MSE	0.1857	0.0630	0.3104
Validation MSE	0.9117	1.4882	0.7878
Test MSE	1.0476	1.4819	0.9565
Total Accuracy	62.63 %	59.54 %	60.59 %
Converter Accuracy	52.93 %	28.13 %	56.93 %
Non-converter Accuracy	72.08 %	90.13 %	64.16 %

Table 4.17: Best Performances for Patient-Conversion-Accuracy Approach with Interpolation & Extrapolation

CHAPTER 5

CONCLUSIONS AND RECOMMENDATIONS

5.1 Accuracy

This neural network approach produced accuracy (best performance) of approximately 71% when using the standard MLP architecture and 69% when using the localized architecture (with fitted scores for training) and as high as ~79% when using the raw scores for training. This accuracy is slightly better than what was obtained when a linear-kernel SVM (~77%) was used with the region-based features (Aksu et al., 2011). Refer to tables 4.11, 4.4 & 4.12

It is interesting to note that the classification performance for a linear neural network (localized architecture) was 74%. And, when we viewed the problem as a Linear Mean Squares problem the accuracy obtained was as high as 80% and the standard deviation performance measure was slightly worse than what was obtained with fitted scores. Refer to tables 4.1 & 4.14

5.2 Age as a feature

For the localized architecture, using age (age during the first visit) as an input to the network did not help improve the performance. In fact, the performance was slightly poorer when age was included (69% as compared to 66% classification accuracy – tables 4.4 & 4.5). On the other hand, for the standard MLP architecture, using age helped improve performance slightly (71% as compared to 67% - tables 4.6 & 4.7). For the LMS approach, including age helped improve the performance by approximately 3%. Even though, the standard MLP neural network architecture had better classification accuracy, the test set MSE was comparably higher.

5.3 Difficulty in doing prognosis

When the MRI scans from all the visits were used (diagnosis), we could achieve accuracy as high as 92%. Comparing this to the 79% accuracy we obtained (with a reasonable standard deviation in predicting conversion) for prognosis tells us how difficult prognosis is for this particular problem.

APPENDIX

Creating a Custom Neural Network in MATLAB

It is very simple to create a standard MLP neural network in MATLAB. All we have to do is execute the following command:

```
net = newff(input_sample, output_sample, no_of_neurons);
```

This command will create a Multilayer Perceptron based on the three arguments provided to it. The size of `output_sample` will decide how many neurons there will be in the output layer. The `no_of_neurons` can be an array which decides the number of the hidden layers and the size of each layer. For example, `no_of_neurons` = [10, 5] means that there will be 10 neurons in the first layer which will be connected to the inputs and the second layer will have 5 neurons whose inputs will be connected to the outputs of the first layer and there will be an output layer whose size, as mentioned already, will be decided based on the size of the `output_sample`. By default, all the layers except the output layer will have a non-linear transfer function while the output layer will have a linear transfer function. For most cases, we do not need any customization to the network.

If one wishes to create a custom neural network, one has to start from scratch and has to set each and every setting. Reference 9 explains how to create a custom network using the Neural Network Toolbox Version 3.0. The current version of the toolbox is 6.0 and has several changes from the 3.0 version.

The following code is an example of how to create a custom network.

```
net = network;
net.numInputs = 100;

for i = 1 : 100
    net.inputs{i}.size = no_of_layers;
end

net.numLayers = 102;

for i = 1 : 100
    net.layers{i}.size = 1;
end

net.layers{101}.size = hidden_neurons;
net.layers{102}.size = 1;
net.biasConnect = ones(102,1);

for i = 1 : 100
    net.inputConnect(i,i) = 1;
end


for i = 1 : 100
    net.layerConnect(101, i) = 1;
end

net.layerConnect(102, 101) = 1;
```

```
net.outputConnect(102) = 1;

for i = 1 : 100
    net.layers{i}.transferFcn = input_txr_fcn;
end

net.layers{101}.transferFcn = hidden_txr_fcn;
net.layers{102}.transferFcn = output_txr_fcn;

net.initFcn = 'initlay';
for i = 1 : 102
    net.layers{i}.initFcn = 'initwb';
    net.biases{i}.initFcn = 'rands';
end

for i = 1 : 100
    net.layerWeights{101,i}.initFcn = 'rands';
end
net.layerWeights{102,101}.initFcn = 'rands';

net.performFcn = 'mse';
net.trainFcn = 'trainlm';
net.divideFcn = 'dividerand';
net.divideParam.trainRatio = 0.9;
net.divideParam.valRatio = 0.1;
net.divideParam.testRatio = 0.0;
net.plotFcns = {'plotperform','plottrainstate'};
```

After creating an empty network (which is an object), we have to set all its parameters. The first parameter to set is the Number of Input Sources. This can be done by setting the parameter `net.numInputs` to the required value. In our case, we have a total of 100 input sources. It is important to note that this parameter sets the number of input sources and not the size of the input array. The number of elements in the input vector can be set by setting the parameter `net.inputs{i}.size` where i varies from 1 to `net.numInputs`. In our case, it can be 2 or 3 depending on whether age input is included and we have 100 such input sources.

The next parameter of interest is `net.numLayers`. The definition of layer here is not exactly the same as we have used it before. When we have only one input source, they are the same. But, if we have more than one input source, then the definition will change. In our case, we have a total of 100 input sources. Also, we have a hidden layer and an output layer. We have one layer for every input source. So, in this case we have a total of 102 layers.

The next parameter decides how many neurons are there in each layer. This can be done using the parameter `net.layers{i}.size`. To make the neurons have a bias input, we have to set the value of `net.biasConnect` connect for all the layers.

Next, we make the connections between the input sources & the input layer, input layer & hidden layer and output layer and output link. These can be done using `net.inputConnect`, `net.layerConnect` and `net.outputConnect`. To set the transfer function for the layers, we have to use `net.layers{i}.transferFcn`. For a non-linear transfer function, set it to `tansig` and for a linear transfer function, set it to `purelin`.

Next step is to configure how the weights and biases are initialized. For this, first set `net.initFcn` to `initlay`. Then, for every layer set the parameter

net.layers{i}.initFcn and net.biases{i}.initFcn. The layer initialization function can be set to initnw or initwb. More information on those functions can be found in Ref. 8. Also, set the initialization function for the weights between layers using net.layerWeights{j,i}.initFcn where layers j's output goes to input of layer i. These parameters can be set to rands which initializes them to random values between -1 and 1.

Next, we have to set the performance function which is used during training and validation. To use mean squared error, set net.performFcn to mse. For setting the training algorithm, configure the parameter net.trainFcn. The most commonly used training function is trainlm.

Next, to configure what percentage of the data set should be used for training, validation and testing, set net.divideParam.trainRatio, net.divideParam.valRatio and net.divideParam.testRatio.

Finally to configure what plotting functions should be invoked during the training and once the training is done, set the value of net.plotFcns. For example, set it to plotperform which plots the performance function as a function of epochs for training, validation and test set. If you set it to plottrainstate, you will see how the mse decreases epoch by epoch as training progresses. One can also have both the plotting functions together.

BIBLIOGRAPHY

1. Alzheimer's Disease Neuroimaging Initiative (ADNI), www.loni.ucla.edu/ADNI www.adni-info.org
2. Yaman Aksu, David J. Miller, George Kesidis, Don C. Bigler and Qing X. Yang, "An MRI-Derived Definition of MCI-to-AD Conversion for Long-Term, Automatic Prognosis of MCI Patients", *CSE Tech. Report*, 2011.
3. N. Schuff, D. Tosun, P.S. Insel, G.C. Chiang, D. Truran, P.S. Aisen, C.R. Jack, Jr., M. W. Weiner, the Alzheimer's Disease Intiative, "Non-Linear time course of brain volume loss in cognitively normal and impaired elders", *Neurobiology of Aging*, 2010.
4. Simon Haykin, *Neural Networks: A Comprehensive Foundation*. Second Edition, Prentice Hall International, Inc., 1999.
5. Richard O. Duda, Peter E. Hart and David G. Stork, *Pattern Classification*. Second Edition, John Wiley and Sons, New York, 2001.
6. Stuart Russell and Peter Norvig, *Artificial Intelligence A Modern Approach*. p. 578.
7. Derrick Nguyen and Bernard Widrow, Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights. *Proceedings of the International Joint Conference on Neural Networks*, 3:21–26, 1990.
8. Howard Demuth, Mark Beale and Martin Hagan, *Neural Network Toolbox™, User's Guide* The MathWorks, Inc., Natick, MA, revised for version 6.0.3 edition, September 2009. http://www.mathworks.com.
9. C. Davatzikos, P. Bhatt, L. M. Shaw, K. N. Batmanghelich, J. Q. Trojanowski, "Prediction of MCI to AD conversion, via MRI, CSF biomarkers, and pattern classification", *Neurobiology of Aging*, 2010.

10. Y.Y. Chou, N. Leporé, C. Avedissian, S. K. Madsen, X. Hua, C. R. Jack Jr., M. W. Weiner, A. W. Toga, P. M. Thompson, and the Alzheimer's Disease Neuroimaging Initiative, "Mapping Ventricular Expansion and its Clinical Correlates in Alzheimer's Disease and Mild Cognitive Impairment using Multi-Atlas Fluid Image Alignment", *Proc. SPIE*, vol.7259, 725930, 2009.
11. C. Misra, Y. Fan, C. Davatzikos, "Baseline and longitudinal patterns of brain atrophy in MCI patients, and their use in prediction of short-term conversion to AD: Results from ADNI", *NeuroImage* 44, pp.1415-1422, 2009.
12. Y.Wang, Y. Fan, P. Bhatt, C. Davatzikos, "High-dimensional pattern regression using machine learning: From medical images to continuous clinical variables", *NeuroImage* 50, pp.1519-1535.
13. Somesh Srivastava, *Introduction to the Matlab Neural Network Toolbox 3.0* http://www.eecs.umich.edu/~someshs/nn/matlab_nn_starter.htm
14. Clinical Dementia Rating: http://www.alz.washington.edu/NONMEMBER/cdr2.html
15. Folstein MF, Folstein SE, McHugh PR. ""Mini-mental state". A practical method for grading the cognitive state of patients for the clinician". *Journal of psychiatric research* **12** (3): 189–98, 1975.
16. D.J. Miller, Private Communication, February 28, 2011