Piotr BERA*

APPLYING NEURAL NETWORK IN COMPUTING FILLING COEFFICIENT OF FOUR-STROKE INTERNAL COMBUSTION ENGINE

SUMMARY

Neural networks consist of many simple elements operating in parallel. In supervised training they are capable of finding their own solution to a particular problem, given only examples of proper behavior. It is a very useful method of solving complex, non-linear problems. The following article discusses the usage of artificial neural network to compute the value of filling coefficient of four-stroke internal combustion engines as the function of crankshaft rotational speed and throttle opening angle. The paper presents the idea of a static, two-layer feedforward network trained with the basic backpropagation algorithm in which the weights and biases are updated in the direction of the negative gradient. The article discusses network architecture and data structure, training parameters and result analysis.

Keywords*: neural network, supervised training, backpropagation, internal combustion engine, filling coefficient*

ZASTOSOWANIE SIECI NEURONOWEJ DO OBLICZANIA WSPÓŁCZYNNIKA NAPEŁNIENIA CYLINDRA CZTEROSUWOWEGO SILNIKA SPALANIA WEWNĘTRZNEGO

Sieci neuronowe zbudowane są z dużej liczby prostych elementów działających równolegle. Uczenie z nauczycielem pozwala sieci znaleźć nowe rozwiązanie konkretnego problemu tylko na podstawie zestawu znanych poprawnych zachowań. Jest to skuteczna metoda rozwiązywania złożonych, nieliniowych zagadnień. W poniższym artykule przedstawiono przykład wykorzystania sztucznej sieci neuronowej do obliczania wartości współczynnika napełnienia cylindra czterosuwowych silników spalinowych spalania wewnętrznego w funkcji prędkości obrotowej wału korbowego i kąta otwarcia przepustnicy. Przedstawiono statyczną, dwuwarstwową sieć trenowaną podstawową metodą wstecznej propagacji błędów, w której wartości wag i progów zmieniają się w kierunku ujemnego gradientu na powierzchni błędu. W artykule omówiono budowę sieci i strukturę danych uczących, parametry trenowania oraz analizę wyników.

Słowa kluczowe*: sieć neuronowa, trenowanie z nauczycielem, wsteczna propagacja błędów, silnik spalinowy, współczynnik napełnienia cylindra*

1. INTRODUCTION

The filling coefficient is the ratio of the actual mass flow rate to the ideal mass flow rate. In four-stroke spark ignition engines without charging, working on homogenous mode, the torque is proportional to the value of the filling coefficient. The ideal mass flow rate can be calculated on a basis of air temperature and pressure in the intake manifold, displacement volume and crankshaft rotational speed. The actual mass flow rate is computed with the use of mass air flow sensor (Serdecki 2001).

The value of the coefficient depends on many construction and exploitation factors. Those in the first group include: number of valves and their shape, valve timing, diameter and length of intake manifold, engine cooling and exhaust system, etc. In the second group the two most important factors are: crankshaft rotational speed and throttle opening angle (Fig. 1) (Zając *et al.* 2001).

The value of filling coefficient is very important because it influences easy and safe use of the car. The torque and power diagram are widely known only for a full opened throttle. But it is a very rare case in common use of the car. Usually full range of throttle opening angle is used.

2. EXAMPLE OF COMPUTING FILLING COEFFICIENT WITH USE OF NEURAL NETWORK

By using an artificial neural network, it is possible to compute the filling coefficient for the whole range of engine speed and any throttle angle having only a finite number of measurements for a specified throttle opening values.

For solving this particular problem a two-layer, static feedforward network is used.

2.1. Network architecture

There are two inputs: x_1^1 – revolutions [rpm] and x_2^1 – throttle valve opening angle [degrees]. The hidden layer consists of M log-sigmoid artificial neurons (Fig. 2). There is one linear neuron in output layer. Such a network is capable of approximating any function. After proper training, it gives reasonable answers when presented with new inputs that it has never seen before.

* AGH University of Science and Technology, Faculty of Mechanical Engineering and Robotics, Department of Mechanical Engineering and Robotics, al. A. Mickiewicza 30, 30-059 Krakow, Poland; pbera@agh.edu.pl

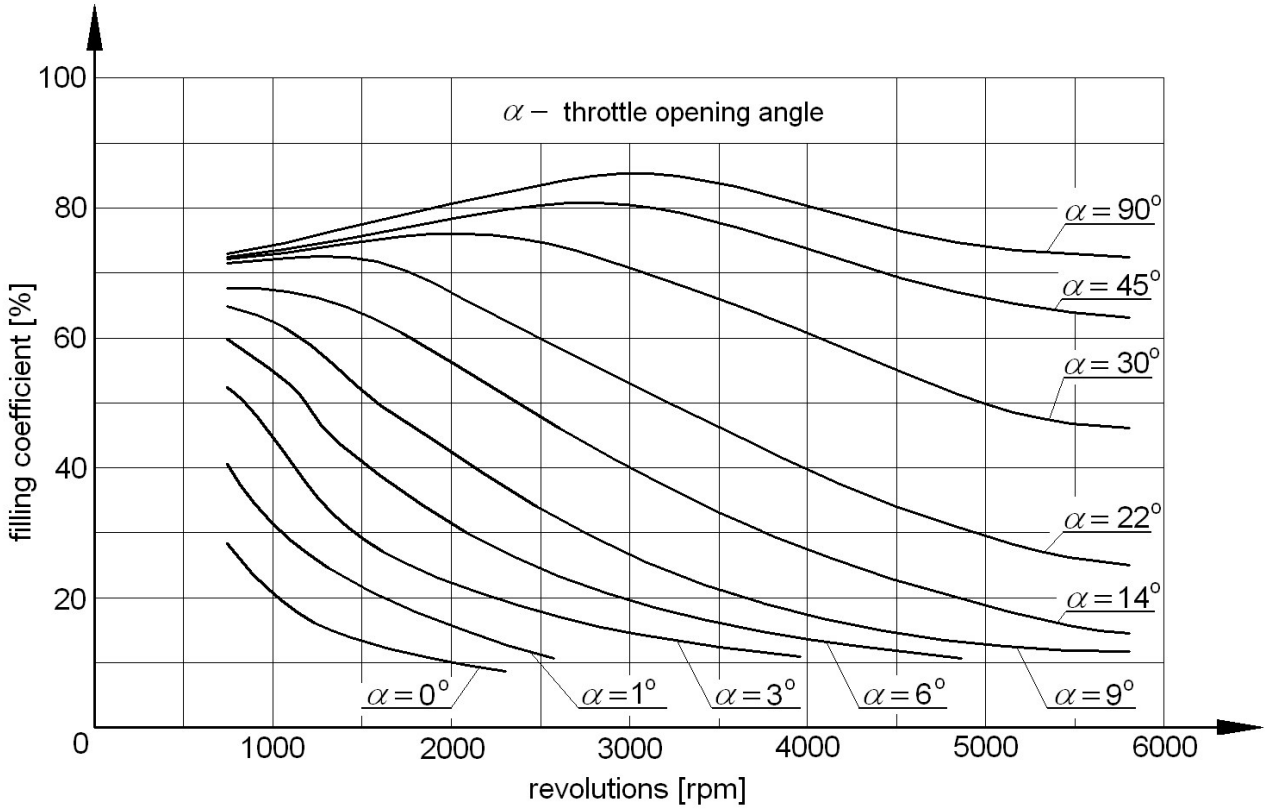

Fig. 1. Filling coefficient vs revolutions for specified throttle opening angles

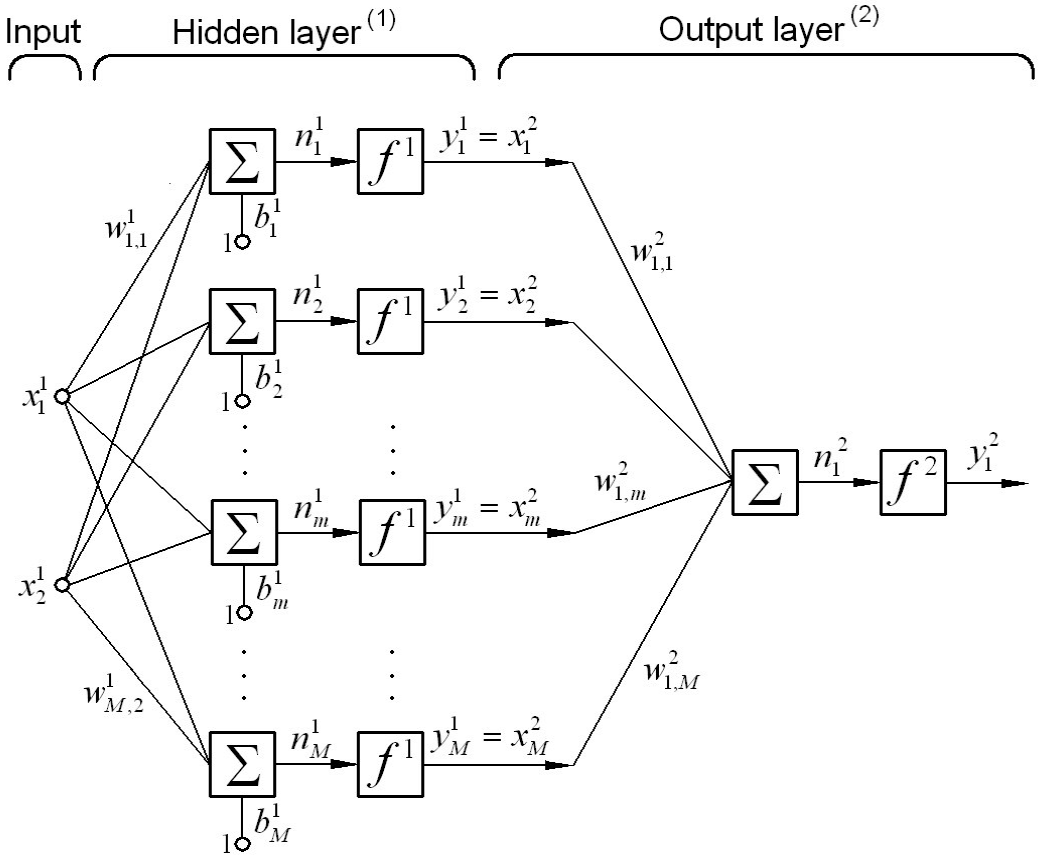

Fig. 2. A two-layer network with two inputs, *M* hidden neurons and one output

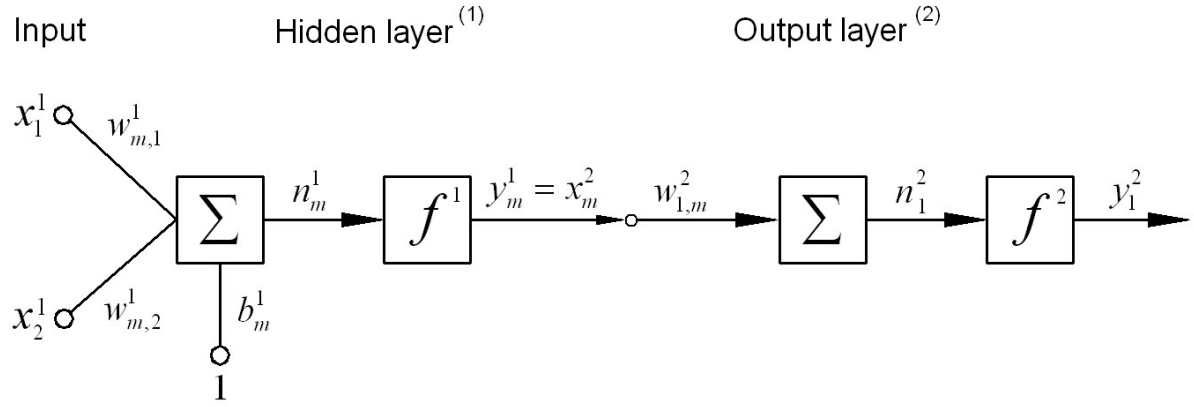

Fig. 3. One representative connection from input through hidden layer to output

Particular elements of Figure 3 are explained below:

$$\mathbf{X} = [x_1^1, x_2^1]^T \tag{1}$$

$$\mathbf{W}^1 = \begin{bmatrix} w_{1,1}^1, & w_{1,2}^1 \\ w_{2,1}^1, & w_{2,2}^1 \\ & \cdots \\ w_{m,1}^1, & w_{m,2}^1 \\ & \cdots \\ w_{M,1}^1, & w_{M,2}^1 \end{bmatrix} \tag{2}$$

$$n_m^1 = \sum_{i=1}^{2} w_{m,i}^1 \cdot x_i^1 + b_m^1 \tag{3}$$

f^1 – log-sigmoid transfer function:

$$y_m^1 = f^1(n_m^1) = \frac{1}{1+e^{-n_m^1}} \tag{4}$$

$$\mathbf{W}^2 = [w_{1,1}^2, w_{1,2}^2 \ldots w_{1,m}^2 \ldots w_{1,M}^2] \tag{5}$$

f^2 – linear transfer function:

$$y_1^2 = f^2(n_1^2) = n_1^2 \tag{6}$$

2.2. Gradient descent training and backpropagation algorithm

Before the training, the weights and biases have random values. Learning a neural network means computing an appropriate set of weights and biases that find proper solution. During the training process they move in the direction of negative gradient, which is the direction of the steepest descent of the error surface (Tadeusiewicz 1993). The global minimum in weight space means best solution. It is shown on a simple example below (start point depends on initial values of weights) (Fig. 4).

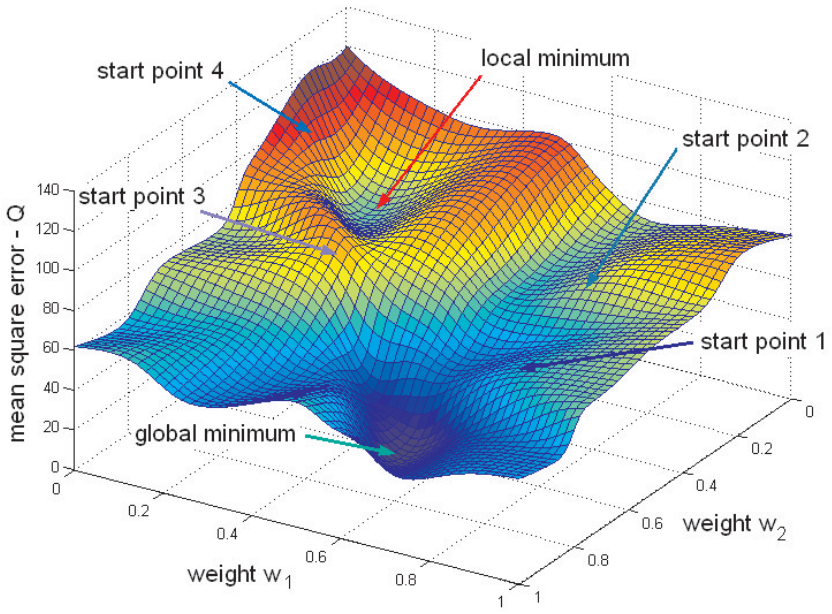

Fig. 4. The hypothetical error surface Q in weight space w_1, w_2

During training process weights and biases are updated to minimize Q – the average squared error between the network outputs y_1^2 and proper answers t:

$$Q = \frac{1}{L} \cdot \sum_{j=1}^{L} Q^{(j)} \tag{7}$$

where:

$$Q^{(j)} = (t^{(j)} - y_1^{2(j)})^2 \tag{8}$$

and:

$$y_1^{2(j)} = f^2\left[\sum_{m=1}^{M} w_{1,m}^2 \cdot \left[f^1\left(\sum_{i=1}^{N} w_{m,i}^1 \cdot x_i^1 + b_m^1\right)\right]\right] \tag{9}$$

where:

i – number of input,
N – total number of inputs (N = 2),
j – number of data from training set,
L – total number of data in training set.

According to gradient descent method (Tadeusiewicz 1993) weight change in output layer can be written as:

$$\Delta w_{1,m}^2{}^{(j)} = w_{1,m}^2{}^{(j+1)} - w_{1,m}^2{}^{(j)} = -\eta \cdot \frac{\partial Q^{(j)}}{\partial w_{1,m}^{2(j)}} \tag{10}$$

The equation above can be written in a different way:

$$\frac{\partial Q^{(j)}}{\partial w_{1,m}^{2(j)}} = \frac{\partial Q^{(j)}}{\partial y_1^{2(j)}} \cdot \frac{\partial y_1^{2(j)}}{\partial w_{1,m}^{2(j)}} = \frac{\partial Q^{(j)}}{\partial y_1^{2(j)}} \cdot \frac{\partial y_1^{2(j)}}{\partial n_1^{2(j)}} \cdot \frac{\partial n_1^{2(j)}}{\partial w_{1,m}^{2(j)}} \tag{11}$$

First factor in equation above equals:

$$\frac{\partial Q^{(j)}}{\partial y_1^{2(j)}} = -\left(t^{(j)} - y_1^{2(j)}\right) = -\delta_1^{2(j)} \tag{12}$$

The last factor has the form:

$$\frac{\partial n_1^{2(j)}}{\partial w_{1,m}^{2(j)}} = x_m^{2(j)} \tag{13}$$

$$\frac{\partial y_1^{2(j)}}{\partial n_1^{2(j)}} = 1 \tag{14}$$

To implement gradient descent training the transfer function must be differentiable. That is why linear function is used in output layer:

$$y_1^2 = f^2(n_1^2) = n_1^2 \tag{15}$$

Finally weight change in output layer equals:

$$\Delta w_{1,m}^2{}^{(j)} = \eta \cdot \left(t^{(j)} - y_1^{2(j)}\right) \cdot \left[\frac{1}{1+\exp(-n_m^1)}\right] \tag{16}$$

where:

$$n_m^1 = \sum_{i=1}^{2} w_{m,i}^1 \cdot x_i^1 + b_m^1 \tag{17}$$

It was easy to compute the error for the output layer. It is also possible to find error for each hidden neuron using backpropagation algorithm. During this phase, the output error is multiplied by the same weights as it happened when input was propagated forward through the network. For the example discussed in the article it is:

$$\delta_m^{1\,(j)} = w_{1,m}^2 \cdot \delta_1^2 = w_{1,m}^2 \cdot \left(t^{(j)} - y_1^{2(j)}\right) \tag{18}$$

Because hidden neurons have log-sigmoid transfer function it is necessary to calculate its derivative:

$$\begin{aligned} y &= f(n) = \frac{1}{1+e^{-n}} \\ y' &= f'(n) = \frac{d}{dn}\frac{1}{1+e^{-n}} = \\ &= \frac{1}{(1+e^{-n})} \cdot \left(1 - \frac{1}{(1+e^{-n})}\right) = y \cdot (1-y) \end{aligned} \tag{19}$$

$$\frac{df(n_m^1)}{dn_m^1} = y_m^1 \cdot \left(1 - y_m^1\right) \tag{20}$$

The final formula for the weight change in hidden layer has the following form:

$$\begin{aligned} &\Delta w_{m,i}^1{}^{(j)} = \\ &= \eta \cdot w_{1,m}^{2(j)} \cdot \left(t^{(j)} - y_1^{2(j)}\right) \cdot \left(1 - y_m^{1\,(j)}\right) \cdot x_i^{1(j)} \cdot y_m^{1\,(j)} \end{aligned} \tag{21}$$

where η – learning rate.

Finally all weight changes can be written as follows:

$$\Delta w_{m,1}^1{}^{(j)} = \eta \cdot \delta_m^{1\,(j)} \cdot \left(1 - y_m^{1\,(j)}\right) \cdot y_m^{1\,(j)} \cdot x_1^{1(j)} \tag{22}$$

$$\Delta w_{m,2}^1{}^{(j)} = \eta \cdot \delta_m^{1\,(j)} \cdot \left(1 - y_m^{1\,(j)}\right) \cdot y_m^{1\,(j)} \cdot x_2^{1(j)} \tag{23}$$

$$\Delta w_{1,m}^2{}^{(j)} = \eta \cdot \delta_1^{2(j)} \cdot \left[\frac{1}{1+\exp(-n_m^1)}\right] \tag{24}$$

$$\Delta b_m^{1\,(j)} = \eta \cdot \delta_m^{1\,(j)} \cdot \left(1 - y_m^{1\,(j)}\right) \cdot y_m^{1\,(j)} \cdot 1 \tag{25}$$

2.3. Data structure and training parameters

The supervised training of neural network requires a set of examples of inputs x_1^1, x_2^1 and proper outputs t. From the data equivalent to those in Figure 1 the training set in created. Each column of matrix **X** presents a single input to the network. In the first row there is a crankshaft rotational speed and in the second row there is a throttle opening angle. Random values from the full matrix containing 220 columns are shown below:

$$\mathbf{X} = \begin{bmatrix} 750 \ldots 2000 \ldots 6000 \ldots 750 \ldots 5000 \ldots 750 \ldots 3000 \ldots 6000 \\ 0 \quad \ldots \quad 0 \quad \ldots \quad 14 \quad \ldots 22 \ldots \quad 22 \quad \ldots 90 \ldots \quad 90 \ldots 90 \end{bmatrix} \tag{26}$$

Actual answers of the network y_1^2 are compared to the values from target matrix **T**. Random columns equivalent to columns in matrix **X** above:

$$\mathbf{T} = [28 \ldots 10 \ldots 15 \ldots 72 \ldots 29 \ldots 75 \ldots 85 \ldots 74] \tag{27}$$

Training set consisting of matrices **X** and **T** have 220 examples of proper behavior (10 throttle angles per 22 values of revolutions). In batch mode weights and biases are updated after the whole training set (all columns from matrices **X** and **T**) has been shown to the network. Values of revolutions and throttle angles are scaled to range (0, 1) so as not to make one of them more important.

Before training weights and biases are initialized. Their values are from range (–3, 3) to fulfill the whole output range (0, 1) of log-sigmoid function (Fig. 5).

It has two positive aspects. Firstly, every neuron having different weight approximates a different part of input space. Secondly, for values from range $(-\infty, -3)$ and $(3, +\infty)$ the weight adjustment, which is proportional to the derivative, will be close to zero so the training process comes in this case to a standstill.

Start value of the learning rate is set to $\eta = 0.02$, whereas it changes during the training. There are 200 training epochs, meaning that the whole training set is shown to the network 200 times. It is enough to achieve the mean square error $Q \approx 0.1$. The diagram of mean square error is shown in Figure 6.

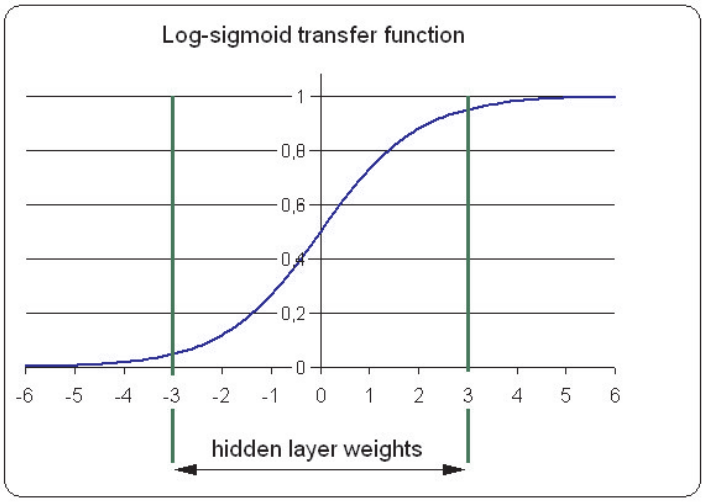

Fig. 5. Initial weights values in case of log-sigmoid transfer function

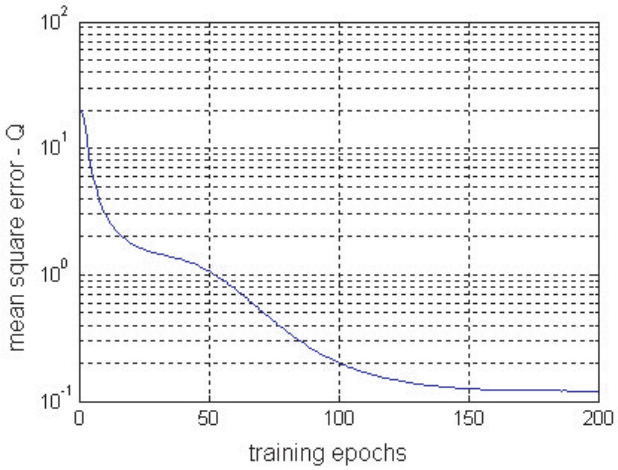

Fig. 6. Mean square error Q vs training epochs

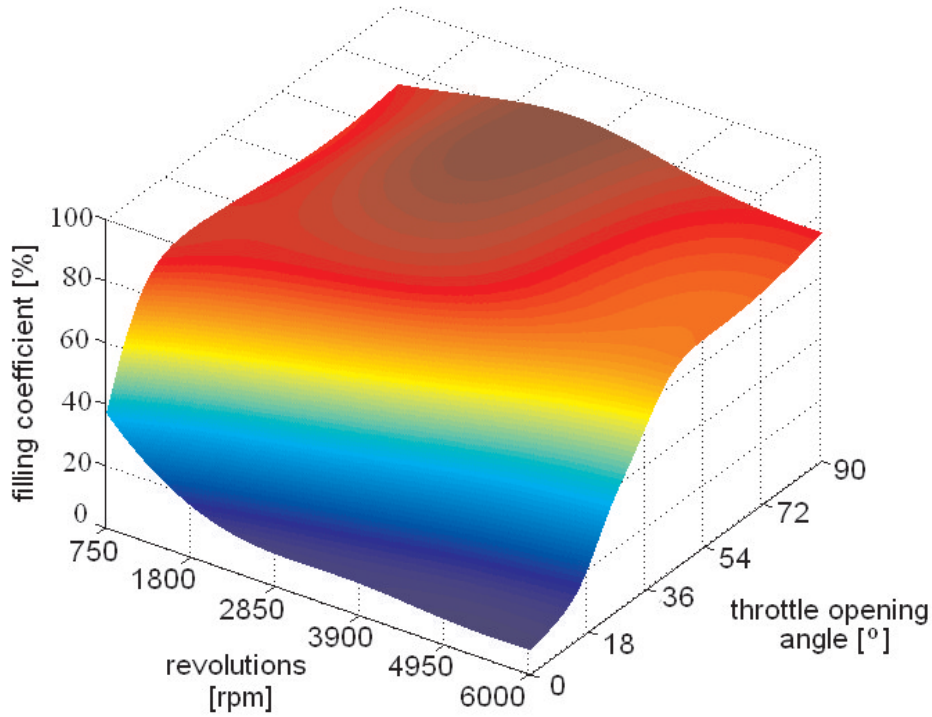

Fig. 7. Filling coefficient vs crankshaft revolutions and throttle opening angle

Number of hidden neurons is set to $M = 10$. It is optimum number because it allows to avoid getting stuck in local minima (Fig. 4) on error surface and it is not too big, so the training set is not overfitted.

It can be easily seen that training process makes the Q smaller. It means that training parameters are correct. In the final 50 epochs, the error rate is almost constant, so training can be finished.

After training the net is simulated with all values from input range and the graph of filling coefficient seen in Figure 7 is received.

2.4. Important aspects of programming backpropagation network

Although the backpropagation learning algorithm is powerful, in order to achieve satisfying solution some aspects should be considered.

The most troublesome is optimum learning rate. Too small learning rate will result in long training process, whereas too big can cause instability of the network. To avoid this problem the method of variable learning rate was implemented. The optimum value is calculated for every training step (MathWorks 2010).

Another issue is the number of hidden neurons. A number that is too small makes the network incapable of approximating problem with satisfying accuracy. Besides the error surface of complex network has many hills and valleys, so there is a risk that network can get trapped in local minimum when learning starts for example in start point 4 (Fig. 4). Suggested possibility is to increase the number of hidden neurons. This will lead to higher dimensionality in error space and it decreases the chance to get trapped in local minimum (learning is more likely to start in start point 1, 2 or 3 – Fig. 4). Of course the number of hidden units can not be too big because after overreaching a limit, the problem with local minima will occur again. Besides, the training process will take much more time and the network can overfit the data (it means that the network gives very good answers to the points from training set but makes big mistakes while being simulated with other examples). Problem of weights and biases initialization has been described before (Tadeusiewicz 1993, MathWorks 2010).

The troubles described above can be solved by trial and error method and watching the mean square error Q which is the evaluation of proper training process. Experience in programming neural networks allows finding best solution in only a few steps.

3. CONCLUSIONS

Applying neural network allows solving complex non-linear technical problems where solution depends on many independent factors. Another advantage of using neural network is that solution is available in every point of input space and not only in particular measured points. The method uses easy mathematic functions. Only knowledge of basic derivatives is required. Although the way the weights and biases are updated is not easy to follow, the result of network performance is controlled by observing mean square error during training process. Thanks to this desirable accuracy can be

achieved by changing number of training epochs or number of hidden neurons. Important things while designing a neural network are structure and parameters which strongly influence performance. Usually, a few tests are required before achieving desired accuracy of the solution.

References

Brzózka J., Dorobczyński L. 2008, *Matlab. Środowisko obliczeń naukowo-technicznych*. Wydawnictwo Naukowe PWN, Warszawa, ISBN 978-83-01-15459-2.

Rokosch U. 2007, *Układy oczyszczania spalin i pokładowe systemy diagnostyczne samochodów OBD*. Wydawnictwa Komunikacji i Łączności, Warszawa, ISBN 978-83-206-1657-6.

Serdecki W. 2001, *Badania silników spalinowych: laboratorium*. 2nd ed., Wydawnictwo Politechniki Poznańskiej, Poznań, ISBN 83-7143-053-1.

Tadeusiewicz R. 1993, *Sieci neuronowe*. Akademicka Oficyna Wydawnicza, Warszawa 1993, ISBN 83-85769-03-X.

Zając P., Kołodziejczyk L. M. 2001, *Silniki spalinowe*. Wydawnictwa Szkolne i Pedagogiczne, Warszawa, ISBN 978-83-02-07987-0.

MathWorks, *Neural network toolbox: user's guide* (Release 2010b), 2010, http://www.mathworks.com/ (20.12.2010).