Online Learning Solutions for Freeway Travel Time Prediction

J. W. C. van Lint

***Abstract*—Providing travel time information to travelers on available route alternatives in traffic networks is widely believed to yield positive effects on individual drive behavior and (route/departure time) choice behavior, as well as on collective traffic operations in terms of, for example, overall time savings and—if nothing else—on the reliability of travel times. As such, there is an increasing need for fast and reliable online travel time prediction models. Previous research showed that data-driven approaches such as the state-space neural network (SSNN) are reliable and accurate travel time predictors for freeway routes, which can be used to provide predictive travel time information on, for example, variable message sign panels. In an operational context, the adaptivity of such models is a crucial property. Since travel times are available (and, hence, can be measured) for *realized* trips only, adapting the parameters (weights) of a data-driven travel time prediction model such as the SSNN is particularly challenging. This paper proposes a new extended Kalman filter (EKF) based online-learning approach, i.e., the *online-censored EKF* method, which *can* be applied online and offers improvements over a delayed approach in which learning takes place only as realized travel times are available.**

***Index Terms*—Advanced traffic information systems (ATIS), extended Kalman filter, online learning, recurrent neural networks, state space neural networks, traffic information, travel time prediction.**

I. Introduction

THERE is an increasing need for advanced traffic information systems (ATIS) that can provide road users and traffic managers with accurate and reliable real-time traffic information. This paper focuses on one particular brand of traffic information, that is, short-term predictions of travel time on freeways, which can be applied, for example, for real-time freeway ATIS, such as variable message signs (VMSs) at bifurcations. As outlined in [1], travel time is the product of highly dynamic and nonlinear traffic processes over space and time, which are (inherently) *a priori* unknown. The travel time $y_{k,i}$ for a vehicle i departing during period k on some route in a traffic network is the result of the traffic conditions (speeds, flows, densities) along the route at time periods $p \in \{k, \ldots, k + y_{k,i}\}$. These traffic conditions may be influenced by many internal or external factors affecting both traffic demand and route capacity along the route during these periods, some of which are clearly beyond the ability of the analyst to predict (e.g., incidents, accidents). Similarly, the expected travel time $y_k = \langle y_{k,i} \rangle_i$ for vehicles departing at k is a result of traffic conditions during periods $p \in \{k, \ldots, k + y_k\}$. Travel time prediction, hence, implicitly requires predicting—to the degree that this is possible—those future traffic conditions along the route of interest. This poses a "chicken-and-egg" type of problem since the length of the prediction horizon is equal to the travel time that we wanted to predict in the first place. In [1]–[3], comprehensive overviews are given on how different strands of travel time prediction approaches tackle this problem. These strands involve (traffic simulation) *model-based approaches* (e.g., DynaMIT [4], DynaSMART, [5], BOSS [6]) and *instantaneous approaches* [7]–[9]. The latter ignore the time dynamics altogether by assuming stationary traffic conditions for an indefinite time period, whereas model-based approaches predict speeds or flows for as long as is required to derive a travel time estimate on the route of interest. A third strand of travel time prediction approaches according to [1] and [2] uses intelligent inductive (data-driven) models that are able to directly learn the complex traffic dynamics from the data on the route of interest. Many successful efforts have been reported in the latter category, including support vector regression approaches [10], generalized linear regression [7], [11], nonlinear time series [12], state-space models and Kalman filters [13], [14], feedforward neural networks [15], [16], and recurrent neural networks [17], to name a few.

A typical class of data-driven travel time prediction models is the so-called state-space neural network (SSNN) proposed in [1]–[3]. Previous research illustrates that combined with simple data preprocessing algorithms, the SSNN model predictions gracefully deteriorate under increasing amounts of missing or unreliable input data [1] and produce an online estimate for the reliability of each prediction by means of confidence intervals [3]. This is useful in cases when, for example, too many input data are missing or when unusual traffic circumstances (incidents, accidents) occur. Under these conditions, naturally, the model makes larger errors, while at the same time, the confidence intervals grow larger, providing the model user with an online indication of the quality of its predictions.

Although [1], [3], and [18] showed that this SSNN method outperforms instantaneous travel time models by a large margin and provides similar (and similarly good) results as other state-of-the-art travel time prediction models, a disadvantage, from a practical point of view, is that for training, large amounts of training data are required (in one application, at least 30–60 workdays of input–output data [18]). Besides the data storage and computational requirements associated with offline

Manuscript received January 8, 2007; revised April 13, 2007, August 15, 2007, October 14, 2007, and November 7, 2007. This work was supported by the Advanced Traffic Monitoring project under the Dutch Transumo program (www.atmo.tudelft.nl). The Associate Editor for this paper was Y. Wang.

The author is with the Department of Transport and Planning, Faculty of Civil Engineering and Geosciences, Delft University of Technology, 2600 Delft, The Netherlands (e-mail: j.w.c.vanlint@tudelft.nl).

Digital Object Identifier 10.1109/TITS.2007.915649

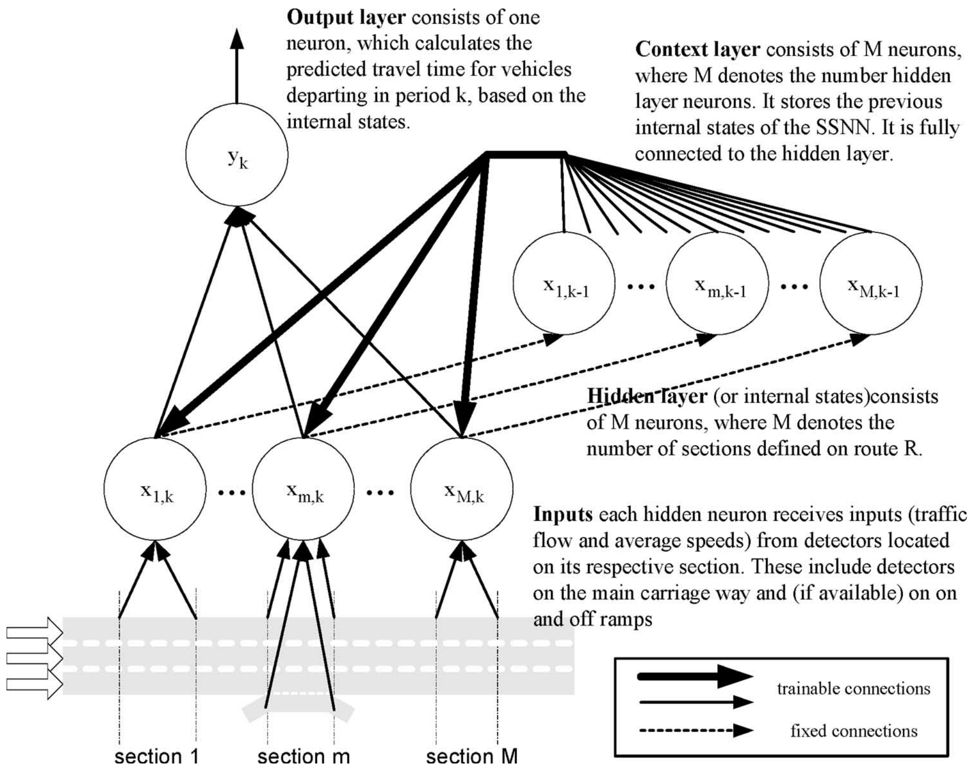
Output layer consists of one neuron, which calculates the predicted travel time for vehicles departing in period k, based on the internal states.
Context layer consists of M neurons, where M denotes the number hidden layer neurons. It stores the previous internal states of the SSNN. It is fully connected to the hidden layer.
y_k
$x_{1,k-1}$ … $x_{m,k-1}$ … $x_{M,k-1}$
Hidden layer (or internal states)consists of M neurons, where M denotes the number of sections defined on route R.
$x_{1,k}$ … $x_{m,k}$ … $x_{M,k}$
Inputs each hidden neuron receives inputs (traffic flow and average speeds) from detectors located on its respective section. These include detectors on the main carriage way and (if available) on on and off ramps
trainable connections
fixed connections
section 1
section m
section M

Fig. 1. State-space neural network for freeway travel time prediction.

training schemes, a second more fundamental problem arises in cases in which physical changes occur either in the traffic system itself or in the monitoring system providing the input data for the SSNN. This paper, therefore, addresses the question whether it is possible to train the SSNN model in an *online* fashion such that it maintains its capabilities as an accurate and robust predictor but is adaptive to changes in either the underlying traffic processes or the surveillance system providing its input data.

Generally speaking, online learning algorithms are a special class of *incremental* learning algorithms. These incremental learning algorithms adapt the model weights (parameters) $\boldsymbol{\psi}$ after observing a single input–output (target) pattern $\{\mathbf{u}_k, d_k\}$. In contrast, *batch* learning algorithms (which are used to train the SSNN models in [1]–[3]) adapt model weights after observing an entire batch of input–output data $\{\mathbf{u}_k, d_k\}$, $k = 1, 2, \ldots$. As a consequence, batch algorithms can only be applied offline. In the case where, at each time instant k, both $\mathbf{u}_k$ and d_{k-1} are available, incremental algorithms can be applied online in a so-called one-step-ahead prediction procedure. Roughly, such an online learning algorithm reads as follows, where $y_{k+1} = G(\boldsymbol{\psi}, \mathbf{u}_k)$ depicts a data-driven model.

1) Make a prediction $y_k = G(\boldsymbol{\psi}_k, \mathbf{u}_k)$.
2) Set $k := k + 1$, and update model weights $\boldsymbol{\psi}_{k-1}$ with error $\varepsilon_{k-1} = d_{k-1} - y_{k-1}$ yielding the updated weights $\boldsymbol{\psi}_k$.
3) Go to step 1.

In a travel time prediction context, this one-step-ahead procedure is clearly not applicable since a realized (actually measured) travel time d_k is not available at time instant $k + 1$ but, in fact, after $k + d_k$ time periods. The consequence is that no standard online learning procedures can be applied to online travel time prediction.

This paper proposes two online learning algorithms based on the extended Kalman filter (EKF) that are tailored to solve this problem. Although the algorithms described can be applied to any data-driven travel time prediction model, here, we demonstrate them with the SSNN model. In Section II, the mathematical structure of this proposed model is outlined. Next, the standard EKF algorithm for parameter estimation is introduced along with two online variations, which will be evaluated on the basis of real data in the second part of this paper. This paper closes with conclusions and recommendations for further research.

II. Mathematical Structure of the SSNN Model

The structure of the SSNN model [17] is schematically outlined in Fig. 1. Each hidden neuron in the SSNN model is thought to "represent" a particular section along the route. Each of these neurons receives *current* input signals from detectors on the associated section only and past signals (stored in the context layer) from all other hidden neurons.

Mathematically, the SSNN can be formulated as follows. First, every hidden neuron j calculates its state (its output) for the next time step k using the latest available input signals $\mathbf{u}_{k-1}$ and outputs $\mathbf{x}_{k-1}$ of all hidden neurons, i.e.,

$$x_{j,k} = f\left(w_{j0} + \sum_{h=1}^{H} w_{jh}^{x} x_{h,k-1} + \sum_{i=1}^{I_j} w_{ji}^{u} u_{ij,k-1}\right) \qquad (1)$$

where w_{jx} depicts the weight from neuron i to neuron or input x. Second, the (only) output neuron calculates the expected travel time with the current states of all of the H hidden neurons, i.e.,

$$y_k = h\left(v_0 + \sum_{j=1}^{H} v_j x_{j,k}\right) \tag{2}$$

where v_j depicts the weight from neuron j to the output neuron; f and h are the hidden and output layer transfer functions, for which we choose logistic sigmoid and identity function, respectively, i.e.,

$$\begin{aligned} f(z) &= 1/\left(1+\exp(-z)\right)\\ h(z) &= z. \end{aligned}$$

As a side effect of its structure, the SSNN model only requires *current* measurements as inputs (rather than time series of inputs), which makes it easier to implement than neural models with more complex input configurations. Second, the SSNN topology is based on the geometry and detector layout of the route of interest, which makes the model generic in terms of mathematical structure and applicable on all freeway stretches, given that these are equipped with detectors.

III. Algorithms for SSNN Training

A. Batch Training Algorithm

1) Bayesian-Regulated Levenberg Marquardt (LMBR) Algorithm: In previous studies [1], [3], [18], the SSNN model is trained with a second-order (batch) Bayesian-regularized [19] training algorithm. This method aims at minimizing the following cost function:

$$C = \beta \sum_{N_k} (\varepsilon_k)^2 + \alpha \sum_{N_\psi} (\psi_n)^2 \tag{3}$$

where

$$\varepsilon_k = d_k - y_k \tag{4}$$

denotes the model error (desired-model prediction), and ψ_n, $n = 1, 2, \ldots, N_\psi$ are the elements of weight vector $\boldsymbol{\psi}$ of size N_ψ containing all SSNN weights. In (3), β and α are hyperparameters controlling the contribution of each of the two components in the cost function. This cost function, hence, balances between minimizing the sum of squared errors (first component) with the sum of squared weights (second component) based on the notion that larger weights make the model more sensitive and increase the risk of overfitting the training data. The weights in the model are updated after a batch of N_k inputs and outputs is presented according to the Levenberg–Marquardt (LM) [20] weight update rule, i.e.,

$$\boldsymbol{\psi}^{\text{new}} = \boldsymbol{\psi}^{\text{old}} - \left[\beta\mathbf{H} + (\mu+\alpha)\mathbf{I}\right]^{-1}\left[\mathbf{J}^T\boldsymbol{\varepsilon} + \alpha\boldsymbol{\psi}^{\text{old}}\right] \tag{5}$$

where

$$\begin{aligned} \mathbf{J} &= \frac{\partial \mathbf{y}}{\partial \boldsymbol{\psi}}\\ \mathbf{H} &= \frac{\partial^2 \mathbf{y}}{\partial \boldsymbol{\psi}^2} \approx \mathbf{J}^T\mathbf{J} \end{aligned}$$

are the first and (approximated) second derivatives (Jacobian and Hessian) of the SSNN output (batch) with respect to its weights, respectively, and $\boldsymbol{\varepsilon}$ denotes a vector of N_k prediction errors. $\mathbf{J}$ can be straightforwardly calculated through back-propagating the network outputs y_k, each producing one column of the Jacobian matrix. In (5), μ represents another hyperparameter that is adjusted during training and that balances the algorithm between gradient decent (large μ) and approximate Newton (small μ). Note that the inverse Hessian $\mathbf{H}$ can be interpreted as a variance–covariance matrix of the SSNN weights ($\boldsymbol{\Sigma} = \mathbf{H}^{-1}$). Furthermore, since the SSNN is a *dynamic* neural network, calculating $\mathbf{J}$ requires either (truncated) back-propagation through time (BPTT) or real-time recurrent learning or other methods that incorporate the internal recurrence in the SSNN model. For the implementation details of these and other recurrent neural network training algorithms, see [21]–[23]. In our case, we use BPTT, where we truncate the time recursion to 15 discrete time steps, which appeared to work best in our experiments.

MacKay [19] argues that both output errors and weights can be interpreted as Gaussian noise processes with prior variances of $1/\beta$ and $1/\alpha$, respectively, and shows that minimizing cost function (3) is equivalent to maximizing the posterior probability density of the weights *given* the training data, hyperparameters, and all other assumptions Ω (e.g., model structure), which reads

$$P(\boldsymbol{\psi}|D,\alpha,\beta,\Omega) = \frac{P(D|\boldsymbol{\psi},\alpha,\beta,\Omega)P(\boldsymbol{\psi}|\alpha,\beta,\Omega)}{P(D|\alpha,\beta,\Omega)}$$

which, in turn, is equivalent to minimizing the log posterior

$$C^* = -\log\left(P(\boldsymbol{\psi}|D,\alpha,\beta,\Omega)\right) \tag{6}$$

where D depicts the available training data $\{\mathbf{u}_k, y_k\}$, $k = 1, \ldots, N_k$. The benefit of translating cost function C (3) into C^* (6) is that Bayes' rule (used to calculate the posterior) automatically embodies Occam's razor [19], and that minimizing C^*, thus, leads to the simplest setting of ψ that is still warranted by the data D. In [19] and [24], it is shown that the maximum likelihood estimates for α and β can be expressed as follows:

$$\begin{aligned} \alpha &= \frac{\gamma}{\sum_{N_\psi}(\psi_n)^2}\\ \beta &= \frac{N_k - \gamma}{\sum_{N_k}(\varepsilon_k)^2} \end{aligned} \tag{7}$$

where

$$\gamma = N_\psi - \alpha \cdot \text{trace}(\boldsymbol{\Sigma})$$

denotes the number of *effective* weights. Note that γ lies in between zero and N_ψ (the total number of weights).

2) Implementation Issues—Model Ensembles: There are a number of reasons why the training algorithm above is cumbersome to implement in practice. The most important is that for any reasonably sized route, for example, one that is equipped with 20–30 detectors, the amount of training data rapidly becomes very large, and training becomes either unfeasible or at least very time consuming. This is particularly due to the calculation and the inversion of the Hessian in (5). A cheap yet robust solution is to partition the training data set into L random subsets which contain, on average, B (< 1) times the total number of available training records [3]. Subsequently, L SSNN models, each with exactly the same mathematical structure, are trained on these random subsets, resulting in an ensemble of L SSNN models. This procedure is called *random subsampling* and is described in [25]. The mean prediction of this ensemble on a particular input pattern is then equal to

$$y_k = \frac{1}{L}\sum_{n=1}^{L} y_k^n \tag{8}$$

where y_k^n denotes the output of model n calculated with (1) and (2). As a bonus, confidence intervals

$$y_k(p) \pm c \times \sigma_k^y \tag{9}$$

around this mean ensemble prediction can be constructed, where

$$\sigma_k^y = \sqrt{\frac{B}{(L-1)}\sum_{n=1}^{L}\left[y_k^n - y_k\right]^2}$$

denotes the standard error of the mean. The term B underneath the square root scales the standard error with the size $(100 \times B\%)$ of the random subsets. Note that what is subsampled here are not single input–output data patterns but entire afternoon sequences of input–output data patterns (see Section V for more details in the test case). The rationale is that the temporal evolution of the input–output relationship needs to be preserved to have the SSNN model correctly infer the travel times.

Although subsampling clearly reduces the computational burden of the training problem, the ensemble algorithm is still (computationally) costly, particularly in terms of calculating and inverting $\mathbf{H}$ in (5). Moreover, the offline training algorithm inevitably requires a large database of input and output patterns to be available beforehand. In case of changes in either the physical traffic processes (e.g., different speed limits, extra lanes) or in the traffic monitoring system (new or obsolete detectors), one would effectively have to wait until such a database is compiled before the SSNN model(s) can be retrained. Section III-B discusses an incremental learning algorithm, which allows for much faster adaptation of the model to such new situations.

B. Incremental EKF Training Algorithm

In an incremental learning context, the weights $\boldsymbol{\psi}_k$ are updated after a single input–output $\{\mathbf{u}_k, y_k\}$ pattern is observed. The underlying idea is that the weights $\boldsymbol{\psi}_k$ are assumed to correspond to a stationary process (a random walk), i.e.,

$$\hat{\boldsymbol{\psi}}_k = \hat{\boldsymbol{\psi}}_{k-1} + \mathbf{r}_{k-1} \tag{10}$$

and that the SSNN model makes a nonlinear observation, i.e.,

$$y_k = G(\mathbf{u}_k, \boldsymbol{\psi}_k) \tag{11}$$

on its weights $\boldsymbol{\psi}_k$, where G represents the entire SSNN mapping, and r_k represents a zero-mean Gaussian white noise term. Given the state space formulation of (10) and (11), the well-known EKF equations can be applied to update the weights in an incremental fashion and, at the same time, maintain an estimation error covariance matrix $\boldsymbol{\Sigma}_\mathbf{k}$ of the weights. Assuming that the noise in process (10) and the model errors ε_k in (4) are additive Gaussian white noises, and that ε_k is uncorrelated to the variance in the targets σ_d^2, the EKF algorithm can be employed to minimize the following cost function:

$$C = \sum_{N_k} \varepsilon_k^2. \tag{12}$$

The EKF algorithm reads as follows.

1) Initialization. First, the weights $\boldsymbol{\psi}$ and the error covariance matrix $\boldsymbol{\Sigma}$ are initialized. For initializing $\boldsymbol{\psi}$, for example, the Nguyen–Widrow method [26] can be used, whereas $\boldsymbol{\Sigma}$ is usually initialized by a large diagonal matrix, which reflects the fact that we have no prior knowledge on the weight setting. Restricting to setting diagonal elements only implies that we initially assume independence among weights, i.e.,

$$\hat{\boldsymbol{\psi}}_0 = E[\boldsymbol{\psi}] \tag{13}$$

$$\boldsymbol{\Sigma}_0 = E\left[(\boldsymbol{\psi} - \hat{\boldsymbol{\psi}}_0)(\boldsymbol{\psi} - \hat{\boldsymbol{\psi}}_0)^T\right]. \tag{14}$$

Now, for $k = 1, 2, \ldots$, recursively apply the following time and measurement updates.

2) Time update (or prediction step)

$$\hat{\boldsymbol{\psi}}_{k|k-1} = \hat{\boldsymbol{\psi}}_{k-1} \tag{15}$$

$$\boldsymbol{\Sigma}_{k|k-1} = \boldsymbol{\Sigma}_{k-1} + \mathbf{R}_{k-1}^r, \qquad \mathbf{R}_{k-1}^r = E\left[\mathbf{r}_k\mathbf{r}_k^T\right]. \tag{16}$$

3) Measurement update (correction step)

$$\widehat{y}_k = F(\mathbf{x}_k, \hat{\boldsymbol{\psi}}_{k|k-1}) \rightarrow \varepsilon_k = d_k - \widehat{y}_k \tag{17}$$

$$\mathbf{K}_k = \frac{\boldsymbol{\Sigma}_{k|k-1}\mathbf{J}_k^T}{\boldsymbol{\Sigma}_{k|k-1}\mathbf{J}_k^T\boldsymbol{\Sigma}_{k|k-1}^T + r_k^\varepsilon}, \quad r_k^\varepsilon = E\left[\varepsilon_k^2\right] \tag{18}$$

$$\hat{\boldsymbol{\psi}}_k = \hat{\boldsymbol{\psi}}_{k|k-1} + \mathbf{K}_k\varepsilon_k \tag{19}$$

$$\begin{aligned}\boldsymbol{\Sigma}_k &= \boldsymbol{\Sigma}_{k|k-1} - \mathbf{K}_k\mathbf{J}_k\boldsymbol{\Sigma}_{k|k-1}\\ &= (\mathbf{I} - \mathbf{K}_k\mathbf{J}_k)\boldsymbol{\Sigma}_{k|k-1}.\end{aligned} \tag{20}$$

Since r_k^ε reflects a variance estimate of the measurement errors (that is, the inherent noise in the targets plus the noise resulting from the SSNN model), we propose the following smoothed estimate:

$$r_{k+1}^\varepsilon = (1-\lambda)r_k^\varepsilon + \lambda(\varepsilon_k + \varepsilon^0)^2 \tag{21}$$

where ε^0 is a constant depicting the inherent target noise, and λ is a smoothing constant $\ll 1$. The effect of (21) is similar to that of a variable learning rate $(\sim 1/r_k^\varepsilon)$ in standard back-propagation algorithms [21]. The smaller the r_k^ε, the larger the weight updates. Last, note that the Kalman gain $\mathbf{K}_k$ (18) can be interpreted as follows:

$$\mathbf{K}_k = \frac{\text{variance weights}}{\text{variance outputs}} \times \text{sensitivity model to weights.}$$

Thus, what the EKF algorithm effectively does is that it recursively updates $\boldsymbol{\psi}_k$ with a factor $\mathbf{K}_k$, which balances the uncertainty in the models' weights with the total uncertainty (noise) in the measurement equation, which is also a function of $\boldsymbol{\Sigma}_{k|k-1}$. For example, large model uncertainty and small output uncertainty imply large weight updates. Conversely, large output uncertainty and small model uncertainty result in small weight updates.

C. Online Applicable EKF Algorithms

As explained in Section I, the incremental learning EKF algorithm described above cannot be applied online for the travel time prediction task due to the fact that travel time prediction is not a one-step-ahead prediction problem. In this section, we propose two algorithms that solve this problem. The first is a delayed EKF approach, and the second one involves the use of the so-called *censored* observations.

1) Online-Delayed EKF Algorithm: The rationale behind this algorithm is straightforward. A weight update is applied only at time instants k for which realized travel times are available. This inherently delayed learning algorithm reads as follows.

1) At time period p, determine the departure time period $k = \text{floor}(p - d_k)$ for which the last (arrival) travel time d_k was available.
2) Update weights based on inputs (speeds, flows) and targets (travel time d_k) available at period k.
3) Now, predict travel time at time period p with updated weights.

Inherently, this delayed learning method will cause the travel time prediction model to lag behind, at least in the early stages of learning.

2) Censored EKF Algorithm: Consider that at some time period p, the last realized travel time d_m is available from vehicles departing at period m, where $m = p - d_m$. Although, for periods k, $m < k < p$, no realized travel times are available yet, a *censored* observation (in fact, a lower-bound value) is given by

$$d_k > d_k^*(p) = p - k. \tag{22}$$

Although the true prediction error $\varepsilon_k = d_k - y_k$ [where $y_k = G(\boldsymbol{\psi}_k, \mathbf{u}_k)$ in (11)] is not available, again, a *censored* observation of this error is given by

$$\varepsilon_k^*(p) = d_k^*(p) - y_k. \tag{23}$$

Due to (22), (23) represents a monotonically increasing lower bound of the true error ε_k, i.e.,

$$\varepsilon_k^*(p) < \varepsilon_k, \quad m < k < p. \tag{24}$$

At each time period $p > k$ for which no realized travel time d_k of vehicles departing at k is available, the censored error (24) provides an incremental estimate of the model prediction error. Letting

$$\xi_k(p) = \varepsilon_k^*(p) - \varepsilon_k^*(p-1) > 0 \tag{25}$$

where

$$\sum_{p=k+1}^{m+d_m+1} \xi_k(p) = \varepsilon_k$$

implies that for a particular departure time k for which no realized travel time is available, the weights $\boldsymbol{\psi}_k$ can be updated stepwise at each $p > k$ by substituting (25) into (19). Such an update is retained if this update indeed improves the model performance, that is, if

$$d_k^*(p) - G(\mathbf{u}_k, \boldsymbol{\psi}_k) > d_k^*(p) - G(\mathbf{u}_k, \boldsymbol{\psi}_{k+1}) \tag{26}$$

which is the case if and only if

$$G(\mathbf{u}_k, \boldsymbol{\psi}_{k+1}) > G(\mathbf{u}_k, \boldsymbol{\psi}_k). \tag{27}$$

In all other cases, the update is discarded. Constraint (27) implies that if the parameter update results in a larger predicted travel time than before, it is retained; otherwise, it is discarded, in which case, $\varepsilon_k^*(p)$ must be reset to zero. Intuitively, this procedure makes sense. For example, in cases where travel times (of, for example, 10 min) are an order-of-magnitude larger than the unit of discrete time k (of, for example, 1 min), the lower bound of (22) will initially (as p is only a few time steps away from k) be much smaller than free-flow travel times. Adapting the weights to these clearly underestimated travel times would not improve performance at all. In situations of congestion buildup, during which travel times tend to increase, it is clear that, according to (27), updates are retained only if these contribute to the increasing trend. In case of declining congestion, during which travel times tend to decrease, (27) has no effect since, in those cases, realized travel times will become available increasingly faster.

Last, note that at any particular time period p, there will be a number of past time periods k for which no realized travel times are available yet. This means that per time period p, possibly more than one weight can be applied with censored errors. In this paper, this is done sequentially, whereas at each update, (27) is evaluated with respect to the last weight update, which could also have been applied during p.

D. Regularization/Weight Constraining

We adopt a heuristic approach proposed in [27] to avoid the SSNN weights become too large and, hence, force the SSNN mapping to remain smooth. Essentially, this procedure has a similar effect as the Bayesian regularization scheme presented before, that is, it prevents the model from overfitting and helps it maintain a more smooth and general mapping. A likely consequence of such a constraint learning algorithm is that it leads to a model that performs slightly worse than one trained in an unconstrained fashion.

The weight constraining algorithm works as follows. Consider the mappings $\boldsymbol{\psi}^{\alpha} = \varphi(\boldsymbol{\psi}, \alpha)$ and $\boldsymbol{\psi} = \varphi^{-1}(\boldsymbol{\psi}^{\alpha}, \alpha)$, which transform the original weights $\boldsymbol{\psi} \in [-\infty, +\infty]$ to constrained weight space $\boldsymbol{\psi}^{\alpha} \in [-\alpha, +\alpha]$ and vice versa. Given that

1) φ is a continuous differential function over $[-\infty, +\infty]$;
2) φ^{-1} exists and is continuous over $[-\alpha, +\alpha]$;
3) $\boldsymbol{\psi}^{\alpha} \to \boldsymbol{\psi}$ as $\alpha \to \infty$;

we can assess model performance (17) in the constrained weight space and update the weights according to (18) and (19) in the unconstrained weight space with just a few minor changes in the algorithm.

We first *unconstrain* the weights after prediction step (17) with $\boldsymbol{\psi} = \varphi^{-1}(\boldsymbol{\psi}^{\alpha}, \alpha)$ and adjust the Jacobian ($\mathbf{J} = dy_k/d\boldsymbol{\psi}$) with a factor $d\varphi(\boldsymbol{\psi}, \alpha)/d\boldsymbol{\psi}^{\alpha}$ to account for the sensitivity of the model to the constraint weights, which are actually used to predict the output. After updating the now unconstrained weights (19), these are transformed back into the constrained weight space with $\boldsymbol{\psi}^{\alpha} = \boldsymbol{\varphi}(\boldsymbol{\psi}, \alpha)$ and used in the next prediction step. In the following, (28) is used to translate and constrain the weights from the interval $[\beta - \alpha, \beta + \alpha]$ to $[-\infty, +\infty]$ (based on [27]):

$$\varphi(\boldsymbol{\psi}) = \beta + \frac{\boldsymbol{\psi}}{1 + |\boldsymbol{\psi}|/\alpha}$$
$$\varphi^{-1}(\boldsymbol{\psi}^{\alpha}) = \frac{\boldsymbol{\psi}^{\alpha} - \beta}{1 - |\boldsymbol{\psi}^{\alpha} - \beta|/\alpha} \tag{28}$$

where we set $\beta = 0$ so that weights are constrained around zero.

IV. Experimental Setup

A. Data

In this study, an SSNN travel time prediction model is built for the 7-km three-lane A13 southbound freeway stretch between The Hague and Delft, The Netherlands. We selected the data that are representative for regular congestion and, therefore, chose all (available) congested weekday afternoon periods (between 14:00 and 20:00) in 2004. Note that in all selected peak periods, congestion occurred in which the travel time during congestion was at least twice as high (i.e., > 10 min) than the free-flow travel time (around 4 min). Note also that no additional information was available on the occurrence of, for example, incidents or accidents. The source data consist of speeds and flows from dual inductive loops that are installed, on average, every 500 m along this freeway stretch and come from the Regiolab–Delft traffic data server [28]. As inputs, spot mean speeds and vehicular flow per minute from each detector along the main carriageway are used. For targets, travel times are estimated (offline) with the so-called piecewise linear speed-based (PLSB) trajectory method [29]. Although, in the following, these PLSB travel times are used as "ground truth" targets, there is no guarantee that these provide an unbiased estimate of real travel times (see, e.g., [30]), which, unfortunately, were not available for this paper.

B. Model Design

Given the used inputs—spot mean speeds and flows (over the entire carriageway)—from a total of 14 consecutive dual loops on the A13 southbound freeway mentioned above, the SSNN model structure used below is straightforwardly derived. It consists of 13 hidden units, each receiving input signals associated with the 13 consecutive freeway sections, which are each enclosed by upstream and downstream detectors. The context layer also consists of 13 units and is fully connected to the hidden layer. A schematic overview is given in Fig. 1.

A few *a priori* remarks must be made with respect to the relation between the SSNN design and the route length and discrete time interval that are used. Longer discrete time intervals yield a smoother travel time (target) curve, which is probably easier to track (and leads to better performance) with an online learning method than a target curve on the basis of short discrete time instants. However, for an individual traveler confronted with these courser travel time predictions, the errors would probably grow larger since his or her experienced travel time would, on average, be available *later* to the model.

A similar argument can be made with regard to route length. Since the SSNN model is only fed with data from the route itself (see Fig. 1), it can only respond to travel time changes if these are "visible" in these data. This implies that, on one hand, longer routes would yield an easier prediction task since it is more likely on longer routes that the cause of a travel time increase (congestion) is visible *earlier* in the data. On the other hand, longer routes make the travel time prediction inherently more difficult since on longer routes, longer travel times occur (and, hence, a larger delay before these are realized).

A full investigation into the effects of model design and data configuration is beyond the scope of this paper.

C. Training and Testing

For comparison, we tested five similar SSNN models with the two online EKF algorithms with varying degrees of weight constraining, that is, with α set to 1, 5, 10, 15, and ∞ (infinite—implying no weight constraining). Each of these ten models (5° of weight constraining $\times$ 2 online learning algorithms) was adapted online on a test data set (*B*) that is compiled from a total of 65 6-h (14:00–20:00) peak periods. The rest of the data (150 6-h peak periods-data set *A*) were used to train an ensemble of ten SSNN models with the offline LMBR batch training algorithm described above. The size of each of the ten subsampled training data sets was 20% of the total data set (i.e., 30 afternoon peaks on average). The ensemble was then tested on the same data set (*B*) as the online EKF models. Last, all models are compared against two baseline models. The first is a naive prediction model, i.e., the so-called instantaneous travel time, which is defined as

$$TT_k^{\text{inst}} = \sum_{M} \frac{L_m}{v_{k,m}} \tag{29}$$

TABLE I
PERFORMANCE ONLINE-DELAYED EKF SSNN MODELS ON DATA SET B

Weight constraining	R^2_{perc} [%]	RMSE [s]	Bias [s]	RRE [s]
α=1	80,7	118	-14,0	117
α=5	80,7	118	-13,3	117
α=10	80,8	118	-13,4	117
α=15	80,8	118	-13,3	117
$\alpha=\infty$	80,9	117	-13,6	116

TABLE II
PERFORMANCE ONLINE-CENSORED EKF SSNN MODELS ON DATA SET B

Weight constraining	R^2_{perc}[%]	RMSE [s]	Bias [s]	RRE [s]
α=1	82,6	112	-12,6	111
α=5	82,6	112	-12,3	111
α=10	82,4	112	-12,4	112
α=15	82,3	113	-12,7	112
$\alpha=\infty$	82,3	113	-12,2	112

where $v_{k,m}$ depicts the mean of the speeds measured at the upstream and downstream detectors of section m during period k. The second baseline model is the day-to-day average travel time, which is given by

$$TT_k^{\text{hist}} = \frac{1}{N_D}\sum_{j=1}^{N_D} d_k^j. \tag{30}$$

As performance criteria, the percentage explained variance (the squared correlation coefficient $R^2 \times 100\%$) and the root mean square error (RMSE, in seconds) are used. The RMSE can be decomposed in a bias and a residual (random) error [root residual error (RRE)] as follows: $\text{RMSE}^2 = \text{Bias}^2 + \text{RRE}^2$, in which the former indicates structural errors and the latter indicates residual (random) errors. In the following, the performance indicators are listed with their formulas:

$$\text{RMSE} = \sqrt{\frac{1}{N_k}\sum_{k=1}^{N_k}(y_k - d_k)^2}$$

$$\text{Bias} = \overline{y} - \overline{d} = E[y_k] - E[d_k]$$

$$\text{RRE} = \sqrt{\frac{1}{N_k}\sum_{k=1}^{N_k}\left((y_k - \overline{y}) - (d_k - \overline{d})\right)^2}$$

$$R^2_{\text{perc}} = \frac{100 \times \text{Cov}(Y,D)^2}{\text{Var}(Y)\text{Var}(D)}, \quad Y=\{y_k\}_{k=1}^{N_k},\ D=\{d_k\}_{k=1}^{N_k}.$$

In this case, a *negative* bias implies that the model structurally *underestimates* the targets.

V. RESULTS

A. Predictive Performance

Tables I and II show the performance of the SSNN models trained with the online-delayed and online-censored EKF algorithms under varying weight constraints on data set B.

TABLE III
MEAN ENSEMBLE PERFORMANCE OF OFFLINE LMBR SSNN MODELS AND BASELINE MODELS ON DATA SET B

	R^2_{perc}[%]	RMSE [s]	Bias [s]	RRE [s]
SSNN ensemble mean	86.5	100	-5.03	100
Instantaneous travel time	75.9	167	21.2	166
Hist. average travel time*	37.3	274	-81	261
Hist. average travel time**	39.0	249	~ 0	249

* taken over (test) dataset *B*; ** taken over (training) dataset *A*

Table III shows the mean ensemble performance of ten SSNN models trained with the offline LMBR algorithm along with the performance of an instantaneous travel time estimate and two historical averages over data sets A and B, respectively. The latter, of course, is, by definition, unbiased and provides us with the amount of variance (equal to RMSE^2) in the test set *itself*. The first conclusion from these results is that all SSNN models, as well as the instantaneous travel time estimate, explain a significant amount of the day-to-day variance in travel times and significantly improve over a historical day-to-day average travel time.

Second, the SSNN models that are trained online with the two EKF algorithms perform (slightly) worse than the ensemble of models that were trained offline with LMBR. This is no surprise since in the offline batch training variant, the model is allowed "to see" the entire training data set many times, whereas in both EKF variants, each input–output pattern is used only once. Nonetheless, the SSNN models that are trained with the online-delayed and online-censored EKF methods still outperform the naive predictor (instantaneous travel time) on all performance measures.

To illustrate the results in Tables I–III, Figs. 2 and 3 show the performance of both online EKF methods during two typical weekday afternoon peak periods in 2004 with mild and severe congestion, respectively. In both figures, the thick gray line denotes the "true" travel time curve, the black line denotes the mean ensemble prediction of the SSNN models trained with the offline LMBR algorithm, and the solid gray line and the dashed black line depict the predictions of the two online EKF-adapted SSNN models, both without weight constraining ($\alpha = \infty$). For readability, all data have been smoothed with a 5-min two-sided exponential filter. Clearly, in Figs. 2 and 3, the LMBR ensemble mean tracks the onset and the demise of congestion faster and more accurate than the two online EKF models.

A third observation from Table I (online-*delayed* EKF) and Table II (online-*censored* EKF) is that the latter outperforms the former—albeit slightly—for all degrees of weight constraining. These improvements pertain to all performance indicators and are in the order of a few percent. Figs. 2 and 3 substantiate the conclusion that, in general, the online-censored EKF algorithm performs slightly better than the online-delayed one. In both figures, the censored curve is smoother and tracks true travel time faster and more accurately than the online-delayed curve, albeit the differences are small.

A final and perhaps more counterintuitive result is that for neither of the two online EKF algorithms, weight constraining (regularization) appears to have a significant effect (positive or

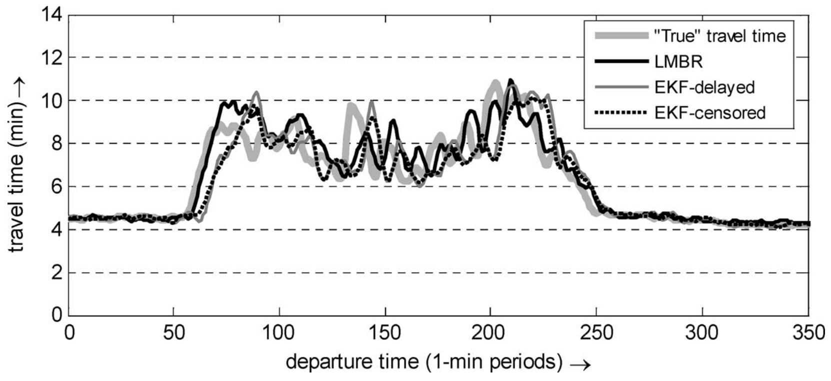

Fig. 2. Example performance of online-delayed and online-censored versus offline LMBR SSNN travel time prediction models under mild congestion. Note that for readability, all data (including targets) have been filtered with a 3-min two-sided exponential average.

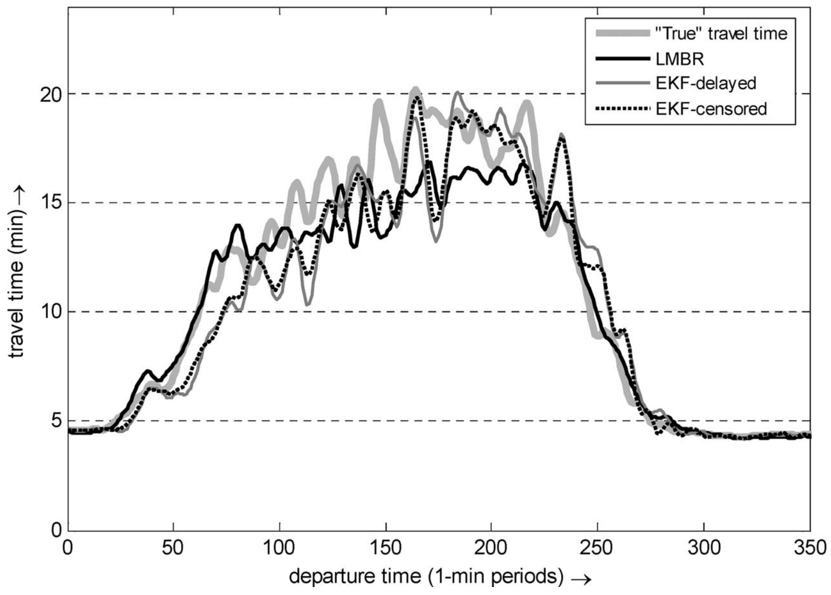

Fig. 3. Example performance of online-delayed and online-censored versus offline LMBR SSNN travel time prediction models under severe congestion. Note that for readability, all data (including targets) have been filtered with a 3-min two-sided exponential average.

negative) *on the performance*. In Section V-B, we will discuss this issue in more detail.

B. Discussion of Results and Possible Improvements

The results above give rise to a number of issues and questions. A selection of these is addressed below.

1) Offline-Trained Models Outperform Models That Are Adapted Online: This result is a direct and expected consequence of the travel time prediction problem, in which the targets (realized travel times) are delayed. This problem, in fact, increases with the magnitude of travel times and the speed with which travel time increases. The faster and higher the travel times go up, the longer it takes before an online learning method can track it and accordingly adapt the model weights [compare Fig. 2 (mild congestion) and Fig. 3 (severe congestion)], particularly in the early stages of learning. An *offline*-trained model will be more responsive to changes in the inputs under the premise that the underlying traffic patterns (input–output relationships) were present in the training data set. Of the two online algorithms, the online-censored EKF algorithm introduced above enables a slightly more responsive and smoother SSNN (travel time prediction) model than the online-delayed EKF learning method, albeit the difference is not very large.

2) Some Notes on the Usefulness of Weight Constraining in an Online Learning Context: Tables I and II show that weight constraining does not affect the SSNN performance more than marginally, neither when applied to the online-delayed algorithm nor when applied to the online-censored EKF algorithm. In both cases, none of the performance indicators substantially differ under different degrees of weight constraining. One conclusion is that weight constraining in an online learning context is not necessary or is at least not very beneficial. A second conclusion is that there exist many weight configurations, which lead to an equally well-performing SSNN model.

To illustrate, Table IV shows the maximum weight of the five SSNN models after being adapted with the online-censored EKF algorithm. As expected, the maximum weight increases in the case where weight constraining is relaxed. Clearly, each of these nearly equally well-performing models has a very different weight setting.

TABLE IV
MAXIMUM WEIGHT IN SSNN MODELS AFTER ONLINE-DELAYED EKF ADAPTATION FOR DIFFERENT DEGREES OF WEIGHT CONSTRAINING

α=1	α=5	α=10	α=15	$\alpha=\infty$
0,90	2,22	3,36	4,67	6,95

More tentatively, one might also conclude that the weight constraining procedure succeeds in preventing the SSNN model from overfitting the data without compromising its predictive performance. To appreciate this—as said, tentative—claim, one must realize that weight constraining effectively makes a parameterized model *less sensitive* (less responsive) to the prediction errors with which its parameters are adapted. In an online learning context, one might subsequently expect that no weight constraining (implying maximum sensitivity to output errors) would lead to better performance. In our experiment, it appeared, however, that weight constraining does not deteriorate performance. On the basis thereof, weight constraining (regularization) is useful since it keeps the model as simple (and smooth) as possible without compromising predictive performance. This is particularly relevant to ensure a degree of robustness in response to missing data (incidents and accidents). Under such conditions, a very sensitive model (with unconstrained weights) might produce erratic predictions. Note that more extensive research is required to further investigate this issue.

3) Possible Improvements and Extensions to the Online-Censored EKF Learning Algorithm: A possible improvement to the online-censored EKF algorithm could lie in making the process noise [see (16)] *adaptive* (dependent on, for example, the output error), similar to what is proposed for the measurement noise [see (21)]. This may be beneficial since in the EKF algorithm described above, the combination of process and measurement noise parameters governs the speed and the magnitude with which the algorithm tracks the target curve (true travel time) and, hence, adapts the model weights. In [27], a number of suggestions to this end are given. Similarly, the parameter α of the weight constraining function itself could be made adaptive and simultaneously updated with the weights, analogous to what happens in the LMBR algorithm. Practically, a second EKF then runs parallel to the one governing the weight updates.

4) Online-Censored Algorithm in a Wider Perspective: Last, the censored algorithm presented here may offer improvements in online learning in a more general context. Recall that the EKF algorithm is classically used for state estimation, that is, for adapting unobserved state variables $\mathbf{x}_t$ on the basis of observable errors on output variables $\mathbf{y}_t$, where both constitute the following state-space mapping:

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t, \mathbf{r}_t) \quad \text{(state dynamics)} \qquad (31)$$
$$\mathbf{y}_t = g(\mathbf{x}_t, \varepsilon_t) \quad \text{(output equation).} \qquad (32)$$

Online parameter estimation (online learning) can be considered to be a special case of state estimation, in which the state dynamics (31) are modeled as a random walk [see (10) and (11)]. In cases where the output equation relates travel time $(\mathbf{y}_t)$ to the unobserved state [whatever this may be: vehicle densities, road capacities, or, for example, origin–destination (OD) flows], the censored algorithm presented above is applicable, also when the state dynamics are governed by other more complex equations than a random walk, as was the case in this paper. Examples of such more involved state dynamics include macroscopic traffic flow models (e.g., [31]) or queuing models. A typical application then would be traffic state estimation (or, e.g., OD matrix estimation) on the basis of different data sources (data fusion) such as travel time measurements (with cameras or Global System for Mobile Communications/Global Positioning System equipped probe vehicles) and inductive loop data (spot speeds and flows). The censored algorithm in that case would allow the modeler to update the state on the basis of travel times as frequently as on the basis of loop data, which most probably leads to a smoother and more accurate state estimate. Further research in this area is, however, required to substantiate this claim.

VI. CONCLUSION

Since travel times are available and can be measured for *realized* trips only, adapting the parameters of a data-driven travel time prediction model, such as the SSNN, *online* is particularly challenging. Travel time prediction is not a one-step-ahead prediction problem and cannot be solved by standard incremental learning algorithms. To our knowledge, this paper is the first to present online learning algorithms that adequately deal with the inherently delayed travel time prediction problem.

It was found that a new algorithm, i.e., the so-called online-*censored* EKF algorithm, performs slightly better than the online-*delayed* EKF algorithm on a large data set of actual data from a heavily congested freeway route in the Netherlands. Both methods outperform a naive online method (instantaneous travel time), and both outperform the historical average by far. Although SSNN models that are trained with both online EKF algorithms perform slightly worse than an *offline*-trained SSNN model, the gain of such online-learning algorithms is large. It alleviates the modeler from preparing and executing offline training procedures, and more importantly, it creates a robust and adaptive travel time prediction model, which is able to adapt to changes in either the underlying traffic processes or the monitoring system collecting the models' input and output data.

To improve the generalization, a weight constraining (regularization) method was introduced. From the results, it appeared that this procedure does not affect the results more than marginally. We argue, however, that weight constraining in an online context may, nonetheless, be beneficial in that it leads to smoother models that still perform well.

Last, we argued that the online-censored algorithm may yield improvements not only in online parameter fitting but also, more generally, in online traffic *state estimation* and data fusion in cases in which delayed detector data (such as travel time) are used.

ACKNOWLEDGMENT

The author would like to thank the anonymous reviewers for their detailed comments and remarks, which have largely improved the quality and clarity of this article.

REFERENCES

[1] J. W. C. van Lint, S. P. Hoogendoorn, and H. J. Van Zuylen, "Accurate travel time prediction with state-space neural networks under missing data," *Transp. Res. Part C, Emerg. Technol.*, vol. 13, no. 5/6, pp. 347–369, Oct.–Dec. 2005.

[2] J. W. C. van Lint, "Reliable travel time prediction for freeways," Ph.D. dissertation, TRAIL Res. School, Delft Univ. Technol., Delft, The Netherlands, 2004. p. 302.

[3] J. W. C. van Lint, "A reliable real-time framework for short-term freeway travel time prediction," *J. Transp. ASCE*, vol. 132, no. 12, pp. 921–932, Dec. 2006.

[4] M. Ben-Akiva, M. Bierlaire, D. Burton, H. N. Koutsopoulos, and R. Mishalani, "Network state estimation and prediction for real-time transp. management applications," presented at the Transp. Res. Board Annu. Meeting, Washington DC, 2002. CD-ROM.

[5] H. S. Mahmassani, *DynaSMART-X Home*, vol. 2004. Austin, TX: Univ. Texas, 2004.

[6] S. A. Smulders, A. Messmer, and W. J. J. Knibbe, "Real-time application of METANET in traffic management centres," presented at the 6th World Congr. Intell. Transp. Syst. (ITS), Toronto, ON, Canada, 1999.

[7] X. Zhang and J. A. Rice, "Short-term travel time prediction," *Transp. Res. Part C, Emerg. Technol.*, vol. 11, no. 3/4, pp. 187–210, Jun.–Aug. 2003.

[8] J. Rice and E. Van Zwet, "A simple and effective method for predicting travel times on freeways," in *Proc. IEEE Conf. Intell. Transp. Syst.*, Oakland, CA, 2001, pp. 227–232.

[9] J. A. C. Van Toorenburg, "ASTRIVAL Functionele specificatie Algoritme," in Rijtijd en Filelengteschatter voor Meetvak (in Dutch). Rotterdam, The Netherlands: AVV Transp. Res. Centre, Ministry Transp., Public Works Water Manage., Nov. 1998.

[10] C.-H. Wu, C.-C. Wei, D.-C. Su, M.-H. Chan, and J.-M. Ho, "Travel time prediction with support vector regression," in *Proc. IEEE Conf. Intell. Transp. Syst.*, Shanghai, China, 2003, pp. 1438–1442.

[11] H. Sun, H. X. Liu, H. Xiao, R. R. He, and B. Ran, "Short-term traffic forecasting using the local linear regression model," presented at the Transp. Res. Board Annu. Meeting, Washington DC, 2003. CD-ROM.

[12] H. M. Al-Deek, M. P. D'Angelo, and M. C. Wang, "Travel time prediction with non-linear time series," in *Proc. 5th Int. Conf. Appl. Adv. Technol. Transp.*, Reston, VA, 1998, pp. 317–324.

[13] S. I. J. Chien and C. M. Kuchipudi, "Dynamic travel time prediction with real-time and historic data," *J. Transp. Eng.-ASCE*, vol. 129, no. 6, pp. 608–616, Nov./Dec. 2003.

[14] A. Stathopoulos and M. G. Karlaftis, "A multivariate state space approach for urban traffic flow modeling and prediction," *Transp. Res. Part C, Emerg. Technol.*, vol. 11, no. 2, pp. 121–135, Apr. 2003.

[15] D. Park, L. Rilett, and G. Han, "Spectral basis neural networks for real-time travel time forecasting," *J. Transp. Eng.*, vol. 125, no. 6, pp. 515–523, Nov./Dec. 1999.

[16] L. R. Rilett and D. Park, "Direct forecasting of freeway corridor travel times using spectral basis neural networks," *Transp. Res. Rec.*, vol. 1752, pp. 140–147, 2001.

[17] J. W. C. van Lint, S. P. Hoogendoorn, and H. J. Van Zuylen, "Freeway travel time prediction with state-space neural networks—Modeling state-space dynamics with recurrent neural networks," *Transp. Res. Rec.*, vol. 1811, pp. 30–39, 2002.

[18] J. W. C. van Lint and M. Schreuder, "Travel time prediction for VMS panels—Results and lessons learnt from a large-scale evaluation study in the Netherlands," presented at the Transp. Res. Board Annu. Meeting, Washington DC, 2006. CD-ROM.

[19] D. J. C. MacKay, "Probable networks and plausible predictions: A review of practical Bayesian methods for supervised neural networks," *Network: Comput. Neural Syst.*, vol. 6, no. 3, pp. 469–505, Aug. 1995.

[20] M. T. Hagan and M. B. Menhaj, "Training feedforward networks with the Marquardt algorithm," *IEEE Trans. Neural Netw.*, vol. 5, no. 6, pp. 989–993, Nov. 1994.

[21] C. M. Bishop, *Neural Networks for Pattern Recognition*. London, U. K.: Oxford Univ. Press, 1995.

[22] D. Rumelhart, G. Hinton, and R. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing*. Cambridge, MA: MIT Press, 1986, ch. 8.

[23] R. J. Williams and D. Zipser, "Gradient-based learning algorithms for recurrent networks and their computational complexity," in *Back-Propagation: Theory, Architectures and Applications*, Y. Chauvin and D. E. Rumelhart, Eds. Hillsdale, NJ: Erlbaum, 1995, ch. 13, pp. 433–486.

[24] H. H. Thodberg, "A review of Bayesian neural networks with an application to near infrared spectroscopy," *IEEE Trans. Neural Netw.*, vol. 7, no. 1, pp. 56–72, Jan. 1996.

[25] D. N. Politis, J. P. Romano, and M. Wolf, "On the asymptotic theory of subsampling," *Stat. Sin.*, vol. 11, no. 4, pp. 1105–1124, 2001.

[26] D. Nguyen and B. Widrow, "Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights," in *Proc. Int. Joint Conf. Neural Netw.*, 1990, pp. 21–26.

[27] S. Haykin, *Kalman Filtering and Neural Networks*. New York: Wiley, 2001.

[28] T. H. J. Muller, M. Miska, and H. J. Van Zuylen, "Monitoring traffic under congestion," presented at the Transp. Res. Board Annu. Meeting, Washington DC, 2005. CD-ROM.

[29] J. W. C. van Lint and N. J. Van der Zijpp, "Improving a travel time estimation algorithm by using dual loop detectors," *Transp. Res. Rec.*, vol. 1855, pp. 41–48, 2003.

[30] R. Li, G. Rose, and M. Sarvi, "Evaluation of speed-based travel time estimation models," *J. Transp. Eng.*, vol. 132, no. 7, pp. 540–547, Jul. 2006.

[31] Y. Wang and M. Papageorgiou, "Real-time freeway traffic state estimation based on extended Kalman filter: A general approach," *Transp. Res. Part B*, vol. 39, no. 2, pp. 141–167, Feb. 2005.

J. W. C. (Hans) van Lint received the M.Sc. degree in civil engineering and informatics and the Ph.D. degree in "reliable freeway travel time prediction" in 1997 and 2004, respectively, both from Delft University of Technology (DUT), Delft, The Netherlands.

After working as a Software Engineer and Consultant, he joined the Municipality of Rotterdam as the Project Manager of the regional traffic information center while pursuing the Ph.D. degree part time at DUT. He is currently an Associate Professor with the Department of Transport and Planning, Faculty of Civil Engineering and Geosciences, DUT. His main research interests include traffic flow theory, travel time estimation and prediction, data fusion and Bayesian optimization, traffic state estimation and prediction, and the development and application of artificial intelligence in transportation.