CHAPTER 4 OBJECTIVE FUNCTIONS

4.1 Introduction

In this chapter, the objective functions of SOURCAO are examined. Because traffic signal algorithm is based on traffic surveillance (except fixed time control in baseline studies), how to process the surveillance detector data to serve two objectives is addressed in Section 4.2 and Section 4.3. The first objective of the safety consideration is surveyed in Section 4.2. The next four sections (4.3- 4.7) are formulated for the fulfillment of the second objective: traffic signal optimization near grade crossings. Section 4.3 introduces the traffic surveillance data for the objective function. Section 4.4 formulates the objective function assuming that all the variables are available. Section 4.5 presents how to forecast the objective functions with historical data of some variables by neural network. Section 4.6 establishes the neural network weight training scheme. Finally, neural network implementation is highlighted in Section 4.7.

4.2 Safety Surveillance

In SOURCAO, the traffic signal control agent promotes the safety of HRGCs. The intelligent agent within SOURCAO perceives the environment through the surveillance detectors. The selection of the next phase is based on queues near grade crossings and overall traffic assessment. The different detector positions in the links configure the different surveillance functions. SOURCAO fosters four types of surveillance as shown in Table 4-1.

As shown in Figure 4-1, two types of surveillance are needed for the two different lane groups (the left turning bay lane and the through right lane) to foster the left turn phase and through/right turn phase. For the left lane, turning movement surveillance is configured. While for the through/right turn lane groups, queue detection is set up. A special surveillance, full detection, which takes advantage of data from queue detection, provides important information for the inference engine. In addition, a presence detector

is added to detect whether vehicles are stopped at a grade crossing before the arrival of a train.

Table 4-1 Types of Surveillance in SOURCAO

Type	Surveillance	Output	Detection Description
1	Queue	0=No vehicle 1= At least one vehicle	The number of vehicles queued in the link
2	Full	-1=Full 0=Not full	The link is full of vehicles or not
3	Turning movement	0=No turning vehicle detected 1= At least one vehicle at turning bay	The number of vehicles queued on the turning bay
4	Grade Crossing Stall Vehicle	0= No stall vehicle detected 1= At lease one stalled vehicle detected	Any stalled vehicle on grade crossing

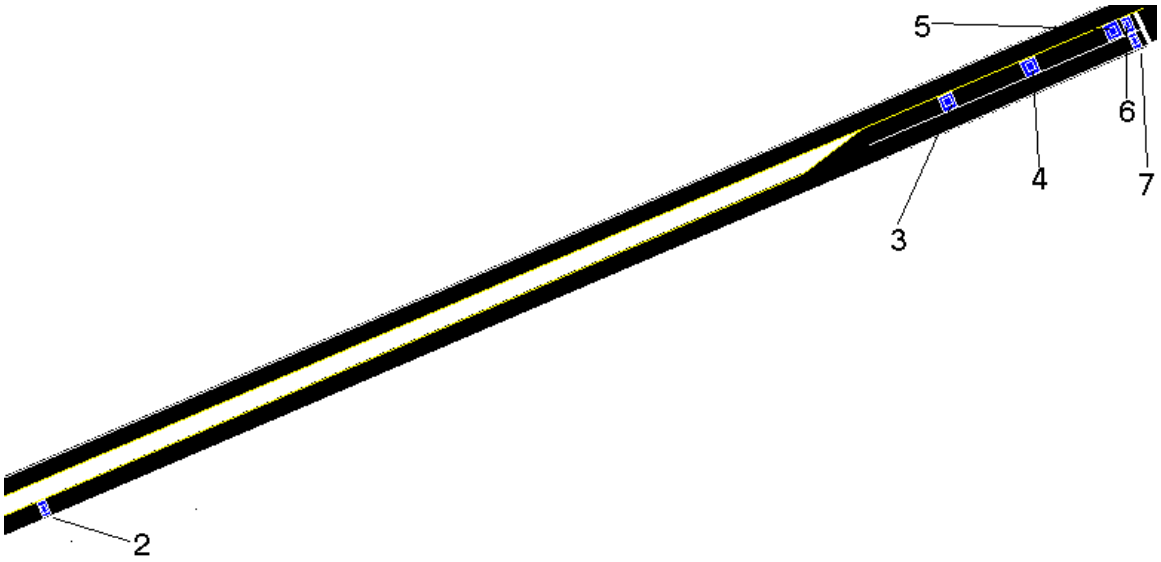

Figure 4-1 Layout of Link Surveillance Detectors

4.2.1 Turning Vehicle Detection

As shown in Figure 4-1, a series of the present detectors (numbered 3, 4 and 5) sense the turning vehicles. The distance between the detectors is three times the average vehicle length. If a vehicle is sensed by any detector, the surveillance function reports '1'; otherwise, '0' is reported.

4.2.2 Vehicle Queue Length Detection

As shown in Figure 4-1, in any time, the number of vehicles in the link is measured by:

$$D(t)=D(t-1)+ D_e(t)-D_x(t)-D_T(t) \tag{4-1}$$

Where:

D(t) is the number of the current vehicles in the link. If *D(t)* is more than one, the surveillance detector outputs 1;

D(t-1) is the number of the vehicles in the link in the previous time step;

D_e is the number of the entrance detector counts (Detector 2 in Figure 4-1);

D_x is the number of the exit detector counts (sum of Detectors 6 and 7 in Figure 4-1);

D_T is the number of the turning vehicle counts, as measured in Section 4.2.1.

4.2.3 Full Detection

The purpose of this detection is to check if a link is full of vehicles. As shown in Figure 4-1, the maximum number of through vehicles (N_{max}) can be estimated by dividing the length between Detector 2 and 6 (L_L) by the average vehicle length (L_v) and the number of lanes (N_L). In addition, two adjacent detectors are placed at the appropriate position near the entrance of the link to detect the speed of vehicles. If the speed is below a threshold value and the number of the detected vehicles in (4-1) divided by the number of vehicles in (4-2) is above a threshold value (0.8), the link is considered "full".

$$N_{\max} = \frac{L_L}{N_L \bullet L_V} \tag{4-2}$$

4.2.4 Grade Crossing Detector

If the presence detector on a grade crossing is continuously "on," the grade crossing is considered "occupied."

4.3 Traffic Delay Surveillance

In SOURCAO, the delays are modeled according to lane groups in a link. The vehicles on different lane groups respond to different phases. If more than one lane groups serve the same link, turning bay delays are calculated first, and the main approach delay is the difference of the link delays and the turning bay delays. This section presents the turning bay surveillance in Section 4.3.1, and the link delay surveillance in Section 4.3.2. The link delay model is discussed in Section 4.4.

4.3.1 The Turning Bay Delay Surveillance

As shown in Figure 4-1, the turning bay delay surveillance is configured the same as turning movement surveillance in Section 4.2. If a vehicle is presented in the first detector, the number of vehicles in the queue is counted as three. Likewise, if two detectors are actuated, the number of vehicles in the queue is counted as six, etc. It is reasonable to assume that the turning vehicles stop close to the stop bar. In this way, the error for counting the turning vehicles might be limited to the number of presence detectors. After the number of vehicles in the queue is collected, the delays of the turning vehicles are accumulated by multiplying the number of vehicles by the polling interval. In SOURCAO, the detector surveillance data are polled in every simulation step (1 second/step).

4.3.2 Link Delay Surveillance

According to the delay model in Section 4.4, the surveillance data needed are the number of the vehicles and the time stamps recorded at the entrance and the exit detectors. For example, the link delay surveillance in Figure 4-1 is implemented through Detectors 6, 7 and 2.

Since SOURCAO is evaluated through CORSIM, CORSIM detector data are utilized. Detector 2 is the entrance detector and Detectors 6 and 7 are the exit detectors. All link surveillance detectors are coded as the passage detectors in CORSIM. As shown in Figure 3-1, the detector counts are accessed through the DTCON variable in FORTRAN.

The input from the surveillance detectors is the only source of data needed for the delay model in Section 4.4. The surveillance detectors are the most common and economic available data sources. Therefore, it is possible to implement the system in the

real traffic world. In addition, the number of vehicles *D(t)* in the link at time *t* can be calculated from (4-1). *D(t)* is used as Multilayer Perceptron (MLP) input.

4.4 Delay Model Formulation

Based on the surveillance data available from Section 4.3, the objective function is formulated in this section. Section 3.2.3 establishes the relationship between Measurements of Effectiveness (MOEs) and the delay objective function. The assumptions and symbols of delay function are presented in Section 4.4.1. The rest of the section is the details of the proposed mathematical formula to calibrate the delays.

4.4.1 Assumptions

The following assumptions and symbols are made to model the proposed traffic delay calibration:

- The delay is defined in Section 3.2.3, and the acceleration delays are excluded;
- The projected time interval is (t_a, t_b), during which the link delays are accumulated. In SOURCAO, t_a is always the current simulation time and t_b is two minutes (120 seconds) after that;
- The time stamps for a vehicle (i) entering (t_{i1}) and exiting (t_{i2}) the link can be recorded and forwarded to the system to process the delays;
- When the vehicles enter a link, they obey first-in/first-out queue definition. This rule is applied to approximately match the exit time and entry time;
- The vehicle length is L_L, the link length is L, and the free flow speed is V_f;
- m_1+ m_3 is the number of the vehicles that are in the queue before the start of the interval (t_a, t_b);
    - m_1 is the number of the vehicles exiting during the interval;
    - m_3 is the number of the vehicles that could not exit during the interval ;

- $m_1 + m_{21} + m_{22}$ is the number of the vehicles exiting from the link during the interval;
  - m_1 is the number of the vehicles that are in the queue before the start of the interval;
  - m_{21} is the number of the vehicles in the link but not in the queue before the start of the interval;
  - m_{22} is the number of the vehicles entering the link during the interval;
- $m_{22} + m_{42} + m_{52}$ is the number of the vehicles entering the link during the interval;
  - m_{22} is the number of the vehicles exiting the link during the interval;
  - m_{42} is the number of the vehicles entering the queue before the end of the interval, but could not exit;
  - m_{52} is the number of the vehicles entering the link but could not join the queue at the end of the interval;
- $m_{21} + m_{41} + m_{51}$ is the number of the vehicles in the link but not in the queue before the start of the interval;
  - m_{21} is the number of the vehicles exiting the link during the interval;
  - m_{41} is the number of the vehicles that could not exit the link but moving to the queue before t_b;
  - m_{51} is the number of the vehicles that enter the link after t_a, could not reach the queue before t_b, and could not exit the link;
- $m_3 + m_{41} + m_{42}$ is the number of the vehicles that reach the queue but could not exit the link during the interval;
  - m_3 is the number of the vehicles in the queue at the time t_a;
  - m_{41} is the number of the vehicles in the link but not in the queue at the time t_a;
  - m_{42} is the number of the vehicles entering the link during the interval;
- $m_2 = m_{21} + m_{22}$ is the number of the vehicles exiting the link during the interval. Those vehicles are not in the queue at the start of interval;

- m_{21} is the number of the vehicles in the link but not in the queue before the start of the interval;
- m_{22} is the number of the vehicles entering the link during the interval;

- $m_4 = m_{41}+m_{42}$ is the number of the vehicles that reach the queue but could not exit the link during the interval. Those vehicles are not in the queue at the start of interval;
  - m_{41} is the number of the vehicles in the link but not in the queue at the time t_a;
  - m_{42} is the number of the vehicles entering the link during the interval;
- $m_{5=} m_{51}+ m_{52}$ is the number of the vehicles not even reached the queue during the interval (t_a, t_b); therefore they could not exit the link;
  - m_{51} is the number of the vehicles not in the queue but in the link at the time t_a;
  - m_{52} is the number of the vehicles that enter the link during the interval;

4.4.2 Vehicles Left in the Link in Past (before t_a start)

There are five cases where the vehicles enter a link before time t_a. In the proposed research, t_b is always 120 seconds after t_a.

- Being in the link and in the queue before t_a and exit the link before t_b *(m_1);*
- Being the link and in the queue before t_a but could not exit the link before t_b *(m_3);*
- Being in the link, not in the queue before t_a but exit the link before t_b *(m_{21});*
- Being in the link, not in the queue before t_a but join the queue and could not exit the link before t_b*(m_{41});*
- Being in the link, not in the queue before t_a but could not join the queue before t_b *(m_{51}).*

Case one

There are (m_1+m_3) vehicles in the queue at the time t_a. m_1 vehicles in the queue exit the link before t_b and m_3 vehicles could not exit the link. Delays for corresponding vehicles can be expressed as (4-3), Where m_1 should be found according to (4-4).

$$d = \sum_{m_1} \left(t_{i2} - t_a - \frac{iL_L}{V_f} \right) = \sum_{m_1} t_{i2} - m_1 t_a - \frac{m_1^2 L_L}{2V_f} \tag{4-3}$$

$$m_1 \in t_{i1} \le t_a - \frac{L - iL_l}{v_f} \quad and \quad t_{i2} < t_b \tag{4-4}$$

Case two

In this case, (m_1 + m_3) vehicles are in the queue before t_a, of which m_1 vehicles are processed in Case 1. Unfortunately, some vehicles could not exit (m_3). The distance traveled by m_3 vehicles is m_1*L_L, Where m_3 should be found according to (4-6).

$$m_3 \in t_{i1} \le t_a - \frac{L - iL_l}{v_f} \quad and\ t_{i2} > t_b \tag{4-5}$$

$$d = \sum_{m_3} \left(t_b - t_a - \frac{m_1 L_L}{V_f} \right) = m_3 (t_b - t_a) - \frac{m_1 m_3 L_L}{V_f} \tag{4-6}$$

Case three

For m_{21} vehicles entering the link before t_a and having not reached the queue, the distance to the stop bar at time t_a is $L-(t_a-t_{i1})*V_f$, Where m_{21} should be found according to (4-8). Since those vehicles exit at time $t_{i2,}$ the delay d can be approximated in (4-7).

$$
\begin{aligned}
d &= \sum_{m_{21}} \left(t_{i2} - t_a - \frac{L-(t_a - t_{i1})*V_f}{V_f} \right) \\
&= \sum_{m_{21}} \left(t_{i2} - t_a - \frac{L}{V_f} + (t_a - t_{i1}) \right) \\
&= \sum_{m_{21}} (t_{i2} - t_{i1}) - \frac{m_{21} L}{V_f}
\end{aligned}
\tag{4-7}
$$

$$
m_{21} \in t_{i1} \geq t_a - \frac{L - iL_l}{v_f} \quad and \quad t_{i2} \leq t_b \tag{4-8}
$$

Case four:

Like Case 3, for the vehicles that enter the link before t_a but have not reached the queue, the distance to the stop bar at time t_a is $L-(t_a-t_{i1})*V_f$. However, m_{41} vehicles could not exit although they joined the queue before the stop bar.

$$
\begin{aligned}
d &= \sum_{m_{41}} \left(t_b - t_a - \frac{L-(t_a - t_{i1})*V_f - (i+m_3)L_L}{V_f} \right) \\
&= \sum_{m_{41}} \left(t_b - t_a + (t_a - t_{i1}) - \frac{L-(m_3 L_L + iL_L)}{V_f} \right) \\
&= \sum_{m_{41}} (t_b - t_{i1}) - m_{41} \frac{L - m_3 L_L}{V_f} + \frac{m_{41}^2 L_L}{2V_f}
\end{aligned}
\tag{4-9}
$$

Case five:

If the vehicles could not reach the queue (m_{51}) at the end of t_b, the delay needs not be accumulated.

4.4.3 Vehicles Entering a Link During (t_a, t_b)

There are three cases for the vehicles entering a link before time t_a:

- Entered and exited the link in the interval of (t_a, t_b);
- Entered and joined the queue in the interval, but could not exit the link;
- Entered and could not join the queue.

Case six

If m_{22} vehicles exit the link during (t_a, t_b), the delay can be approximated as:

$$d = \sum_{m_{22}} \left(t_{i2} - t_{i1} - \frac{L_L}{V_f} \right)$$
$$= \sum_{m_{22}} t_{i2} - \sum_{m_{22}} t_{i1} - \frac{m_{22} L}{V_f}$$

(4-10)

where:

$t_{i1} >= t_a$ *and* $t_{i2} <= t_b$

$t_B > t_{m_1 2}$

Case seven

In this case, there are m_{42} vehicles entering the queue. (4-11) could be applied for the delay calibration.

$$d = \sum_{m_{42}} \left(t_b - t_{i1} - \frac{L - (m_3 + m_{41} + i)L_L}{V_f} \right)$$
$$= m_{42} t_b - \sum_{m_{42}} t_{i1} - m_{42} \frac{L - (m_3 + m_{41} + m_{42}/2)L_L}{V_f} \tag{4-11}$$

Where:

$$m_{42} \in t_{i1} \le t_b - \frac{L - iL_l}{v_f} \text{ and } t_{i2} > t_b$$

$$t_B > t_{m_1 + m_{22}}$$

Case eight

In this case, there are m_{52} vehicles entering the link. Since they have not reached the queue, the delay is zero.

4.5 Neural Network Delay Forecasting

One recent development in artificial intelligence is the neural network, in which the human brain is mimicked in some ways. The neural network is defined as a massively parallel-distributed processor that has a natural propensity for storing experimental knowledge and making it available for use. It resembles the brain in two respects:

- Through a learning process, the network acquires knowledge;
- Inter-neuron connection strengths, known as “the synaptic weights,” are used to store the knowledge.

The neural network is extremely useful in pattern recognition, in which case the stall vehicles could be detected through the neural network architecture. It was also used in traffic forecasting and traffic signal control (Section 2.5.1).

4.5.1 The Objective Function by Neural Network Forecasting

In Section 4.4, the objective function is defined as the summation of the link delays in the next two minutes. In the same section, the delays d from time t_a to t_b can be approximated from the link surveillance input.

$$d = f(S) \tag{4-12}$$

Where

$$S = (s_1, s_2, s_3 ..., s_n)^T \tag{4-13}$$

s_i is the collection of number of vehicles entering and exiting the link and the time stamps;

N is the time step (1 second in SOURCAO) during the projected time intervals t_a and t_b.

The relationship *f(S)* is established in Section 4.4 based on available surveillance data. Unfortunately, the next two-minute surveillance detector data are not available until two minutes later. This chapter is formulated to resolve such an issue -- forecasting the delays in the next two minutes by applying an artificial neural network.

To simplify the question, it is assumed that the future vehicles entering and exiting a link are the functions of the historical surveillance data *S*, the historical link delays *D* and the current and future intersection traffic control device status. In

SOURCAO, the signal phase timing P (phase ID and phase length) and grade crossing information G (close time and open time) represent the traffic control variables.

For S_k, the past two-minute surveillance data are utilized. Calculated at the current time, at one-minute and at two-minutes before the current time, the number of vehicles contributes to the delay forecast. Those data are represented by S_0, S_{-1}, and S_{-2} by links. The delay D_k (by phase and link) is represented by the delay in the last one-minute D_0, and the delay in one-minute before the last minute D_{-1}. Three phases ahead (P_0, P_1 and P_2) are taken as the variables. Each phase is represented by the phase length. The phase is organized by intersection. Current grade crossing closure information G_0 (the open time and the closed time) and the next time closure information G_1 compose G.

Since P is optimized in SOURCAO, it is distinguished as the control variable X (4-15). Others are named as the state variable U (4-14). In other words, the delay function can be represented as in (4-16).

$$U=(S_0, S_{-1}, S_{-2}, G_0, G_1, D_0, D_{-1}) \tag{4-14}$$

$$X=P=(P_0, P_1, P_2) \tag{4-15}$$

$$d=f(X;U) \tag{4-16}$$

The above delay function (4-16) is approximated by an MLP neural network. Section 4.5.2 describes the mathematical model of MLP and its derivation. In Section 4.6, an approach named “back-propagation” to train the MLP weights is discussed. Following that section, an objective-oriented implementation is introduced in Section 4.7. After the neural network training, the delay function can be expressed in (4-17). A detailed formation will be presented from (4-20) to (4-27):

$$d=f(X;U;W)$$

(4-17)

Where:

W is the weights obtained from method described in this section.

4.5.2 Multilayer Perceptrons

The MLP is widely used to approximate a function. The universal Approximation Theorem (Haykin 1999) is stated as:

Let j (.) be a non-constant, bounded, and monotone-increasing function. Let I_{m0} *denote the* m_0*-dimensional unit hypercute* $[0,1]^{m0}$*. The space of continuous functions on* I_{m0} *is denoted by* $C(I_{m0})$*. Then, given any function fÉ* $C(I_{m0})$ *and* $e>0$*, there exist an integer M and sets of real constants* a_I, b_I *and* w_{ij}*, where* $I=1,\ldots,m_0$ *such that we may define*

$$F(x_1,...,x_{m_0}) = \sum_{1}^{m_1} a_i j\left(\sum_{1}^{m_0} w_{ij}x_j + b_i\right)$$

(4-18)

as an approximate realization of f(.); that is

,

$$\left|F(x_1,...,x_{m_0}) - f(x_1,...,x_{m_0})\right| < e$$

(4-19)

for all $x_1, x_2,\ldots, x_{m0}$ *that lie in the input space.*

By applying the above theorem, neural network approximation can be expressed as:

$$d = f(j(X;U,W))$$

(4-20)

Where

X: Control variable (phase length in SOURCAO) (4-15);

U: State variables (current and historical detector and delay input) (4-14);

$W_{:}$ Weights;

f: Utility function. In the proposed research, the most commonly used function (4-21) is adapted.

$$f(x) = \frac{1}{1+e^{-x}}$$

(4-21)

For the single layer perceptrons, usually j could be expressed in (4-22).

$$j(X;U,W) = \sum \left(UW_u + XW_x\right)$$

(4-22)

For the two-layer perceptrons, the middle-layer perceptrons $V=\{v_i\}$ are added, as shown in (4-23), (4-24) and (4-27):

$$\begin{aligned} v_i &= f\left(j_i(X;U,W)\right) \\ &= f\left(\sum \left(UW_u^i + XW_x^i\right)\right) \end{aligned}$$

(4-23)

$$V = \begin{pmatrix} v_1 \\ \\ v_n \end{pmatrix}^T = \begin{pmatrix} f\left(\sum\left(UW_u^1 + XW_x^1\right)\right) \\ f\left(\sum\left(UW_u^2 + XW_x^2\right)\right) \\ \cdots \\ f\left(\sum\left(UW_u^n + XW_x^n\right)\right) \end{pmatrix}^T \tag{4-24}$$

Where

$$X = \left(x_1, x_2 \ldots x_m\right)^T \tag{4-25}$$

$$\left(W_x^1, W_x^2, \ldots W_x^n\right) = \begin{pmatrix} w_{11} & w_{21} & \cdots & w_{n1} \\ w_{12} & w_{22} & & w_{n2} \\ \cdots & \cdots & \cdots & \cdots \\ w_{1m} & w_{2m} & \cdots & w_{nm} \end{pmatrix} \tag{4-26}$$

$$d = f\left(j\left(V, W_v\right)\right) = f\left(\sum VW_v\right) \tag{4-27}$$

$$W_v = \left(w_1^v, w_2^v, w_3^v, \ldots, w_m^v\right)^T \tag{4-28}$$

$$V = \left(v_1, v_2, \ldots, v_m\right) \tag{4-29}$$

n is number of input nodes and *m* is the number of middle nodes.

In SOURCAO, after the training *W* is attained, the neural network function (4-27) is used as the objective function. In other words, the function *d=f(X;U,W)* is minimized. When searching the optimal solution of *X*, most of the algorithms require the first order

of the derivation. The rest of this section presents the derivation of the function d with respect to X.

First of all, the utility function (4-21) could be derived as:

$$\begin{aligned} f'(t) &= \left(\frac{1}{1+e^{-t}} \right)' \\ &= \frac{e^{-t}}{(1+e^{-t})^2} \\ &= f(t) \bullet (1 - f(t)) \end{aligned} \tag{4-30}$$

Next, it is noticed that the control variable X exists in the input perceptrons only. The perceptrons in other layers don't connect to input perceptrons directly. The chain rule of derivation has to be applied repeatedly. For the output layer, deriving (4-27):

$$\begin{aligned} \frac{\partial d}{\partial X} &= \frac{\partial f(j(V, W_v))}{\partial V} \frac{\partial V}{\partial X} \\ &= \sum W_v \frac{\partial f}{\partial j} \frac{\partial V}{\partial X} \end{aligned} \tag{4-31}$$

For the input layer, deriving (4-24):

$$\frac{\partial V}{\partial X} = \frac{\partial \begin{pmatrix} v_1 \\ v_2 \\ \dots \\ v_n \end{pmatrix}^T}{\partial X}$$

$$= \left(W_x^1, W_x^2, \dots W_x^n\right) \begin{pmatrix} \frac{\partial f^n}{\partial j} \\ \frac{\partial f^n}{\partial j} \\ \dots \\ \frac{\partial f^n}{\partial j} \end{pmatrix}^T$$

$$= W_x \begin{pmatrix} v^1 \bullet (1-v^1) \dots & \dots\ v^1 \bullet (1-v^1) \\ v^2 \bullet (1-v^2) \dots & \dots v^2 \bullet (1-v^2) \\ \dots & \dots \\ v^n \bullet (1-v^n) \dots & \dots v^n \bullet (1-v^n) \end{pmatrix}^T$$

$$= W_x \bullet V'$$

(4-32)

Combining (4-31) and (4-32), the final derivative (4-33) can be expressed as:

$$\frac{\partial d}{\partial X} = d(1-d)\sum w_v \frac{\partial V}{\partial X}$$
$$= d(1-d) W_v W_x V'$$

(4-33)

If the number of layers is more than two, the derivation becomes tricky, as discussed as follows.

The assumptions are made that there are N layers of the perceptrons. $v_i(0)$ ($i=1,\dots,m_0$) denotes the input perceptrons. $v_i(1)$ ($i=1,\dots,m_1$) denotes the perceptrons immediately next to the input perceptrons etc. $v_i(n)$ ($i=1,\dots,m_n$) denotes the output perceptrons. Actually, in the proposed research case, there is only one output perceptron

(m_n=1), the network delay. In general, on the *kth* layer, $w(v_i(k)->v_j(k-1))$ represents the weight connecting perceptron $v_i(k)$ to perceptron $v_j(k-1)$.

The deriving process starts from the input layer to the output layer.

In the first layer (input layer), for any ***j,*** the perceptron $v_j(1)$ connects to all input perceptron $v_i(0)$s*:*

$$V_j(1) = f\left(\sum_i W\left(V_j(1) -> V_i(0)\right) \bullet V_i(0)\right) \tag{4-34}$$

Deriving the above equation:

$$\frac{\partial v_j(1)}{\partial v_i(0)} = w\left(v_i(1) -> v_j(0)\right) \bullet \left(1 - v_j(1)\right) \bullet v_j(1) \tag{4-35}$$

Where

$w(v_i\ (1)\ ->v_j\ (0))$ is the weight from the perceptron $v_i\ (1)$ to $v_j\ (0)$ on the first layer;

i=1,2,…m_0 (output layer);

j=1,2,…m_1 (first layer).

In the second layer, for any j, the perceptron $v_j(2)$ connects to all perceptron $v_i(1)s:$

$$v_j(2) = f\left(\sum_i w\left(v_j(2) -> v_i(1)\right) \bullet v_i(1)\right) \tag{4-36}$$

Deriving the above equation:

$$\frac{\partial v_j(2)}{\partial v_i(1)} = w\left(v_j(2) -> v_i(1)\right) \bullet (1 - v_j(2)) \bullet v_j(2) \tag{4-37}$$

Now the chain rule is applied:

$$\begin{aligned}
&\frac{\partial v_j(2)}{\partial v_k(0)} = \sum \frac{\partial v_j(2)}{\partial v_i(1)} \frac{\partial v_i(1)}{\partial v_k(0)} \\
&= \sum_i w\left(v_j(2) -> v_i(1)\right) \bullet \left(1 - v_j(2)\right) \bullet v_j(2) \bullet w\left(v_i(1) -> v_k(0)\right) \bullet \left(1 - v_i(1)\right) \bullet v_i(1) \\
&= \left(1 - v_j(2)\right) \bullet v_j(2) \bullet \sum_i w\left(v_j(2) -> v_i(1)\right) \bullet w_i\left(v_i(1) -> v_k(0)\right) \bullet \left(1 - v_i(1)\right) \bullet v_i(1)
\end{aligned} \tag{4-38}$$

The same process is applied to the third layer:

$$v_j(3) = f\left(\sum_i w\left(v_j(3) -> v_i(2)\right) \bullet v_i(2) \right) \tag{4-39}$$

$$\frac{\partial v_j(3)}{\partial v_i(2)} = w\left(v_j(3) -> v_i(2)\right) \bullet \left(1 - v_j(3)\right) \bullet v_j(3) \tag{4-40}$$

$$\frac{\partial v_j(3)}{\partial v_k(0)} = \sum \frac{\partial v_j(3)}{\partial v_i(2)} \frac{\partial v_i(2)}{\partial v_k(0)} \tag{4-41}$$

Substituting (4-38) and (4-40) into (4-41), we can solve (4-41).

In general, the above process is applied to solve the derivation of the perceptron function $v_j(l)$ in any layer l with respect to $v_k(0)$ if $\partial v_i(l-1)/\partial v_k(0)$ is obtained beforehand, as shown below.

In the layer *l*, for any j, a perceptron $v_j(l)$ connects to all perceptron $v_i(l\text{-}1)$s with weight $w(v_j(l) \rightarrow v_i(l\text{-}1))$ on the *(l-1)th* layer*:*

$$v_j(l) = f\left(\sum_i w\left(v_j(l) -> v_i(l-1)\right) \bullet v_i(l-1)\right) \tag{4-42}$$

Deriving the above equation:

$$\frac{\partial v_j(l)}{\partial v_i(l-1)} = w\left(v_j(l) -> v_i(l-1)\right) \bullet \left(1 - v_j(l)\right) \bullet v_j(l) \tag{4-43}$$

Applying the chain rule to (4-42):

$$\frac{\partial v_j(l)}{\partial v_k(0)} = \sum \frac{\partial v_j(l)}{\partial v_i(l-1)} \frac{\partial v_i(l-1)}{\partial v_k(0)} \tag{4-44}$$

In (4-44), since $\partial v_i(l\text{-}1)/\partial v_k(0)$ is calculated in the previous step, $\partial v_i(l)/\partial v_k(0)$ can be solved.

For the output perceptron, the delay can be directly expressed as follows:

$$d = v(n) = f\left(\sum_i w\left(v(n) -> v_i(n-1)\right) \bullet v_i(n-1)\right) \tag{4-45}$$

Finally, *d'* can be solved since $\partial v_i(l\text{-}1)/\partial v_k(0)$ is already gotten in the (n-1)th layer derivation:

$$\frac{\partial d}{\partial v_k(0)} = \frac{\partial v(n)}{\partial v_k(0)} = \sum \frac{\partial v(n)}{\partial v(n-1)} \frac{\partial v_i(n-1)}{\partial v_k(0)} \tag{4-46}$$

To verify the correctness of the above process, for an MLP of two layers with the number of perceptrons in (input, middle, output) layers of (3,2,1), we can use the above notation to confirm (4-33):

Applying (4-35), we generate (4-47), (4-48) and (4-49):

$$\frac{\partial v_1(1)}{\partial v_1(0)} = w\left(v_1(1) -> v_1(0)\right) \bullet \left(1 - v_1(1)\right) \bullet v_1(1) \tag{4-47}$$

$$\frac{\partial v_1(1)}{\partial v_2(0)} = w\left(v_1(1) -> v_2(0)\right) \bullet (1 - v_1(1)) \bullet v_1(1) \tag{4-48}$$

$$\frac{\partial v_1(1)}{\partial v_3(0)} = w\left(v_1(1) -> v_3(0)\right) \bullet (1 - v_1(1)) \bullet v_1(1) \tag{4-49}$$

Applying (4-37), we obtain (4-50), (4-51) and (4-52):

$$\frac{\partial v_2(1)}{\partial v_1(0)} = w\left(v_2(1) -> v(0)\right) \bullet (1 - v_2(1)) \bullet v_2(1) \tag{4-50}$$

$$\frac{\partial v_2(1)}{\partial v_2(0)} = w\left(v_2(1) -> v_2(0)\right) \bullet (1 - v_2(1)) \bullet v_2(1) \tag{4-51}$$

$$\frac{\partial v_2(1)}{\partial v_3(0)} = w(\ )(v_2(1) -> v_3(0)) \bullet (1 - v_2(1)) \bullet v_2(1) \tag{4-52}$$

Applying (4-46), the following equations are generated:

$$\frac{\partial d}{\partial v_1(0)} = d(1-d)\left(w(v(2)->v_1(1)) \bullet \frac{\partial v_1(1)}{\partial v_1(0)} + w(v(2)->v_2(1)) \bullet \frac{\partial v_2(1)}{\partial v_1(0)} \right)$$
$$= \left(d(1-d)\begin{pmatrix} w(v(2)->v_1(1)) \bullet w(v_1(1)->v_1(0)) \bullet v_1(1) \bullet (1-v_1(1)) + \\ w(v(2)->v_2(1)) \bullet w(v_2(1)->v_1(0)) \bullet v_2(1) \bullet (1-v_2(1)) \end{pmatrix} \right)$$

(4-53)

$$\frac{\partial d}{\partial v_2(0)} = d(1-d)\left(w(v(2)->v_1(1)) \bullet \frac{\partial v(1)_1}{\partial v(0)_2} + w(v(2)->v_2(1)) \bullet \frac{\partial v(1)_2}{\partial v(0)_2} \right)$$
$$= \left(d(1-d)\begin{pmatrix} w(v(2)->v_1(1)) \bullet w(v_1(1)->v_2(0)) \bullet v_1(1) \bullet (1-v_1(1)) + \\ w(v(2)->v_2(1)) \bullet w(v_2(1)->v_2(0)) \bullet v_2(1) \bullet (1-v_2(1)) \end{pmatrix} \right)$$

(4-54)

$$\frac{\partial d}{\partial v_3(0)} = d(1-d)\left(w(v(2)->v_1(1)) \bullet \frac{\partial v_1(1)}{\partial v_3(0)} + w(v(2)->v_2(1)) \bullet \frac{\partial v_2(1)}{\partial v_3(0)} \right)$$
$$= \left(d(1-d)\begin{pmatrix} w(v(2)->v_1(1)) \bullet w(v_1(1)->v_3(0)) \bullet v_1(1) \bullet (1-v_1(1)) + \\ w(v(2)->v_2(1)) \bullet w(v_2(1)->v_3(0)) \bullet v_2(1) \bullet (1-v_2(1)) \end{pmatrix} \right)$$

(4-55)

If we use the notation expressions of *w* above, (4-33) can be replaced by (4-56), (4-57) and (4-58). (4-53), (4-54) and (4-55 can be unified as (4-33).

$$W_x = \begin{pmatrix} w(v_1(1)->v_1(0)) & w(v_1(1)->v_2(0)) & w(v_1(1)->v_3(0)) \\ \\ w(v_2(1)->v_1(0)) & w(v_2(1)->v_2(0)) & w(v_2(1)->v_3(0)) \end{pmatrix}^T$$

(4-56)

$$W_v = \begin{pmatrix} w(v(2)->v_1(1)) \\ w(v(2)->v_2(1)) \end{pmatrix}$$

(4-57)

$$V' = \begin{pmatrix} v_1(1) \bullet (1 - v_1(1)) & v_1(1) \bullet (1 - v_1(1)) \\ v_2(1) \bullet (1 - v_2(1)) & v_2(1) \bullet (1 - v_2(1)) \end{pmatrix}$$

(4-58)

4.6 Back-Propagation Training

Back propagation is a highly popular weight update algorithm applied to train MLP (Haykin 1999). The weights are adjusted in a way that the error function (4-59) can be minimized.

$$E(w) = \frac{1}{2} \sum_h (d_h - y_h)^2$$

(4-59)

Where:

h is the index to the output data; in SOURCAO, $h=1$ since only entire network delay is approximated. The S and h can be removed here and $y=v(N);$

y_h is the hth observed delay;

d_h is the calculated delay provided by (4-20), since $h=1$,

$$d=u(n) \tag{4-59}$$

This section describes how the weights can be updated by using back-propagation training. For the output layer (layer N), as shown in (4-45), the derivation of function $E(w)$ with respect to w can be found as follows:

$$\begin{aligned} \frac{\partial E}{\partial w(v_i(n) -> v_j(n-1))} &= (d - y) \bullet \frac{\partial d}{\partial y} \\ &= (d - y) \bullet f'(v_i(n)) \bullet v_i(n) \end{aligned}$$

(4-60)

Therefore, the updating rule of *w* in the output layer can be expressed in (4-61) and weights can be achieved in (4-66):

$$\begin{aligned}&\Delta w\left(v_i(n)->v_j(n-1)\right)\\&=r\frac{\partial E}{w\left(v_i(n)->v_j(n-1)\right)}\\&=r\bullet(d-y)\bullet f'(v_i(n))\bullet v_i(n)\end{aligned}$$

(4-61)

Where ρ is the updating learning rate.

For the layer immediate to the output layer (Layer *n-1*), the derivation of the function *E(w)* with respect to *w* can be found in:

$$\frac{\partial E}{\partial w\left(v_i(n-1)->v_j(n-2)\right)}=\frac{\partial E}{\partial v_i(n-1)}f'\left(v_i(n-1)\right)\bullet v_i(n-1)$$

(4-62)

$$\begin{aligned}\frac{\partial E}{\partial v_i(n-1)}&=(d-y)\frac{\partial d}{\partial y}\frac{\partial y}{\partial v_i(n-1)}\\&=(d-y)\bullet f'(v(n))\bullet w\left(v(n)->v_i(n-1)\right)\end{aligned}$$

(4-63)

Combining (4-62) and (4-63), the updating rules can be expressed in (4-64) and weights can be achieved in (4-65):

$$\begin{aligned}&\Delta w\left(v_i(n-1)->v_j(n-2)\right)\\&=r\bullet(d-y)\bullet f'(v(n))\bullet w\left(v(n)->v_i(n-1)\right)\bullet f'\left(v_i(n-1)\right)\bullet v_{i-1}(n-1)\\&=r\bullet\left(d(n-1)-y(n-1)\right)\bullet f'\left(v_i(n-1)\right)\bullet v_{i-1}(n-1)\end{aligned}$$

(4-64)

Where

$$d(n-1) - y(n-1) = \left(d(n) - y(n)\right) \bullet f'(v(n)) \bullet w\left(v(n) -> v_i(n-1)\right)$$
$$d(n) = d$$
$$y(n) = y$$

(4-65)

and

$$w\left(v_i(n) -> v_j(n-1)\right) = w\left(v_i(n) -> v_j(n-1)\right) + \Delta w\left(v_i(n) -> v_j(n-1)\right)$$

(4-66)

From (4-66), the error in a hidden layer is back-propagated from the hidden layer to the output layer. For the rest of the hidden layers, the same approach is used to back-propagate the error to the output layer.

In (4-59), only one data point is considered put into training. Usually, one of the data points is randomly withdrawn to receive the training. However, we can average weight update *Dw* in the training process by minimize the summation of the errors for all data points in (4-67). The approach is usually called "batch training" or "epoch training."

$$E(w) = \frac{1}{2}\sum_m (d_m - y_m)^2$$

(4-67)

Where

M is the index to all training data points.

4.7 Multilayer Perceptrons Implementation

There are dozens of back-propagation implementations in the published neural network books using different programming languages (FORTRAN, C, C++ and MATLAB) in a variety of platforms (WINDOWS, UNIX etc.). One of the implementations is particularly interesting and is chosen to interface with the SOURCAO.

Rogers (1997) took an object-oriented approach to implement the neural network architectures. Two reusable neural network base classes -- *Base_Node* and *Base_Link* classes -- represented the perceptrons and the connections between the different layers. Rogers unified most of the neural network architectures by taking full advantage of the OO concepts of classes, objects, inheritance, virtual function, polymorphism, and dynamic binding, etc.

It is natural to take a link-node view of the neural network. For example, for the MLP, we can view the weight w as the attributes of the link and the perceptron $V_i(L)$ as an attribute of a node.

There are several issues in neural network implementations. This section addresses normalization of training data, the speed of training, the architecture parameters and the training algorithms. The test data set in this section comes from first three evaluation cases. The input nodes are shown in Table 4-2 and the output node is the entire network delay in seconds per vehicle. The training processes in this section are intended to explore the training techniques only and do not converge when the tests end.

Table 4-2 Neural Network Training Input Nodes

Variable Type	Name	Number	History	Number of Variables
Control Variables	Phase Length	By the intersection	Next three phases	3*Numberof OptimizedSignal =6
State Variables	Phase ID and Offset	By control signal	Current	2*Numberof OptimizedSignal =4
	Number of vehicles in the link	By the link queue	Current, previous second and the second before previous	3*Numberof Queue=36
	Number of vehicles entered and exited a link during an interval		Two intervals before current time	4*Numberof Queue=48
	Link Delay		Two intervals before current time	2*Numberof Queue=24
	Turning Queue Delay	By the turning queue	Two intervals before current time	2*Numberof TurningQueue=8
	Network Delay		Two intervals before current time	2
	Grade Crossing	Close/open	Current/next	4
Total				132

Normalization of training data. During the author's experiences in the neural network training, it is noticed that if the input data had not been normalized, the neural network training would not converge.

The speed of training. The greatest challenge to neural network training is the training time. It is especially true for a real time system like traffic control. SOURCAO utilizes the delay forecast based on neural network training. However, for the delay forecast, the author believes the traffic pattern on the road does not change everyday. The trained neural network weights should last for a certain period (say, a week or a month). Therefore, the slow convergence of the weights is not a factor limiting the neural network application in this research.

The architecture parameters. Some algorithms can automatically configure some neural network architecture parameters like the number of layers and the number of hidden nodes etc. Virtually, those algorithms take a practical approach to grow the layers and perceptrons from a small size, for example, Cascade-correlation learning architecture (Fahlman and Lebirere 1990) or structure-level adaptation (Lee, Peterson and Tsai 1990). Although those algorithms are not implemented in the training process described above, the idea is applied here by comparing the different architectures.

The different network parameters may result in different convergence speed. A test is performed to determine two- and three-layer architectures. As shown in Figure 4-2, the x-axle represents processor time (how much processor time that the calling process has used) and the y-axle represents the error measured by (4-59). The Error is the summation of the square of the difference of MLP calibrated delays (minutes/second) and the proposed calibrated delay (minutes/second). From the test, it seems that there is almost no difference in training speed for two- and three-layer architectures (represented with blue and red lines, respectively). In Figure 4-2, the MLP training is intended to test the training speed between two- and three-layer architectures only and training does not converge at the end of test.

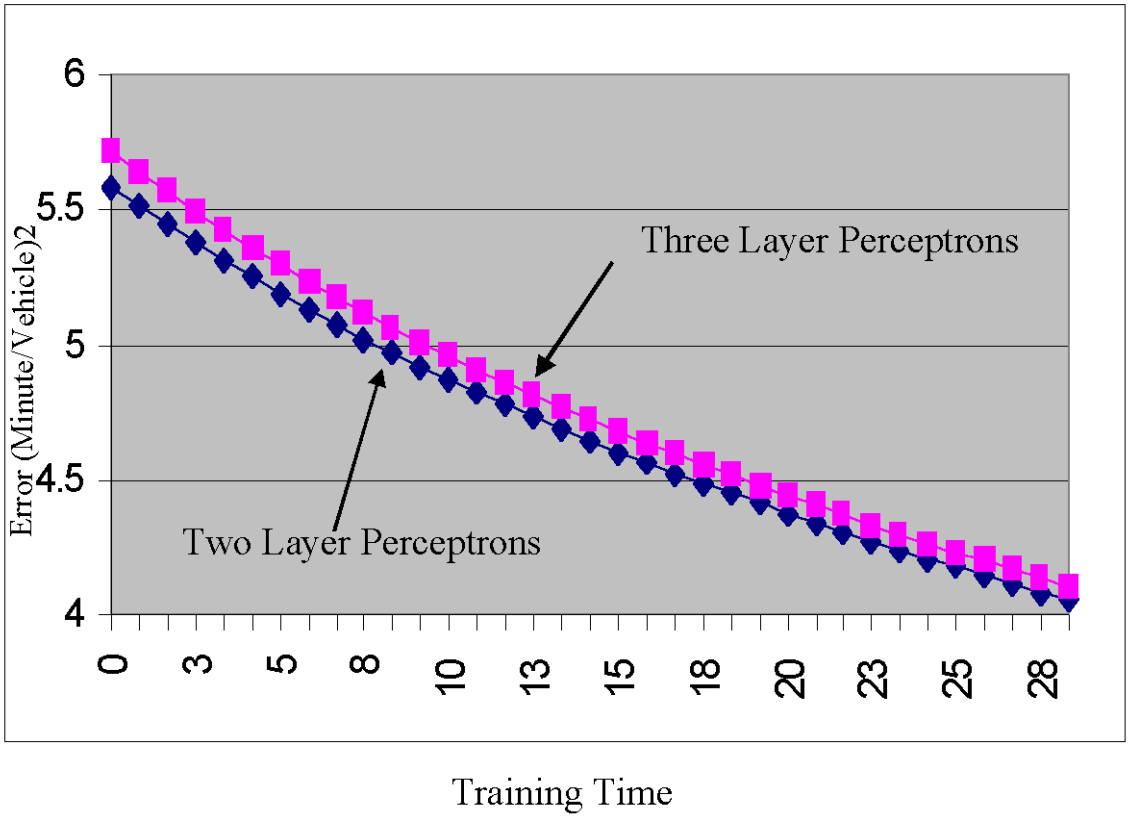

Figure 4-2 Two and Three Layer MLP Training Speed[3]

Training algorithms. There are two approaches to train the MLP network (Haykin 1999): Supervised Learning and the Optimization. As indicated above, the speed of the training is not the major concern in this research. There is no comparison between the two types of the approaches. However, a comparison is made between the two commonly used algorithms in supervised learning category: the back-propagation and the batch updating rules. Figure 4-3 shows the convergence of the back-propagation with time (with the learning rate of 0.01). However, in the batch training, the error fails to decline although there is a wide range of the learning rates: from 0.1 to 0.0000001. In

[3] : The time is measured by the Microsoft Visual C++ 6.0 clock() function, with this configuration of the computer: Intel PII 333 Mobile CPU, 64M RAM, Windows 98 OS.

Figure 4-3, the MLP training is intended to test the training only and training does not converge at the end of test.

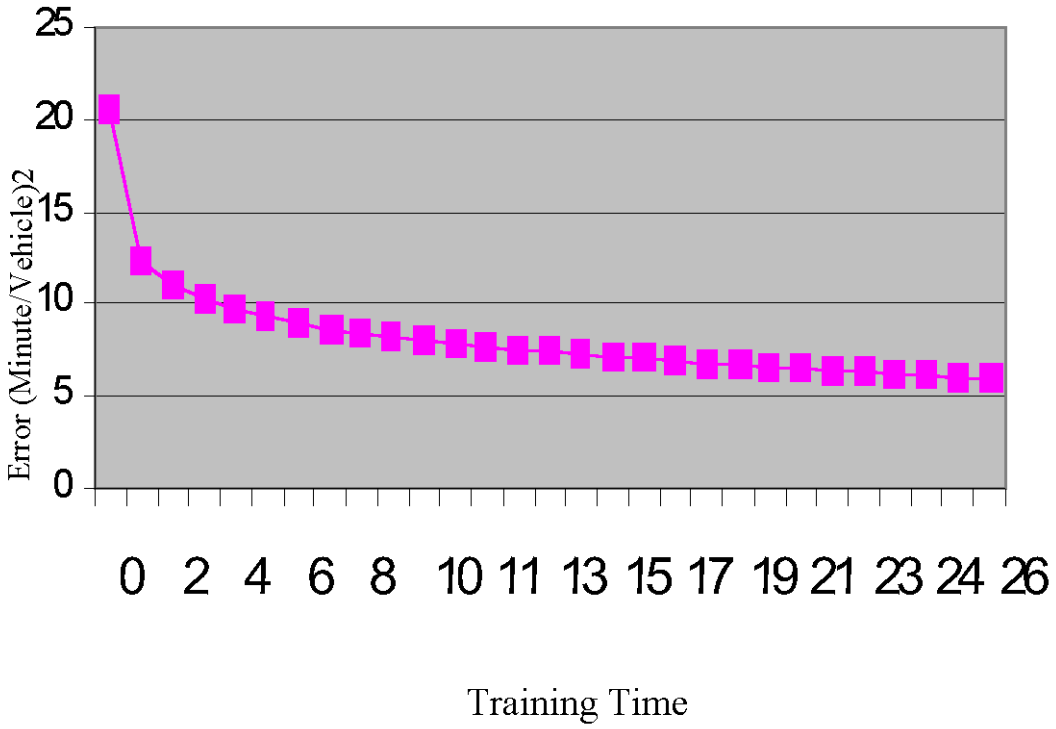

Figure 4-3 Declining Error with the Time in Back-Propagation[3]

The simple back-propagation is usually superior to the batch since minimizing the error in a random data point allows a wider search space and avoids the local minimum. The complexity of the batch training does not improve the quality of the solution, especially for such a large and sophisticated problem in SOURCAO. The test here just demonstrates this property.