

Algebraic Inversion of an Artificial Neural Network Classifier

Travis Wiens, Rich Burton and Greg Schoenau *

University of Saskatchewan - Department of Mechanical Engineering
Saskatoon, Sk, Canada

Abstract. Artificial neural networks are, by their definition, non-linear functions. Typically, this means that it is impossible to find a closed-form solution for the inverse function of a neural network. This paper presents a special form of neural network classifier that allows for its algebraic inversion in order to find the boundary between classes. The control of the fuel-air ratio in a spark ignition engine is given as an example.

1 Introduction

An artificial neural network (hereafter referred to as simply a neural network) is a complex system, made of simple identical non-linear parallel elements [1]. Typically, this complexity and non-linearity do not permit one to find an algebraic solution for the inverse of a neural network; that is, to solve for what inputs will result in a particular output. This paper presents a form of neural classifier which permits one to solve this problem in order to find a closed-form solution for the boundary, under certain conditions. This is achieved by using a generalized neural network [2] with certain weights strategically set to zero. An example is used to illustrate the technique: the control of fuel-air ratio in a spark ignition (SI) engine.

2 Problem Definition

Static neural networks are often used as non-linear function approximators or classifiers. For example, in the control of a spark ignition engine, one may wish to classify whether the fuel-air ratio injected into the engine will be rich (too much fuel for complete combustion) or lean (too little fuel), given operating conditions of intake manifold pressure, P_m , engine speed, N_e , and the control action of fuel injector pulse width, t_i . In this case, a neural network model of the "plant", G_p , would be set up as

$$\hat{y} = G_p(P_m, N_e, t_i) \quad (1)$$

where \hat{y} is an estimate of a two-state oxygen sensor output, with 0 signifying a lean air-fuel ratio and 1 rich. One would then take a sample of data and train the

*The authors would like to acknowledge the support of NSERC, in the form of a PGS D Scholarship; the Saskatchewan Research Council for providing equipment and expertise; and General Motors Alternative Fuels for the loan of the test vehicle. Thanks are also given to Natural Resources Canada and Precarn Inc who funded initial work in this area.

network weights in G_p so that the error between the estimated sensor output, \hat{y} and the measured sensor output, y is minimized.

Stoichiometric fuel-air control is based on maintaining the fuel-air ratio near the stoichiometric point, where there is just enough fuel for complete combustion, at the boundary between rich and lean. Thus, given operating points P_m and N_e , the controller must find the value for t_i on the “decision boundary”. This injection pulse width is called the stoichiometric pulse width, t_{is} , and the controller’s estimate of it is \hat{t}_{is} . The control function, G_c , is then the inverse of the plant function:

$$\hat{t}_{is} = G_c(P_m, N_e) = G_p^{-1} \quad (2)$$

or the solution of the equation

$$G_p(P_m, N_e, \hat{t}_{is}) = 0.5 \quad (3)$$

for t_{is} . This corresponds to the transition point between rich ($G_p = 1$) and lean ($G_p = 0$) operation.

There are a number of methods of inverting a nonlinear function, but typically they involve iterative numerical solutions, which are not suitable for real-time control. For example, for an 8-cylinder engine running at 6000 RPM, this calculation must be completed at a minimum of every 2.5 ms on processors with low computational power. The ideal solution would be to be able to algebraically invert the network.

3 Proposed Solution

The proposed solution to the problem outlined above involves the use of a special kind of neural network, known as a generalized neural network, or GNN [2]. Unlike the familiar multilayer perceptron (MLP), which has discrete layers, each of which is only connected to the previous layer, the neurons of a GNN are organized in a line, with each neuron connected to all the neurons to the left of it, as shown in Figure 1. Thus, for a network with four inputs (three inputs plus a 1 for bias), the first hidden neuron, x_5 , would have four inputs; the next neuron, x_6 , would have these four inputs plus the output of x_5 , and so on, such that the output neuron, x_N , in an N -neuron network, would have $N - 1$ inputs. Each neuron in the network is a typical artificial neuron: a non-linear sigmoidal function applied to the weighted sum of the inputs. In this case, a $(1 - e^{-x})^{-1}$ sigmoid was used to scale the outputs to a range of 0 to 1, but any sigmoid may be used.

For the example given above, the equation for the network output is

$$\hat{y} = \text{sig}(W_1x_1 + W_2x_2 + W_3x_3 + W_4x_4 \dots + W_5x_5 + \dots + W_{N-1}x_{N-1}) \quad (4)$$

where W_n is the weight on the n^{th} input x_n , in a network with N neurons, and $\text{sig}(x) = (1 - e^{-x})^{-1}$. If one sets up the network as shown in Figure 1, $x_1 = P_m$, $x_2 = N_e$, $x_3 = t_i$, (all scaled to the range (0,1)), $x_4 = 1$ (for a bias) and the

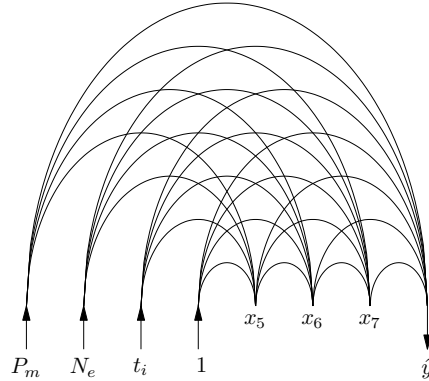


Fig. 1: A Generalized Neural Network (GNN) is a string of neurons, shown here with inputs on the left and outputs on the right. Each neuron is connected to every neuron to left of it. This architecture eliminates the need to determine how many layers to use and how many neurons should be in each layer.

output is scaled such that $y = 0$ is lean and $y = 1$ is rich. Equation 4 now becomes

$$\hat{y} = \text{sig}(W_1 P_m + W_2 N_e + W_3 t_i + W_4 \dots + W_5 x_5 + \dots + W_{N-1} x_{N-1}) \quad (5)$$

The problem is now to solve this equation for \hat{t}_{is} at the boundary, where $\hat{y} = 0.5$. Knowing that $\text{sig}(0) = 0.5$, one may rewrite this equation as

$$\hat{y} = 0.5 = \text{sig}(W_1 P_m + W_2 N_e + W_3 \hat{t}_{is} + W_4 \dots + W_5 x_5 + \dots + W_{N-1} x_{N-1}) \quad (6)$$

$$(7)$$

or

$$0 = W_1 P_m + W_2 N_e + W_3 \hat{t}_{is} + W_4 \dots + W_5 x_5 + \dots + W_{N-1} x_{N-1} \quad (8)$$

If one separates as the hidden neuron outputs into

$$\Phi(P_m, N_e, t_i) = W_5 x_5 + W_6 x_6 + \dots + W_{N-1} x_{N-1}, \quad (9)$$

equation 8 can be manipulated as follows

$$0 = W_1 P_m + W_2 N_e + W_3 \hat{t}_{is} + W_4 + \Phi(P_m, N_e, \hat{t}_{is}) \quad (10)$$

$$\hat{t}_{is} = \frac{-1}{W_3} (W_1 P_m + W_2 N_e + W_4 + \Phi(P_m, N_e, \hat{t}_{is})). \quad (11)$$

Unfortunately, as Φ is a nonlinear function of t_i , this is as far as one can proceed algebraically on a typical network. However, it is possible to proceed if one removes the dependence of Φ on t_i by eliminating (or constraining to zero) the weights that connect t_i to the hidden neurons (but not the output neuron), as shown in Figure 2. Equation 11 then takes the form of

$$\hat{t}_{is} = \frac{-1}{W_3}(W_1 P_m + W_2 N_e + W_4 + \Phi(P_m, N_e)) \quad (12)$$

which is a closed-form solution for the estimated stoichiometric injection pulse width for operating conditions of intake manifold pressure and engine speed.

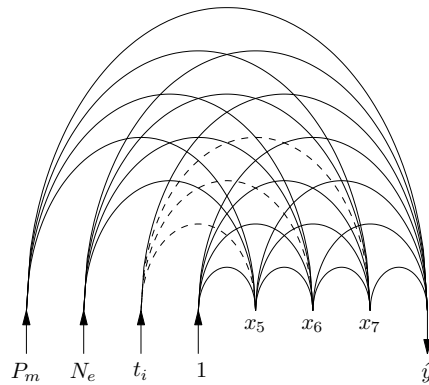


Fig. 2: By setting the weights connecting t_i and the hidden layers to zero (shown as dashed lines), it becomes possible to invert the network. Note that the connection between t_i and the output neuron is not severed.

There are number of a consequences of zeroing the weights connecting one input to the hidden neurons. The weighted sum for the output neuron will be a linear function of t_i . While the neuron output will still be a non-linear function, it will be monotonically increasing or decreasing. This eliminates a class of problems with non-monotonic class boundaries, but eliminates the possibility of multiple solutions. One additional positive consequence of this architecture is that the solution given in Equation 12 can be viewed as a linear equation, augmented by a non-linear term Φ . This can be exploited for initialization or analysis of the network. For example, it is known that the equation for t_{is} is strongly linear with P_m , with a small contribution from N_e and other non-linearities [3][4]. Therefore, plausible initialization values are given by setting $-W_1/W_3$ and $-W_4/W_3$ to match the linear dependence on P_m and setting $W_2, W_5, W_6, \dots, W_N$ to zero or small values. Further details may be found in [5].

4 Simulation Example

The inversion method described in the previous section was used to identify and control a simulated V8 engine following the model in [4], assuming a fuel of

natural gas. The engine was fed inputs of intake manifold pressure and engine speed that were previously recorded from actual city driving. Upon each injection, the controller used online backpropagation[1][6] to train a GNN to match the simulated sensor signal (which included a pure time delay). The controller then used the inverted network (equation 12) to determine an estimate of the stoichiometric pulse width for the next injection. A network with 15 hidden neurons was used for the model.

The results of the simulation study are shown in Figures 3 and 4. Figure 3 shows the relative air-fuel ratio during training. This is the air-fuel ratio divided by the stoichiometric air-fuel ratio. The controller has initially poor performance with approximately 30% error, but quickly improves. After 60 minutes, the error improves to approximately 2%, before eventually reaching a steady state error of approximately 0.03%, as shown in Figure 4.

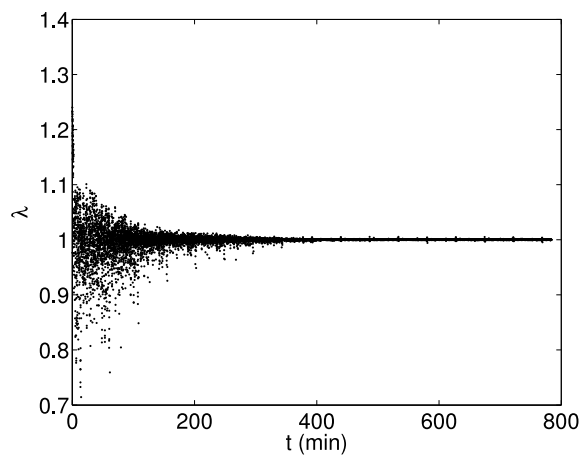


Fig. 3: The relative air-fuel ratio, λ , defined as the measured air-fuel ratio divided by the stoichiometric air fuel ratio, for a simulated V8 engine was controlled using the network inversion scheme introduced in this paper.

With regard to computation speed, in a separate experiment this algorithm was implemented on a Motorola MPC555 microcontroller. The calculation of the stoichiometric pulse width took 0.335 ms, well within the requirements for realtime operation. Since each inversion calculation takes approximately the same time as one forward network calculation, each iteration of a numerical solution can be expected to take at least the same time. Therefore, an iterative solver would only be allowed seven iterations to find its solution in the 2.5 ms time allotted between injections, which would probably not provide the required accuracy for fuel-air control. This is especially true as gradient-based iterative solvers can not be used since they have problems finding the transition point of functions that have a sharp transition with very small slopes in areas away from the transition point. For example, the bisection method would have an unacceptably large error of 0.78% after 7 iterations [7].

