Neural Network Based Software System for Hemodynamic Simulation

Olga MILJKOVIC                                Milan TUBA                          Milos IVANOVIC
Faculty of Computer Science           Faculty of Computer Science       Faculty of Natural Science
University Megatrend Belgrade       University Megatrend Belgrade      University of Kragujevac
Bulevar umetnosti 29                       Bulevar umetnosti 29                Radoja Domanovića 12
SERBIA                                            SERBIA                                    SERBIA
omiljkovic@megatrend.edu.rs                   tubamilan@ptt.rs                     mivanovic@kg.ac.rs

Abstract: - This paper describes a software system for hemodynamic simulation of the blood flow through an aneurysm with rigid walls. Human health significantly depends on the status of blood vessels, however measuring in vivo the important hemodynamic factors in the arterial blood flow, such as wall shear stress, pressure and velocity in blood vessels, is very difficult. Finite Element Mesh (FEM) of the observed blood vessel is a starting point in a patient specific blood flow simulation and measurement of these hemodynamics. A feed-forward neural network is proposed here that performs required computations in each point of the predetermined geometry of the mesh. The architecture of that neural network is two layers with sufficient number of neurons in the hidden layer. Optimal value is empirically determined. The first step in the training process is assembling input vectors and the corresponding target vectors, obtained as an output of MedCFD application. Among many variations of backpropagation training functions implemented by generalizing the Widrow-Hoff learning rule, created network object uses Levenberg-Marquardt algorithm. Training continues until the network can associate input vectors with appropriate output vectors. Finally, network is able to simulate and give good responses to new inputs.

Key-Words: - Neural Network, Backpropagation, Finite Element Mesh, Hemodynamics, Levenberg-Marquardt algorithm

1 Introduction
Precise knowledge of the hemodynamic factors, such as blood pressure, velocity, wall shear stress, residence time and impingent flow is valuable for understanding of formation and rapture of pathological changes on blood vessels and in the outcome of their endovascular and surgical treatment. There is a growing trend to practice minimally invasive endovascular procedures in measuring these important hemodynamic variables. Finite Element Method (FEM), as one of the image-based computational method, makes possible to carry out hemodynamic simulation without measuring in vivo. Finite Element Mesh of the observed blood vessel is a starting point in blood flow simulation and measurement of important hemodynamics [1]. MedCFD application uses FEM and generates precise structured Finite Element Mesh of the blood vessel, determines its geometry and gives its 3-D graphical figuration. MedCFD performs hemodynamic simulation and gives as a result the values of the relevant hemodynamic parameters in each mesh point.

The neural network is a structure that predicts a value $y$ for a given input vector $x$ with the weights $w$ being the coefficients. If we choose these weights to minimize the mean square error using observations in a training set, these weights would simply be the least squares estimates of the coefficients. The weights in neural nets are also often designed to minimize mean square error in a training data set. Neural network is useful for functional prediction and system modeling. Network with biases, a sigmoid layer, and a linear output layer is capable of approximating any function with a finite number of discontinuities. The inputs and outputs of MedCFD application are vectors for training of the Neural Networks which will be able to perform hemodynamic simulation and give results by values of the blood pressure, velocity and wall shear stress in each mesh point of the blood vessel.

2 MedCFD Application
The program MedCFD (developed by Supercomputing center in University of Kragujevac, Serbia) is an interactive program for the simulation of blood flow through an aneurysm with rigid walls. Using the MedCFD, it is possible to determine and evaluate the effects of flow characteristics. It was
found that the blood flow through an aneurysm is very complex, with reattachment and separation points. Such saddle points can be determined. The program MedCFD provides automatic generation of a realistic geometric model of an aneurysm (Fig.1.). It helps in predicting the aneurysm growth and the danger of rupture. Users of this program could be vascular surgeons, physicians and biomedical engineers.

Figure1. Model of an aneurysm

Finite Element Method is a discretization of the geometric definition of the object into valid finite elements. This process consists of node and element generation. Recognizing and creating contours of the object from medical images for automatic mesh generation precedes FEM. 2-D and 3-D object can be modeled by several graphical primitives like triangles, rectangles, cubes, blocks, cylinders [1]. Finite Element Mesh is a starting point in blood flow simulation and measurement of important hemodynamic variables. MedCFD is used for mesh generation of elements and nodes. Aneurysm and smoothness parameters are used for determination of a geometry of the blood vessel with pathological change. Mesh and physical parameters (layer division, number of layers and velocity multiplier) define a full Finite Element Mesh, whose graphical figuration is shown with 3-D cross section [2].

MedCFD application offers the option of Finite Element analysis which lists the values of important hemodynamic factors.

3 The Gradient Descent Algorithm

Neural networks are useful for functional prediction and system modeling where the physical processes are not understood or are highly complex. Networks with biases, a sigmoid layer, and a linear output layer are capable of approximating any function with a finite number of discontinuities [3].

Multiple layers of neurons with nonlinear transfer functions allow the network to learn nonlinear and linear relationships between input and output vectors. The disadvantage of neural nets is that they require a lot of data to give good confidence in the results. With higher numbers of inputs, the number of connections and hence the complexity increase rapidly. Multilayer nets are much harder to train than single layer networks. That is, convergence is much slower and speed-up techniques are more complicated.

In order to train neural networks by gradient descent algorithm, we need to be able to compute the gradient G of the loss function with respect to each weight wij of the network. It tells us how a small change in that weight will affect the overall error E. To find the gradient G for the entire data set, we sum at each weight the contribution over all the data points. We can then subtract a small learning rate µ of G from the weights to perform gradient descent. The algorithm follows [4].

1. Initialize all weights to small random values.
2. REPEAT until done
   1. For each weight w$_{ij}$ set $\Delta w_{ij} = 0$
   2. For each data point $(x,t)$
      1. set input units to x
      2. compute value of output units
      3. For each weight $w_{ij}$ set $\Delta w_{ij} = \Delta w_{ij} + (t_i - y_i) y_j$
   3. For each weight $w_{ij}$
      set $w_{ij} = w_{ij} + \mu \cdot \Delta w_{ij}$.

The algorithm terminates once we are at, or sufficiently near to, the minimum of the error function, where $G = 0$.

4 Backpropagation

Backpropagation was created by generalizing the Widrow-Hoff learning rule to multiple-layer networks and nonlinear differentiable transfer functions. Input vectors and the corresponding target vectors are used to train a network until it can approximate a function, associate input vectors with
specific output vectors \[5\], \[6\]. Standard backpropagation is a gradient descent algorithm, in which the network weights are moved along the negative of the gradient of the performance function.

For layered feedforward networks that are fully connected, each node in a given layer connects to every node in the next layer, is often more convenient to write the backprop algorithm in matrix notation rather than using more general graph form. In this notation, the biases weights, net inputs, activations, and error signals for all units in a layer are combined into vectors, while all the non-bias weights from one layer to the next form a matrix \(W\). Layers are numbered from 0 (the input layer) to \(L\) (the output layer). The backprop algorithm then looks as follows:

1. Initialize the input layer: \(y_0 = x\).
2. Propagate activity forward: for \(i = 1, 2, ..., L\) \(y_i = f_i(W_i y_{i-1} + b_i)\) where \(b_i\) is the vector of bias weights.
3. Calculate the error in the output layer: \(\delta_L = t - y_L\).
4. Backpropagate the error: for \(i = L-1, L-2, ..., 1\) \(\delta_i = (W^T_{i+1} \delta_{i+1}) \cdot f'_i(\text{net}_i)\) where \(T\) is the matrix transposition operator.
5. Update the weights and biases: \(\Delta W_i = \delta_i y_{i-1}^T \Delta b_i = \delta_i\).

This notation is significantly more compact than the graph form, even though it describes exactly the same sequence of operations \[4\].

The simplest implementation of backpropagation learning updates the network weights and biases in the direction in which the performance function decreases most rapidly - the negative of the gradient. One iteration of this algorithm can be written

\[x_{k+1} = x_k - \alpha_k \cdot g_k\]

where \(x_k\) is a vector of current weights and biases, \(g_k\) is the current gradient, and \(\alpha_k\) is the learning rate. In the batch mode all of the inputs are applied to the network before the weights are updated \[7\]. There are generally four steps in the training process: assembling the training data, creating the network object, training the network and simulating the network response to new inputs \[8\]. A new input leads to an output similar to the correct output for input vectors used in training that are similar to the new input being presented.

### 5 Levenberg-Marquardt Algorithm (LMA)

Levenberg-Marquardt algorithm was designed to approach second-order training speed without having to compute the Hessian matrix \[9\]. When the performance function has the form of a sum of squares, then the Hessian matrix can be approximated as

\[H = J^T \cdot J\]

and the gradient can be computed as

\[G = J^T \cdot E\]

where \(J\) is the Jacobian matrix that contains first derivatives of the network errors with respect to the weights and biases, and \(E\) is a vector of network errors.

The Jacobian matrix can be computed through a standard backpropagation technique. The Levenberg-Marquardt algorithm uses this approximation to the Hessian matrix in the following update

\[x_{k+1} = x_k - (J^T \cdot J + \mu \cdot I)^{-1} J^T \cdot E\]

When the scalar \(\mu\) is zero, this is just Newton's method. When \(\mu\) is large, this becomes gradient descent with a small step size. \(\mu\) is decreased after each successful step and is increased only when a tentative step would increase the performance function \[3\]. The performance function will always be reduced at each iteration of the algorithm.

The parameters for network training function that updates weight and bias values according to MatLab implementation of Levenberg-Marquardt optimization are:

- maximum number of epochs to train
- performance goal
- maximum validation failures
- factor to use for memory/speed
• minimum performance gradient number of epochs for training
• initial value for \( \mu \)
• parameter by which initial value for \( \mu \) is multiplied, whenever the performance function is reduced by a step
• parameter by which initial value for \( \mu \) is multiplied, whenever a step would increase the performance function
• the parameter \( \text{mem}_\text{reduc} \) is used to control the amount of memory used by the algorithm [10], [11].

If the initial value for \( \mu \) becomes larger than \( \text{mu}_\text{max} \) parameter, the algorithm is stopped. This algorithm appears to be the fastest method for training moderate-sized feedforward neural networks.

6 Neural Reasoning in Hemodynamic simulation

Multiple layers of neurons with nonlinear transfer functions allow the network to learn nonlinear and linear relationships between input and output vectors. Networks with biases, a sigmoid layer, and a linear output layer are capable of approximating any function with a finite number of discontinuities [12], [13]. This is the architecture of the neural networks which compute the values of hemodynamic factors in the arterial blood flow.

MedCFD application uses FEM and generates precise structured Finite Element Mesh whose geometry is determined by groups of parameters is shown in the Fig.2. The aneurysm and smoothness parameters are given on the right picture.

MedCFD application performs hemodynamic simulation and lists the results of Finite Element analysis - velocity, pressure and wall shear stress for all mesh points. Fig.4. shows Finite Element analysis report.

If the initial value for \( \mu \) becomes larger than \( \text{mu}_\text{max} \) parameter, the algorithm is stopped. This algorithm appears to be the fastest method for training moderate-sized feedforward neural networks.

![Figure 2. Defining of an aneurysm geometry](image1)

3-D aneurysm model is shown in the Fig.3.

![Figure 3  3D cross-section of the aneurysm](image2)

![Figure 4 Computational window](image3)

Neural network, responsible for corresponding FEM point, will approximate the results of MedCFD application in this point. MedCFD input/output variables are input/output vectors for training of all Neural Networks.

Input training vector consists of five parameters: time period, B, C, aneurysm_length and velocity multiplier. Output training vector consists of seven parameters: \( V_x, V_y, V_z \) (velocity coordinates), \( P \) (pressure) and \( S_x, S_y, S_z \) (shear stress coordinates).

In order to gain the best resulting fit, MatLab application searches for the appropriate number of hidden units. It starts with architecture of number of neurons equals 10 and grows each of the networks, adding one by one additional hidden unit, with weights from inputs and to outputs. It initializes new weights randomly and resumes training. The process continues until no significant gain is achieved and the improvement in the error over one epoch falls below some threshold. Using this criteria, it reaches number of neurons equal to 97. After this number,
there is no further enhancement. Adding additional neurons in the hidden layer causes no better correlation between network results and output training parameters and only prolongs the training time. Learning with 97 hidden units is considerably harder than learning with smaller number, and takes significantly longer time. Having too large hidden layer can degrade the network’s performance. This is a clue for stopping.

The first step in training a feedforward network is to create the network object. Weights and biases have been initialized and the network is ready for training. Backpropagation is used to calculate the Jacobian jX of the training performance with respect to the weight and bias variables X. Each variable is adjusted according to Levenberg-Marquardt,

\[ jj = jX * jX, \quad jk = jX * E, \quad dX = -(jj+I*mu) \cdot jk \]

where E is the error and I is the identity matrix. The adaptive value mu is increased until the change above results in a reduced performance value [3]. Early stopping is a method for improving generalization. It is a technique based on dividing the data into three subsets. The first subset is the training set used for computing the gradient and updating the network weights and biases. The second subset is the validation set. When the validation error increases for a specified number of iterations, the training is stopped, and the weights and biases at the minimum of the validation error are returned. As the validation data is independent of the training data, network performance is a good measure of generalization, and as long as the network is learning the underlying structure of the data, performance on the validation set will improve with training. Once the network stops learning things which are expected to be true of any data sample and learns things which are true only of this sample, performance on the validation set will stop improving, and will typically get worse. The third subset is the test set. It is used to verify the network design [14]. Test vectors are used as a further check that the network is generalizing well, but do not have any effect on training.

Finally, Neural Network Reasoning application performs hemodynamic simulation at any mesh point and gives a suitable report by the window shown in Fig. 5. This information is a recommendation in prevention of fatal consequences which can occur when a blood vessel breaks. It is a clue in a further clinical analysis and medical treatment of blood vessels with pathological changes.

Figure 5 Neural Network Reasoning window

7 Conclusion
Importance of measuring hemodynamic factors in blood vessels is valuable for understanding formation and rapture of pathological changes on blood vessels and in the outcome of their endovascular and surgical treatment. There is a growing trend to practice minimally invasive endovascular procedures in measurement of these factors. Finite Element Method (FEM), as one of image-based computational method, is one of such methods which can carry out hemodynamic simulation. Based on FEM, MedCFD software tool, using all needed parameters which describe the Finite Element Mesh, generates 3-D precise geometry of the observed blood vessel, performs hemodynamic simulation and gives a list of derived hemodynamics - blood pressure, velocity and shear stress coordinates, in each mesh point.

Neural network is useful for functional prediction and system modeling. Network with biases, a sigmoid layer, and a linear output layer is capable of approximating any function with a finite number of discontinuities. MedCFD outputs and corresponding inputs compose input and output training vectors for Neural Network created in each mesh point, with the goal to determine hemodynamic factors in the arterial blood flow. Standard backpropagation as a gradient descent algorithm in which the network weights are moved along the negative of the gradient of the performance function is implemented. Levenberg-Marquardt algorithm, as a fast numerical optimization technique, is used here as a Neural Network training technique. It gives the best results according to a regression analysis between the network response and the corresponding targets.
Optimal network architecture regarding number of neurons in the hidden layer is empirically determined and tested.

Set of Neural Networks in each mesh point represents useful software tool in finite element analysis of relevant hemodynamics in the arterial blood flow. Further software improvement and upgrading include other available Neural Network training methods and a right selection of number of neurons in the hidden layer in network architecture. Using suggested methodology, it could be possible to determine an approximation of any functionality between input arterial blood parameters and relevant hemodynamics.

Acknowledgment: This research is supported by Project 144007, Ministry of Science, Republic of Serbia.

References: