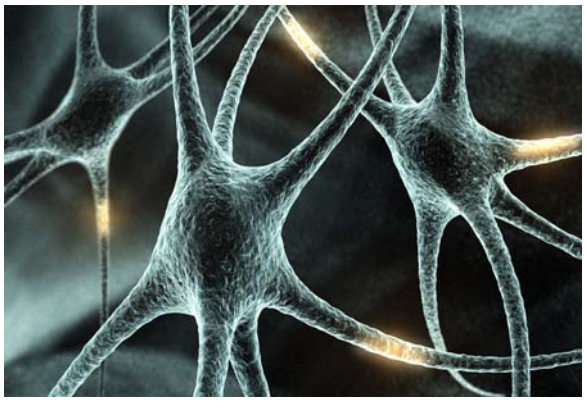
Technical University of Crete
Department of Electronics & Computer Engineering
Systems Division

Diploma Thesis

Use of artificial neural networks in sequential models of membrane proteins of the SWISS-PROT database for the detection of transmembrane segments



by

Athanasios C. Kalpakas
(R.N. 2000030088)

Chania, July 2006

Diploma Thesis

July 17, 2006

Use of artificial neural networks in sequential models of membrane proteins of the SWISS-PROT database for the detection of transmembrane segments

Athanasios C. Kalpakas
Dept. of Electronics & Computer Engineering,
Technical University of Crete
(R.N. 2000030088)

supervised by
Prof. Manolis Christodoulou,
Technical University of Crete

Technical University of Crete
Department of Electronics &
Computer Engineering

Systems Division
Department of Electronics &
Computer Engineering

Technical University of Crete

Department of Electronics and Computer Engineering

The undersigned hereby certify that they have read and recommend to the Faculty of Undergraduate Studies for acceptance a diploma thesis entitled "Use of artificial neural networks in sequential models of membrane proteins of the SWISS-PROT database for the detection of transmembrane segments" by Athanasios C. Kalpakas in partial fulfillment of the requirements for the degree of Bachelor of Science.

Dated: July 2006

Supervisor: _____
Prof. Manolis Christodoulou

Readers: _____
Prof. Michalis Zervakis

_____
Assoc. Prof. Vangelis Georgiou

Technical University of Crete

Department of Electronics and Computer Engineering

Date: **July 2006**

Author: **Athanasios C. Kalpakas**

Title: **Use of artificial neural networks in sequential models of membrane proteins of the SWISS-PROT database for the detection of transmembrane segments**

Department: **Electronics and Computer Engineering**

Degree: **B.Sc. (Hons)** Convocation: **July** Year: **2006**

To my parents, Christos and Amalia…

Acknowledgements

This very interesting diploma thesis was completed after several hundred hours of thoughtfulness, brainstorming, experiments, tests and creative work. It was supported by the Systems division of Electronics and Computer Engineering (ECE) Department of the Technical University of Crete under the supervision of Professor Manolis Christodoulou. Without his constant encouragement and confidence in me it would have been impossible to finish.

My gratitude goes to Giorgos Ioannidis, MSc in Electronics and Computer Engineering, for his enlightening discussions and constant support without which the progress of this work would have been at stake. Of inestimable value for my study was Konstantia Zarkogianni, PhD-candidate in Electronics and Computer Engineering for sharing her knowledge and providing flexible ideas on various problems.

Special thanks go to my sister Penelope Kalpaka, Biologist, MSc in Environmental Engineering, for sharing her valuable scientific experience with me. Moreover, I feel the urge to thank all my friends for believing in my potentials and for the nice and revitalizing moments we shared during work breaks. Last but not least, I would like to thank my father, Christos Kalpakas, my mother Amalia, my brother Elias and my sister Penelope for their psychological support, patience and confidence. I am proud to have such a family which gave me valuable advice, inspiration and support when I needed it most.

Abstract

Geneticists try to find the truth behind the genomes that contain the blueprint for all parts of life's machinery. All these genes store 'digital' information represented in serial format where every group of digits create a specific type of entity. Next challenge will be corresponding DNA data to the various types of proteins; thus deriving meaningful knowledge for the understanding of biological systems. Proteins will give us answers for the human evolution while others, such as membrane proteins, will reveal the developing mechanisms of diseases, such as muscle disease, deafness, blindness, diabetes, arthritis, and cancers.

Currently, artificial neural networks are used to model human organs and predict diseases from digital scans. Transmembrane segment topology is crucial for visualizing protein's folding into space; thus understanding it's role inside and outside of the cell. Unfortunately conventional identification processes use laboratory methods which are slow; most of them are inaccurate while still not using all the available information and cost a fortune. For that reason, a feed forward neural network was constructed using the back-propagation algorithm aiming at the prediction of transmembrane segments (transmembrane helices) in membrane proteins from a single amino acid sequence. More than 300 proteins were used to train the network. Variable input and output lengths urged the implementation of a sliding-window technique, converting variable protein lengths to a fixed number of inputs/outputs. Several configurations were tested and optimal parameters, such as learning rate, hidden units and window size were

elaborated. Finally, the system was evaluated based on its ability to efficiently predict the topology of new, unknown membrane proteins.

Περίληψη

Οι Γενετιστές προσπαθούν να ανακαλύψουν τα μυστικά που κρύβουν τα γονίδια τα οποία αποτελούν τα λεπτομερή προσχέδια για όλους τους μηχανισμούς της ζωής. Όλα τα γονίδια αποθηκεύουν «ψηφιακή» πληροφορία, η οποία αναπαριστάται σε σειριακή μορφή. Κάθε ομάδα ψηφίων μπορεί να παράγει μια διαφορετική οντότητα. Η επόμενη πρόκληση θα είναι να γίνει η πλήρης αντιστοίχιση του γενετικού υλικού με τους διάφορους τύπους πρωτεϊνών που κωδικοποιεί. Κάποιες πρωτεΐνες θα μας βοηθήσουν να δώσουμε απαντήσεις για την κατανόηση πολύπλοκων βιολογικών συστημάτων, ενώ άλλες, όπως οι διαμεμβρανικές πρωτεΐνες θα μας βοηθήσουνε να κατανοήσουμε καλύτερα τους μηχανισμούς που διέπουν τις διάφορες αρώστειες, όπως μυοσκελετικές παθήσεις, κώφωση, τύφλωση, διαβήτης και καρκίνος.

Προς το παρών, τα τεχνητά νευρωνικά δίκτυα χρησιμοποιούνται για να μοντελοποιούν ανθρώπινα όργανα ή για να κάνουν πρόγνωση ασθενειών μέ τη βοήθεια ψηφιακών σαρώσεων. Η θέση των διαμεμβρανικών τμημάτων είναι πολύ σημαντική για την αναπαράσταση της τριτοταγούς δομής της πρωτεΐνης με σκοπό την κατανόηση του ρόλου της μέσα αλλά και έξω από το κύτταρο. Δυστυχώς οι συμβατικές μέθοδοι ανίχνευσης διαμεμβρανικών τμημάτων εντάσσονται στα πλαίσια εργαστηριακών πειραμάτων, είναι πολυδάπανες και οι περισσότερες είναι ανακριβείς μιας και δεν μπορούν να εκμεταλλευτούν τον όγκο των πληροφοριών. Γι' αυτό τον λόγο, ένα κατασκευάστηκε ένα feed-forward νευρωνικό δίκτυο με τη χρήση του αλγορίθμου back-propagation, με στόχο την πρόβλεψη των διαμεμβρανικών τμημάτων (διαμεμβρανικών ελίκων) στις μεμβρανικές πρωτεΐνες, με βάση την

αλληλουχία αμινοξέων. Περισσότερες από 300 προτεΐνες χρησιμοποιήθηκαν για να εκπαιδεύσουν το δίκτυο. Το πρόβλημα των μεταβλητών μηκών των πρωτεϊνών αντιμετωπίστηκε με την τεχνική του κυλιόμενου-παραθύρου, μετατρέποντας τα μεταβλητά μήκη των πρωτεϊνών σε σταθερό αριθμό εισόδων και εξόδων. Αρκετές διατάξεις δοκιμάστηκαν και οι βέλτιστοι παράμετροι ρυθμίστηκαν, όπως ο ρυθμός εκπαίδευσης, οι κρυφοί νευρώνες και το μέγεθος του παραθύρου του νευρωνικού. Τελικά, το σύστημα αξιολογήθηκε με βάση την ικανότητά του να προβλέπει αποτελεσματικά την ακριβή θέση όλων διαμεμβρανικών τμημάτων σε μεμβρανικές πρωτεΐνες.

Contents

Chapter 1

Introduction

DNA is the centre of evolution in all living organisms and it is organized in chains of pairing amino acids. These chains form the chromosomes, which are responsible for the shape, structure and function of an organism, from tiny eukaryotes cells to animals and humans. DNA decoding, based on the genetic code, creates different types of proteins which are major functional units of the cells. Proteins differ in type and role according to their DNA 3D folding in space. Membrane proteins, for example, gather unique characteristics. Studying their role, shape and behavior is critical for many diseases such as cancer. This folding is still a major predictive problem in biology and systems biology.

However, before we move on to that level, it is crucial to identify the primary structure of the protein and its transmembrane parts. Identification is a very costly, problematic and time-consuming procedure due to the vast amount of information and the need for accuracy. Geneticists and biologists used to work on an extremely reduced data set for weeks until they would come up with some convincing results.

Nowadays, the growth of Bioinformatics, biotechnology and Systems Biology offer advanced tools and algorithms for classification, pattern recognition and prediction problems; Neural networks are an inextricable associate of these tools. Algorithms would accept a set of unknown protein inputs and would output the exact topology of its TM segments. This diploma thesis is based on the same algorithmic pattern and deals with the detection of TM segments in membrane

proteins. Using neural networks, the detections of TM segments is a simple task: no more endless experiments are needed, the cost is minimized and the vast amount of information can at last be exploited.

This project has six chapters: it begins with biology fundamentals: a description of the genetic material, how DNA is decoded into amino acids and how amino acids connect to each other for the synthesis of proteins. The genetic code is presented to explain protein structure and membrane proteins are introduced. The potential of a digital representation of DNA and protein sequences is realized. Chapter 3 initially describes the transition from biological to artificial neurons and then focuses on the artificial neural networks. Basic principles, architectural models and main function parameters are thoroughly analyzed. Back-propagation algorithm is extensively described as it will be the basic learning algorithm used for training. Chapter 4 describes the problem core, main emergent issues and suggested solutions. It explains the suggested information processing plan and describes the architecture and basic parameters of the artificial neural network used as the basis of our predicting system. Optimization methods are discussed at the end of this Chapter. Chapter 5 analyzes the overall results and evaluates system's performance by examining the mean square error, sensitivity and specialty. In Chapter 6, final remarks and conclusions are reported and future perspectives are examined.

Chapter 2

DNA decoding

Decoding of the DNA that constitutes the human genome has been widely anticipated. Since the first bacterial genome sequence completion in 1995 [1] and the first eukaryote (yeast) in 1996 [2] scientists try to find the truth behind the genomes that contain the blueprint for all parts of life's machinery. In 2003 researchers managed to identify all the approximately 25.000 genes in human DNA [3]. All these genes store 'digital' information represented in serial format where every group of digits create a specific type of entity. In the case of the DNA we have the four letter coding: A, T, C, G (corresponding to the four bases) while in protein level we use the twenty letter coding (corresponding to the twenty amino acids). After having revealed the sequences of human DNA the next challenge will be corresponding DNA data to proteins; thus deriving meaningful knowledge for the understanding of biological systems. This will give us answers for the human evolution, the developing mechanisms of diseases, such as muscle disease, deafness, blindness, diabetes, arthritis, and cancers. Furthermore, we will be able to understand the interplay between the environment and heredity in defining the human condition.

2.1 DNA structure

DNA is a macromolecule created by the union of smaller molecules named nucleotides. Each nucleotide consists of three sub-groups: a phosphate group, a base containing nitrogen attached to a 5-carbon sugar (deoxyribose). (Figure 2.1) Several nucleotides are joined in ester links named phosphodiester bonds forming a chain of deoxyribose nucleic acid (DNA).

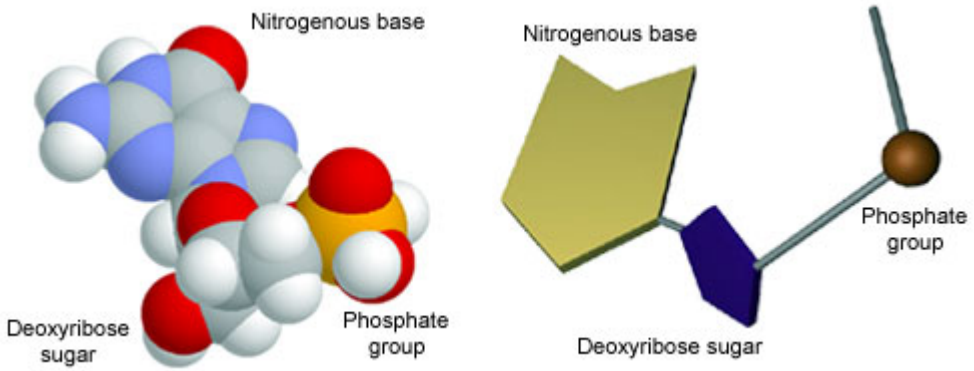


Figure 2.1 *Space-filling visualization (left) and 3D representation of a nucleotide (right)*

These bonds link the hydroxyl on the 3'- carbon pentose of the first nucleotide with the phosphate group which is linked on the 5'- carbon pentose of the next nucleotide in order to form the DNA backbone. The first nucleotide of every polynucleotide chain always has a free phosphate group connected on 5'- carbon pentose and the last nucleotide always has a free hydroxyl on the 3'- carbon pentose (Figure 2.2). For that reason, the first end of polynucleotide chain is called 5'-end and the other end is called 3'-end.

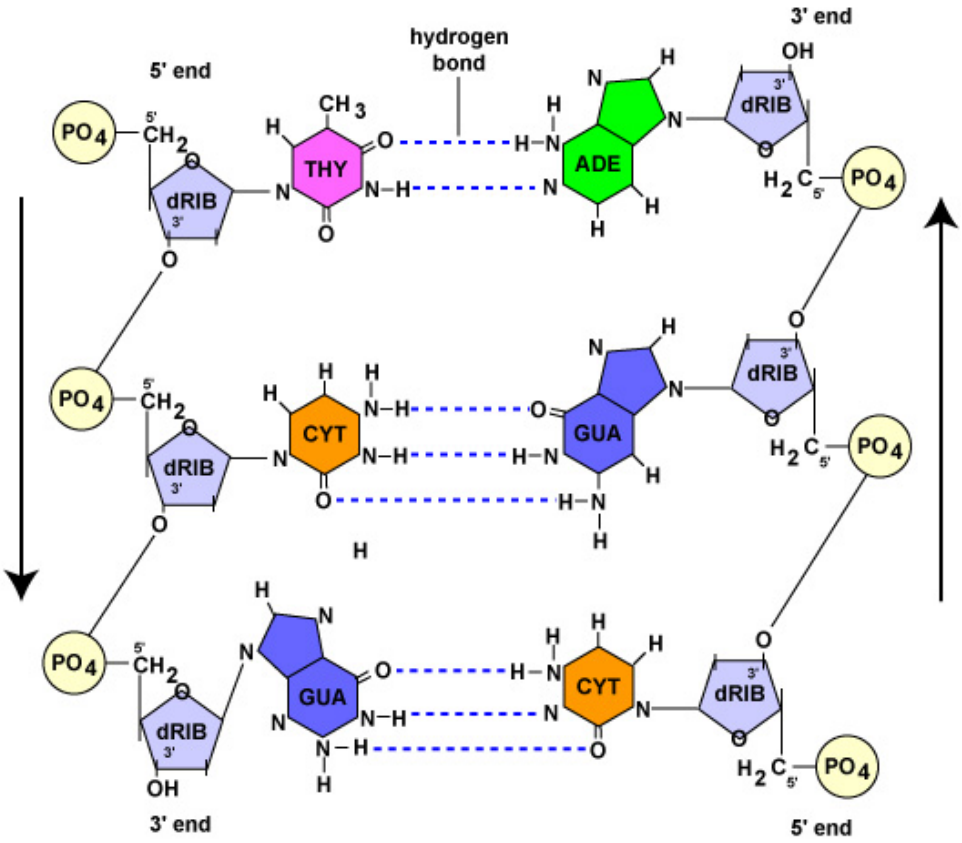


Figure 2.2 *Orientation of the two nucleotide chains that form the molecule of DNA. The left chain has a 5' – 3' orientation and the complementary clone has a 3' – 5'*

James Watson and Francis Crick (1953) suggested a model to explain the structure of the DNA, presuming that the molecule of the DNA consists of two polynucleotide chains each coiled round the same axis to form a double helix in a right-handed spiral [7]. The double helix has a steady backbone with alternating sugar-phosphate sequence. The backbone is external and has hydrophilic behavior while the bases are found in the interior of the structure of DNA showing hydrophobic behavior.

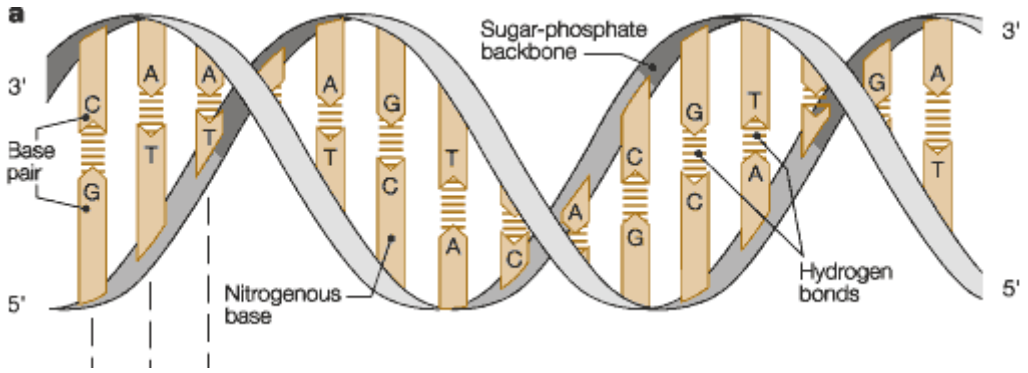


Figure 2.3 *DNA double helix*

Four different nitrogenous bases extend away from chains, and stack atop each other, like the steps of a spiral staircase (Figure 2.3). These bases are: 'A' for adenine (a purine), 'C' for cytosine (a pyrimidine), 'G' for guanine (a purine) and 'T' for thymine (a pyrimidine) (Figure 2.4). The two polynucleotide chains run in opposite direction because the direction (or polarity) of the first chain is 5'-3' (top to bottom) and the direction of the second chain is 5'-3' (bottom to top). Furthermore, one chain is complementary to the other meaning that if a cytosine forms one member of a pair, on either chain, then the other member must by guanine; similarly for adenine and thymine. The above expression forms the rule of complementation of bases. Consequently, the two chains of the DNA are complementary (Figure 2.2).

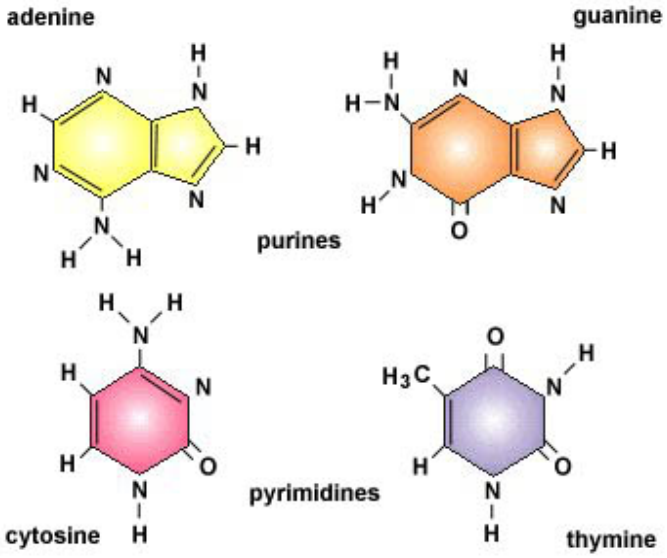


Figure 2.4 *The four nitrogenous bases which are found in DNA molecule*

In addition, hydrogen bonds link the pair of bases of the nucleotides. Therefore, the macromolecule keeps the structure of the right-handed direction, double helix in space. Within the DNA double helix, Adenine forms two hydrogen bonds with Thymine on the opposite strand, and Guanine forms three hydrogen bonds with Cytosine on the opposite strand [4] (Figure 2.2).

DNA is the genetic material of every cell and most of the viruses. In human cells DNA is organized in structures, called 'chromosomes' (Figure 2.5). Within the DNA molecule there is genetic information that determines the characteristics of an organism which is organized in functional units, called 'genes' [6]. DNA molecule preserves and transfers the genetic information from cell to cell and from organism to organism due to the feature of self-duplication. The process of DNA copying is feasible because of the complementation of bases. The expression of the genetic information is achieved by controlling the synthesis of proteins.

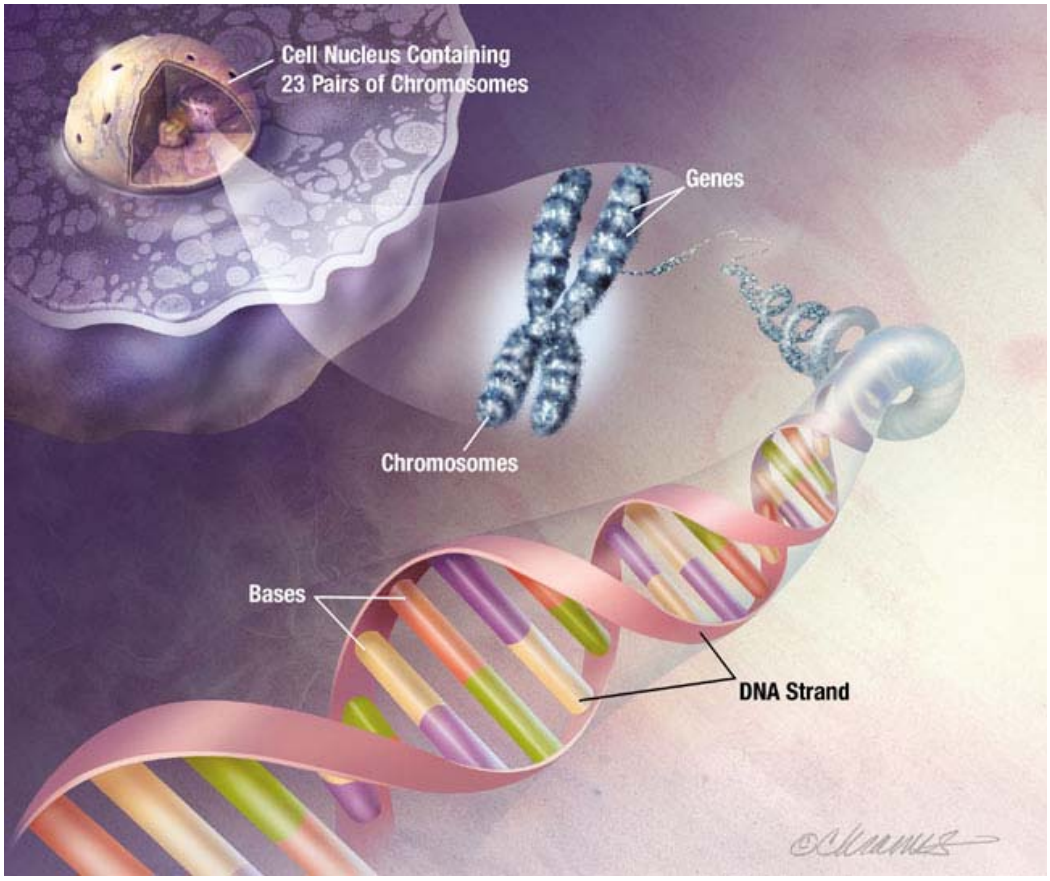


Figure 2.5 *DNA molecule is coiled and packed to form a chromosome inside a nucleus.*

2.2 Proteins

Referring to shape and functions, proteins are the most popular and multidimensional macromolecules. These molecules are created by the union of smaller elements called 'amino acids'. Amino acids are joined to other amino acids by homopolar bonds (Figure 2.6).

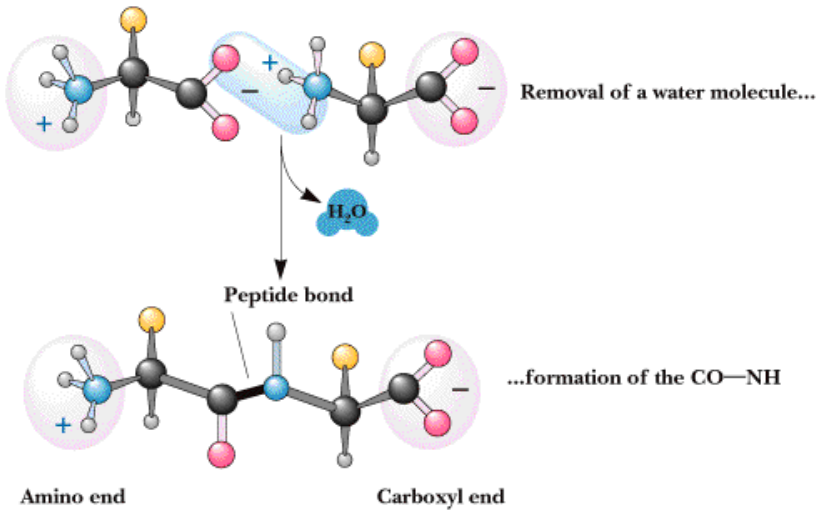


Figure 2.6 *Synthesis (polymerization) of two amino acids forming a dipeptidic chain after the creation of peptidic bond*

Twenty different types of amino acids are combined together to produce each time a different sequence; hence producing a vast amount of variant proteinic molecules. Amino acids molecule consists of two functional groups, a constant and a variable. The constant functional group holds a hydrogen atom, a carboxylic acid (-COOH) and an amino (-NH2) attached to the same tetrahedral carbon atom, while the variable group holds a distinct R-group [7]. Distinct R-group has a different chemical structure for each of the twenty amino acids (Figure 2.7).

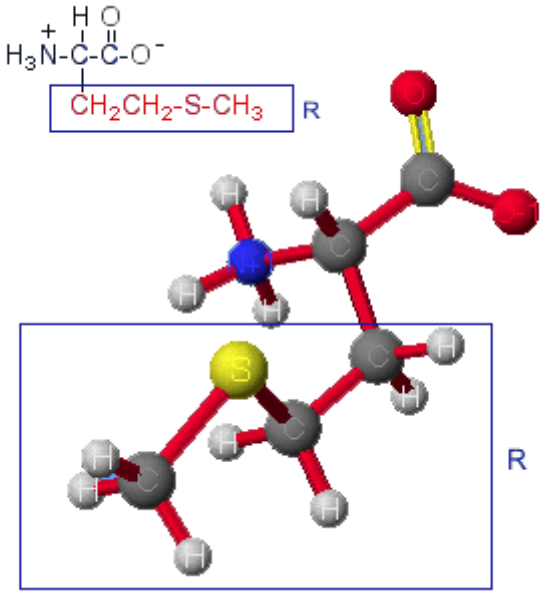


Figure 2.7 *3D model of the molecule of the amino acid methionine (met)*

Apart from primary structure, proteins as well as polynucleotides have a quaternary structure (three-dimensional structure involving the association of two or more polypeptide chains into a multi-subunit structure). It is this three-dimensional structure that allows proteins to function. The final folding of the proteinic molecule in space which will specify the proper function of the protein is defined by the amino acids sequence in the peptide chain and stabilized by peptide bonds among amino acids R-groups [8].

There are more than thirty thousand different proteins in humans, each one of them having a special biological role to play. Metabolism, multiplication and other major cell functions are based upon the action of these amazing molecular tools [7].

2.3 Genetic code

Information stored in genes is transferred from the nucleus of the cell to outer cell organelles with the help of another bio-molecule, called 'messenger RNA' (or m-RNA), so as the protein synthesis can be accomplished. Genetic

information stored in the DNA molecule is transcribed into m-RNA due to the complementation of nucleotide bases. Then the single helix m-RNA strand gets out of the cell's nucleus and connects to sub-cell organelles called 'ribosomes'. Ribosomes (found on the endoplasmic reticulum and free within the cell) are responsible for the synthesis of proteins (Figure 2.8). Ribosomes scan the m-RNA chain and read the sequence of bases responsible for the amino acids sequence. The correlation between bases and amino acids is based on a code mostly known as the genetic code (Figure 2.9). Therefore, during protein synthesis a translation between "base language" and "amino acid language" is taking place.

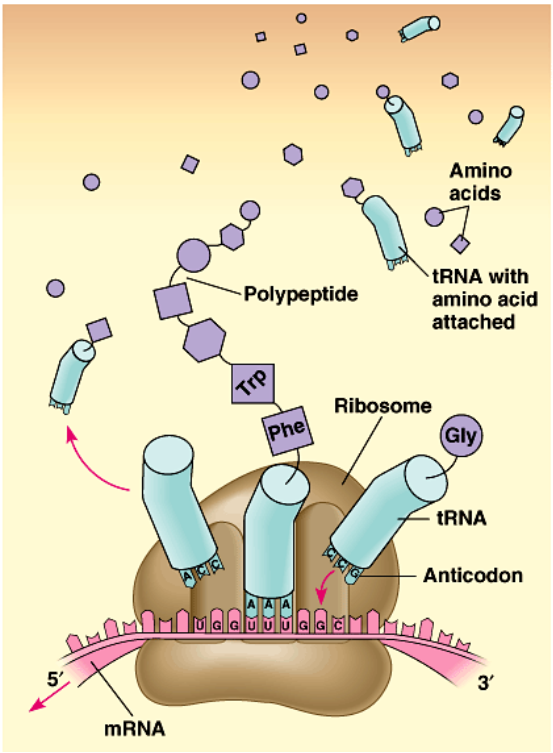


Figure 2.8 *m-RNA molecule attached to a ribosome is translated for the synthesis of a peptide chain (proteins usually consists of more than one peptide chain)*

According to the genetic code, three nucleotides (triplet) correspond to a single amino acid due to the fact that there are twenty different types of amino acids in proteins that must come up after the combination of the four nucleotides ($4^3 = 64$). All possible combinations that come up when three nucleotides define a single amino acid is sixty four; thus the twenty different types of amino acids found in living organisms are over covered by the above combination. For that reason,

the Genetic Code is also called 'Triplet Code'. Three nucleotides form a codon and the last decodes an amino acid [7].

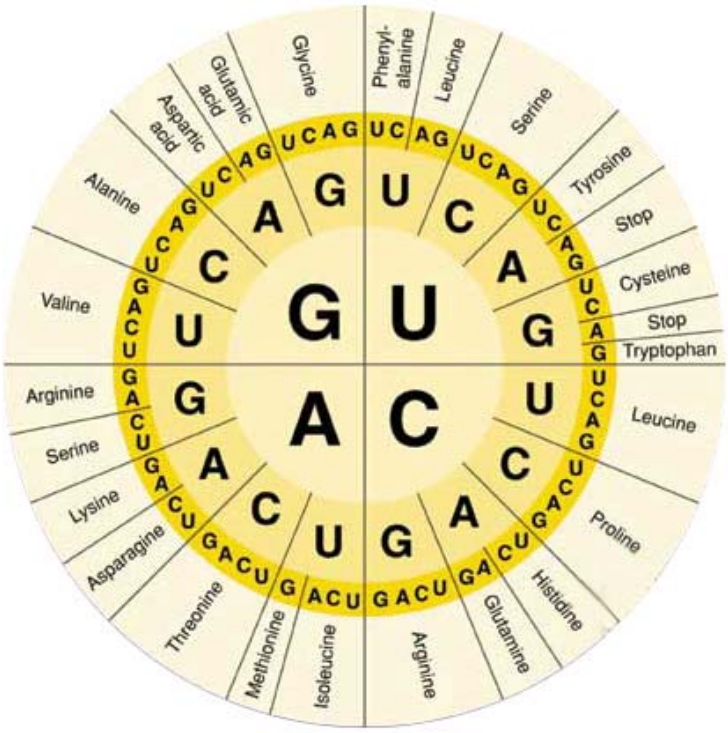


Figure 2.9 *The Genetic Code – Triplets (codons) correspond to amino acids*

Genetic code is continuous meaning that m-RNA is read constantly in groups of three nucleotides without any dropped nucleotides. Furthermore, every nucleotide belongs to a single codon. All living organisms have the same genetic code; therefore the genetic code is characterized as universal. Another characteristic of the genetic code is that it is degenerated with the exception of two amino acids (methionine and tryptophan); the other eighteen amino acids are encoded by two to six different codons. Codons which encode the same amino acid are called 'synonyms'. Genetic code has a start and a stop codon. The start codon is AUG (valid for all organisms) and encodes methionine. There are also three end codons: UAG, UGA and UAA. Their presence in the molecule of m-RNA marks the termination of the synthesis of the polypeptide chain. Nevertheless, no amino acid is encoded by the end codons. The term 'codon' refers not only to m-RNA but also to the gene that produces it. For example, start

codon AUG corresponds to start codon ATG of the chain of DNA on which the gene is located.

As previously mentioned, the segment of a gene and its equivalent m-RNA segment that encodes a polypeptide chain starts with a start codon and ends with an end codon. In superior species, genes are discontinuous meaning that between the start and end codon of a gene, there are nucleotide sequences that interpolate among genomic DNA segments, holding no information. The above "worthless" segments are called 'introns' whereas the information source segments are called 'exons' [6]. During the activation of a gene for the production of protein introns are neutralized through a process called 'splicing'. Figure 2.10 shows an example of a protein genesis from a gene that contains introns and exons. The whole segment of the DNA that consist the gene transcribes itself into a m-RNA molecule. Right after this precursor m-RNA molecule leaves the nucleus and before it reaches the ribosomes, the process of "splicing" takes place. All introns are cut off and all exons are connected. Finally, a new (mature) m-RNA molecule is constructed with a start codon on one end and an end codon on the other end.

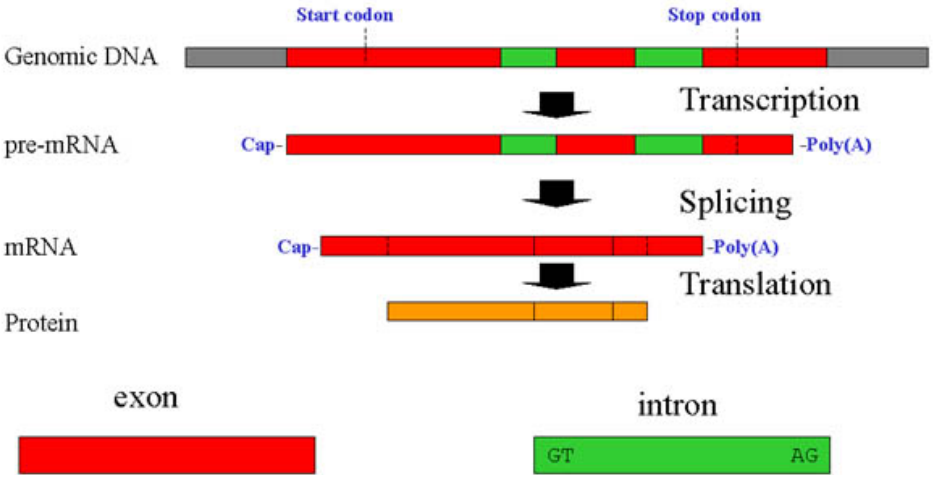


Figure 2.10 *Splicing procedure*

Every DNA segment which is activated for decoding in order to produce the proper protein is a result of the combined presence or absence of appropriate controlling proteins that usually link at the ends of genes [5]. It is worthy of note that the synthesis of proteins is an economical procedure because a cell is able to

produce vast amounts of multiple proteins from a single gene through the procedure of 'alternative splicing' (more than one third of all human genes may be affected by alternative splicing [9]) (Figure 2.11).

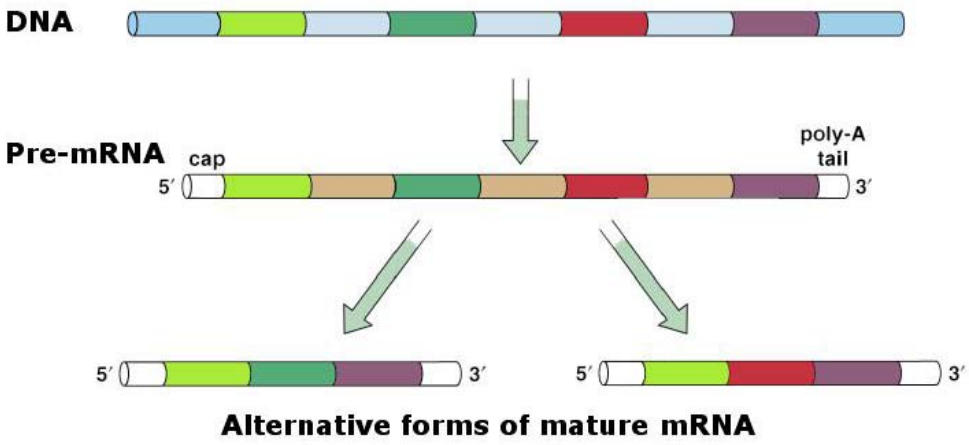


Figure 2.11 *Alternative splicing procedure*

Primary goal of bioinformatics is to identify the role of every nucleotide segment of DNA chain. Furthermore, knowing the role of specific amino acid segments and the implementation of the genetic code via reverse procedure we can identify the nucleotide regions and the genes that will eventually produce a specified protein. This project follows the above direction aiming at the detection of transmembrane segments (or helices) in unknown proteins.

2.4 Transmembrane proteins

The membrane of a cell consists of lipids and proteins also known as membrane proteins. There are two types of membrane proteins: integrals and peripherals (Figure 2.11). Integral membrane proteins that span across the lipid bilayer are called 'transmembrane' whereas other proteins are peripherally linked to the lipid bilayer that consist the cell membrane.

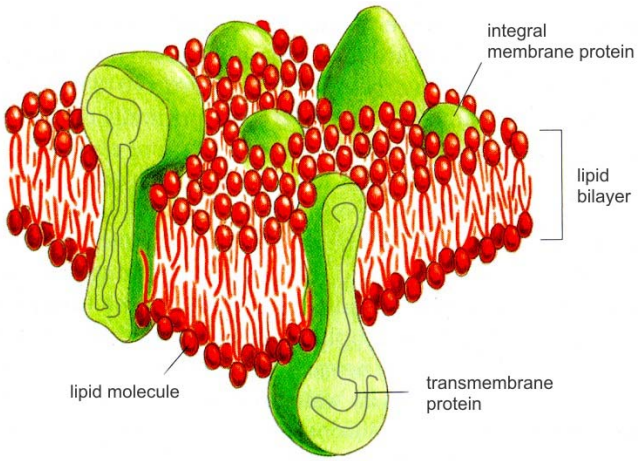


Figure 2.11 *Model of a cell membrane showing the lipid bilayer and membrane proteins crossing it*

The most interesting type of proteins is the integral. Proteins can penetrate the under layer more than once. For example, according to Figure 2.12, 'A' protein has one transmembrane segment, 'B' protein has four, 'C' protein has six and 'D' protein has four. However, the hydrophobic domain of the protein resides in the oily core of the membrane, while hydrophilic domains protrude into the watery environment inside and outside the cell. Transmembrane proteins often have their N-terminal on the exoplasmic face and their C-terminal on the cytoplasmic face. Many transmembrane proteins have multiple membrane spanning alpha helix segments which anchors them to the membrane. The biological role of these proteins is to form a hydrophilic (polar) channel where specific ions and molecules can pass through it [6].

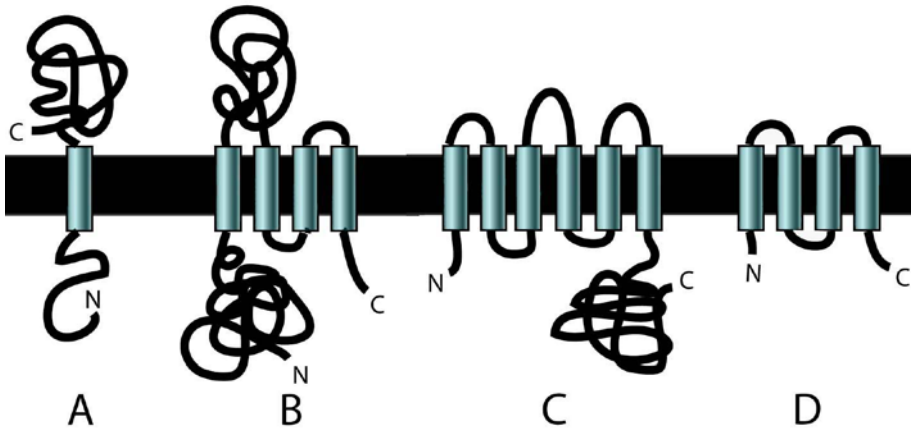


Figure 2.12 *Schematic diagram of transmembrane protein topology. Proteins vary in the number of transmembrane helices and in the number and size of N-terminal, C-terminal, and interhelical domains.*

Analyses of complete genome sequences indicate that over 25% of an organism's proteins are embedded in cellular membranes. Transmembrane proteins play vital roles in cell-to-cell communications, transmembrane signaling, ion transport and maintenance of cell structure and are the targets for the majority of pharmaceuticals in use today. While transmembrane proteins may comprise as much as twenty percent of an organism's genome, they comprise only a fraction of a percent of known protein structures. Thus, the need for accurate and efficient structure prediction is crucial for this group of proteins. For example, the misfolding of specific transmembrane proteins can result in disease, such as in cystic fibrosis. In spite of the vast importance of transmembrane proteins, there are far fewer structures and molecular mechanisms known for transmembrane proteins than for soluble proteins. This difference is due to the presence of hydrophobic sequences that can make it difficult to express and isolate large amounts of these proteins and makes them refractory to many biochemical and structural methods. This project, using the artificial neural networks will try to accurately predict the topology, size and number of transmembrane sequences in unknown transmembrane proteins which are basic definition parameters of proteins' structure.

Chapter 3

Artificial Neural Networks

An Artificial Neural Network (ANN) is an information processing network that attempts to model the architecture and operation of human neural networks. Both human neural networks and artificial neural networks are a collection of connected neurons that interact to produce an action: for human brain's neural networks, mnemonic functions and key brain operations (vision, taste, hearing) are the basic actions while for artificial neural networks, actions would involve sales forecasting, risk management and optical recognition. In spite the fact that 'neural networks' is a biological term, it is commonly used for artificial neural networks as well. Also, artificial neural networks, like people, are based upon training and learn by example [10]; thus meaningful results can derive from complicated or imprecise data. It is this striking ability of ANNs that can recognize patterns and predict solutions that are too complicated for a computer program or a human eye to notice. While architecture of neural networks can vary greatly from type to type, different learning methods can be used to solve specific problems. Currently, artificial neural networks are used to model human organs and predict diseases from digital scans. In the next few years, ANNs are said to lead the way in biomedical systems.

3.1 Introduction to Neural Networks

A Neural Network is an interconnected assembly of simple processing elements (neurons) working together to solve specific problems [10]. The architecture and operation of such information processing networks are based on the human nervous system (Figure 3.1). Like human brain, every new task, a repetition of a common task and learning of a task in neural networks can cause an alternation of the neuro-connections: some will be weakened, others will be reinforced and new ones will be created. Furthermore, according to the type of neurons, some of them are fired with excitation and others with inhibition in a variation of levels: for example very sensitive neurons may fire with very little input while other may fire within the limits of specific threshold. The addition after the final adjustment of all connecting weights will eventually define the proper action (output) for given stimuli (inputs) [11].

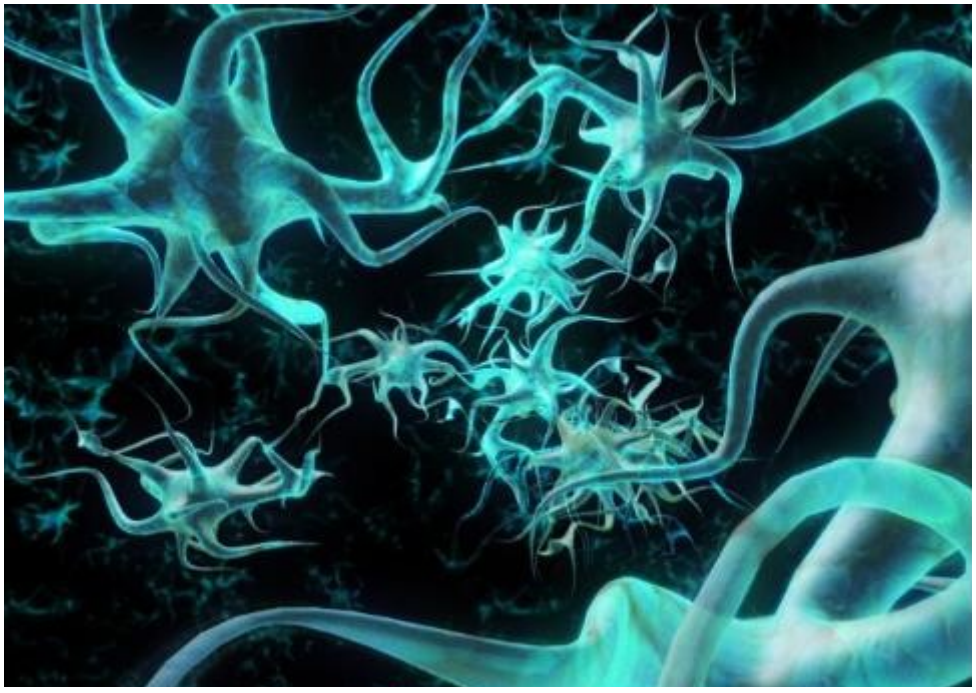


Figure 3.1 *Three-dimensional representation of human brain's neural network*

Back in 1943, McCulloch and Pitts (1943), based on their knowledge on neurology, developed the very first model of neural network which was consisted of simple neurons with fixed thresholds. After that, many engineers and neuroscientists had invested time and efforts in a promising and emerging

technology. In 1958, Rosenblatt designed the perceptron, the first three-layered neural network comprised of an input, middle and output layer. During the sixties, there was a deep frustration and prejudice against neural networks which lasted until the early seventies. However, in the late eighties, the re-emergence of this technology was a fact: many new algorithms were invented (including the popular back-propagation algorithm and the error correction method) and extensive research project were initiated in US, Europe and Japan. Since then, the progress was great and the commercialization of neural applications attracted funds and attention.

3.2 From biological to artificial neurons

Billions of biological neurons are used to train the human brain. The structure and function of such a biological neuron are quite simple yet perfect: a set of branching structures, called 'dendrites' are connected to the cell body, called 'soma'. In the opposite direction, a long, thin stand called axon extends away from the cell body. At the end of each axon there are fine structures, called 'synapses' that help two neurons to connect (Figure 3.2). Signals that have previously collected from the dendrites are transmitted between neurons via electrical pulses traveling along the axon to the synapse. Then, the signal activity is converted into excite or inhibit effect for the neurons. Depending on each neuron's previous state and threshold a specific change on its synapse effectiveness occurs. This constant process enables learning.

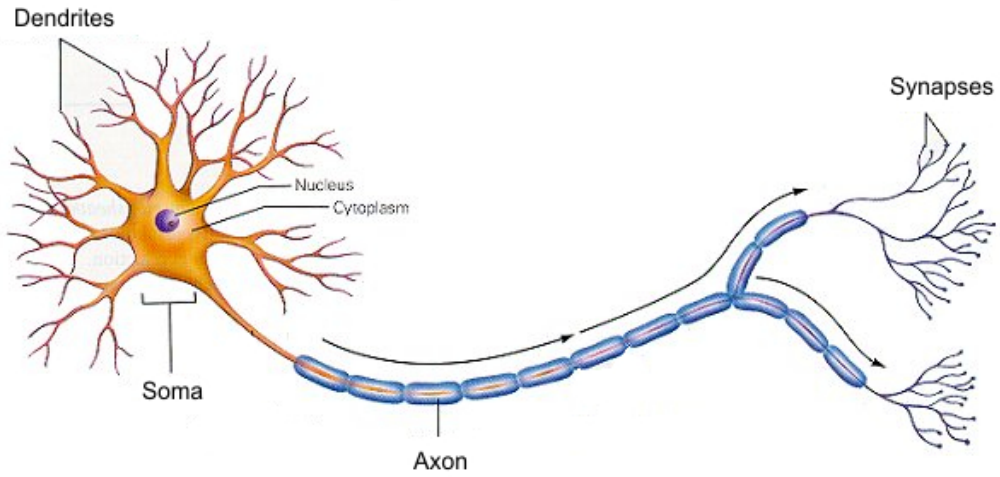


Figure 3.2 *Dendrites of a neuron send the received impulses through the axon to the synapses where neurotransmitters are released to stimulate other neurons.*

Practically, the above description of natural information processing can be generally modeled as input signals (stimulus in synapses), multiplied by some weight (strength of synapse) producing an activation unit (summation of weights), that gives an output response according to a certain threshold [12].

3.3 Architecture

Similarly to the nervous systems, neurons (often called 'nodes') are the keystones of neural networks and incorporate three major functional features: connection strength, excitation/inhibition and transfer function. Connection strengths (or 'weights') are modified by learning: new connections among neurons are created; others are being strengthened or weakened according to specific inputs. The above is closely connected to excitation or inhibition [11]. Each neuron is excited or inhibited which means that its activation could level up the connected neurons or level them down. Also, the transfer function is responsible of determining the response (output) of a neuron: according to the input value and the threshold, a neuron may fire in small inputs or not fire at all or even fire up vigorously till the specified threshold and then fire a little (Figure 3.3).

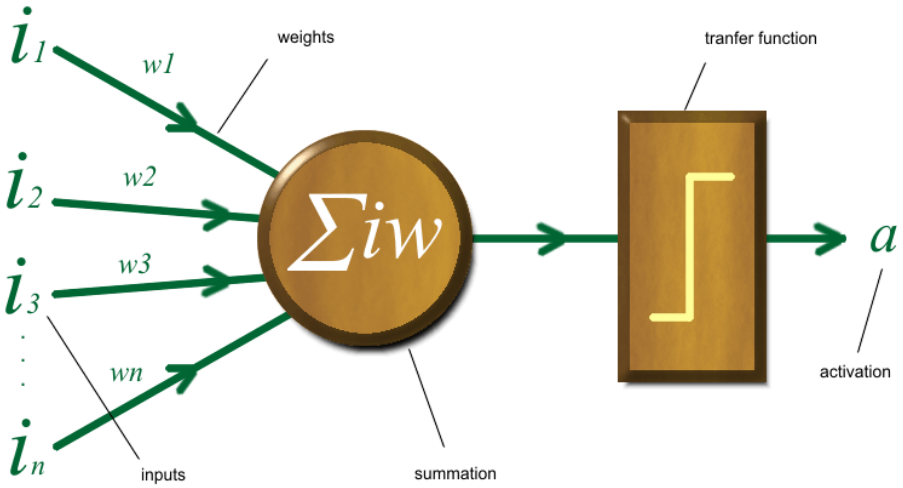


Figure 3.3 *A simple artificial neuron with i weighted inputs and a transfer function*

Activation *a* is given by summing up weighted (wi) inputs (xi):

$$a = \sum_{i=1}^{n} w_i x_i$$

The output *y* is given by thresholding the activation:

$$y = \begin{cases} 1 & \text{if} \quad a \geq \theta \\ 0 & \text{if} \quad a < \theta \end{cases}$$

Many of these artificial neurons construct an artificial neural network that can be either single-layer (the inputs are fed directly to the outputs via a series of weights) or multi-layer (each neuron in one layer has directed connections to the neurons of the subsequent layer). There are two different types of neural network organization: the feed-forward networks and the feedback networks [13]. Since this project utilizes multi-layer, feed-forward networks we will focus on the first type.

3.3.1 Feed-forward networks

While feedback neural networks allow information to travel in both directions, feed-forward networks allow information to travel one-way from input to output without any loops (feedbacks). The output of a layer does not affect the same layer at any point. Feed-forward networks are constructed in a three-layered structure (Figure 3.4): the first layer has a set of input nodes, the second has one or more layers of 'hidden' nodes and the third has a set of output nodes. Initial data are "captured" at the input nodes and initial activation is assigned to nodes (usually numbers representing the level of activation) [11]. Then, the activated information is passed through the network and weights and transfer functions determine the activation level of each node. Each node sums up all the activation levels it receives from all previous nodes and passes the output (new activation level) to the next node.

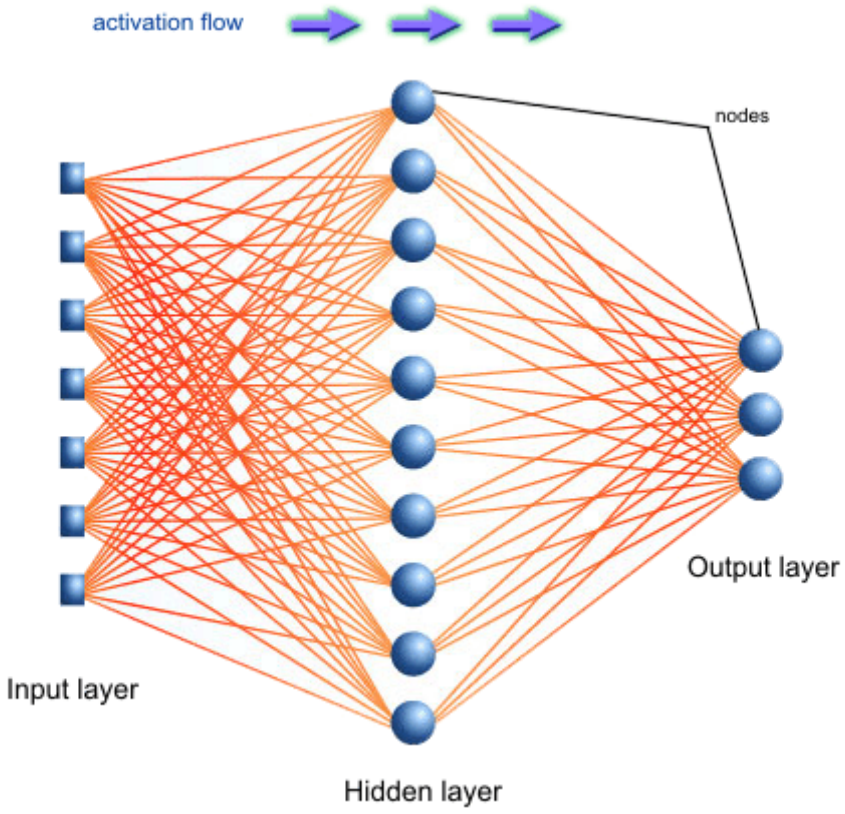


Figure 3.4 *A feed-forward network with three layers*

As mentioned earlier, the flow of the network is one-way and travels from input layer, through hidden layers, to the output layer. This procedure is the beginning of a training session that should result (in a sense) an output similar to the input.

3.4 Learning

Learning is basically a process of adjusting connection weights. In adaptive networks (capable of changing their connection weights), learning can be either 'supervised' or 'unsupervised'. During supervised learning each output unit is told what the desired output value should be. If actual and desired outputs are compared then it is easy to calculate an error which will guide the adjustment of weights (error correction, stochastic learning). In unsupervised learning there is no external teacher. Network uses only local information while it self-organizes its data (Hebbian learning, competitive learning). The most popular learning algorithm for supervised training of multi-layer feed-forward neural networks is called 'back-propagation' [13].

3.4.1 The Back-Propagation algorithm

The Back-Propagation (or Error-Backpropagation, or backprop) algorithm was first introduced by Werbos in 1974 and named after the fact that error signals are propagated backwardly, layer by layer, through the entire neural network. Algorithm's main idea is to adjust weights of each node in such a way that the error between actual and desired output is reduced (Figure 3.5). During this process and while weights are modified back-propagation constantly watches error behavior and stops if error has reached a minimum point. This minimization is achieved by using the gradient descent method [12].

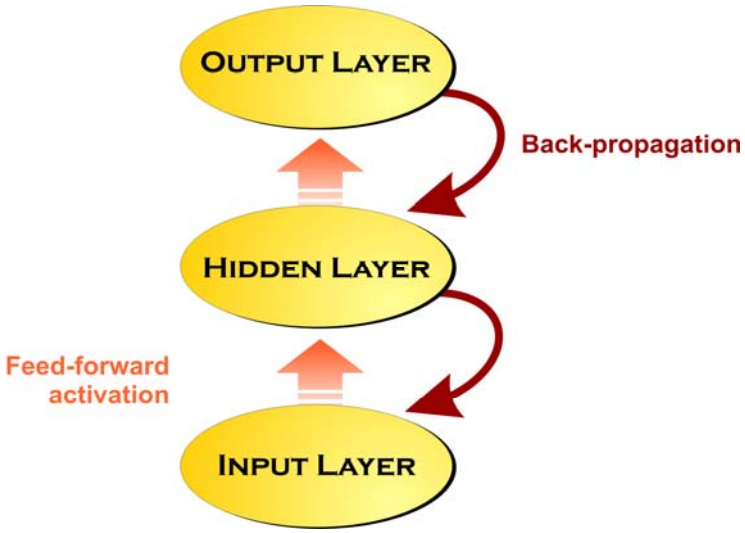


Figure 3.5 *The Back-Propagation algorithm updates weights backwardly aiming at the minimization of error rate.*

Training a neural network with the back-propagation algorithm consists of two phases: the forward pass and the backward pass. During the forward pass initial input data are fed to the network and "local" outputs are calculated. These outputs are fed into the next layer; finally, the data reach the output layer while at the same time connecting weights are fixed. During the backward pass, mean square error is calculated and all the weights are backwardly adjusted so as actual output meets desired output. This error minimization process needs several to several thousands iterations (or epochs) to complete. Furthermore, training mode can be on-line or batch. On-line training adjusts weights per input while batch training adjusts weights when all inputs are given [10].

A simple programming description of the algorithm is:

Loop

for *each training pattern*

Train

end *for*

Until *the error is "acceptably low"*

In general, an output node follows the below steps before deciding what to respond. First, it computes the total weighted input x_j, using the formula:

$$X_j = \sum_i y_i W_{ji}$$

where y_i is the activity level of the j^{th} unit in the previous layer and W_{ij} is the weight of the connection between the i^{th} and the j^{th} unit. Next, the unit calculates the activity y_j using some function of the total weighted input:

$$y_i = \frac{1}{1+e^{-x_i}}$$

Finally, the network computes the error E, which is defined by the expression:

$$E = \frac{1}{2}\sum_i \left(y_i - d_i\right)^2$$

where y_j is the activity level of the j^{th} unit in the top layer and d_j is the desired output of the j^{th} unit.

The back-propagation algorithm is completed in four steps: First step is to compute how fast the output error (EA) changes, that is the difference between the actual and the desired output:

$$EA_j = \frac{\partial E}{\partial y_i} = y_i - d_j$$

Second step is to compute how fast the error of inputs to output changes (EI). EA from step 1 is multiplied by the rate at which the output of a unit changes as its total input is changed:

$$EI_j = \frac{\partial E}{\partial x_j} = \frac{\partial E}{\partial y_j} \times \frac{dy_j}{dx_j} = EA_j y_j \left(1 - y_j\right)$$

Third step is to compute how fast the output weight error changes (EW). EI from step 2 is multiplied by the activity level of the unit from which the connection emanates:

$$EW_{ji} = \frac{\partial E}{\partial W_{ji}} = \frac{\partial E}{\partial x_j} \times \frac{\partial x_j}{\partial W_{ji}} = EI_j y_i$$

Fourth, compute how fast the previous layer error changes allowing back propagation to be applied to multilayer networks. EI in step 2 is multiplied by the output weights and then summed to compute the overall effect of previous layers to the output units.

$$EA_i = \frac{\partial E}{\partial y_i} = \sum_j \frac{\partial E}{\partial x_j} \times \frac{\partial x_j}{\partial y_i} = \sum_j EI_j W_{ji}$$

Once we know the EA of a unit, we can use second the third step to compute the EWs on its incoming connections [12].

3.5 Generalization

Sometimes neural networks are over trained meaning that the network has memorized the training data but it has not learned to generalize to new data. Therefore, testing results are poor and unpredicted. This phenomenon is called over-fitting and it is a common problem when using the back-propagation learning method. The simplest method to avoid such a problem is to choose a network which will be just large enough to adequately fit all the data. However, this is a very difficult task and it is hard to predict. Two other methods for preventing neural networks from over-fitting are regularization and early stopping [14].

3.5.1 Improving generalization using Early Stopping

Early stopping (or 'stopped training') is a technique for improving generalization. In early stopping, available data is divided intro three sub sets: the training set, the validation set and the testing set (Figure 3.6).

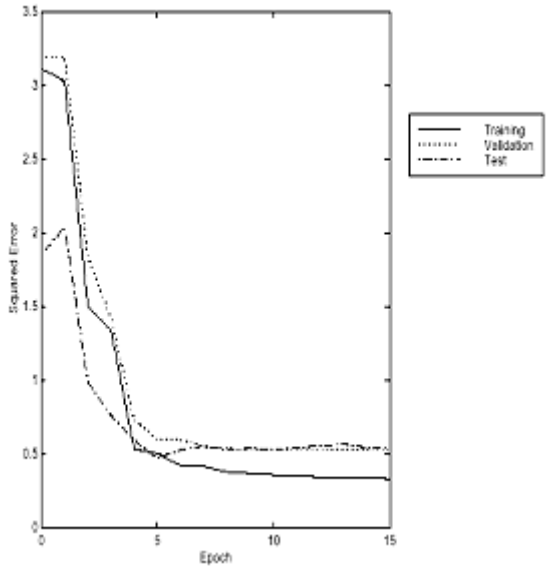


Figure 3.6 *Sample training stops at epoch 15 when validation error "starts to go up"*

Training set is responsible for teaching the neural network by modifying all connecting weights. The validation set is responsible of stopping training when validation error starts to rise, meaning that the network has just started to over-fit data. With testing set we can receive an unbiased, actual network response to new data and compare different ANN models. Testing set contrary to the validation set is not used during training so it is a better estimate of the generalization error. Early stopping is very popular because it is fast, needs only the size of the validation set and can be applied to networks in which the number of weights far exceeds the sample size [14].

Chapter 4

Materials and methods of transmembrane segments detection system

Designing a neural network is basically a matter of defining the problem well in mind; one must recognize the values that would influence the outcome of the system, shape them, feed them into a carefully constructed neural network and evaluate the results for optimal performance. Mathworks® Matlab® provided us with a powerful environment and useful tools such as the Neural Network and Bioinformatics toolbox in order to cope with the demands of the design. All learning algorithms, improved functions and commonly used neural network scripts needed ad hoc were included in the latest Matlab® release (Release 14, August 2005).

4.1 Information gathering

The most difficult aspects of the neural network are: a) decision on the information type used and b) decision on how to collect that information. Furthermore, the precision of these decisions will also affect the network's overall performance. Accurate and well organized data will eventually produce very

promising results. However, if the database is inconsecutive, deficient or corrupt then the outcome of the neural net will be misleading or absolutely erroneous. Information gathering needs hard work and thorough search for valid data that will uniquely characterize the problem and match all the solution criteria posed during the neural net design.

For the implementation of a transmembrane (TM) segment prediction system using neural networks, our goal was to gather as much information (proteins) as possible in order to create a wrought training that would effectively generalize to new, unknown proteins. This vast amount of information should also be accurate, confirmed and widely accepted. Keeping all the above basic principles in mind, the popular and most-cited Swiss-Prot Protein Knowledgebase (release 48.1, September 2005) was chosen to be the source of raw protein data. Swiss-Prot contains approximately 481 fully annotated membrane (MEM) protein sequences with known topologies. Sequences are selected by searching the entire database using the feature keyword 'TRANSMEM'. However, not all protein sequences are used since some of them have not fully-identified transmembrane segments (usually called transmembrane helices) or have unknown or missing information.

4.1.1 Data set

The useful information collected will be the basis for teaching (training), evaluating (validation) and scoring (testing) the system (neural network). That information is often called data set. Our data set consists of four hundred and ten (410) membrane proteins (human and non-human) which have at least one transmembrane segment and meet the criteria posed during information gathering (Figure). As explained in Chapter 2, proteins (in primary structure) are represented by a sequence of amino acids. Protein size is not fixed; therefore, protein length can vary from a few hundred to several thousands of amino acids. Our data set includes membrane proteins with variable lengths of one hundred (100) to about three thousands five hundred (3500) amino acids which can contain from one (1) to fourteen (14) transmembrane segments. TM segment length also varies from ten

(10) to thirty (30) amino acids, with an average length of about twenty two (22) and a standard deviation of approximately three (3) amino acids.

The complete data set is then divided into three sub sets: a training set containing three hundred (300) membrane proteins, a validation set containing fifty five (55) membrane proteins and a testing set containing the rest fifty five (55) membrane proteins. The method of cross-validation [15] is used to train the neural network. This means that the protein order within the sub sets is not fixed. Membrane proteins are randomly selected to form three different groups of three sub-sets each (Figure 4.1).

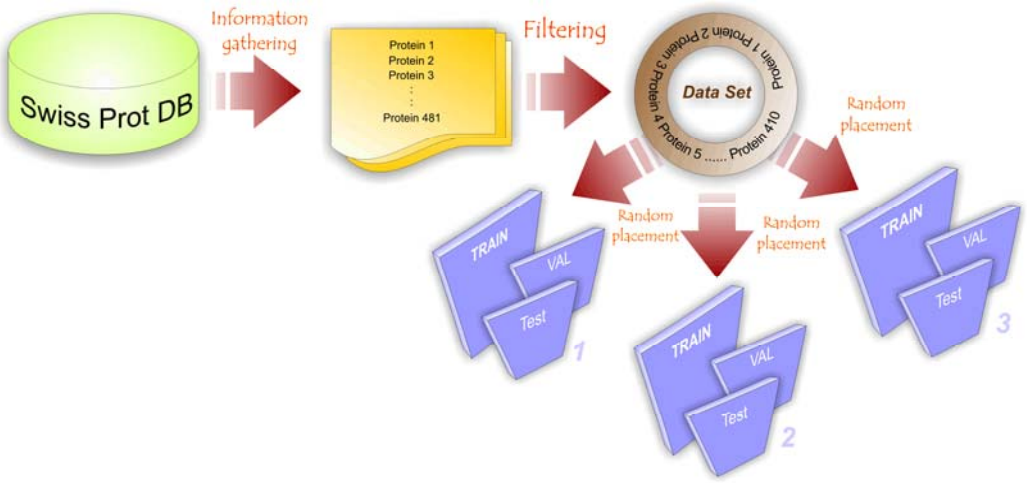


Figure 4.1 *Schematic overview of information processing plan*

Since the suitable data are found, we proceeded to choose the appropriate variables that may be influential; in other words, which variables to use, and how many (and which) cases to gather. This process is guided mainly by intuition and/or experience. As we will examine later on, (in 'Pre-processing of data'), all numeric data had to be fixed into a specific range (this usually requires scaling) and all non-numeric data had to be represented numerically.

4.1.2 Encoding scheme

Encoding of the protein data is a very elaborative process that will determine the successful outcome of this effort. Protein data had to be numerically transformed into values in such way, that every value should be uniquely definitional of its role within the protein structure and also descriptive of the relations among neighboring values. Since membrane proteins are characterized by high discission of their transmembrane segments due to the hydrophobicity of amino acids, an encoding scheme using amino acids' hydrophobicity indices could be suggested. Such an encoding scheme based on laboratory experiments is shown in Table 4.1.

Amino acids	Letter code	Hydrophobicity indices
Alanine	A	1.8
Arginine	R	-4.5
Asparagine	N	-3.5
Aspartic acid	D	-3.5
Cysteine	C	2.5
Glutamine	Q	-3.5
Glutamic acid	E	-3.5
Glycine	G	-0.4
Histidine	H	-3.2
Isoleucine	I	4.5
Leucine	L	3.8
Lysine	K	-3.9
Methionine	M	1.9
Phenylalanine	F	2.8
Proline	P	-1.6
Serine	S	-0.8
Threonine	T	-0.7
Tryptophan	W	-0.9
Tyrosine	Y	-1.3
Valine	V	4.2

Table 4.1 *Amino acids hydrophobicity values and lettering code*

However, the above encoding scheme wouldn't work since it does not offer enough information about the 'weighted' connections among amino acids and requires the approach to complex models that should clearly explain the relationship between the amino acids (inside or outside TM segments) and the repeated passes of the membrane protein through the lipid bi-layer.

For that reason, the method of propensity of amino acids is used. This representation method - based on the TMALN [16] and BKALN database [17] analysis containing more than 1000 TM segments - associates every amino acid with a propensity value which describes the potential of that very amino acid belonging to a transmembrane region:

$$P_i = \frac{F_i^{TM}}{F_i}$$

where P_i is the propensity value of the i^{th} amino acid, F_i^{TM} the frequency of i^{th} amino acid in TM segments and F_i the frequency of i^{th} amino acid [18]. For a better evaluation of the relativity of TM segments the SHTM (HTM-propensity Scaling) method was presented and the SHTM propensity values ($P_{i,SHTM}$) were used which were estimated using the following equation:

$$P_{i,SHTM} = \frac{F_{i,TMALN}}{F_{i,BRKALN}}$$

where $F_{i,TMALN}$ is the propensity of the i^{th} amino acid to appear in the TMALN database and $F_{i,BKALN}$ is the propensity of the i^{th} amino acid to appear in the BRKALN database.

Values range between +0.1 and +2.3: values above 1 indicate a strong possibility of an amino acid to belong to a TM segments while values below 1 indicate a small possibility. Using the sequences of these values as inputs to our neural network requires normalization through scaling (ranging between 0 and +0.1) (Table 4.2).

Amino acids	Letter code	Normalized $P_{i,SHTM}$
Alanine	A	0. 055
Arginine	R	0.017
Asparagine	N	0.018
Aspartic acid	D	0.0067
Cysteine	C	0.071
Glutamine	Q	0.025
Glutamic acid	E	0.009
Glycine	G	0.04
Histidine	H	0.038
Isoleucine	I	0.1
Leucine	L	0.083
Lysine	K	0.0056
Methionine	M	0.088
Phenylalanine	F	0.1
Proline	P	0.029
Serine	S	0.041
Threonine	T	0.034
Tryptophan	W	0.1
Tyrosine	Y	0.044
Valine	V	0.073

Table 4.2 *Amino acids normalized propensity values*

After encoding the neural network inputs (a sequence of amino acid propensities with the same length as the initial amino acid sequence), we also need to define the outputs and its neural network-based, numerical-compatible representation. The best method is to represent outputs as binary sequences of zeros (0) and aces (1). Within a protein sequence, all TM segments are represented by sequences of aces (1) and zeros (0) elsewhere. The relationship between inputs and outputs is shown in Figure 4.2.

4.1.3 Pre-processing of data

Transmembrane protein information is extracted from Swiss-Prot knowledge base search engine and written into a single ASCII text file. This text file is divided into two ASCII text files: one that contains the input sequences of amino acids along with identity information and description of the proteins and another (output file) that contains the topology of the TM segments per protein (Figure 4.2). Topology information includes a starting and ending pointer of each TM segment. Multiple TM segments within the same protein are represented by multiple entries of topology information for the same protein.

First phase of pre-processing refers to the implementation of the encoding scheme: an algorithm had to read the protein input data file character by character, distinguish and tag the protein sequences and assign the correct propensity value to every amino acid in sequence. At the same time, another algorithm had to read the output (usually called target) file holding the known topologies and create a sequence of zeros (0) and aces (1) with the same length as the input (Figure 4.3).

For example, protein CXCR4_HUMAN is annotated according to the following format (Figure 4.2): first line includes a unique Swiss Prot identification number, followed by the official name and a short description about the type, special characteristics and function of the membrane protein. Identification number has the following format: >sp|YXXXXX|, where Y can be any capital Latin letter and X a one-digit integer number. X and Y then form a six-digit alphanumeric code. The name is written in a single, capital Latin lettered word containing no spaces or special symbols except for the underscore symbol '_'. Description is of a low practical importance; however its informational value was unquestionable. Next lines would include the entire letter-coded sequence of the protein grouped in a block of amino acids, sixty (60) proteins wide. Each description would end with a full stop ('.') (Figure 4.2).

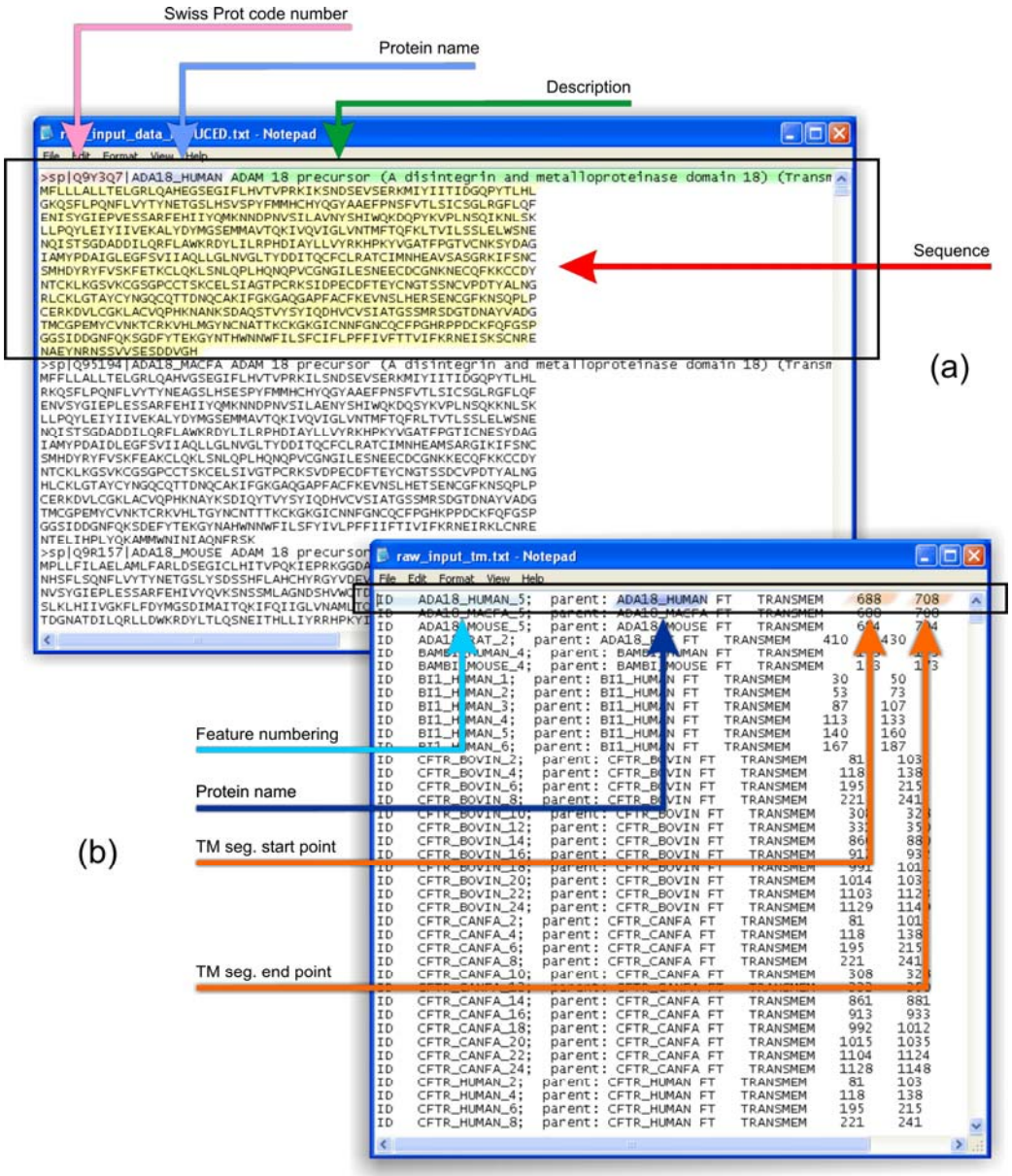


Figure 4.2 *Raw input (a) and target (b) data format*

During the pre-processing stage we had to find a way to distinguish and correlate protein name and sequence. Since every protein started with the symbol '>', has its name placement fixed and its description always ends with a full stop, the algorithmic solution to our problem would be the following:

Step 1: *Read chars from '>' to '|' and exclude them (Swiss Prot coding was indifferent to us).*

Step 2: *Continue reading all chars after '|' until you meet <space> and store them. (that's the protein name).*

Step 3: *Continue reading chars after <space> until you meet a full stop and exclude them (description had not value to us).*

Step 4: *Continue reading all capital chars after '.' (that's the protein sequence)*

Step 5: *Switch to next protein when '>' is found and repeat steps 1 to 4 (which denotes the end of the previous protein and the beginning of next; '>' works as a protein separator).*

Step 6: *Continue executing steps 1 to 5 until the end of file (EOF).*

For the output (target) file a different strategy had to be considering that the format of the output data was different: A line starts with a numbering string of the TM segment of a protein and then the sting 'parent', followed by the parent protein name. Next to the protein name is the string 'FT' followed by 'TRANSMEM'. Next to 'TRANSMEM' there are two integers separated by a <space>: first integer is the start and second integer is the end point of the TM segment (Figure 4.3). Moreover, multiple lines with the same parent name denote multiple TM segments within the same membrane protein. The target data are contiguous meaning that TM segments coming from different proteins are not separated by any symbol or line break (Figure 4.4 shows a line break in Target info between TM segments of different protein only for visual reasons). The algorithmic solution here is the following:

Step 1: *Read chars from the beginning of line to the ':' symbol and exclude them (this information was indifferent to us).*

Step 2: *Continue reading all chars after ':' until you meet <space> and store them. (that's the protein name).*

Step 3: *Continue reading chars after <space> until you meet <space>again, convert them to integer and store them (that's our TM segment start point).*

Step 4: *Continue reading chars after <space> until you meet <space> again, convert them to integer and store them (that's our TM segment end point).*

Step 5: *Switch to next protein when a line breaks and repeat steps 1 to 4*

Step 6: *Continue executing steps 1 to 5 until the end of file (EOF).*

Finally, the entire I/O set of proteins is vectored into two arrays of equal size whereas each row held one protein and each column one amino acid. Thus, the entire set is primary converted into two <410 by maxLength> arrays, where maxLength is the maximum length among the membrane proteins. Proteins with less amino acids than maxLength are filled up with null elements.

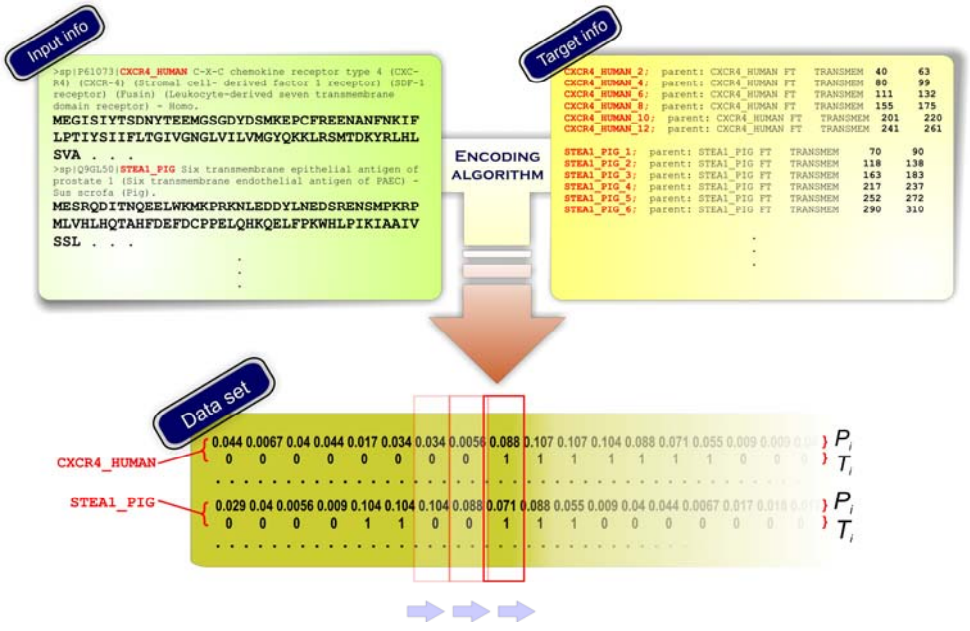


Figure 4.3 *The implemented encoding scheme translates raw protein data into aligned membrane protein sequences*

Second phase of pre-processing had to answer to the question: what kind of reshaping should be applied to the vectored data in order to adequately match the requirements of our neural network system. One major problem we encountered was the variable lengths of membrane proteins. Since every neural network must have a specific number of inputs (and outputs) we had to express the variability of the amino acid sequences to a fixed number of inputs/outputs.

The optimal method chosen to solve that problem is the method of the sliding window. Both input and output vectored sequences are scanned by a sliding window of thirty (30) amino acids creating input and output groups of thirty amino acids. The first input would include amino acid in position one (1) to thirty (30) then the window would slide right by one amino acid and the second input formed would include amino acid in position two (2) to thirty one (31). Third input would be the group three (3) to thirty two (32) and so forth (Figure 4.4).

Hence, every membrane protein is serially read from the primary array producing a sub-array of <WS by (proteinLength-WS)>, where WS is the window size and proteinLength is the length of the protein meaning that the number of sliding windows is equal to the full length of the protein minus the window size. The output vector followed exactly the same pattern. The above method is accordingly applied to training, validation and testing set.

4.2 Neural network system

According to Haykin [19], a Neural Network is a massively parallel-distributed processor that has a natural prosperity for storing experiential knowledge and making it available for use. It is the heart of the predicting system which is responsible for successfully processing the information and making decisions about the outcome of the system. After acquisition and pre-processing of data, this section will discuss in detail the design principles, requirements and specifications of the suggested neural network. Furthermore, issues concerning

layer topology, weight initialization and optimal parameterization of the network will be thoroughly examined.

4.2.1 Description

The most common type of neural network used for prediction systems is the multi layered, feed forward, supervised-learning neural network. This means that our neural net will have multiple layers, processed information will travel from one layer to the next in a forward direction and knowledge is obtained from pairs of inputs and corresponding desired outputs. The difference between the actual and desired response of the net is calculated and the error is then used for weight adjustment until it becomes extremely small (Figure 4.4). Continuous experimentation eventuated in a 30:45:30 feed forward model meaning that there will be three layers: an input layer having thirty (30) neurons, one hidden layer having 45 hidden neurons and an output layer with thirty (30) neurons (Figure 4.5). Each neuron is connected to all the other neurons of the next layer.

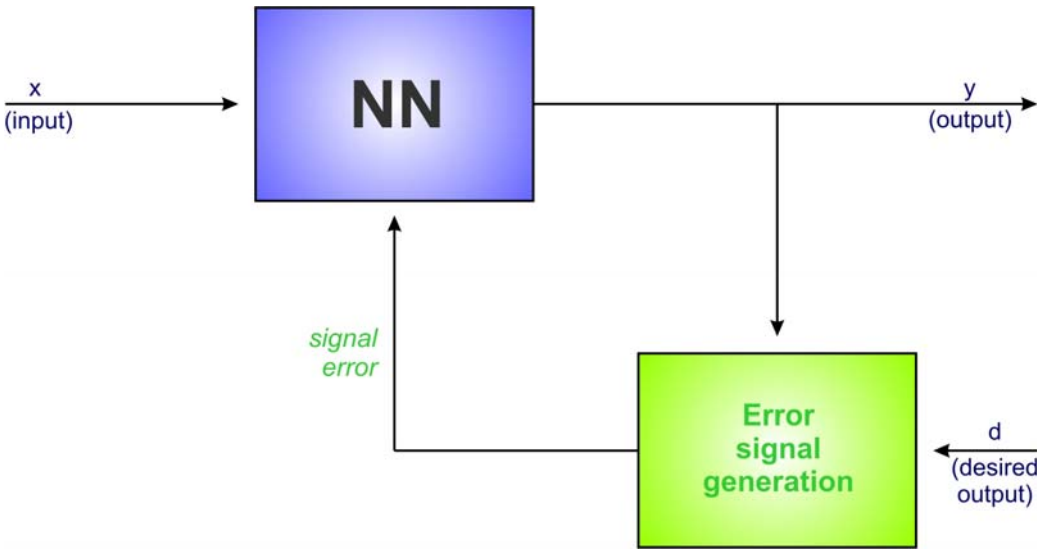


Figure 4.4 *The supervised learning operational system*

The neural network was trained using the back propagation algorithm [12]: inputs of the training set are fed into the neural net, the new neuron values are calculated and through the activation functions they pass on to the next layer until they reach the output layer. Just when the output neurons output their result values the process of weigh adjustment begins. The network compares the desired output values (usually called target values) with the predicted output values and adjusts weights backwardly in order to decrease the mean square error until this difference between target and predicted output values reaches a minimum. The neural network can repeat the above procedure using the entire training set as long as it needs in order to reach that minimum. These repetitions are called epochs (or iterations).

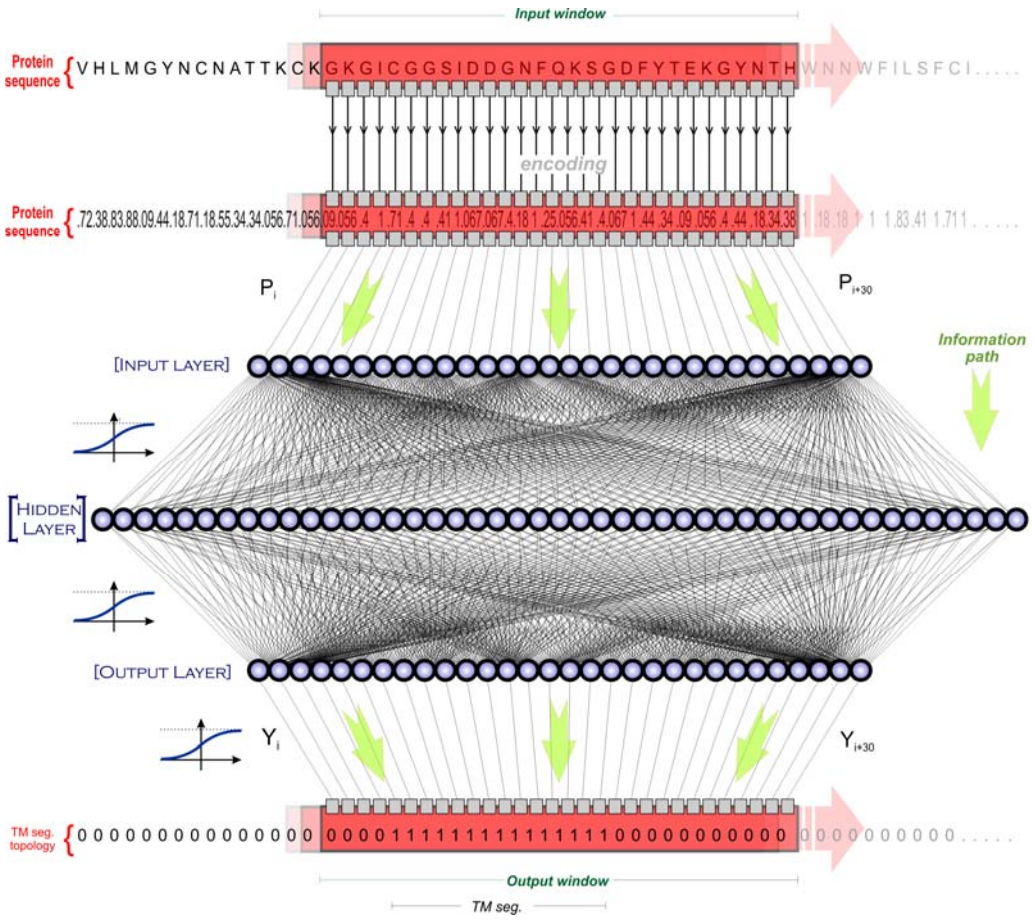


Figure 4.5 *Neural network system architecture using the sliding window input method*

Input range was set between zero (0) and one (1). Since we expected from the network to give us straight answers whether an amino acid belongs to a TM

helix or not, we had to use a transfer function that would work as a switch: zero (0) would mean that an amino acid is definitely outside a TM segment while one (1) would mean that the amino acid is definitely a part of a TM segment. Furthermore, all intermediate output values had to fall into one of the above two categories: for example, an output value of 0.2 would mean that an amino acids has a very small probability to belong to a TM segment; thus it can be considered as an 'outsider' (zero) while an output value of 0.8 would mean that an amino acid has a strong probability (one) to belong to a TM segment. There are two basic transfer functions that have switching behavior: the first is the hyperbolic tangent (tan-sigmoid) function and the second is the log-sigmoid transfer function. The tan-sigmoid (Figure 4.6a) squashes input values between -1 and 1 while the log-sigmoid (Figure 4.6b) squashes input values to 0 and +1 depending on the threshold; in other words, it's a pure switch that can be turned-off (0) or turned-on (1) if the output values are smaller or greater of a given threshold. Thus, unit response was determined by the log sigmoid transfer function for input-to-hidden, hidden-to output and final output layer (Figure 4.7). Threshold is set to 0.5 meaning that values equal or below 0.5 will result in a 0 response, 1 otherwise.

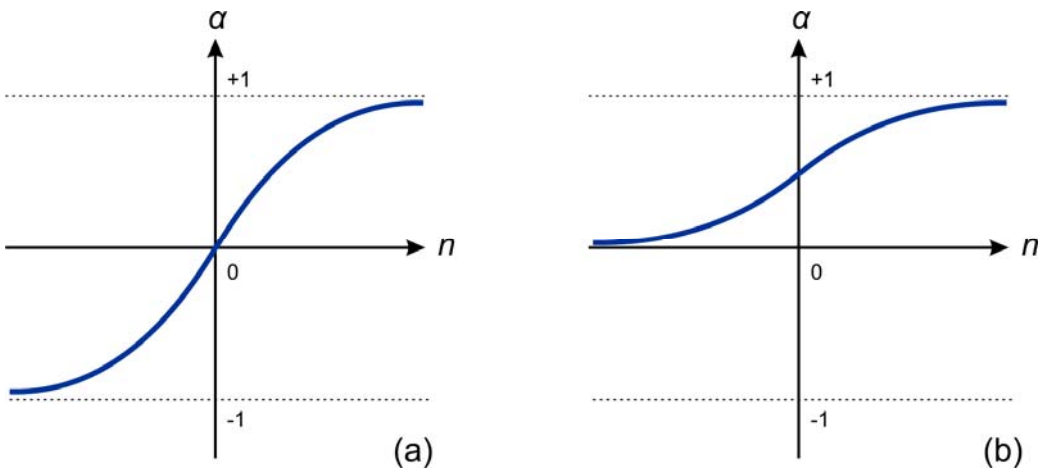


Figure 4.6 *The tan-sigmoid (a) and log-sigmoid (b) transfer functions*

Along with training, another procedure called validation works in parallel. Validation uses the validation set which contains new proteins that haven't been used during training so as to evaluate the response of the system at any time. If this response decreases with time then the learning is performing well and training

procedure continues until the response starts to rise. Should the rising occurs, the network has just started to over fit data and the system can not learn any more.

Finally, testing set is used to evaluate the performance of the neural network system. A brand new set of unknown proteins is fed into the network and the predicted output is compared to the target. These proteins must not be included in either the training or the validation set because we want to check whether the system has reliably generalized.

4.2.2 Implementation and parameterization

The skeleton of the neural network is constructed using the newff command in Matlab® which creates a feed-forward, back-propagation neural network:

$$net = newff(PR,[S1\ S2...SNl],\{TF1\ TF2...TFNl\},BTF,BLF,PF)$$

where PR = R x 2 matrix of min and max values for R input elements, S_i = Size of i^{th} layer for N_l layers, TF_i = Transfer function of i^{th} layer, BTF = Back-propagation network training function, BLF = Back-propagation weight/bias learning function and PF = Performance function.

As mentioned above, the variable size of membrane proteins was a real problem for every fixed input neural network. To deal with the problem we had to correlate the variable lengths with a fixed input window (Figure 4.4) that would read every amino acids by sliding right (by one) through the entire protein sequence n times:

$$n_i = pLength_i - WS$$

where n_i is the number of window slides in the i^{th} protein sequence, $pLength_i$ the sequence length of the i^{th} protein and WS the window size.

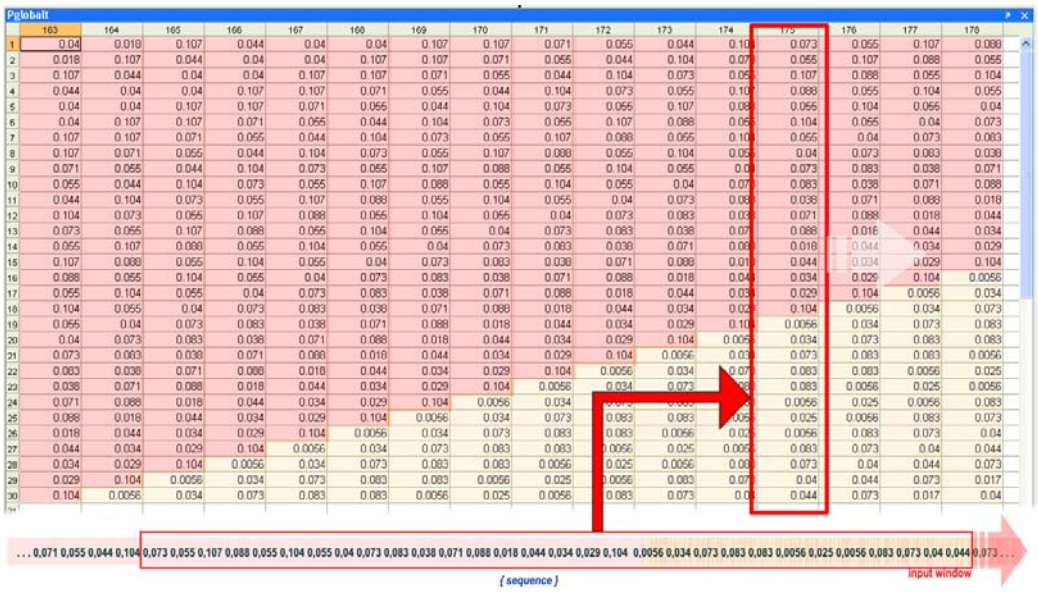
Since the length of a TM segment ranges between ten (10) and thirty (30) amino acids we had to find both an input and output formula that would best fit the TM segments. Basically, the formula should define the size of the sliding

window. After experimentation, the optimal window size was fixed to thirty (30) amino acids which allowed the effective screening of the smallest up to the largest TM segment.

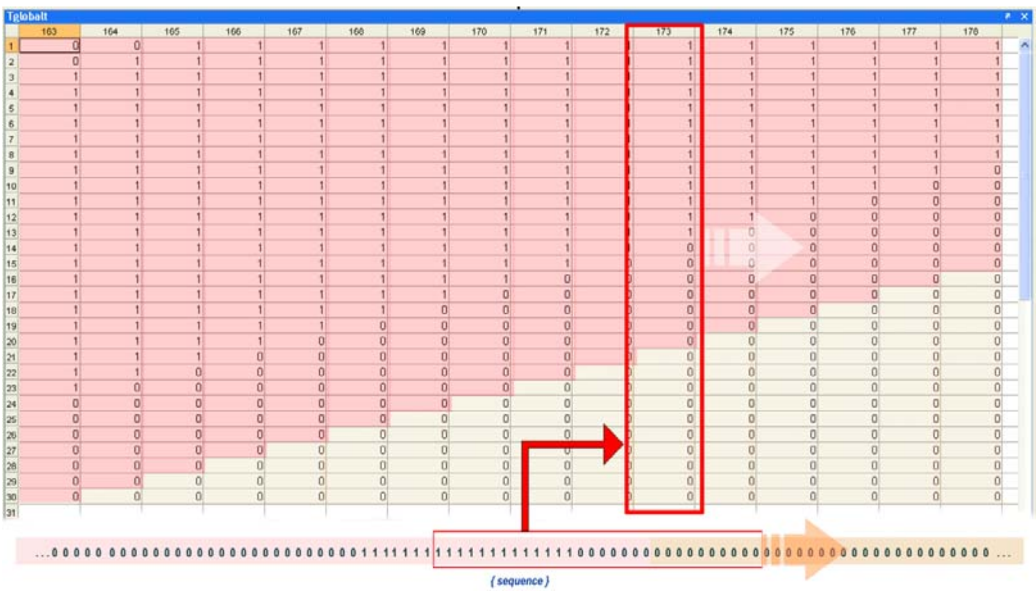
Having calculated the propensities of each amino acid to be positioned inside a TM segment during the encoding phase, we presented an input array with dimensions (D_P):

$$D_p = X \cdot Y_p$$

where $X = WS$ (window size) and $Y_p = \sum pLength_i$. All input data were grouped in a single input array named Pglobal and then randomly divided into the three sub sets: Pglobalt (for training inputs), Pglobalv (for validation inputs) and Pglobalts (for testing inputs). Accordingly, all target data (D_T) were divided into the three corresponding sub sets: Tglobalt (for training targets), Tglobalv (for validation targets) and Tglobalts (for testing targets) (Figure 4.6).



(a)



(b)

Figure 4.7 *Training set input (a) and target (b) arrays. The two contrasting surfaces (pink and white) depict the sliding (by one) of a 30 amino-acid window as it runs through the sequences.*

Passing on to the neural net, strengths (called states) Sj of each connection is weighted and can be modified during the training procedure:

$$S_j = \frac{1}{1+e^{-\left(w_{j0}+\sum_{i=1}^{n} W_{ji} S_i\right)}}$$

where S_j is the state in the j^{th} layer, w_{j0} is the bias and $\Sigma W_{ij} S_i$ the summation of weights. Moreover, we have to define our basic benchmark: the error during the training and testing stage. The following equation explains how the training error rate is calculated:

$$E_{train} = \frac{1}{P_{train}} \sum_{p=1}^{P_{train}} \left(\frac{1}{2} \sum_{i=1}^{N} \left(d_i^p - y_i^p \right)^2 \right)$$

where P_{train} is the net input for the training samples, ½ $\Sigma(d_i^p - y_i^p)2$ the difference between the desired (d_i^p) and the actual (y_i^p) net output sized N. Using the same equation we can calculate the error during the testing stage:

$$E_{test} = \frac{1}{P_{test}} \sum_{p=1}^{P_{test}} \left(\frac{1}{2} \sum_{i=1}^{N} \left(d_i^p - y_i^p \right)^2 \right)$$

where P_{test} is the net input for the testing samples, ½ $\Sigma(d_i^p-y_i^p)2$ the difference between the desired (d_i^p) and the actual (y_i^p) net output sized N. Therefore, the next step to system optimization would focus on the minimization of this error [10,19].

Along with the mean square error calculation, sensitivity (Q_{se}) and specialty (Q_{sp}) factors are the other two basic evaluation criteria of our method. Sensitivity factor is the fraction of the total number of correctly predicted TM segments ($q_{predcor-total}$) divided by the total number of actual TM segments ($q_{act-total}$), while specialty factor is the fraction of the total number of correctly predicted TM segments ($q_{predcor-total}$) divided by the total number of predicted TM segments ($q_{pred-total}$).

$$Q_{se} = \frac{q_{predcor-total}}{q_{act-total}}$$

$$Q_{sp} = \frac{q_{predcor-total}}{q_{pred-total}}$$

During the training stage we engaged validation and testing processes in parallel by feeding the network a two-arrayed structure containing both the validation and test set. All these stages were monitored via both a real time history report and a visual representation of the mean square error (MSE) progress for every five (5) epochs. The visual representation included a real time graphical plot of the progress, deployment and diffusion of the three basic MSEs: the training MSE, the validation MSE and the testing MSE Figure. The testing stage involved the evaluation of the network in predicting unknown proteins through average error monitoring, history report logging and manual sampling verifications of random sequence parts.

4.3 Model optimization

The optimization process is a continuous struggle for perfection and runs through every stage of system's development. Model optimization starts with finding the appropriate encoding scheme for the representation of the information gathered. This is mostly achieved by combining the visualization of the problem with common experience and experimentation. Should we manage to gather satisfactory data and pre-process them successfully (the last will be presumably certified later on), thereafter we need to adjust the parameters of the neural network system so as to achieve best performance. An important factor is the sample size problem, meaning that the training set might not be large enough for teaching the neural network system correctly. So, the need for a big collection of data was essential. Our collection of three hundred (300) membrane protein sequences only for training (approximately 145,000 amino acids) would definitely solve that problem. Window size had also played an important role to prediction effectiveness. Many experiments have taken place utilizing four different input window sizes: 5, 10, 20 and 30 amino acids. As previously analyzed, input window with a length of 30 amino acids matched the criteria. Main system parameters in a feed forward back propagation neural network include: the total number of training samples, number of hidden neurons, type of transfer function applied on each layer, error evaluation, learning algorithm, number of iterations, weights initialization and learning rate.

During the training process, the neural net calculates the mean square error (MSE) for every iteration until it reaches a minimum point. This point (called goal) was set to 0.001 or 0.1%. If training error reaches 0.1% before validation's early stopping does then training stops. On the other hand if the lower limit is not met during training, validation can trigger an early termination of the training as long as it detects data over-fitting. The detection is carried out using the Early Stopping method described in Chapter 3. Number of epochs (or iterations) was set to 1000 meaning that the network can be repeatedly fed with the entire training set up to 1000 times until it reaches a gradient descent minimum.

Optimization continues with the adjustment of the number of hidden units (neurons) in the hidden layer. Hidden units adjustment can be very tricky because

there is always the danger of under-fitting or over-fitting. On one hand, under-fitting usually occurs when there are only a few hidden units available and the neural network is not capable of learning with the given training set. On the other hand, too many hidden units can increase the learning capability of the system but the CPU execution time and memory cost would be significantly greater. Since there is no rule for calculating the exact number of hidden units, experimentation gave us the golden mean: forty five (45) fully connected hidden units. Previous tries which presented lower performance ratio included neighboring models with 35, 40, 42, 48, 50 and 55 hidden units.

Several learning algorithms are tested and the performances to costs ratios are compared. Costs include both execution time and memory resources size. Because of the large data set many algorithms are out of memory such as Levenberg-Marquardt while others like BFGS quasi-Newton performed poorly. Gradient Descent based algorithms (especially with variable learning rate) have a good performance with medium memory requirements but are significantly slower. The Resilient Back-propagation algorithm (or TrainRP) has the best performance to cost ratio among 8 different learning algorithms tested [20]. It is the fastest algorithm with almost the best performance and relatively moderate memory requirements. After some tests to the parameters of the TrainRP algorithm we decided to set the minimum performance gradient (min_grad) parameter to 1e-6, maximum validation failures (max_fail) to 5, learning rate (lr) to 0.01, increment, decrement and initial weight change to 1.2, 0.5 and 0.07 respectively with a maximum of 50 weight changes.

Initialization of weights has a key role to model optimization. The use of small random weights is suggested so as to avoid the saturation effect (identical weights will results in identical weight updates). Using the equation below, we can calculate the weights for every neuron:

$$w_{ji(new)} = w_{ji} + \Delta w_{ji}$$

where w_{ji} is the weigh of the ith neuron of the jth layer, $w_{ji(new)}$ is the new weight of the ith neuron, and Δw_{ji} is the update weight value.

Learning rate needs special attention as it is crucial for an effective learning. Very small learning rates can lead to extremely slow neural network responses

while high learning rates can make the system unstable. Learning rate derives from the below equation:

$$\Delta w_{ji} = -\eta \cdot \delta_j x_{ji}$$

where Δw_{ji} is the update weight value, $\delta_j x_{ji}$ is the error for each unit ij in hidden and output layer and η is the learning rate. Different values of learning rate can change the step of gradient descent towards the error surface minimum, meaning that a small learning rates takes longer to converge to minimum error whereas higher learning rates may converge too fast missing the global minimum [10,19].

Other minor improvements are made including Matlab® scripting acceleration, extension of memory usage of variables, graphical interface enhancements. Due to the fact that the data set are unspeakably huge the process of implementation and optimization of the neural network system is time and resources consuming. However, it is an important procedure that defined the success of our predictive system. The end of the optimization is followed by the testing stage (Chapter 5), where all the qualitative and quantitative characteristics of the results along with the overall performance are analyzed and evaluated.

Chapter 5

Results

Since there are virtually zero tools for defining the optimal parameters of a predictive system, we had to discover those criteria that would produce the best results. The basis of our comparisons was the mean square error and ad hoc experience through various trials and experiments. Important factors that would influence the performance of the system, such as the optimal inputs dimension, number of hidden layers and learning rate, were tested. In the beginning of the testing stage, a small number of these influential factors stayed fixed while others received several testing values. Short after the progress of the tests, more factors stayed fixed while the first optimal values started to transpire. Figures and observations that follow explain the relations of these factors, clarify our systems optimal set of characteristics and reveals its simulated response and performance. Results begin with some basic facts and figures and continue with overall benchmarking and three case studies.

5.1 Statistics

As previously mentioned in Chapter 4, our dataset contained 410 membrane proteins where of 300 were used for training, 55 of them were used for validating and the rest 55 were used for testing the network (Figure 5.1).

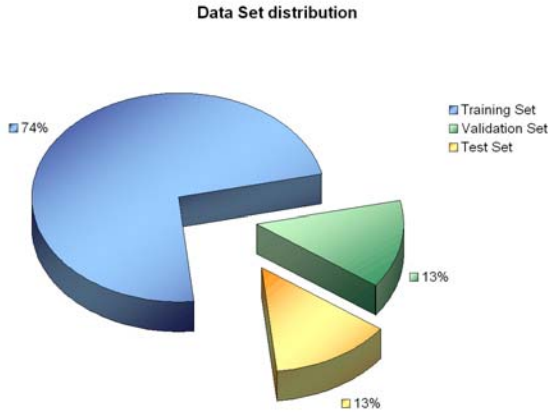


Figure 5.1 *Distribution percentage of the dataset*

Membrane proteins had variable lengths which varied from 100 to 3600 amino acids and different number and sizes of TM segments which also varied from 1 to 14 and from 10 to 30 amino acids respectively (Figure 5.2). The majority of membrane proteins are found to have lengths between 200 and 600 amino acids and gather more TM segments than any other group. Proteins with lengths between 600 and 1000 amino acids follow. It is clear that bigger proteins may usually contain more TM segments. The number of TM segments per protein in proteins whose lengths vary from 1400 to 1800 amino acids is maximized (the small number of proteins bigger than 1800 amino acids does not allow us to generalize safely).

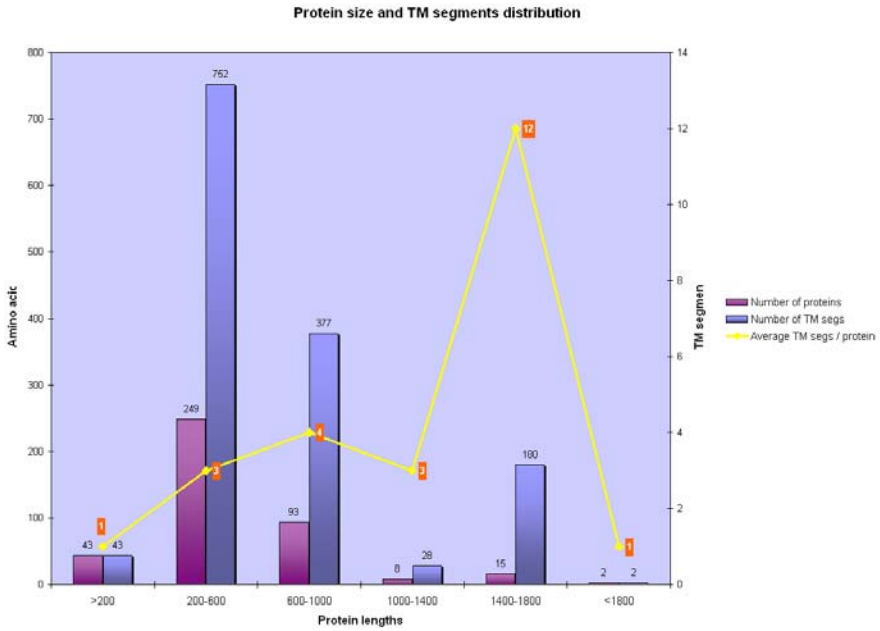


Figure 5.2 *Distribution of protein lengths, total and average number (yellow line) of TM segments per size*

Dataset propensities per amino acid are also represented (Figure 5.3). This is a helpful count in order to observe the appearance frequency of every amino acid in the dataset and check whether the propensities used for input pre-processing matched the propensities found in the dataset. Figure 5.3 shows the dominance of Leucine (L), Serine (S) and Valine (V) over the rest amino acids. Comparing the propensities of this dataset with the propensities used as inputs (Table 4.2) there is an obvious similarity; Phenylalanine (F), Isoleucine (I) and Leucine (L) have a strong probability to be a part of a TM segment whereas Arginine (R), Aspartic acid (D) and Lysine (K) are probably outside a helix. Overall performance trials that follow use the random-ordered Data set with the highest scattering of proteins.

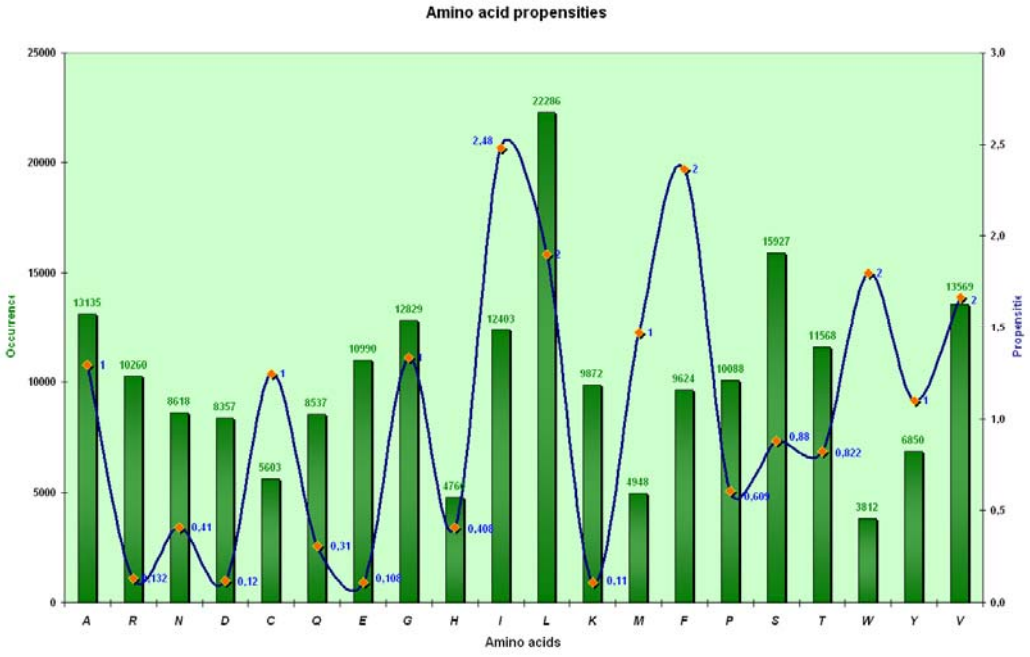


Figure 5.3 *Amino acid occurrence frequencies in dataset (green bars) and their propensities (blue line) for residing in a TM segment*

5.2 Overall performance

Initial tests for studying the behavior of several training algorithm were performed using a sliding window of 10 amino acids on a reduced dataset, because it was more flexible and less time-consuming. As described in Chapter 4, several training algorithms were applied to test system's performance: TrainGDA (adaptive gradient descent), TrainGDX (variable learning rate gradient descent), TrainCGF (conjugate gradient back-propagation), TrainSCG (scaled conjugate gradient back-propagation), TrainRP (resilient back-propagation) and others. Figure 5.4a shows the TrainGDA algorithm progress of train, validation and test MSE till epoch 70 when Early Stopping occurred.

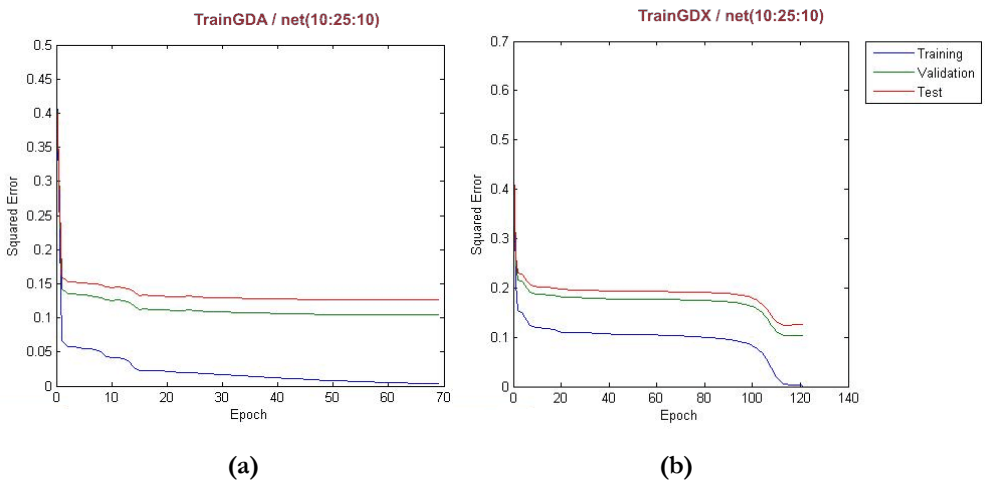


Figure 5.4 *Training, validation and testing MSE for (a) TrainGDA and (b) TrainGDX algorithm per epoch*

Despite the fair performance, TrainGDA evolved quite rapid at first but ended extremely slowly and needed approximately 70 epochs to complete. 70 epochs might not seem many; however the small input size and number of hidden layers must be taken into consideration. Figure 5.4b shows the results of the TrainGDX for the training of the same network. TrainGDX with variable learning rate had slightly worse performance and needed 50 epochs more to stabilize. Next figure (Figure 5.5) shows the progress of TrainCGF and TrainSCG algorithm while training the reduced training set in a 10:25:10 neural network.

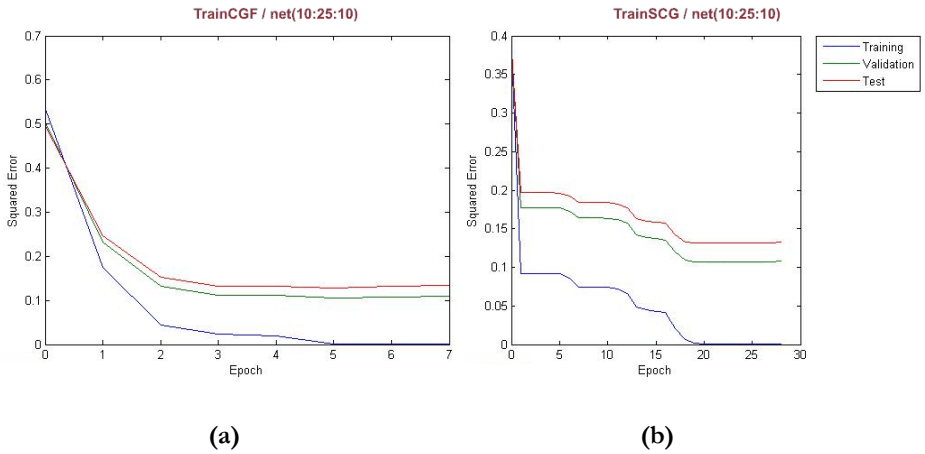


Figure 5.5 *Training, validation and testing MSE for the TrainCGF algorithm per epoch*

Examining the test MSE of TrainCGF (Figure 5.5a) it is visible that the network performed relatively well and needed few epochs to reach the minimum MSE; thus TrainCGF is potential candidate as the optimal training algorithm. However, as proved after many trials on large datasets, TrainCGF algorithm had lower performance compared to others. The TrainSCG algorithm, if compared to previous algorithms, had modest performance and required extra virtual memory to run. Finally, the TrainRP algorithm was applied for testings (Figure 5.6).

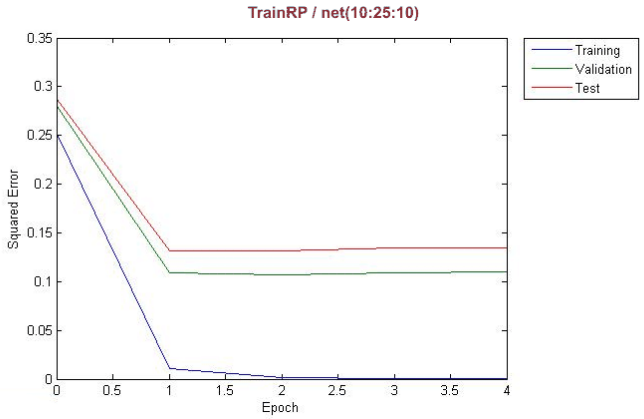


Figure 5.6 *Training, validation and testing MSE for the TrainRP algorithm per epoch*

The results for TrainRP were satisfactory especially when used for much larger 30-input/30-output train sets. Speed was extremely high and memory requirements quite low while, for the given network, test mean square error was kept low.

Trials that followed had to arrive at a decision about the input window length that would definitely affect the size of the dataset (which was in turn unbreakably bonded with the number of units in the hidden layer). Usual practice in prediction systems using neural networks emphasizes the importance of an analogy between the size of the input and hidden layer. Hence, we had to relatively increase the number of hidden units while the input window size was increasing from 10 to 20 and then to 30 amino acids. Three out of approximately twenty different configurations were chosen to depict the performance differences among different architectural patterns. These milestones along with their progress during training, validation and testing and their error response are shown below (Figures 5.7, 5.9, 5.11).

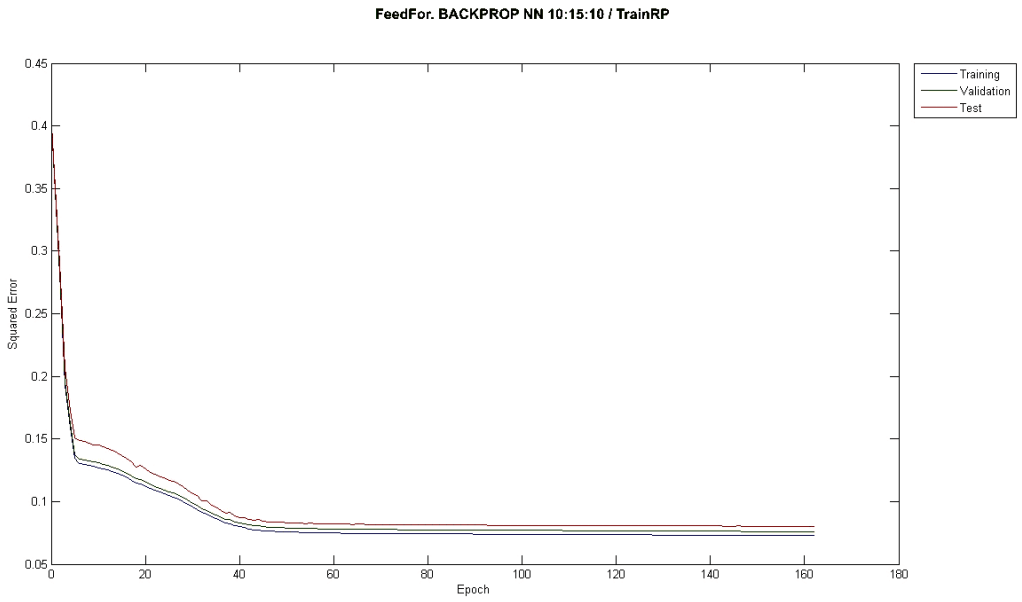


Figure 5.7 *Training, validation and testing MSE per epoch for the 10:15:10 net architecture*

Figure 5.7 shows the recorded error response of our entire test data set in a 10:15:10 neural network scheme; meaning that net had 10 inputs, 15 hidden and 10 output units in input, hidden and output layer respectively. Test MSE and train MSE reached 7.87% and 6.98% respectively, while the network needed about 160 epochs to finish training. The Early Stopping method detected an increase tendency to validation error and immediately terminated the training at epoch 160. Training history log file that recorded the train error progress is shown in Figure

5.8. Learning rate changes were applied from 0.01 to 0.9 but it would not affect the performance significantly; thus it was fixed to 0.01 throughout the trials.

```
TRAINRP, Epoch 0/500, MSE 0.428633/1e-005, Gradient 0.0831843/1e-006
TRAINRP, Epoch 5/500, MSE 0.201154/1e-005, Gradient 0.0338328/1e-006
TRAINRP, Epoch 10/500, MSE 0.122861/1e-005, Gradient 0.00508858/1e-006
TRAINRP, Epoch 15/500, MSE 0.11408/1e-005, Gradient 0.00636811/1e-006
TRAINRP, Epoch 20/500, MSE 0.104014/1e-005, Gradient 0.00653366/1e-006
TRAINRP, Epoch 25/500, MSE 0.0920503/1e-005, Gradient 0.0064255/1e-006
TRAINRP, Epoch 30/500, MSE 0.0820222/1e-005, Gradient 0.00394233/1e-006
TRAINRP, Epoch 35/500, MSE 0.0756261/1e-005, Gradient 0.00455431/1e-006
TRAINRP, Epoch 40/500, MSE 0.07408.............................
.....................................................................
.....................................................................
.................................001/1e-005, Gradient 0.000210255/1e-006
TRAINRP, Epoch 158/500, MSE 0.0699702/1e-005, Gradient 0.000248815/1e-006
TRAINRP, Epoch 159/500, MSE 0.0699463/1e-005, Gradient 0.000238899/1e-006
TRAINRP, Epoch 160/500, MSE 0.0699217/1e-005, Gradient 0.000179104/1e-006
TRAINRP, Epoch 161/500, MSE 0.0699014/1e-005, Gradient 0.00015419/1e-006
TRAINRP, Epoch 162/500, MSE 0.0698926/1e-005, Gradient 0.000176442/1e-006
TRAINRP, Validation stop.
```

Figure 5.8 *Training MSE history per epoch for the 10:15:10 net architecture*

Trying to minimize test MSE some new configurations were tried out involving input window extension. Extending window length caused the need for more hidden units. Figure 5.9 shows the results obtained after the extension of the window length to 20 amino acids and the addition of 20 extra hidden units.

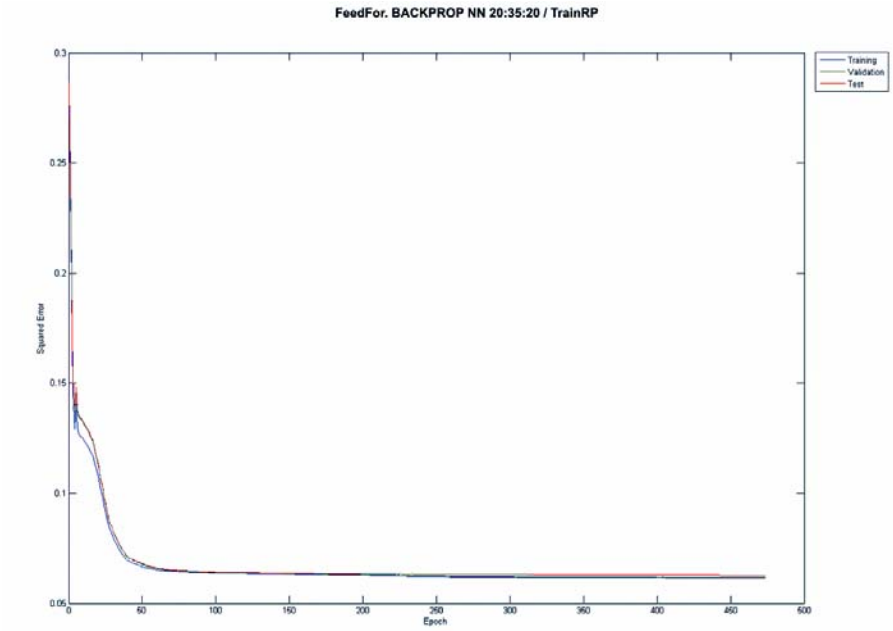


Figure 5.9 *Training, validation and testing MSE per epoch for the 20:35:20 net architecture*

This configuration presented an enthusiastically lower test MSE and increased the overall performance by almost 1 %. Train error dropped down to 6.1% and test error to 6.9%. Less hidden units would increase the MSE by making the network incapable of learning due to under-fit. More hidden units would cause an over-fit of data; thus a decrease in performance. Training history of the above configuration is shown in Figure 5.9. Training needed over 470 iterations to complete.

```
TRAINRP, Epoch 0/500, MSE 0.281669/1e-005, Gradient 0.0601326/1e-006
TRAINRP, Epoch 5/500, MSE 0.140687/1e-005, Gradient 0.0440236/1e-006
TRAINRP, Epoch 10/500, MSE 0.124172/1e-005, Gradient 0.00488219/1e-006
TRAINRP, Epoch 15/500, MSE 0.118721/1e-005, Gradient 0.0107418/1e-006
TRAINRP, Epoch 20/500, MSE 0.107537/1e-005, Gradient 0.00632738/1e-006
TRAINRP, Epoch 25/500, MSE 0.0918364/1e-005, Gradient 0.00754197/1e-006
TRAINRP, Epoch 30/500, MSE 0.0803103/1e-005, Gradient 0.00587332/1e-006
TRAINRP, Epoch 35/500, MSE 0.0736442/1e-005, Gradient 0.00343099/1e-006
TRAINRP, Epoch 40/500, MSE 0.06946.............................
.....................................................................
.....................................................................
.............................862/1e-005, Gradient 0.00016505/1e-006
TRAINRP, Epoch 455/500, MSE 0.0613725/1e-005, Gradient 0.000141134/1e-006
TRAINRP, Epoch 460/500, MSE 0.0613601/1e-005, Gradient 0.000103177/1e-006
TRAINRP, Epoch 465/500, MSE 0.061347/1e-005, Gradient 0.000222276/1e-006
TRAINRP, Epoch 470/500, MSE 0.0613361/1e-005, Gradient 0.000142073/1e-006
TRAINRP, Epoch 474/500, MSE 0.0613273/1e-005, Gradient 0.000116886/1e-006
TRAINRP, Validation stop.
```

Figure 5.10 *Training MSE history per epoch for the 20:35:20 net architecture*

Having in mind that a TM segment's maximum size is 30 and since the increase of the window size had improved system's performance, the next idea was to extend window length to 30 amino acids and increase hidden units. Experimenting with the number of hidden units resulted in a hidden layer consisted of 45 neurons. More than 45 units made the network unstable and exhausted every single memory resource, although a slight improvement was noted. The implementation of the 30:45:30 configuration on the entire data set resulted in a test MSE decrease (Figure 5.10).

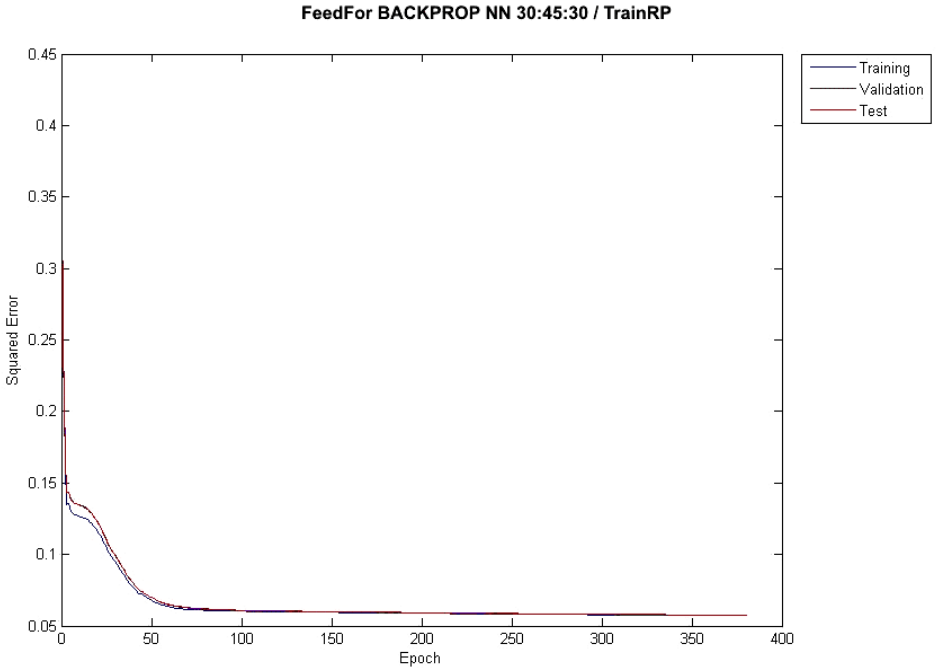


Figure 5.11 *Training, validation and testing MSE per epoch for the 30:45:30 net architecture*

After about 380 iterations early stopping found a global minimum and caused an interrupt: training was terminated forcing train MSE to reach 5.72% and test MSE 5.77%. Figure 5.11 shows a close-up of training, validation and test MSE during the last 110 iterations.

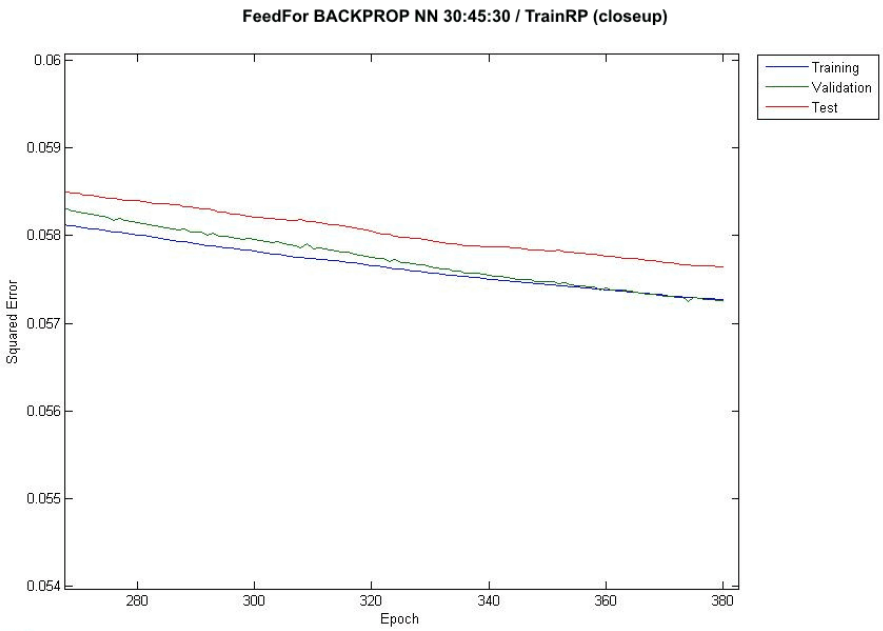


Figure 5.12 *Training, validation and testing MSE during last the 100 epochs for the 30:45:30 net architecture*

The results of the 30:45:30 / TrainRP configuration were extremely satisfactory with overall performance reaching almost 95%. Training history (Figure 5.11) shows the MSE descent per 5 epochs. Similarly to previous configurations, during the first 10 to 15 epochs all MSEs drop quite fast from about 45% to 20% and then to 10-12%. After that, the MSE descents smoothly till the 80th epoch from about 12% to 7.5%; then training continues dropping the validation another 2%-3% before early stopping traces a validation error ascent.

```
TRAINRP, Epoch 0/500, MSE 0.417881/1e-005, Gradient 0.10648/1e-006
TRAINRP, Epoch 5/500, MSE 0.130047/1e-005, Gradient 0.014643/1e-006
TRAINRP, Epoch 10/500, MSE 0.12639/1e-005, Gradient 0.00328666/1e-006
TRAINRP, Epoch 15/500, MSE 0.123407/1e-005, Gradient 0.00855686/1e-006
TRAINRP, Epoch 20/500, MSE 0.11618/1e-005, Gradient 0.00786973/1e-006
TRAINRP, Epoch 25/500, MSE 0.104446/1e-005, Gradient 0.0102068/1e-006
TRAINRP, Epoch 30/500, MSE 0.0945172/1e-005, Gradient 0.00631862/1e-006
TRAINRP, Epoch 35/500, MSE 0.0849155/1e-005, Gradient 0.0046966/1e-006
TRAINRP, Epoch 40/500, MSE 0.07673.....................................
.....................................................................
.....................................................................
.....................................137/1e-005, Gradient 0.000137637/1e-006
TRAINRP, Epoch 360/500, MSE 0.0573833/1e-005, Gradient 0.000245974/1e-006
TRAINRP, Epoch 365/500, MSE 0.057354/1e-005, Gradient 0.00018302/1e-006
TRAINRP, Epoch 370/500, MSE 0.0573196/1e-005, Gradient 0.000170847/1e-006
TRAINRP, Epoch 375/500, MSE 0.0572926/1e-005, Gradient 0.000270288/1e-006
TRAINRP, Epoch 380/500, MSE 0.0572697/1e-005, Gradient 0.000175495/1e-006
TRAINRP, Validation stop.
```

Figure 5.13 *Training MSE history per epoch for the 30:45:30 net architecture*

Figure 5.14 shows the overall output spectrum of the entire test set in three subplots: the actual output spectrum (Figure 5.14a), the predicted output (Figure 5.14b) and the raw network response (Figure 5.14c). In actual output spectrum several thin or thicker green blocks mark the existence of TM segments for each one of the 55 test membrane proteins. Figure 5.14b shows the predicted output of the system which is quite similar to the actual output. The excessive thin, green blocks in the predicted spectrum - before or after the correctly predicted blocks - are included in the overall error and they physically reflect network's local weight weakness to rapidly oscillate between 0 and 1. It is worthy of note that predicted TM segments smaller than 10 or bigger than 30 amino acids are ignored.

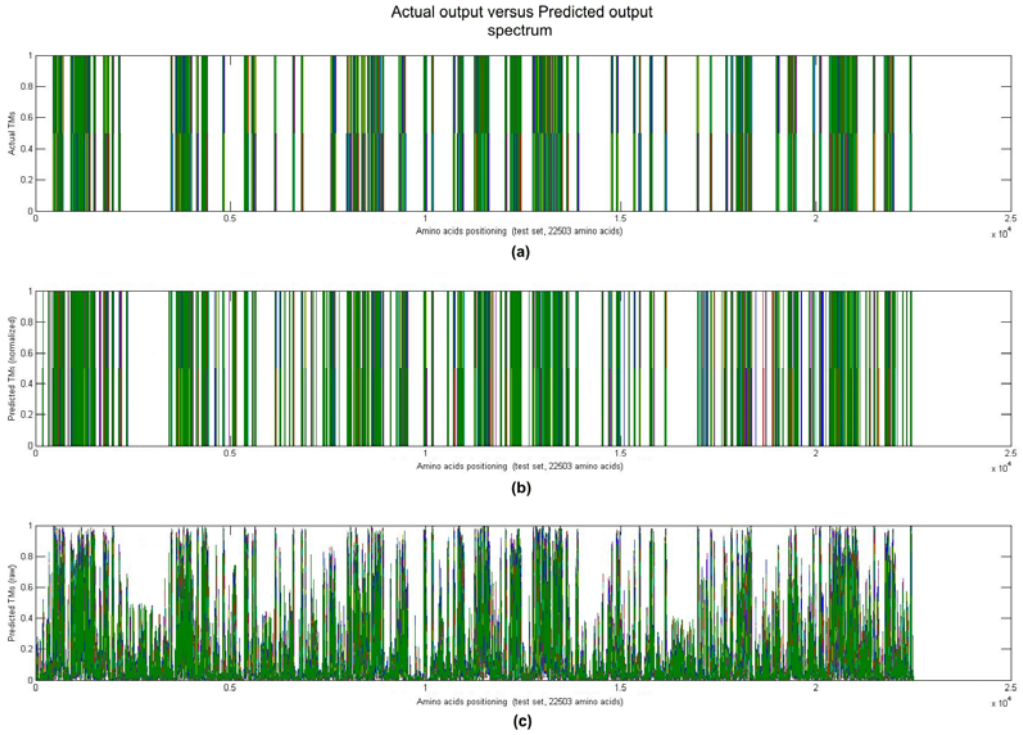


Figure 5.14 *Actual output (a), normalized predicted output (b) and raw output response spectrum for the entire, windowed test set*

Nevertheless, it is necessary to examine how the real output sequences and the above output sequences are tangled with the sliding window method. In reality, test output sequences are <1 by length(testprotein)> vectors. However, according to the sliding window method analyzed in Chapter 4, our systems inputs and outputs were organized in arrays whose rows were equal to window size and whose columns were equal with the number of window slides throughout the test set minus one window (because last window slide stops when its right-end (position 30) hits the last amino acid in sequence without needing to do 30 more slides). Thus, every slide of the window added an extra column to output array. So, the final array would contain many instances of the same amino acid in different slide positions of the window. This means that the second amino acid in sequence would be read twice: once in the beginning of the first window (being element number 2) and twice when the window would slide right by one (being element number 1). In the same way, third amino acid in sequence would be read three times: first as element number 3 of the first window, second as element number 2 of the second window and third as element number 1 of the third window. Although it seems a bit confusing, this method actually re-feeds the same amino

acids into the network thirty times during training (except from the first 30 and last 30 amino acids in sequence), improving learning efficiency and saving execution time by reducing the iterations. Finally, only amino acids' '30-pass' positions will be selected as the final output since in those positions the corresponding input has already re-fed to the network 30 (maximum) times. Figure 5.15 shows the beginning and the end of the input array of protein LRTM2_MOUSE.

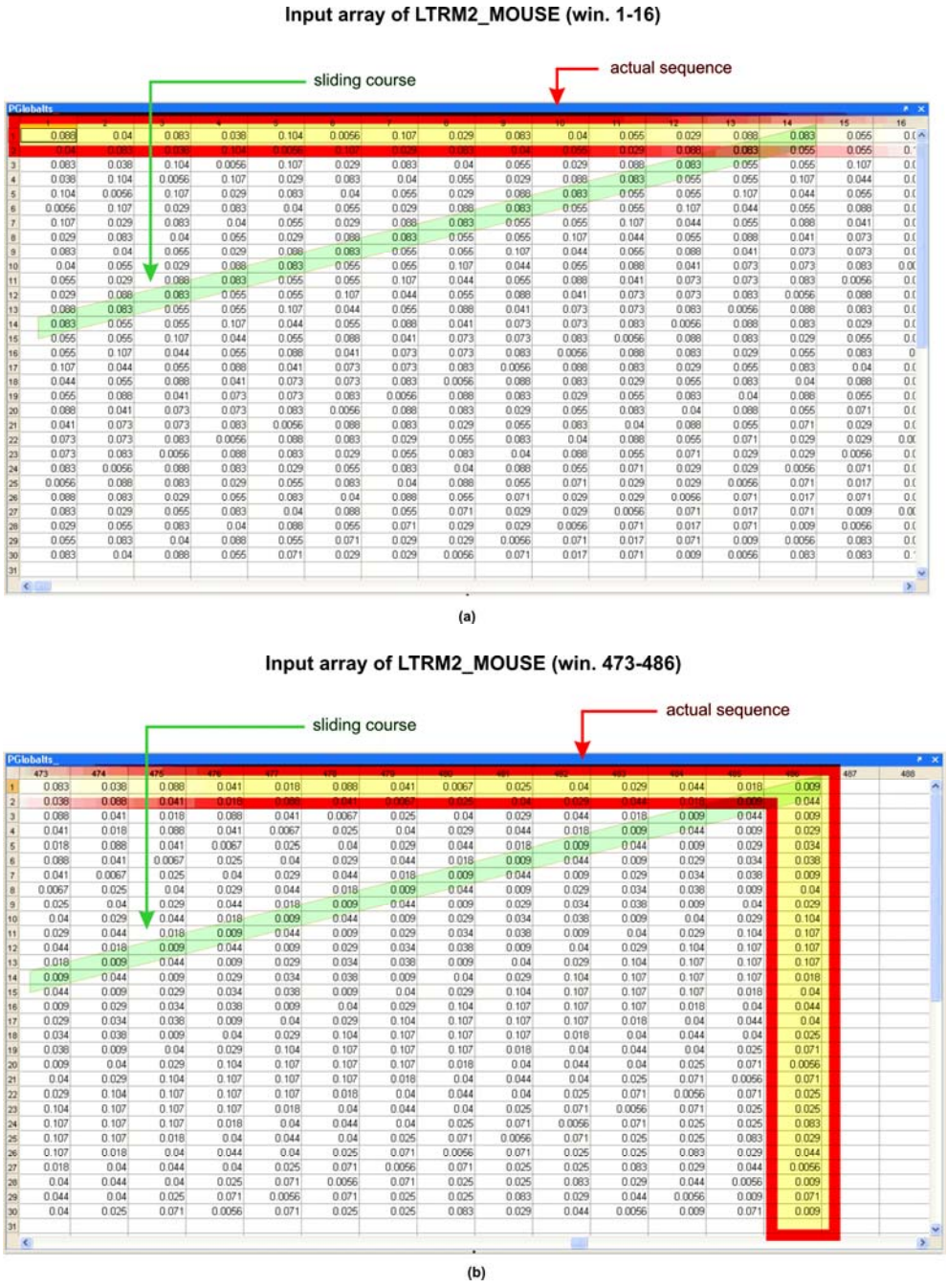
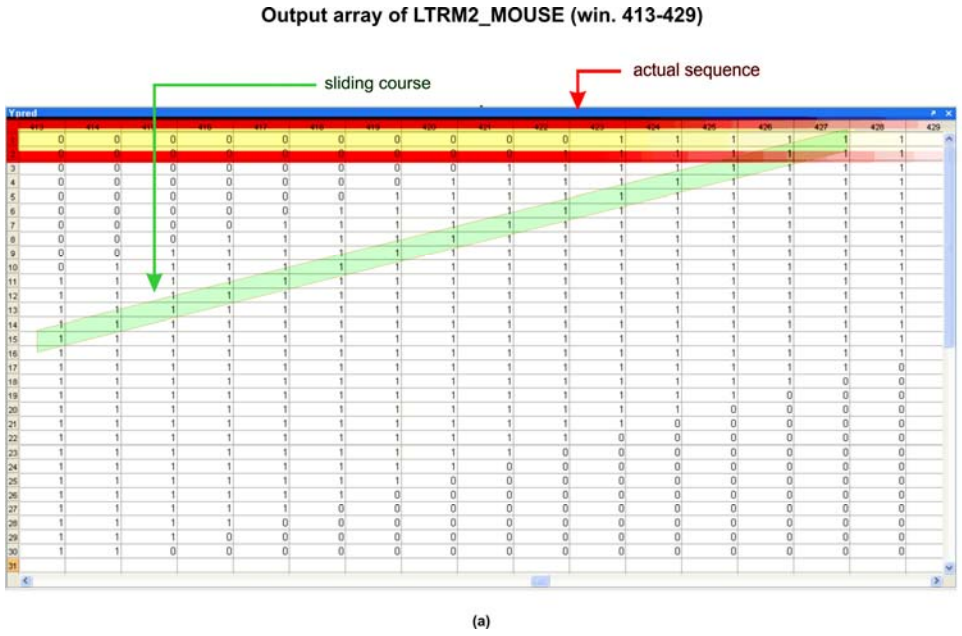


Figure 5.15 *The beginning (a) and the end (b) of protein LTRM2_MOUSE input array. Selected inputs are outlined with a red rectangular.*

In Figure 5.15a the green diagonal shows the position change of the 14th amino acid (L=0.083) and how it is finally selected after being re-fed to the network maximum times for training (maximum for L is 14). Leucine (L) in position 30 (row 30, column 1) would have to re-fed 30 times before it is selected. Note that the 31st amino acid in sequence would be Glycine (0.04) in array position (30, 2). This amino acid will be read 30 times until it diagonally reaches position (1,31) and then placed at first row, within the selection area (red rectangle in Figure 5.15). Since the last 30 amino acids of the sequence would be read only once, selection of inputs completes with the addition of the last window (Figure 5.15b).

The above rules apply to output arrays as well. Both the correct actual and predicted output sequences had to be 'extracted' from the windowed arrays. In accordance with the input 'extraction' method, output sequence was selected from the last response of every amino acid except from the last 30; meaning that the last responses of the system would be more accurate since the decision would be 30 times more 'mature'. Figure 5.16 shows the first 16 (Figure 5.15a) and the last 13 (Figure 5.16b) predicted window outputs of the LRTM2_MOUSE output array.



(a)

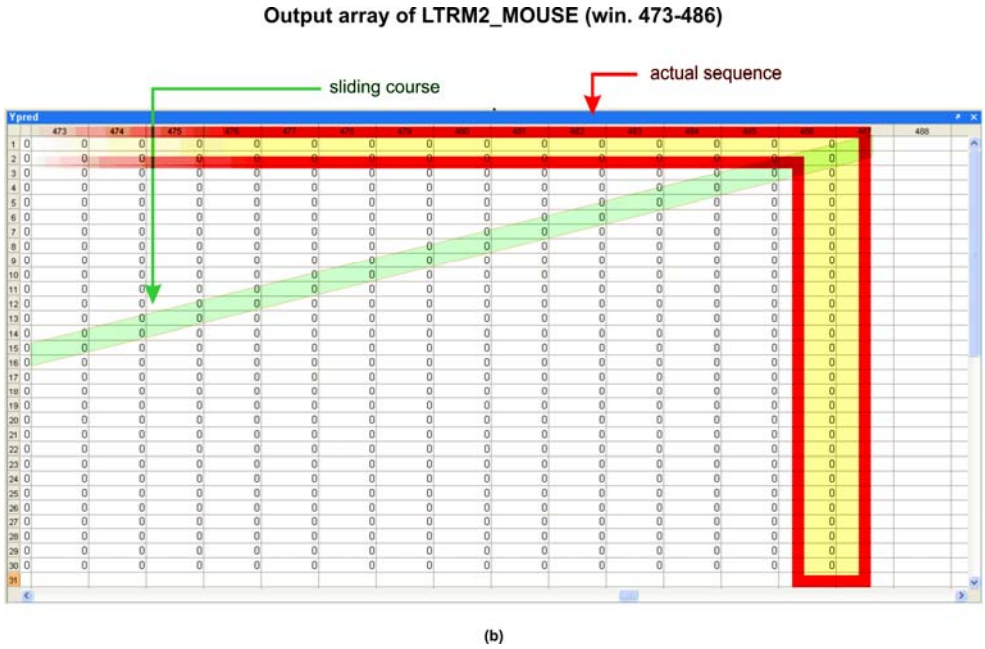


Figure 5.16 *The beginning (a) and the end (b) of protein LTRM2_MOUSE output array. Selected outputs are outlined with a red rectangular.*

After the input / output sequence regulation described above, the evaluation of sensitivity (Q_{se}) and specialty (Q_{sp}) factors can be calculated (equations are referred in Chapter 4). Total number of correctly predicted TM segments was 151 while the total number of actual TM segments was 162 (Table 5.1). The comparison between the actual and predicted output for the entire test set is presented in Figure 5.17. Note that we have excluded TM segments less than 10 and greater than 30 amino acids since these segments are out of TM segment size limits. Finally, overall sensitivity and specialty are:

$$Q_{se} = \frac{q_{predcor-total}}{q_{act-total}} = \frac{151}{162} = 93.20$$

$$Q_{sp} = \frac{q_{predcor-total}}{q_{pred-total}} = \frac{151}{157} = 96.17$$

Overall test MSE stopped at 5.77% and overall performance based on the mean square error reached 94.23%. Next section describes the case study of three proteins and presents the three basic evaluation criteria and local performance scoring of the system.

Test set order	Protein name (Test set)	Dataset order	Final dataset order	q_{act}	q_{pred}	$q_{predcor}$
#1	TMP21_RAT	#356	#351	1	1	1
#2	TMPS4_MOUSE	#357	#357	1	1	1
#3	TMPS4_HUMAN	#358	#356	1	2	1
#4	TMPS2_HUMAN	#359	#352	1	1	1
#5	TMPS3_HUMAN	#360	#354	1	1	1
#6	TMPS2_MOUSE	#361	#353	1	1	1
#7	TMED9_MOUSE	#362	#313	1	1	1
#8	TMM15_HUMAN	#363	#335	14	13	13
#9	LOLE_ECOLI	#364	#139	4	5	4
#10	TMG1_HUMAN	#365	#322	1	1	1
#11	TMPS9_HUMAN	#366	#362	1	1	1
#12	SIDT1_RAT	#367	#201	11	10	10
#13	SCTM1_HUMAN	#368	#195	1	1	1
#14	TM50A_HUMAN	#369	#254	4	3	3
#15	ROR2_HUMAN	#370	#191	1	1	1
#16	TSN7_PANTR	#371	#398	1	1	1
#17	TMPS4_MOUSE	#372	#357	1	2	1
#18	TMG4_HUMAN	#373	#326	1	1	1
#19	NETO2_MOUSE	#374	#172	1	1	1
#20	TMPS6_HUMAN	#375	#360	1	1	1
#21	TM14B_HUMAN	#376	#236	4	4	4
#22	TM16E_HUMAN	#377	#242	8	7	7
#23	TM45A_HUMAN	#378	#246	5	4	4
#24	TM11D_RAT	#379	#227	1	1	1
#25	TMM33_MOUSE	#380	#342	3	3	3
#26	TMCC3_HUMAN	#381	#298	2	1	1
#27	TMPSD_MOUSE	#382	#366	1	1	1
#28	BAMBI_MOUSE	#383	#6	1	1	1
#29	TM60_MOUSE	#384	#266	4	4	4
#30	TM9S2_RAT	#385	#282	9	8	8
#31	LRTM4_HUMAN	#386	#149	1	1	1
#32	**CXCR4_BOVIN**	**#387**	**#29**	**7**	**7**	**7**
#33	TM9S4_HUMAN	#388	#285	9	8	8
#34	STM1_SCHPO	#389	#211	1	1	1
#35	TSN15_HUMAN	#390	#384	4	4	4
#36	LETM1_MOUSE	#391	#129	1	1	1

#37	*EFR1_MACFA*	*#392*	*#45*	*2*	*1*	*1*
#38	*EFCB_PAPAN*	*#393*	*#39*	*2*	*2*	*2*
#39	*TM55B_RAT*	*#394*	*#262*	*2*	*3*	*2*
#40	*ROR2_DROME*	*#395*	*#190*	*1*	*1*	*1*
#41	*LRTM1_PONPY*	*#396*	*#143*	*1*	*1*	*1*
#42	*TM59_MOUSE*	*#397*	*#264*	*1*	*1*	*1*
#43	*ENW1_HYLPI*	*#398*	*#85*	*2*	*2*	*2*
#44	*MS4A7_HUMAN*	*#399*	*#165*	*4*	*3*	*3*
#45	*MTRP_HUMAN*	*#400*	*#166*	*4*	*5*	*4*
#46	***FZOL_SCHPO***	***#401***	***#106***	***2***	***2***	***2***
#47	*SMS1_MOUSE*	*#402*	*#206*	*5*	*5*	*5*
#48	*ENK7_HUMAN*	*#403*	*#66*	*2*	*2*	*2*
#49	*TM9S3_MOUSE*	*#404*	*#284*	*9*	*8*	*8*
#50	*TM47_XENLA*	*#405*	*#252*	*4*	*5*	*4*
#51	*TM50B_PONPY*	*#406*	*#258*	*4*	*4*	*4*
#52	*LRTM2_HUMAN*	*#407*	*#144*	*1*	*1*	*1*
#53	*TMED4_HUMAN*	*#408*	*#303*	*1*	*1*	*1*
#54	*T4S1_PONPY*	*#409*	*#217*	*4*	*4*	*4*
#55	***LRTM2_MOUSE***	***#410***	***#145***	***1***	***1***	***1***
			Total	***162***	***157***	***151***

Table 5.1 *Network's overall scoring per protein for the entire test set (sample test proteins are colored orange)*

If we compare the actual output with the network's response we will notice high prediction scores when the TM segments are scattered within the protein but we will also notice some 'noise' among the transitions of proteins since the instant weight adaptation from zeros to aces or the reverse are virtually impossible due to the fact that weights need some time to gain or loose force respectively.

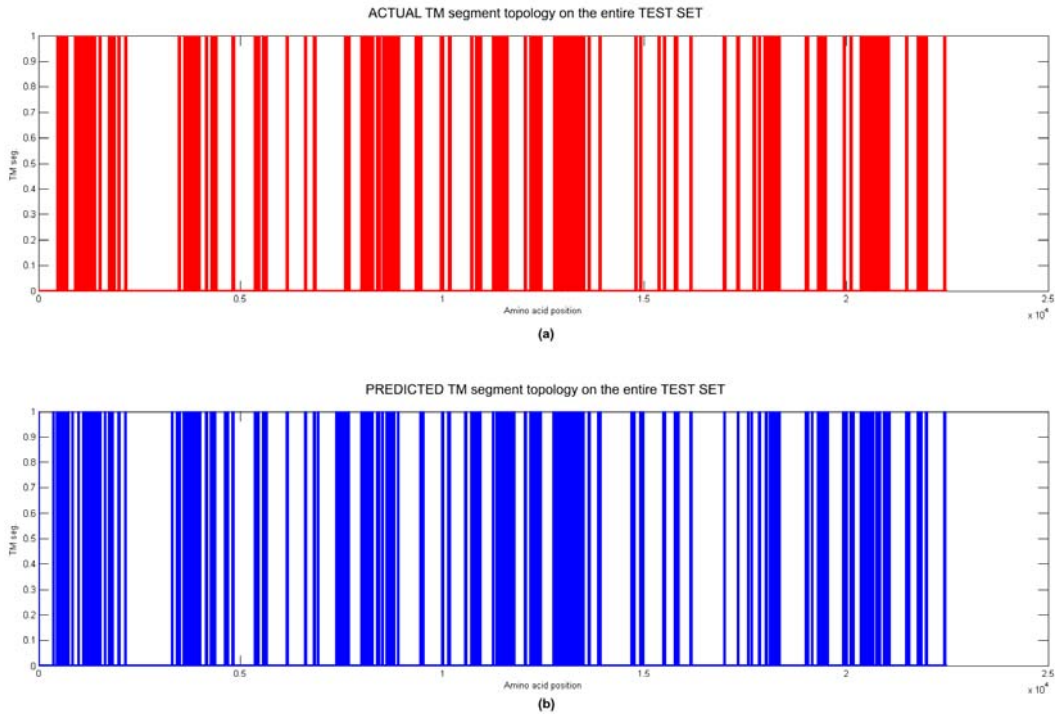


Figure 5.17 *Overall comparisons between the actual (a) and the predicted (b) output for the entire test set*

5.3 Protein sample tests

The network was simulated for a sample set of three proteins: CXCR4_BOVIN, FZOL_SCHPO and LRTM2_MOUSE. Each protein had a different length, was of a different type and had a different number of TM segments with variable lengths; in other words the three proteins had a completely different primary structure. Although overall performance was more or less calculated, protein examples would make the objective of this project more comprehensible.

5.3.1 C-X-C chemokine receptor type 4 (CXCR4_BOVIN)

CXCR4_BOVIN (#29 in our initial dataset) is a multi-pass membrane protein that works as a receptor for the C-X-C chemokine CXCL12/SDF-1. Its

role is to transduce a signal by increasing the intracellular calcium ions level. It exists in bovines and it can be found in brain, heart, lung, kidney and liver tissue. Its primary structure is 353 amino acids long and contains 7 TM segments. A membrane protein with 7 TM segments will be a good evaluation test for multiple segment detection.

TM	*Actual TM segment position*	*Pred. TM segment position*	*Abs. error*
#1	41-64	40-62	3
#2	81-100	82-100	1
#3	112-137	111-139	3
#4	156-176	156-176	0
#5	202-221	202-220	1
#6	242-262	242-263	1
#7	287-308	287-308	0

Table 5.2 *Actual and predicted TM segment topology for CXCR4_BOVIN*

Figure 5.18 shows the actual (red line) and the predicted TM segment topology (blue line). The small noises around the segments are faulty aces and they are ignored since any group of aces smaller than 10 cannot define a TM segment. The network missed some aces and so some predicted segments are slightly displaced.

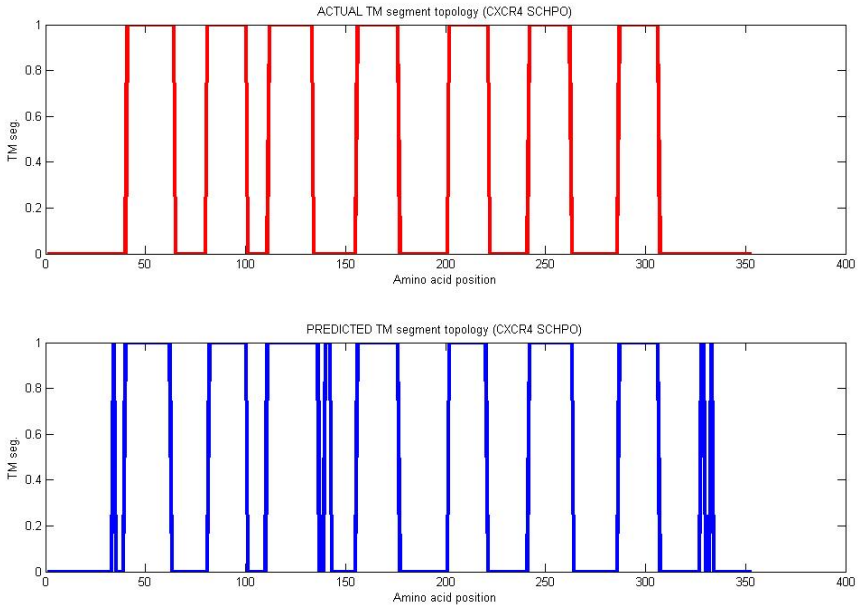


Figure 5.18 *Actual and predicted TM segment topology for FZOL_SCHPO protein.*

It is very interesting to see how the network elaborates the input data and tries to respond. According to Table 5.2, the first actual TM segment of the CXCR4_BOVIN protein is located in position 41 to 64 and the predicted topology is slightly shifted in position 40 to 62. Figure 5.19 shows the predicted output array and focuses on the transition effect between the end of the first TM segment and the beginning of the non-TM region. Unfortunately, weights didn't "hold" in order to keep the ace up to position 64 and zeros started to reappear.

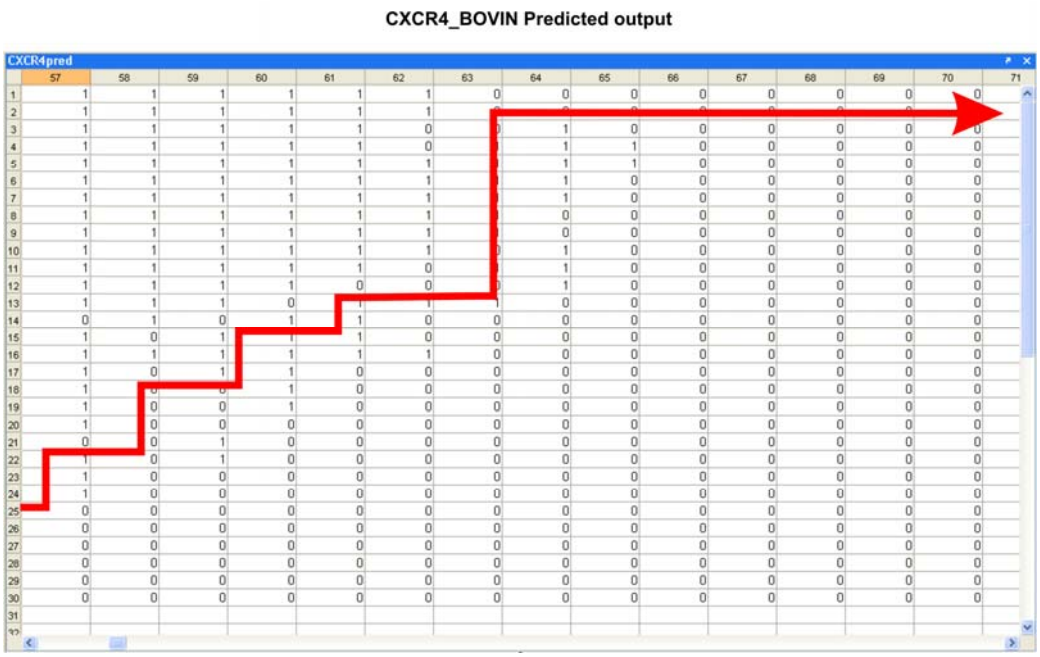


Figure 5.19 *Predicted output array for CXCR4_BOVIN. Red line shows the transition between 1 and 0 while the TM segment fades away.*

5.3.2 Transmembrane GTPase fzo-like protein (FZOL_SCHPO)

FZOL_SCHPO (#106 in our initial dataset) is a multi-pass protein that belongs to the mitofusin family. It exists in saccharomyces and it is usually located in the mitochondrial outer membrane; it probably mediates mitochondrial fusion which is an important step in mitochondria morphology. Protein length is 758 amino acids with 2 TM segments.

TM	*Actual TM segment position*	*Pred. TM segment position*	*Abs. error*
#1	596-616	597-617	2
#2	636-656	637-655	2

Table 5.3 *Actual and predicted TM segment topology for FZOL_SCHPO*

Actual versus predicted output response for FZOL_SCHPO is shown in Figure 5.20 This case proves that the system can handle quite well large protein lengths with a small number of TM segments. Furthermore, two TM segments are quite close, so the density of the TM segments is high adding extra difficulty to the predictor. Two aces in position 340 and 341 of the predicted sequence are ignored since they do not meet the criteria for a valid TM segment.

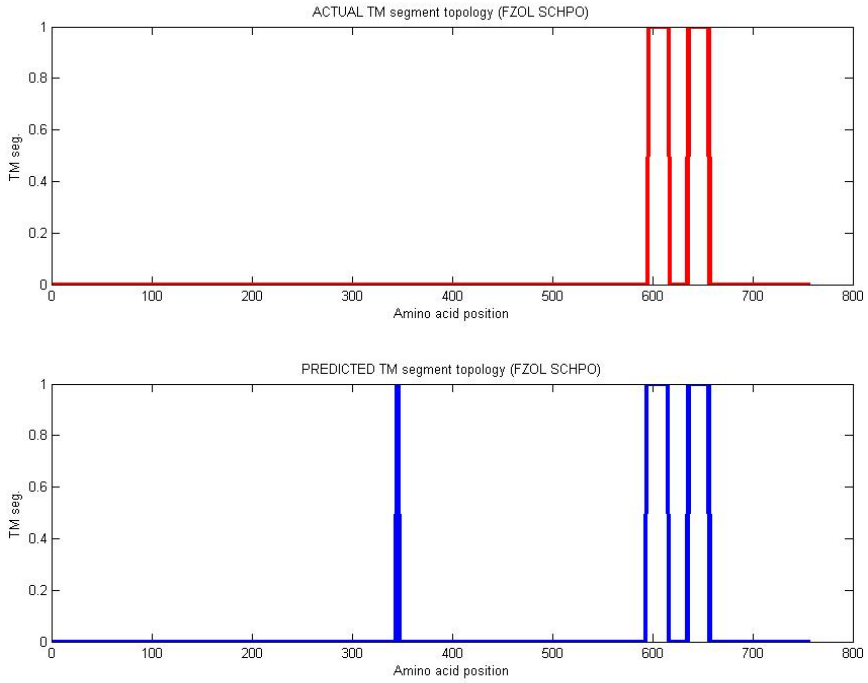


Figure 5.20 *Actual and predicted TM segment topology for FZOL_SCHPO protein.*

Figure 5.21 shows the predicted output array of FZOL_SCHPO from position 334 to 347. The red circle in Figure 5.21 explains the single line that appears in Figure 5.19 and is away from the correctly predicted TM segments. These aces falsely appear in position 340-341, they are not part of a group of aces larger than 10 (minimum size for a default TM segment); therefore they are completely ignored.

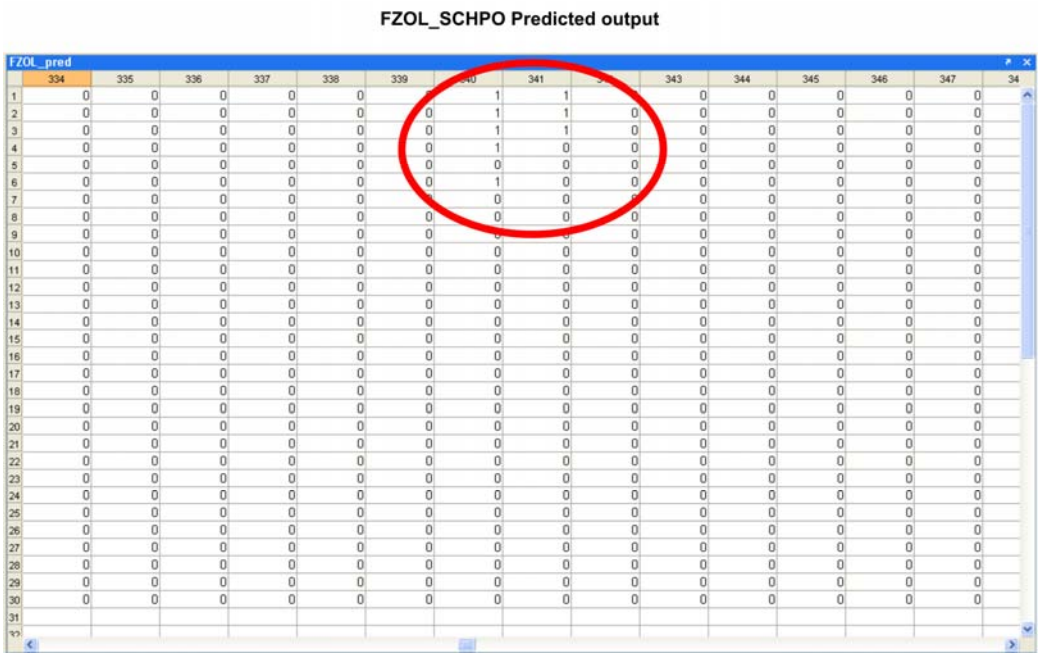


Figure 5.21 *Predicted output array for FZOL_SCHPO. Red circle shows a region of aces that has been ignored due to its limited size.*

5.3.3 Leucine-rich repeat transmembrane neuronal protein (LRTM2_MOUSE)

LRTM2_MOUSE (#145 in our initial dataset) is a single-pass membrane protein that belongs to the LRTM family and exists in mice. Possibly, it plays a role in the development and maintenance of the vertebrate nervous system and is expressed in neuronal tissues. Its sequence length is 515 amino acids and has only 1 TM segment. We wanted to test how system would react in a single TM segment and how fast it could reach few aces (1) in a plethora of zeros. Table 5.4 shows the actual and the predicted TM segment position in protein.

TM	*Actual TM segment position*	*Pred. TM segment position*	*Abs. error*
#1	422-442	423-442	1

Table 5.4 *Actual and predicted TM segment topology for LRTM2_MOUSE*

Figure 5.22 shows the actual (red line) and the predicted (blue line) output of the system. Results are extremely good given the fact that aces in position 220 and 320 are excluded; minimum TM segment size (10 amino acids) is not met, so they do not form a TM segment.

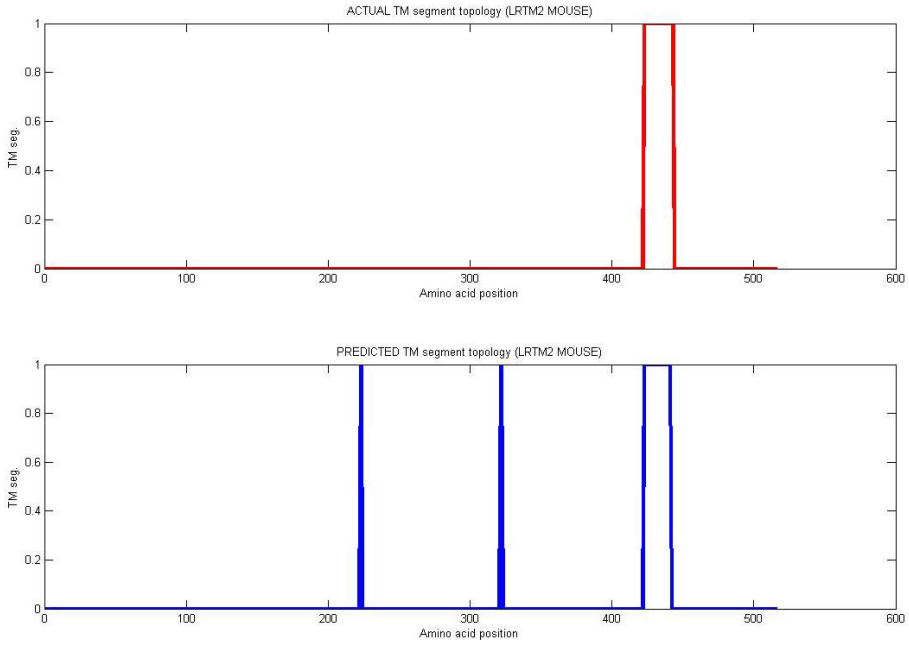


Figure 5.22 *Actual and predicted TM segment topology for LRTM2_MOUSE protein.*

In Figure 5.23, network weights are increased fast in order to give 1 as output at position 422. Unfortunately, the increase was not quick enough and position 422 had just lost an ace. Aces appearance will continue down to position 443 until weights rapid change will produce zeros again.

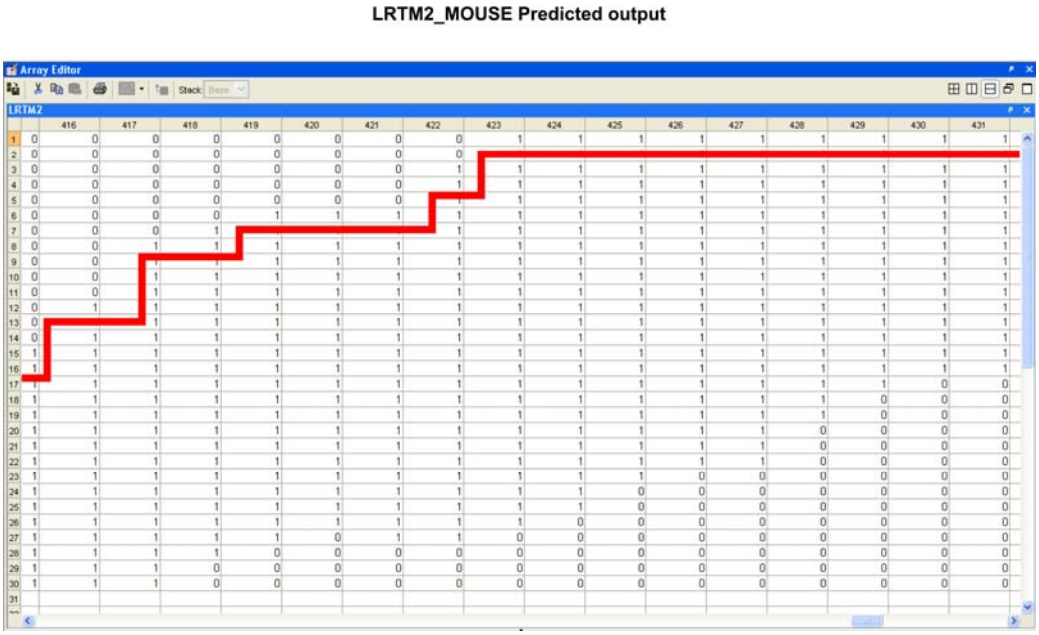


Figure 5.23 *Predicted output array for LRTM2_MOUSE. Red line shows the change between 0 and 1when while the TM segment approaches.*

Chapter 6

Discussion

Membrane proteins are the main communication gateways of the cell. Armed with the ability to span the membrane bi-layer of cells, membrane proteins form communication channels via which cells can effectively receive or send information to other cells. The spanning part of membrane proteins is called transmembrane segment (or transmembrane helix). Proteins may contain one or several transmembrane segments. The size and frequency of these segments determine the structure and role of proteins. Understanding the role of membrane proteins – namely predicting its transmembrane segments – is invaluable for the development of drugs, antibodies or other reagents that will affect the function of proteins. This research suggests a complete, neural network-based prediction system that efficiently reads an unknown human and/or non-human membrane protein and detects the number and position of its transmembrane segments.

6.1 Conclusions

Contrary to traditional laboratory analyses and other statistical methods, the neural network-based transmembrane segments prediction can offer quite high

approximations if supplied with enough information. Apart from quantity, information used to train the network must have quality. Approximately 410 updated, accurate, uncorrupted and fully annotated protein data were extracted from the Swiss-Prot knowledgebase in raw text format and formed our dataset. The dataset is split into three subsets: one training, one validation and one test set. The training set teaches the neural network by providing all the collected, known protein information available. Validation set is responsible for checking the network's response periodically, in order to spot minimum error's exact location. When the network's error starts rising again, validation stops the training procedure and locks the minimum mean square error. Test set evaluates the performance of the system and defines the optimal prediction model.

The original protein data is organized in text files which contain: the protein name, id, description, sequence and TM region pointers. There were three major problems that had to be solved: the encoding of data, the variable protein lengths and the optimal neural network model. Data encoding involves: a) the selection of the best representation of amino acids to numbers and b) the correlation between amino acids and their output topology. Our encoding scheme uses the SMTP method, where each amino acid (input) is replaced by a unique value ranging between 0 and 0.1. This value represents the possibility of an amino acid to belong to a transmembrane segment. Output sequence is a binary sequence of zeros and aces and it is created by placing zeros in non-transmembrane and aces in transmembrane regions. Then, the encoded information passes through a pre-processing phase where output sequences are aligned with input sequences in order to be fed into the network. The inputs and outputs which vary in length are converted into fixed-sized arrays through a sliding-window technique, since neural networks only allow fixed size inputs and outputs. The optimal size of the sliding window is found to be 30 amino acids. All amino acids in input and output sequences are serially scanned in groups of 30 and fed into the network for training. The network is trained and the optimal parameters are calculated after several hundreds of small or large-scale evaluation trials. The output of the neural network is produced by a sequence of zeros and aces: if no transmembrane segment is found, zeros are marking the region whereas if a transmembrane segment is found, aces are marking the region. Predicted outputs are reshaped back to their natural dimensions and are compared with the actual output. This

evaluation contributes to the optimization procedure through the mean square error minimization (mean square error is the fraction of the error sum of squares divided by the total number of data points). The results include the evaluation of the prediction net and the calculation of the mean square error, sensitivity and specialty indicator. Sensitivity (Qse) expresses the number of correctly predicted TM segments divided by the total number of actual TM segments, while specialty (Qsp) expresses the number of correctly predicted TM segments divided by the sum of correctly and incorrectly predicted TM segments.

A variety of figures and tables present the final results of this project. Evaluation results such as sensitivity and specialty are compared with the results of different methods. The predicted transmembrane segment topology is compared with the actual topology for the entire test set. Many segments are found in the designated position while others are slightly shifted by one to few amino acids – some segments are not found at all. Performance ejects to 94.23% and the test MSE is held down to 5.77%. Sensitivity is equal to 93.20% and specialty is equal to 96.17%. Moreover, careful examination of individual protein tests confirms that the prediction of the exact topology of transmembrane segments is feasible. In some membrane proteins, there is an exact much between the actual and the predicted output. Many transmembrane segments are predicted slightly shifted by few amino acids. If the shift exceeds the absolute error of 10 amino acids then the transmembrane segment is considered non-predicted. Comparing our results with results coming from other predictive methods, such as statistical analysis and wavelets, a significant improvement of sensitivity and specialty factors can be noticed. Statistical analyses are considered outdated, depend too much on the dataset and score relatively low because of the strict cut-offs of the hydrophobicity indicator. The WCP1 method [21], which is a prediction method based on wavelets, when applied to the same transmembrane detection problem, it produces sensitivity equal to 84.61% and specialty equal to 95.6% [22]. Additionally, the data set used in this project is about five times larger than the one used in the WCP1 method. Hence, it offers great stability to the network and makes training more reliable.

6.2 Future research

As Bill Toomey (Olympic gold medalist and six times world champion in decathlon) once said "It is always possible to improve". In our case, improvement can be summarized to the threefold: database growth, encoding optimizations and network enhancements. A larger protein database can provide the network with more information for the training process and therefore the prediction will become more dependable. Also, different encoding schemes can be tested including new amino acid representation and numbering which will influence network's neural strengths and weight alterations. Since there are virtually no rules in constructing a prediction network, different neural network architectures can be implemented involving a different input and output organization, more hidden layer units or even different learning methods.

From a biological point of view, there is a great interest for the decoding of ORF regions. ORFs (open reading frames) are regions of chromosomes with no STOP triplet that encode proteins with unknown structure and functions, during the splicing process. It has been proved that almost one third of these ORF regions are translated into membrane proteins. Hence, we can identify these regions by defining the structure and role of their unknown membrane proteins.

References

[1] Fleischmann, R. D. et al. "Whole-genome random sequencing and assembly of Haemophilus influenzae Rd". *Science* 269, 1995, pp. 496-512.

[2] Goffeau, A., Barrell, B. G., Bussey, H. et al. "Life with 6000 genes". *Science* 274, 1996, pp. 546-567.

[3] Venter, J. C. et al. "The Sequence of the Human Genome". *Science* 291 (5507) February 2001, pp. 1304-1351

[4] Tobin, A. J. and Morel R. E. *Asking about Cells*. Fort Worth, Sauders College Publishing , 1997

[5] Watson, J. et al *Molecular Biology of the Gene*. 4th ed. San Francisco, Benjamin Cummings Publishing Company, 2001

[6] Thomopoulos, G. N. *Cell Biology*. 1st ed. Thessaloniki, University Studio Press, 1995

[7] Kastritsi, K. D. *Introduction to Biology*. Thessaloniki, Kyriakides Bros Publishers, 1992

[8] Georgatsou, I. G. *Introduction to Biochemistry*. 3rd ed. Thessaloniki, Yahoudi-Yapouli Publishers, 1993

[9] Modrek, B. and Lee, C. "A genomic view of alternative splicing". *Nat Genet*, 2002, vol. 30, pp. 13-19 ,

[10] Gurney, K. *An Introduction to Neural Networks*. London, UCL Press, 1996

[11] Martin, T. H. et al *Neural Network Design*. Boston, MA PWS Publishing, 1996

[12] Stergiou, C. and Siganos, D. "Neural Networks". *SURPRISE Journal*, Imperial College, 1996, vol. 4

[13] Bishop, C. M. *Neural Networks for Pattern Recognition*. Oxford, Oxford University Press, 1995

[14] Demuth, H. et al *Neural Network Toolbox for use with MATLAB*. Version 4.0.6 (MATLAB R14 SP3), Mathworks Inc., 2005

[15] Plutowski, M., Sakata, S., and White, H., "Cross-validation estimates IMSE", *Advances in Neural Information Processing Systems*, Morgan Kaufman, 1993 pp. 391-398

[16] Hoffman, K. and Stoffel, W. "TMbase a dataset of membrane spanning proteins segments", *Biol. Chem.*, Hoppe-Seyler, 1993, pp. 166-171

[17] Goldman, N., Thorne, J and Jones, D.T "Using evolutionary trees in protein secondary structure prediction and other comparative sequence analyses", *Mol. Biol.*, 1996, pp. 487-494

[18] Pasquier, C. and Hamodrakas, S. J. "An hierarchical artificial neural network system for the classification of transmembrane proteins", *Protein Engineering*, 1999, vol. 12, pp. 631-634

[19] Haykin, S. *Neural Networks: A Comprehensive Foundation* 2nd Edition, Prentice Hall, 1998

[20] Riedmiller, M., and H. Braun, "A direct adaptive method for faster backpropagation learning: The RPROP algorithm" Proceedings of the IEEE International Conference on Neural Networks, San Francisco, 1993

[21] Odgen T. and Parzen E., "Data Dependent Thresholding in Nonparametric Regression with Change-point Applications", 1994, pp. 1-25

[22] Panas S. and Zarkogianni K., *Detection of transmembrane regions in membrane proteins using wavelets*, Diploma Thesis, Aristotle University of Thessaloniki, 2003