Transportation Research Part C 91 (2018) 176–191

Contents lists available at ScienceDirect

Transportation Research Part C

journal homepage: www.elsevier.com/locate/trc

Vehicle classification from low-frequency GPS data with recurrent neural networks

Matteo Simoncini, Leonardo Taccari*, Francesco Sambo, Luca Bravi, Samuele Salti, Alessandro Lori

Verizon Connect Research, Via Giovanni Paisiello, 20, Florence, Italy

ARTICLE INFO

Keywords:
Vehicle classification
GPS
Sequence classification
Recurrent neural networks

ABSTRACT

The categorization of the type of vehicles on a road network is typically achieved using external sensors, like weight sensors, or from images captured by surveillance cameras. In this paper, we leverage the nowadays widespread adoption of Global Positioning System (GPS) trackers and investigate the use of sequences of GPS points to recognize the type of vehicle producing them (namely, small-duty, medium-duty and heavy-duty vehicles). The few works which already exploited GPS data for vehicle classification rely on hand-crafted features and traditional machine learning algorithms like Support Vector Machines. In this work, we study how performance can be improved by deploying deep learning methods, which are recently achieving state of the art results in the classification of signals from various domains. In particular, we propose an approach based on Long Short-Term Memory (LSTM) recurrent neural networks that are able to learn effective hierarchical and stateful representations for temporal sequences. We provide several insights on what the network learns when trained with GPS data and contextual information, and report experiments on a very large dataset of GPS tracks, where we show how the proposed model significantly improves upon state-of-the-art results.

1. Introduction

Inferring the type of vehicles in a road network, a problem typically referred to as *vehicle classification* in the literature, is a fundamental task in several applications, such as surveillance systems, traffic management, emission control, and urban planning (Gupte et al., 2002). Depending on the application, several definitions of vehicle categories can be considered, ranging from very broad ones (*e.g.*, motorcycles, cars, buses), up to fine-grained categorization (*e.g.*, make and model). A standard reference for the definition of classes of vehicles is the 13-category vehicle taxonomy proposed by the Federal HighWay Administration (FHWA) of the United States, which is based on the vehicle weight, length, axles number and axles distances (Wyman et al., 1985). Although the rules have been revised over the years by companies and agencies (Hallenbeck et al., 2014), the FHWA 13 vehicle categories are still used as a standard in many applications.

Traditional approaches for vehicle classification use different types of hardware sensors and classification methods, depending on the application context and the granularity of the desired classification. When physical components can be installed along a road, techniques using fixed-location sensors, such as pneumatic tubes, inductive loop detectors, piezoelectric sensors and Weigh-in-motion

* Corresponding author.
E-mail addresses: matteo.simoncini@verizonconnect.com (M. Simoncini), leonardo.taccari@verizonconnect.com (L. Taccari), francesco.sambo@verizonconnect.com (F. Sambo), luca.bravi@verizonconnect.com (L. Bravi), samuele.salti@verizonconnect.com (S. Salti), alessandro.lori@verizonconnect.com (A. Lori).

https://doi.org/10.1016/j.trc.2018.03.024
Received 12 December 2017; Received in revised form 16 March 2018; Accepted 26 March 2018
Available online 13 April 2018
0968-090X/ © 2018 Elsevier Ltd. All rights reserved.

(WIM) systems can be adopted (Benekohal and Girianna, 2003; Harlow and Peng, 2001; Sun et al., 2013). Some of these approaches can even provide the full 13-class classification with high accuracy, in exchange for a high installation cost. For this reason, some less intrusive and expensive classification devices, such as infrared sensors, acoustic sensors and radar sensors, have been introduced at the cost of a lower accuracy and/or coarser classification. Nevertheless, the accuracy of these devices may be affected by traffic and environmental conditions, or human error during installation (Benekohal and Girianna, 2003; Sun et al., 2013).

A different set of techniques involve performing classification from still images or videos, for instance obtained from surveillance cameras (Kafai and Bhanu, 2012; Ma and Grimson, 2005). The number and kinds of vehicle categories varies, depending also on the resolution of the cameras: for instance, vans, taxis, and passenger cars are considered in Ma and Grimson (2005), while sedans, pickups, and vans in Kafai and Bhanu (2012). For image recognition tasks, a huge performance improvement in recent years has been obtained thanks to deep learning approaches based on Convolutional Neural Networks (CNN), such as those presented in Zhou et al. (2016), where the authors classify vehicles into passengers and other vehicles from noisy and/or dark images, and Gebru et al. (2017), where the authors focus on vehicles found in Google Street View images across the United States. In the latter work, a CNN-based classification algorithm is trained on a very large and diverse dataset of almost 350,000 vehicle images and tested on 27,865 examples, obtaining an average class accuracy of 0.67 for the challenging problem of discriminating 11 vehicle types.

For a thorough discussion on advantages and drawbacks of the existing approaches for vehicle classification involving different hardware and techniques, as well as their expected performance, we refer the interested reader to Benekohal and Girianna (2003) and Sun et al. (2013). To give an idea of the performance that can be obtained with different sensors, the authors of Sun et al. (2013) claim that state-of-the-art approaches typically allow an accuracy between 0.80 and 0.95 for 3–5 vehicle classes. We must note, though, that most of the results in the literature are obtained from experiments on small datasets (sometimes as small as a few dozen examples).

Nowadays, the rise of the Internet of Things and connected cars is enabling new ways to sense a vehicle, *e.g.* through the Global Positioning System (GPS) signals it transmits. GPS data are typically produced by either general-purpose mobile devices (*e.g.*, smartphones) or dedicated GPS tracker devices, traditionally installed on commercial or public transport vehicles (*e.g.*, delivery fleets, taxis, ambulances, buses) (Leduc, 2008), but recently also on personal vehicles.[1] When generated by mobile devices, GPS signals are usually used for navigation or geo-localization purposes, hence they exhibit high sampling rates (of the order of one GPS sample per second). The frequency and quality of the data from sensors in modern smartphones allow for their use in several emerging applications, such as driving behavior analysis (Eren et al., 2012; Paefgen et al., 2012; Vlahogianni and Barmpounakis, 2017). When GPS trackers are used, instead, GPS signals are typically used for remote fleet management, vehicle tracking or anti-theft systems, and lower frequency sampling (of the order of one sample per minute) is often sufficient. The use of low-frequency GPS data allows for the reduction of operational costs due to bandwidth, storage space, and computational power and is therefore very common in industrial applications and commercial fleet management solutions, as well as for insurance companies (Leduc, 2008). Clearly, the technical and economical advantages come at the cost of accuracy: lower frequency sampling means that information on instantaneous speeds are scarce and that it is harder to infer the true path of a vehicle between two reported positions.

In the context of fleet management software, a strong motivation for tackling the problem of vehicle classification lies in extracting information on the connected vehicles that could not be obtained directly from the GPS devices, and that can be used to improve the user experience of the fleet managers. However, GPS data are still relatively unexplored as clues to tackle this problem. One of the few works that explore vehicle classification from GPS data is Sun et al. (2013), that considers a two-class classification problem, distinguishing between passenger cars and delivery trucks, on a small dataset comprising 52 GPS tracks of passenger cars and 84 GPS tracks of trucks. GPS data used in the paper have a sample rate of 3 s, which is relatively high. The proposed approach entails the use of a Support Vector Machine (SVM) classifier, and the authors conclude that acceleration- and deceleration-based features have a greater predictive power than speed-based features. Closely related to vehicle classification is the problem of transportation mode detection (Bolbol et al., 2012; Gonzalez et al., 2008; Xiao et al., 2015; Zheng et al., 2008), though it is worth remarking that distinguishing between travel modes such as *walk*, *bus*, *train* and *car* is in general easier than the finer-grain detection of vehicle classes, as the former can be solved by using highly discriminative features (such as speed, number of heading changes, number of stops, or distance traveled) that may not be equally effective for vehicle classifications. Very few works in this area, an exception being (Bolbol et al., 2012), consider low-frequency data.

In Simoncini et al. (2016), we addressed for the first time the problem of vehicle classification from low-frequency GPS data, provided by devices installed in commercial fleets for vehicle-tracking purposes. A binary SVM classifier was shown to achieve state-of-the-art performance in distinguishing between light-duty vehicles (*i.e.*, cars, SUVs, vans and light duty pickups, that correspond to classes 2–3 of the FHWA categorization) and larger size vehicles (*i.e.*, heavy duty pickups, small trucks, trucks and big trucks, classes 5–12 of the FHWA categorization). Our method started from a set of low-level features for each point of a GPS track, and then aggregated them at the track level, relying on a wide range of aggregation functions such as mean, median, standard deviation and the like, to obtain over 130 high-level features. The most predictive combination of features was then identified based on a recursive feature elimination procedure (Guyon and Elisseeff, 2003).

In recent years, machine learning approaches based on deep neural networks, commonly known as *deep learning* (Goodfellow et al., 2016), have gained a renewed attention thanks to their success in tasks that involve complex input data like images (He et al., 2015) or sequential or temporal data, such as speech recognition (Deng et al., 2013) and automated machine translation (Cho et al., 2014; Wu et al., 2016), becoming the *de facto* standard technique. What typically makes deep learning effective is the presence of

[1] See for instance the recent Verizon Hum https://www.hum.com.

several stacked layers of non-linear processing units, which allow the deep network to learn rich hierarchical representations of the initial raw data in an automated way. Typically, the first layers of a neural network learn to extract some basic and low-level features; proceeding deeper in the network, the layers learn increasingly complex representations, based on the output of the previous layers; finally, the last layers map the encoded state to the desired output, be it a class label, or more complex objects such as images, audio or text (Goodfellow et al., 2016). Recurrent Neural Networks (RNNs) (Bengio et al., 1994; Graves, 2012; Rumelhart et al., 1986) are especially suited for tasks that involve temporal sequences of variable length, such as machine translation, handwriting recognition (Graves and Schmidhuber, 2009) or text classification (Graves, 2012). Alternative approaches are based on the application of convolutional networks (Zhang et al., 2015): although originally developed for image-related data, convolution can be applied across time to sequential data, in order to exploit local correlation of features along the temporal dimension.

Very few works in the literature apply deep learning to GPS data. In Dong et al. (2016), the aim is to characterize driving style, thus associating the correct driver to each GPS sequence. The proposed approach involves the use of stacked IRNN layers (Le et al., 2015) working on sequences of matrices of statistical features obtained from the GPS sequences. Relatively high-frequency GPS data, with a sampling rate of 1 Hz, is used: the authors claim that performance would significantly degrade with signals sampled at less than 1 sample every 10 s. They also found that aggregating the input features with statistical functions over temporal windows is instrumental in obtaining better results. In Song et al. (2016), the authors perform multi-task learning via deep recurrent neural networks to predict the next position and travel mode in a sequence of GPS data points. They show that using deep learning allows for a significant improvement over traditional "shallow" models, such as Gaussian processes and Hidden Markov Models.

In this paper, we assess the effectiveness of the deep learning paradigm for the problem of vehicle classification from low-frequency GPS data. We consider the same type of raw GPS data used in our previous work (Simoncini et al., 2016), but, instead of hand-engineering a large number of high-level features (as done in Simoncini et al., 2016), we let the network learn the most effective high-level representations from raw data. We test several combinations of feed-forward layers and recurrent layers, with Long Short-Term Memory cells (LSTM) (Hochreiter and Schmidhuber, 1997) as building blocks. A thorough experimental assessment is carried out to identify the best configuration and parameters for the deep network. Even the simplest network is shown to perform better than our previous state-of-the-art approach (Simoncini et al., 2016) for the binary problem of distinguishing between light-duty and heavy-duty vehicles. The best performing networks set a new state of the art both for the binary problem and for the finer grained classification of vehicles into three classes, that we introduce in this work. The behavior of the best network is then studied in detail, to gain some insights on what the network has learned and why. We use a new dataset, comprising about 1 million GPS tracks (*i.e.*, one order of magnitude larger than the one we introduced in Simoncini et al., 2016), for a total of more than 55 million GPS positions, about 56 million traveled kilometers, and a total duration of 1.3 million hours.

The remainder of the article is structured as follows: Section 2 presents the methodology we followed and describes the data we used and the deep architectures we tested; experimental results and insights on the learned models are presented in Section 3, while in Section 4 we draw our conclusions.

2. Methodology

2.1. Pre-processing of GPS data

Given a stream of GPS data collected from several heterogeneous GPS devices, we segment it to obtain a finite set of examples for each vehicle. Let us define a *GPS track* as a sequence of points $\{\mathbf{p}_i\}_{i=1}^{n} = \{\mathbf{p}_1,\ldots,\mathbf{p}_n\}$ collected between two stationary states, *i.e.* periods of time when the engine is off: $\mathbf{p}_1$ is the first point collected after the engine is turned on, and $\mathbf{p}_n$ is the last one, obtained just before the engine is turned off again (see Fig. 1 for an example). Our aim is to classify a GPS track as belonging to a certain class (vehicle type).

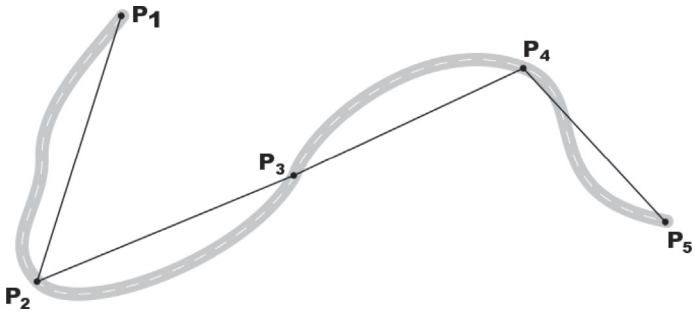
Each GPS point $\mathbf{p}_i$ collected from a device contains the position (latitude and longitude) and a set of additional data: the overall odometer distance of the device, an absolute timestamp, and the instantaneous speed. This set of measures are commonly collected by commercial GPS trackers. Note that we cannot assume uniform sampling rates within a track or across tracks, as data collected by heterogeneous GPS devices may have different sampling rates, and even in sequences collected by homogeneous devices the sampling rate can vary according to the vehicle speed or the occurrence of asynchronous triggers, *e.g.* harsh driving events.

We perform a few simple preprocessing transformations on the raw data to obtain the input data for the network. The reason is twofold: on the one hand, this is done to avoid some potential sources of bias in our dataset, presented below; on the other hand, reducing the amount of variation in the data can reduce both generalization error and the size of the model needed to fit the training set (Goodfellow et al., 2016). As also noted by Dong et al. (2016), GPS data in a deep learning context requires some preprocessing to reach satisfactory results. However, our pre-processing is simpler than the sliding window approach they used. We pre-process the GPS points as follows:

- for each GPS sample, we replace the raw (lat,lon) pair with the crow's flight distance from the previous to the current point, computed between two pairs of GPS coordinates $(\mathrm{lat}_1,\mathrm{lon}_1)$ and $(\mathrm{lat}_2,\mathrm{lon}_2)$ with the Haversine formula:

$$2R\cdot\arcsin\left(\sqrt{\sin^2\left(\frac{\mathrm{lat}_2-\mathrm{lat}_1}{2}\right)+\cos(\mathrm{lat}_1)\cdot\cos(\mathrm{lat}_2)\cdot\sin^2\left(\frac{\mathrm{lon}_2-\mathrm{lon}_1}{2}\right)}\right), \tag{1}$$

  where R is the average earth radius (6371 km) and all coordinates are expressed in radians. This is done to remove a first source of bias, since the trained model might generalize poorly to areas not contained in the training dataset if we were using the GPS



	Timestamp	Position	Speed	Odometer
$\mathbf{p}_1$	2016-11-01 16:02:10	(42.13222, -72.55265)	0.0 m/s	128171.02 m
$\mathbf{p}_2$	2016-11-01 16:03:40	(42.13737, -72.54227)	6.1 m/s	128977.99 m
$\mathbf{p}_3$	2016-11-01 16:05:10	(42.12700, -72.54660)	16.3 m/s	130830.12 m
$\mathbf{p}_4$	2016-11-01 16:06:40	(42.11551, -72.55024)	17.1 m/s	132789.96 m
$\mathbf{p}_5$	2016-11-01 16:06:55	(42.10739, -72.54664)	0.0 m/s	133792.06 m

Fig. 1. Example of a GPS track composed by five GPS samples. The sampling frequency varies depending on vehicle speed and it is affected by hardware delay, thus being not constant during the sequence. Note that the reported speed is instantaneous, so it may be affected by traffic or street conditions.

coordinates directly;

- the raw odometer value obtained from the GPS messages contains the traveled distance of the vehicle over its entire life. This value carries information also on the age and the historical usage of the vehicle, poorly generalizing, *e.g.*, to new or underutilized vehicles, and introducing potential bias, *e.g.*, if certain types of vehicles are typically older/over-utilized in our dataset. We avoid this potential issue keeping only the relevant information, that is, the difference of consecutive odometer readings, which represents the distance traveled by the vehicle from the previous GPS position to the current one;
- to simplify the format of data in input to the model, we do not use the timestamp directly but the time elapsed between two consecutive GPS messages, which is not constant in our data;
- we complement the instantaneous speed collected from the GPS device with the *interval speed*, computed as the traveled distance from the previous GPS point over time, since it represents the average speed in the interval and conveys different and less noisy information. Similarly, since the instantaneous acceleration is not available in our raw data, we compute it as both finite differences of instant speed and interval speed. An example of the two types of speed and acceleration is given in Fig. 2. Therefore, starting from the raw data sequences of odometer readings and speeds (denoted as $\{o_i\}_{i=1}^n$ and $\{v_i\}_{i=1}^n$, respectively), we introduce the sequences $\{\widetilde{v}_i\}_{i=1}^n$,$\{a_i\}_{i=1}^n$ and $\{\widetilde{a}_i\}_{i=1}^n$ as the sequences of finite differences approximations to derivatives of distance, instantaneous speeds, and average speeds over time, *i.e.*,

$$\widetilde{v}_i = \frac{o_i - o_{i-1}}{t_i - t_{i-1}}, a_i = \frac{v_i - v_{i-1}}{t_i - t_{i-1}}, \widetilde{a}_i = \frac{\widetilde{v}_i - \widetilde{v}_{i-1}}{t_i - t_{i-1}}. \tag{2}$$

Finally, we add the *Road Type* as a contextual feature for our GPS data. We use the one-hot encoding into a vector of length 7 to present the road type of each GPS location, among `MOTORWAY`, `HIGHWAY`, `TRUNK_ROAD`, `COUNTRY_ROAD`, `CITY_ROAD`, `RESIDENTIAL_ROAD` and `SPECIAL` (here reported in descending order of posted speed). The road type is obtained from the GPS location via a reverse geocoding service.[2] We already showed its effectiveness for vehicle classification in our previous work (Simoncini et al., 2016), where road type ranked among the top features. We do not consider additional contextual data here, such as traffic or weather information, although, if available, that information may be beneficial. As an example, cruising speed is likely to be highly correlated with the vehicle type, but also with traffic congestion. Any point-wise property of a GPS track can be easily added in our framework, as an additional input dimension for the neural network.

To summarize, we obtain a set of 14 inputs for the network from each point in a GPS track:

- *Distance*: odometer distance from the previous point (difference between consecutive odometer readings)
- *Haversine Distance*: distance from the previous point, computed as in Eq. (1)
- *Time*: time (in seconds) from the previous point (difference between consecutive timestamps)
- *Speed* and *Interval Speed*: instantaneous speed (transmitted by the GPS device), and average speed in the last interval, computed as

[2] PTV xServer (http://xserver.ptvgroup.com/). Analogous results can be obtained with free services such as OpenStreetMap (http://www.openstreetmap.org/).

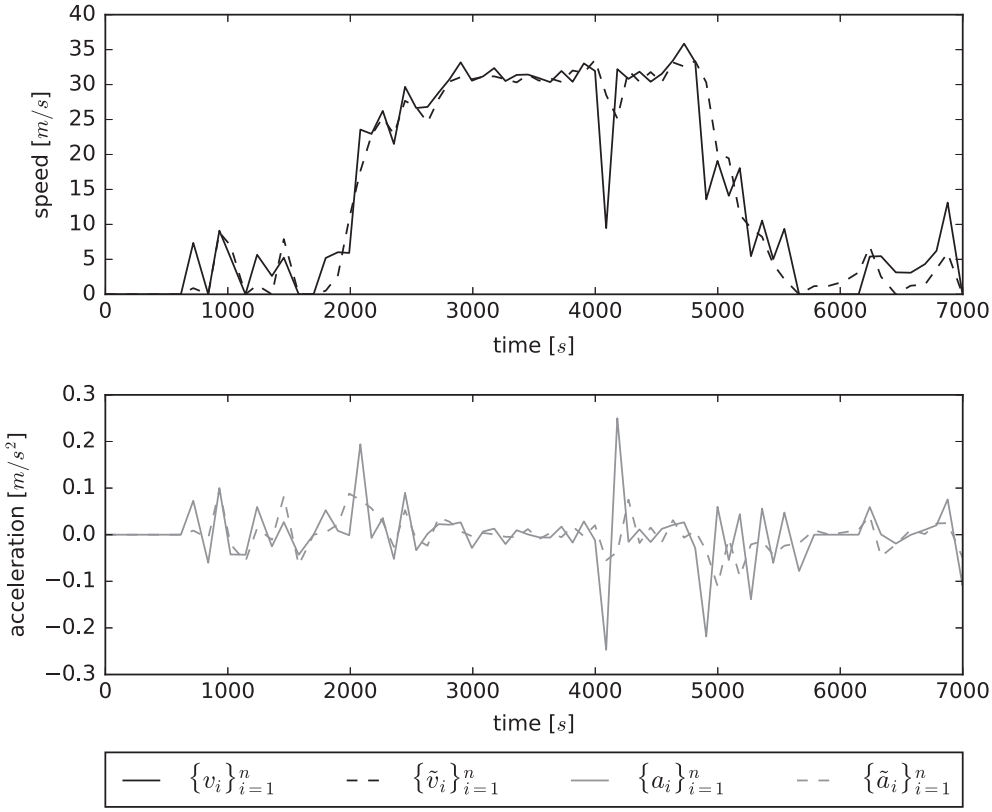


Fig. 2. Example of speed, interval speed, acceleration and interval acceleration sequences.

in Eq. (2)

- *Acceleration* and *Interval Acceleration*: derived accelerations, computed as in Eq. (2)
- *Road Type*: one-hot vector with 7 entries.

We do not use any *global* feature, computed as statistical aggregations or spectral transformations of the point-wise data. In particular, we do not include the total distance which, according to our analysis in Simoncini et al. (2016), was the most effective feature for vehicle classification, nor any other statistical aggregated features, like the median speed, or its standard deviation, that also proved very effective. We do not even include features computed over a sliding-window of the sequence, as commonly done in sequence classification problems (Dong et al., 2016; Kadous, 2002). By keeping the data as point-wise as possible, we want to fully exploit the ability of LSTM-based recurrent neural networks to learn how to extract, encode, and combine the sequential point-wise GPS information, deriving higher-level features automatically.

2.2. Sequence classification with Recurrent Neural Networks

Recurrent neural networks (RNN) are a popular neural network architecture that has been proven effective in many tasks on temporal sequences, such as sequence labeling and machine translation. Generally speaking, an Artificial Neural Network consists of several layers of processing units, called *neurons*, where the output of each neuron in a layer is connected to the input of one or more neurons of another layer. A single neuron implements a function that maps a vector of numeric inputs $x = [x_1,...,x_m]$ to a single numeric output, in the following form:

$$f(x) = \sigma\left(b + \sum_{i=1}^{m} w_i \cdot x_i\right), \tag{3}$$

where $w = [w_1,...w_m]$ is a numeric vector of *weights*, b is a *bias* term and $\sigma(\cdot)$ is a (usually non-linear) *activation* function, such as the sigmoid function, the hyperbolic tangent or a Rectified Linear Unit (ReLU, Nair and Hinton, 2010). Data, as a set of vectors of m real numbers, is fed as input to all neurons in the first layer of the network and the classifier output is determined by the output of the last layer: depending on the specific learning task, the activation function of the last layer is typically the sigmoid function for binary classification, the *softmax* function for multiclass classification, and the identity function for regression (Goodfellow et al., 2016; Hastie et al., 2009). The first layer is usually defined as the *input layer*, the last as the *output layer* and all the others as *hidden layers*.

Given a *training* dataset, *i.e.* a set of $m-$dimensional vectors of examples and one *label* for each example, the weights of the neurons in the network can be adjusted so that the network is able to assign the correct label to the examples in the dataset, a procedure known as training. Neural network training algorithms usually iterate a sequence of steps until convergence: a set of data is fed to the network, which is used to predict a label for each example in the set; the predicted label is compared to the real label through an error function; for each network weight a *gradient* of the error, *i.e.* the partial derivative of the error relative to the specific weight, is computed; each weight is updated according to the following equation:

$$w_i^{t+1} = w_i^t - \alpha \frac{\partial e^t}{\partial w_i}, \tag{4}$$

where w_i^t is the specific weight at iteration t,e^t is the error at iteration t and α is a (possibly time-varying) multiplicative weight known as *learning rate*. For memory and efficiency reasons, it is often convenient to partition the training data in random sub-samples, known as *minibatches*, and iteratively adjust the network weights according to the gradient in each minibatch, a procedure known as Stochastic Gradient Descent (SGD). See Bottou et al. (2016) for more details on optimization methods for machine learning. Several recent improvements have been developed in making adaptive, time varying learning rates (Duchi et al., 2011; Zeiler, 2012): one of the most successful is the Adam method (Kingma and Ba, 2015), which computes individual adaptive learning rates for different weights from estimates of the first and second moments of the gradients.

In *feed-forward networks* the neurons and their connections form a directed acyclic graph. *Recurrent neural networks*, on the other hand, are a subclass of neural network architectures which contain one or more recurring element, *i.e.* links connecting a layer to itself, or more generally connecting downstream layers to upstream layers. Given an input sequence $\mathbf{x} = \{x^1,...,x^T\}$, consisting of T subsequent observations of the feature vector x, the simplest way to define a recurrent hidden layer is to consider a function of the form

$$h^t = \sigma(W \cdot [x^t, h^{t-1}] + b) \tag{5}$$

where x is the vector of inputs, h is the vector of output, W and b are the matrix of weights and the vector of biases, respectively, $[\cdot,\cdot]$ is the vector concatenation operator and σ is the activation function.

This simple RNN architecture has been shown to be rather challenging to train, even with a single recurrent hidden layer: for example, it is affected by the problem of vanishing and exploding gradients (Bengio et al., 1994; Pascanu et al., 2013b), *i.e.* the norm of the gradient might grow exponentially or shrink to zero, preventing any form of learning. To tackle this problem, several solutions have been proposed, such as gradient clipping (Pascanu et al., 2013b), identity initialization with Rectified Linear Unit (ReLU) activations (so called IRNN) (Le et al., 2015), and more complex cell architectures, such as long-short term memory (LSTM) cells (Gers et al., 2000; Hochreiter and Schmidhuber, 1997). Compared to the RNN described by Eq. (5), the main idea behind LSTM cells, as summarized in Eqs. (6)–(11), is to propagate, in addition to the output state h^t, also an internal cell state c^t and a number of gating functions i^t,f^t,o^t, whose weights are trained as well, that allow the network to decide whether and how much to update its internal cell state based on the current input (Eq. 9), and how much of its internal state to actually transfer on the output state (Eq. (11)),

$$f^t = \sigma(W_f [x^t, h^{t-1}] + b_f) \tag{6}$$

$$i^t = \sigma(W_i [x^t, h^{t-1}] + b_i) \tag{7}$$

$$\tilde{c}^t = \tanh(W_c [x^t, h^{t-1}] + b_c) \tag{8}$$

$$c^t = f^t c^{t-1} + i^t \tilde{c}^t \tag{9}$$

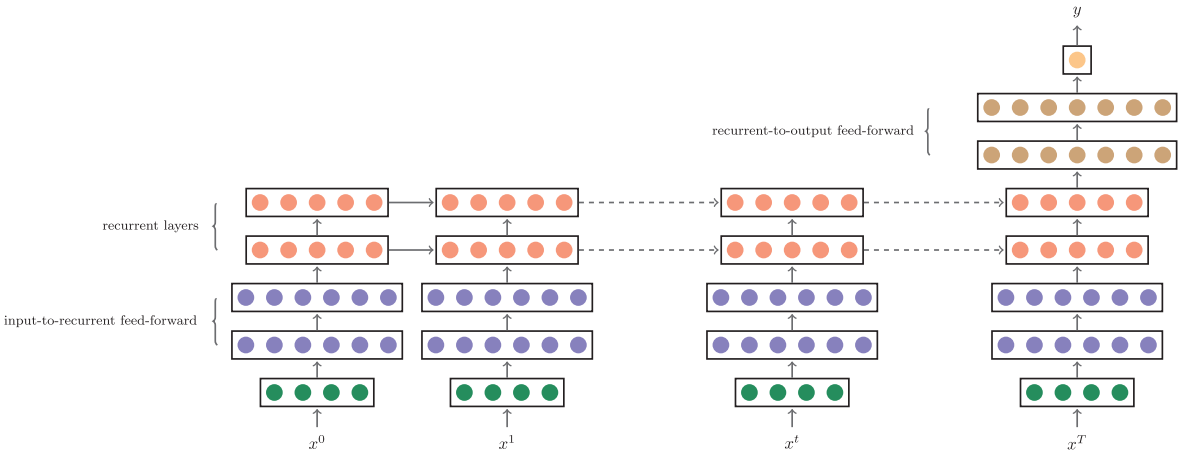
$$o^t = \sigma(W_o [x^t, h^{t-1}] + b_o) \tag{10}$$

$$h^t = o^t \tanh(c^t). \tag{11}$$

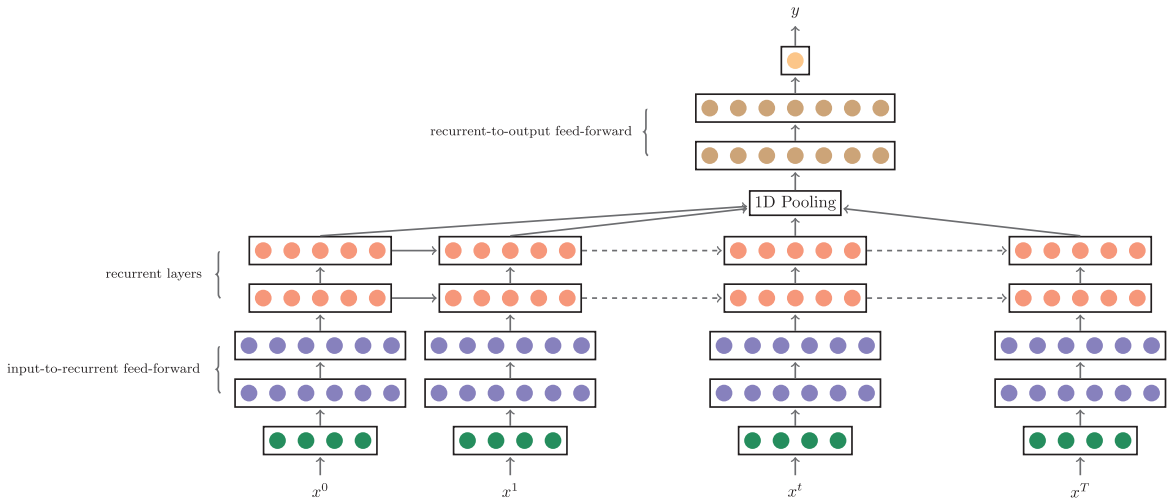
Although several alternatives have been proposed in the last few years, LSTM are still commonly considered as the state-of-the-art for tasks involving sequential data (Goodfellow et al., 2016).

2.3. Proposed architectures

In our vehicle classification problem, we want to classify each GPS track according to the type of vehicle that generates it. Each GPS track corresponds to a sequence of input vectors $\{x^t\}_{t=0}^T$, $x^t \in \mathbb{R}^{14}$ that are fed into our network, computed according to the pre-processing formulas described in Sec. 2.1. Although we consider several configurations in terms of number of layers in the experimental results, the general architecture we adopt is a network with a stack of multiple feed-forward and/or recurrent layers, with three separate groups of layers, loosely inspired by the architectures presented in Pascanu et al. (2013a) but applied to LSTM instead of traditional RNN: an input layer followed by $N_{i\to r}$ ($N_{i\to r} \geqslant 0$) input-to-recurrent fully connected feed-forward layers; N_r ($N_r \geqslant 1$) recurrent LSTM layers; and $N_{r\to o}$ ($N_{r\to o} \geqslant 0$) recurrent-to-output fully connected feed-forward layers, leading to a final output layer. In Fig. 3 we illustrate two versions of our architecture, unrolled in time. The input-to-recurrent set of feed-forward layers operate independently on each temporal sample of the sequence, applying the same function in what can be thought as a feature extraction phase (Chen and Deng, 2014). The set of recurrent layers, on the other hand, operate sequence-wise, applying a function that depends not only on the current input, but also on the state of the layer at the previous timestep. Recurrent layers are meant to learn patterns through time.



(a) An example of NO POOLING architecture.



(b) An example of the architecture with pooling across time.

Fig. 3. Examples of the two recurrent architectures we tested. In both architectures, for each GPS point t, the input layer (green) is fed a vector x^t. Some input-to-recurrent fully connected feed-forward layers (blue) process the input before the recurrent layers. Each LSTM layer (red) receives as input at time t the output of the underneath layer and its internal states h^{t-1} and c^{t-1} from the previous timestep. In the first architecture, at the last timestep T the output of the top LSTM layer goes through a number of additional recurrent-to-output feed-forward layers (brown), before being classified by the final output layer (orange). In the second architecture, at the last timestep T all the outputs across time of the top LSTM layer go through a pooling function, such as the average function, and the pooled output is processed by the recurrent-to-output feed-forward layers and classified in the output layer. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Since the length T of each sequence is not constant, but we are dealing with a sequence-to-one classification problem, we then need to encode the output of the top recurrent layers into a fixed-length vector. A simple and effective option is to use only output of the top LSTM layer after it has seen the whole sequence: this means that the network should accumulate evidence that points to a certain decision, and only the final state vector of the top recurrent layer is considered (see Fig. 3a, NO POOLING). A slightly different option is to use a one-dimensional (1D) pooling across time, aggregating the outputs of the top recurrent layers over the whole time horizon (see Fig. 3b). This way, rather than letting the recurrent layers accumulate evidence over time to produce a single output, we look at the entire history of outputs from the recurrent layers, applying a pooling function to aggregate it. The pooling function can be the average operator, as done, *e.g.*, in Zhang et al. (2015), or the max operator, under the hypothesis that single, highly confident outputs should matter more than the whole sequence.

Once the sequence has been encoded via the recurrent layers into a single, fixed-length vector, a second set of recurrent-to-output fully connected feed-forward layers is used to operate on the encoded sequence and produce the output via a final non-linear activation.

Table 1
Parameters used in training neural networks

Activation functions	ReLU
Optimizer	Adam
Learning rate	1e−3
Gradient clipping	10.0
β_1 (Adam)	0.9
β_2 (Adam)	0.999
Weight initialization feed-forward	Glorot uniform (Glorot and Bengio, 2010)
Weight initialization LSTM	Glorot uniform (Glorot and Bengio, 2010)
Weight initialization LSTM, recurrent path	orthogonal (Saxe et al., 2013)

2.4. Training

Several techniques may be used in order to speed up the training of neural networks and reduce over-fitting. We choose to add batch normalization layers (Ioffe and Szegedy, 2015) before the non-linear activation function of each feed-forward or recurrent layer as a regularization technique. In preliminary tests, batch normalization appeared in our case to be more effective than other equally popular techniques, such as dropout (Srivastava et al., 2014).

As far as the optimization algorithm is concerned, we use Adam (Kingma and Ba, 2015), combined with gradient clipping (Pascanu et al., 2013b) to avoid instability. The Adam algorithm turns out to be quite robust to the input parameters such as learning rate, which is of great help in the parameter tuning phase. The training phase is performed with stratified minibatches of size 64. As activation functions in the feed-forward layers, we use ReLU everywhere, except for the layers in the LSTM, which are sigmoid or *tanh* functions, according to the original architecture. As we are dealing with unbalanced classes, the loss function is weighted by the inverse of the class size, for neural networks and also for baselines. Table 1 reports the parameters used in the training phase of the neural networks.

The models are built and trained with `tensorflow` (Abadi et al., 2015), via the `Keras` wrapper (Chollet, 2015), on Nvidia K80 GPUs as provided on the `p2.xlarge` AWS instances.

3. Experimental results

In our experiments, we consider two variants of the vehicle classification task. First, we focus on a binary classification problem, where we want to discriminate heavy-duty vehicles from light-duty ones, based on the GPS tracks they generated. This is the problem already considered in Simoncini et al. (2016), Sun et al. (2013). Then, we extend the approach to a three-class version of the same task, with a finer classification granularity. In this case, the classifier should distinguish between light-duty, mid-duty and heavy-duty vehicles, as defined below.

3.1. Datasets

To our knowledge, no public low-frequency GPS datasets labeled by vehicle type exist. The dataset we use is made out of GPS tracks from about 100 K vehicles (96,338) tracked by Fleetmatics, a fleet intelligence company now part of Verizon Connect,[3] across the US in January 2016. For each vehicle, we select 10 GPS tracks (as defined at the beginning of Section 2.1) with length no shorter than 20 timesteps, corresponding to roughly 30 min, assuming the average sampling rate is 90 s. Thus, considering the task of vehicle classification from single GPS sequences, we have almost 1 million examples in our dataset: this is one order of magnitude greater than what we used in Simoncini et al. (2016) and among the largest datasets ever used for this task.

Note that the number of sufficiently long sequences generated by each vehicle during the considered month may be larger than 10. However, we choose to include in our dataset the same number of sequences for all the vehicles, in order to avoid bias in the learning stage towards those that would appear a larger number of times in the dataset. At the same time, we have discarded vehicles that did not have at least 10 long enough sequences.

As far as data cleansing is concerned, due to the presence of some outliers in sequence length (often due to device or network issues, *e.g.*, missing "engine off" events), we decide to limit the length of the sequences to a maximum value, discarding the remaining GPS points. This preprocessing may also help training the network, as gradients in RNNs become harder to propagate the longer the input sequence is. We opt for keeping up to 200 GPS points for each sequence, corresponding to roughly 5 h of driving time, which should be enough to let driving patterns emerge and remove almost all outliers due to hardware issues.

For the binary problem, we define the two classes according to the same definitions used in Simoncini et al. (2016): we obtain 77,545 light-duty vehicles (containing cars, SUVs, vans and small/medium-size pickups, roughly corresponding to class 2 and 3 of the FHWA scheme) and 18,793 heavy-duty vehicles (small, medium and large-size trucks, classes 5 and above of the FHWA scheme). The problem is unbalanced, with the light-duty vehicles outweighing the heavy-duty ones about 4 to 1.

For the three classes problem, the vehicles were assigned as follow: 5,593 light-duty vehicles (cars and SUVs, corresponding to

[3] https://www.verizonconnect.com.

class 2 of the FHWA scheme), 78,102 mid-duty vehicles (vans, pickups and small trucks, corresponding to class 3 and 5 of the FHWA scheme) and 12,643 heavy-duty vehicles (medium and large-size trucks, corresponding to class 6–13 of the FHWA scheme). The problem is, again, unbalanced, with the mid-duty vehicles representing about 80% of the examples, and the light-duty class being particularly under-represented.

We split the dataset into training, validation, and test sets of 674,360, 144,500, and 144,520 examples, respectively (roughly a 70–15–15 split). The split is performed in a stratified way, according to the vehicle that generated each sequence: the proportion of GPS tracks belonging to each considered class is the same in the 3 sets, and all the tracks of the same vehicle belong to the same set (to avoid the bias arising from seeing the same vehicle both at training and test time). To train and tune the neural networks while avoiding over-fitting of the training set, we use the training set to learn the parameters, and test the learned model on the validation set to select the best epoch. To tune the other classifiers, we train them with 5-fold cross validation, stratified by vehicle, on the combined training and validation set. Finally, we compare all the classifiers by running them on the test set.

3.2. Performance metrics

When monitoring the performance of the network on the validation set during training, and to evaluate the final performance on the test set, we consider metrics which give meaningful results also with unbalanced class cardinalities. Moreover, we are interested in having good performance on classes regardless of their size, which can be obtained using macro-average metrics. For these reasons, we use as main performance indicator *balanced accuracy* (equivalent to macro-average recall) (Brodersen et al., 2010; Yang, 1999), defined as

$$acc_{bal} := \frac{1}{C}\sum_{c=1}^{C}\frac{TP_c}{N_c}, \tag{12}$$

where C is the number of classes, TP_c is the number of correctly classified samples (true positives) for class c, and N_c is the total number of samples belonging to class c. Beside the desired robustness, this metric also has the advantage of being seamlessly suited to evaluate binary and multi-class models.

In a practical setting, one may also use a cost-sensitive accuracy, where each type of error and correct classification is weighted according to the application goal. For binary models, this is equivalent to changing the decision threshold. Therefore, we also report the ROC Area Under the Curve (AUC), which compares models across all the range of possible decision thresholds, and can give complementary insights into the models performance. While in the binary case the AUC can be computed directly and is invariant to the choice of the positive class, for multi-class models we report the macro-averaged AUC.

3.3. Binary classification

We first report results on the binary classification task of distinguishing between sequences produced by light-duty and heavy-duty vehicles, as described in more detail in Section 3.1.

As a baseline, we use the approach we recently proposed in Simoncini et al. (2016), based on an SVM classifier that uses an optimal set of features found by recursive feature elimination. The selection is performed over a pool of features computed with sequence-level statistical aggregation functions, such as mean, standard deviation, etc. We adopt an RBF kernel, as in Simoncini et al. (2016). Since training the SVM classifier on the full dataset proves rather time consuming, we perform a grid search with 5-fold cross-validation over a subset of the data with 3,080 light-duty vehicles and 7,58 heavy-duty ones to select the SVM parameters. The optimal parameters are $C = 10.0$ and $\gamma = 0.08$.

We also use as baseline a random forest classifier (Breiman, 2001) as implemented in the scikit-learn library (Pedregosa et al., 2011). As it is faster to train than the SVM, to select its parameters we perform 5-fold cross validation over the full dataset: we run a grid search over the number of trees and their maximum depth. The optimal parameters turn out to be 200 trees and 20 levels of depth.

3.3.1. Single sequence classification

Results for the classification of a single sequence are reported in the column `Single seq.` of Table 2 for the baselines and a selection of the network configurations we tested. All the RNN models have been trained for 50 epochs, and we selected the model obtained at the epoch with the best balanced accuracy value on the validation set. Typically, this occurs between the 20th and the 40th epoch for all the models. Training longer only leads to over-fitting.

To define the best architecture and parameters for the network, we started by assessing the performance of a “shallow”, single layer of LSTM neurons. We preliminary experimented with some numbers of neurons for the layer, and we found that 100 neurons perform slightly better than fewer of them, and have similar performance to more neurons while being faster to train. Therefore, we settled on 100 neurons for these layers and the subsequent ones we add. A single layer of LSTM cells already performs better than the Random Forest baseline, and on par with the SVM one.

In Table 2, we first report experiments where we incrementally add new layers with a `NO POOLING` architecture, to assess the improvement obtained with deeper architectures, as suggested in Pascanu et al. (2013a). Interestingly, the presence of at least a fully-connected feed-forward layer between the input and the recurrent layers is crucial to obtain a significant improvement in both considered metrics (*e.g.*, the balanced accuracy jumps from 0.751 to over 0.786). An input-to-recurrent layers allows the network to

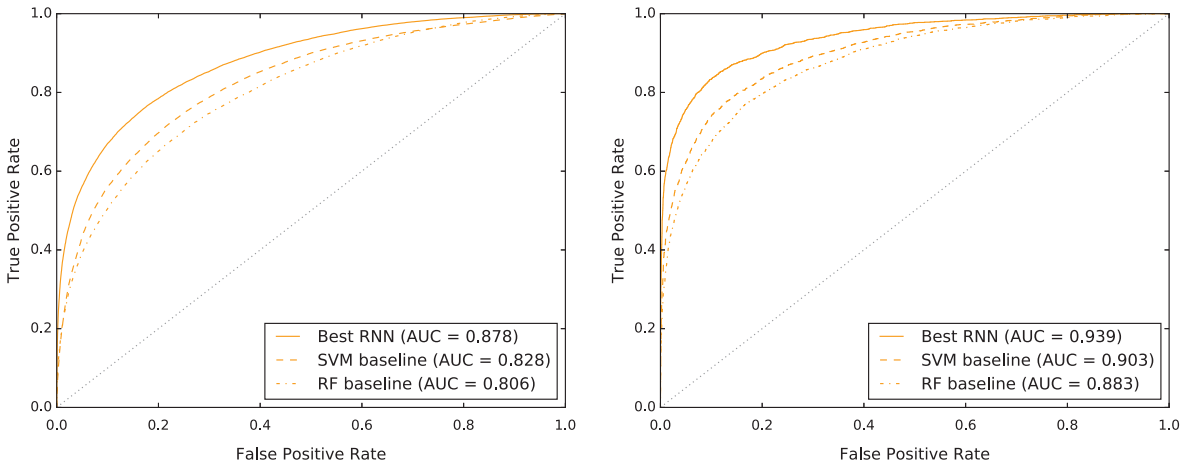
Table 2
Results for the binary classification task when classifying every sequence independently (`Single seq.` columns) and all the sequences of the same vehicle together (`Multiple seq.` columns). In bold, the best result for each metric. Rows marked as `NO POOLING` report results obtained with the architecture described in Fig. 3b, while rows marked as `MAXPOOL`, `AVGPOOL` report results obtained with the architecture with the pooling gate described in Fig. 3b, where the aggregation function is a max or average operator, respectively. In the layers columns for the RNNs, with the notation $L_i \times N_i$ we indicate that the layer has L_i stacked feed-forward or LSTM layers with N_i neurons.

Model	Input → Recurrent FF layers	Stacked LSTM layers	Recurrent → Output FF layers	Single seq.		Multiple seq.	
				acc_{bal}	AUC	acc_{bal}	AUC
RANDOM FOREST				0.726	0.806	0.801	0.883
SVM (Simoncini et al., 2016)				0.750	0.828	0.823	0.903
NO POOLING	–	1 × 100	–	0.751	0.831	0.828	0.905
NO POOLING	–	1 × 100	1 × 100	0.756	0.839	0.835	0.915
NO POOLING	1 × 100	1 × 100	–	0.786	0.871	0.858	0.933
NO POOLING	1 × 100	1 × 100	1 × 100	0.793	0.876	0.863	0.938
NO POOLING	3 × 100	1 × 100	1 × 100	**0.794**	**0.878**	**0.867**	**0.939**
MAXPOOL	3 × 100	1 × 100	1 × 100	0.792	0.877	0.865	0.938
AVGPOOL	3 × 100	1 × 100	1 × 100	0.793	0.877	**0.867**	**0.939**
NO POOLING	5 × 100	1 × 100	1 × 100	0.793	0.877	0.863	0.937
NO POOLING	1 × 100	2 × 50	1 × 100	0.790	0.875	0.862	0.936
MAXPOOL	1 × 100	2 × 50	1 × 100	0.793	**0.878**	0.865	0.938
AVGPOOL	1 × 100	2 × 50	1 × 100	0.792	0.877	0.862	0.937

learn new representations of the input data which are more effective when stored in the recurrent layers. Adding a feed-forward recurrent-to-output layer, between the LSTM layer and the output of the network, allows the model to learn also some useful post-processing of the network hidden state, and slightly increase the model performance. Overall, the configuration with an input-to-recurrent layer and a recurrent-to-output layer surrounding the LSTM layer obtains a significant improvement in terms of balanced accuracy (0.793) and AUC (0.876) with respect to the one with a single recurrent layer only. This is consistent with the findings in Pascanu et al. (2013a), that showed how the performance of RNNs can be improved inserting deep feed-forward layers between the recurrent layers and the input/output.

Let us now also consider networks with the pooling gate, as described in Fig. 3b. As pooling functions, we used the *average* and the *max* operators, which are commonly used in deep neural networks (Goodfellow et al., 2016).

Following on the idea that deeper networks have better performance, we tried to further increase the number of input-to-recurrent layers, whose addition brought the greatest improvement. Indeed, using 3 input-to-recurrent layers gives us a small but consistent improvement across metrics: we obtain models with the best overall performance, reaching almost 80% of balanced accuracy, and almost 0.88 of AUC score, with the `NO POOLING` architecture. Results with the pooling architectures are very similar. By inspecting the ROC curves in Fig. 4a, we can see that the best deep model outperforms the baseline not only at the aggregated AUC level, but for all levels of specificity.



(a) Single sequence. (b) Sequences aggregated by vehicle.

Fig. 4. ROC curves over the test set for the best recurrent neural network and the baselines.

Adding more layers does not appear to help, as shown by the experiment with 5 input-to-recurrent layers. Note that networks with several feed-forward layers are significantly slower to train due to the increasing number of parameters, so we did not experiment with adding more layers.

An alternative way to increase the power of a recurrent neural network is to use more than one recurrent layer (Pascanu et al., 2013a). We tested an architecture with two stacked recurrent layers, where we reduced the number of LSTM cells to 50, to limit the training time and the number of parameters. With two stacked layers, we obtain the best results with the max pooling network, whose performance is very close to the best overall results.

3.3.2. Multiple sequences

Depending on the application, it may be feasible to delay the decision on the vehicle category until more than one GPS sequence have been collected. Indeed, the behavior of a vehicle in a (relatively) short period of time does not only depend on its category, but also on several additional factors, such as the driver behavior, and external conditions (weather, traffic, etc.). Moreover, especially in mid-duty vehicles, the dynamics are highly affected by the load of the vehicles: a pickup motion will appear more like a high-duty vehicle when it is fully loaded, while, when unloaded, it may be hard to distinguish from a sedan – and especially so, given that we are dealing with low-frequency data. Making an informed decision from multiple GPS sequences obtained from the same device would allow to average out outliers or non-typical behaviors.

Therefore, we tested a simple algorithm which makes use of all the 10 GPS tracks that are available for each vehicle in our dataset. First, we classify every track with a single-sequence classification model, trained as described in the previous section. Then, we aggregate the results by averaging the continuous output of the model for the 10 predictions and we threshold the average to obtain the final classification. As shown in the columns `Multiple seq.` of Table 2, this simple aggregation yields a significant boost in classification performance for all the models.

The best model remains the same even when aggregating sequences, and it keeps outperforming the baselines at the aggregated AUC level and at all values of specificity (Fig. 4b).

To further study the effect of the aggregation of several single-sequence outputs, we tested the performance of the method when we artificially limit the number of sequences to be aggregated. We randomly sample without replacement a subset of cardinality $k = 1,\ldots,10$ from the 10 available tracks in the test set for each vehicle, use our best RNN model to obtain k single-sequence classification outputs, and aggregate them. This process is repeated 10 times and the final accuracy averaged. As easily seen in Fig. 5, the balanced accuracy for our best model grows with the number of aggregated sequences, but the gain quickly plateaus after 7/8 tracks, which suggests that 10 sequences can be an effective value in practice.

3.4. Interpretation of the learned binary model

In Fig. 6, we report the evolution of the internal state of the recurrent layer for two sequences belonging to the two classes, to provide some intuitions about what kind of reasoning the network has learned to carry out. In both sequences, the network starts with high confidence in the vehicle being a heavy-duty one: as there is some idling at the beginning of the sequence, the network starts to favor the heavy-duty class, as heavy-duty vehicles tend to have long idling periods at the beginning of the sequences, likely because it takes longer to load them. In both sequences, when the vehicle starts to move the confidence of the network decreases, and it is more or less half-way after the first part of the sequence, where both vehicles travel on motorways with similar speed, above 100 km/h. In the case of the first sequence, the network gains more and more confidence in the vehicle being a heavy-duty one, the more stop-and-restarts the vehicle accumulates: probably, big trucks tend to serve stops without turning their engine off, and the network has learned to exploit this clue. We can see that there are a lot of transitions in the LSTM cell states every time the vehicles starts a new

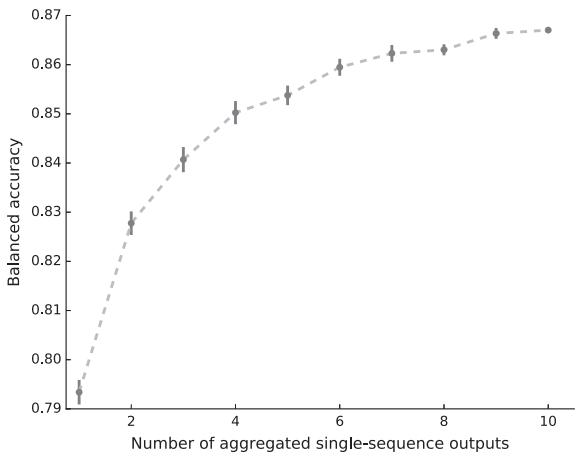
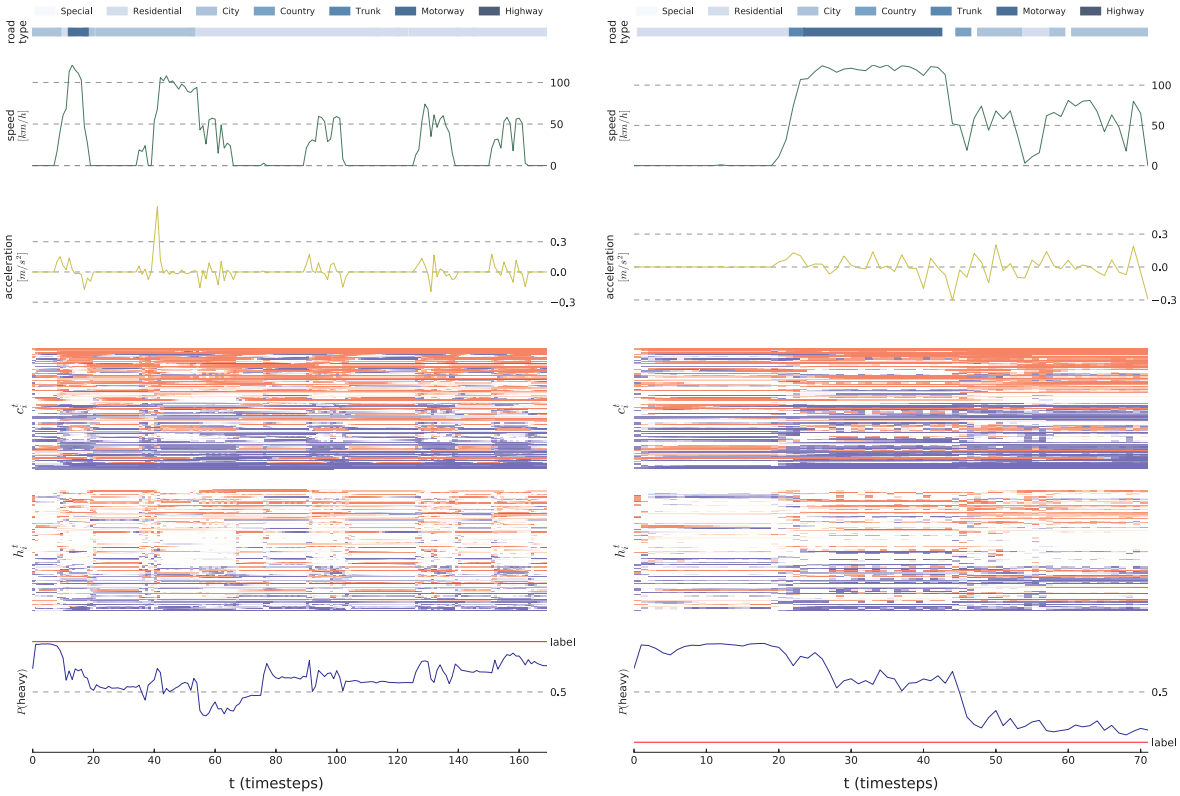


Fig. 5. Balanced accuracy on the vehicles of the test set as a function of the number of single-sequence results that are aggregated for each vehicle (mean ± standard deviation).



(a) Heavy-duty vehicle (b) Light-duty vehicle

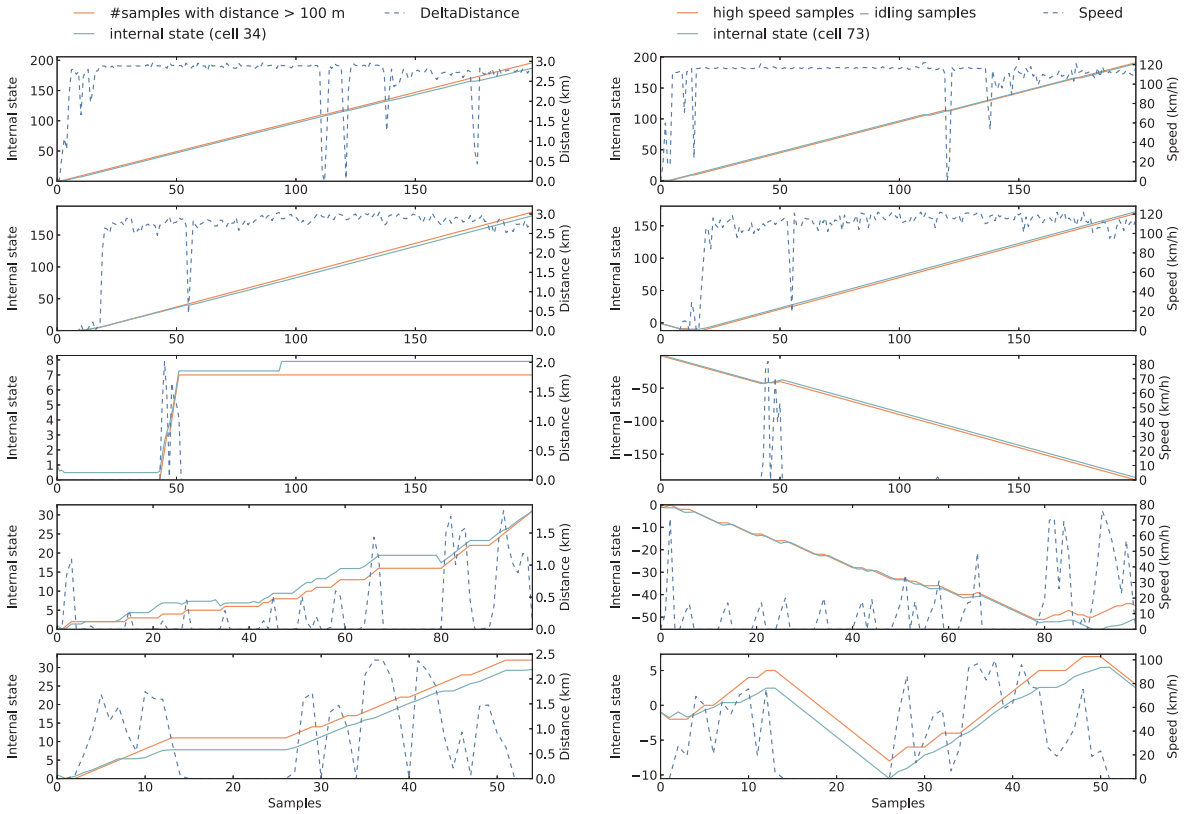
Fig. 6. Input data (road type, speed and acceleration) and LSTM activations over time for the best binary network. More specifically, the top subplot reports the road type where darker means faster (highways are darkest blue, while missing data is in white). The second subplot displays the instant speed in km/h, while the third shows the acceleration computed as the difference between consecutive instant speeds. The two heat maps show on the i-th row the activation of the i-th LSTM cell of the recurrent layer: the first heat map shows the internal cell state c_i^t (normalized between -1 (blue) and $+1$ (red) applying a *tanh* function), the second one the output state h_i^t. Finally, the bottom plot reports the unthresholded output of the network for each time t, *i.e.*, the probability assigned by the network to the input track having been generated by a heavy-duty vehicle up to that moment (heavy-duty $= 1$, light-duty $= 0$). The correct label is displayed as a red line. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

part of its journey, and the output confidence increases accordingly. Instead, for the light-duty vehicle, the network moves strongly in favor of the light-duty class around sample 45. Probably, the sharp deceleration at sample 44, combined with what the network has observed so far, provides enough evidence for the decision. Indeed, in our dataset it is more likely for light-duty vehicles to reach such decelerations, which are high for our sampling frequency. However, it is hard to interpret which events are key for the network decisions. It is also evident from these charts how difficult it is to infer the correct label from the sparse information we collect.

It is also instructive to see examples of what the network has learned to memorize in its recurrent neurons. To identify easily interpretable neurons, we looked for LSTM cells that were highly correlated with simple functions of the input.

First, we show a neuron whose internal state sequence $\{c^t\}_{t=1}^T$ has maximum average correlation, over all the GPS tracks ($\rho = 0.98$), with the function that counts the number of non-zero distance intervals occurred so far (defined as GPS samples where *Distance* is greater than 100 m). We report the evolution of this function and the cell state for five randomly selected input sequences in Fig. 7a, that clearly shows the similarities of the two signals. From our previous work (Simoncini et al., 2016), we knew that the total cumulative distance is an important feature. It is interesting to see that the network has learned to extract a similar information, but has done this totally automatically from the training data.

Another interpretable neuron we found is a cell which is tuned to approximately memorize the difference between the number of high speed events (*i.e.*, *Speed* is greater than 40 km/h) and idling events (*i.e.*, *Speed* is lower than 3 km/h) seen so far. The average correlation between this signal and the cell state across our dataset turns out to be over 0.9. We report in Fig. 7b the evolution of this function and the cell state for the same five input sequences considered in Fig. 7a. As we observed when commenting Fig. 6, a lot of idling samples are a strong indication that the vehicle is a heavy-duty one, therefore it seems reasonable in hindsight to store this information. Overall, this is another interesting example of the ability of the network to figure out on its own what processing of the input features can be effective for the classification task at hand.



(a) LSTM neuron correlated with distance events (b) LSTM neuron correlated with speed events

Fig. 7. Example of LSTM neurons whose internal state is highly correlated with interpretable functions of the inputs. We report the same sequences in both sub-figures. On the left, the internal state of a neuron which approximately counts how many samples have *Distance* greater than a threshold, and the count of samples with *Distance* > 100 meters for reference. On the right, the internal state of a neuron which approximately memorizes the difference between the number of samples with high speed (> 40 km/h) minus the idling samples (*Speed* < 5 km/h).

3.5. Three-class classification

In this section, we report results on the more challenging task of classifying sequences into three classes: light-duty, mid-duty, and heavy-duty vehicles. Although this problem has not been studied before in the literature, we can easily extend both baselines we used for the binary case. The SVM approach we proposed in Simoncini et al. (2016) can be used for multi-class classification with a standard One-versus-All (OVA) algorithm. The optimal parameters are again selected with a grid search on the same subset of the full dataset, as done for the binary problem, obtaining $C = 10$ and $\gamma = 0.0022$. It is even easier to extend the random forest baseline to the multi-class scenario, because it seamlessly supports it and it is actually widely regarded as very competitive out-of-the-box choice for multi-class classification problems (Hastie et al., 2009). As done for the binary problem, we perform a grid search with 5-fold cross-validation over the full training set to select the optimal number of trees and their maximum depth. The optimal parameters in this case turn out to be 300 trees and 10 levels of depth.

The RNN architectures tested in the multi class problem are the same we used for the binary case, the only difference being the final layer, where we replace the sigmoid activation function with the standard 3-way softmax. We use the same number of epochs and the same model selection strategy used for the binary case.

Results are reported in Table 3 for a selection of network architectures and parameters which had the best performance. All the tested configurations significantly outperform the baseline methods in terms of balanced accuracy, both when classifying single sequences and when aggregating multiple sequences. For the three-class problem the architecture with the max pooling gate performs slightly better than the `NO POOLING` one. This suggests that, for harder problems, looking at the highest activation in the entire history of the recurrent layer helps highlighting segments where it is easier to infer the vehicle class.

Let us also use the multi-class problem to provide some results in terms of computational time. The SVM One-versus-All requires more than 10 days of training time on a single CPU (an Intel Xeon E5-2660 v3 @2.60 GHz), while the Random Forest only requires 20 min on the same machine. For the neural networks, each training epoch takes between 1 and 2 h, which translates to several days of training time (between 2 and 5) without early stopping, on Nvidia K80 GPUs as provided on the `p2.xlarge` AWS instances. Concerning the efficiency when predicting the class of each sequence, the SVM is again quite slower, requiring 13 h to classify the

Table 3
Results for the multiclass task when classifying every sequence independently (`Single seq.` columns) and all the sequences of the same vehicle together (`Multiple seq.` columns). In bold, the best result for each metric. Rows marked as `NO POOLING` reports results obtained with the architecture described in Fig. 3a, while rows marked as `MAXPOOL` and `AVGPOOL` reports results obtained with the architecture with the pooling gate described in Fig. 3b, where the aggregation function is an average or a max pooling, respectively. In the layers columns for the RNNs, with the notation $L_i \times N_i$ we indicate that the layer has L_i stacked feed-forward or recurrent layers with N_i neurons.

Model	Input→Recurrent FF layers	Stacked LSTM layers	Recurrent → Output FF layers	Single seq.		Multiple seq.	
				acc_{bal}	AUC	acc_{bal}	AUC
SVM OVA				0.586	0.759	0.657	0.821
RANDOM FOREST				0.580	0.781	0.664	0.848
NO POOLING	3 × 100	1 × 100	1 × 100	0.655	0.820	0.745	0.886
AVGPOOL	3 × 100	1 × 100	1 × 100	0.654	0.820	**0.758**	0.887
MAXPOOL	1 × 100	2 × 50	1 × 100	**0.661**	**0.821**	0.753	**0.888**

144 k GPS sequences in the test set. The RF classifier takes less than 10 s, while the Neural Network roughly takes 5 min, *i.e.* around 2 ms per sequence. Therefore, although the training phase is computationally intensive, the approach based on recurrent neural networks is also extremely attractive in terms of practical applicability in real-time.

3.6. Interpretation of the learned 3-class model

By analyzing the confusion matrix reported in Table 4, we can see that, by using the class weights in the loss function, we are able to obtain a classifier which has good performance also on the minority classes (*i.e.*, light-duty and heavy-duty vehicles). In particular, the classifier is effective at distinguishing between the heavy-duty vehicles and the other classes. However, the mid-duty vehicles are especially hard to separate from the light-duty ones: roughly 40% are incorrectly classified as light-duty. We believe this may be mainly due to the large intraclass variance in how vehicles from the middle class are used. Indeed, mid-duty vehicles such as pickups and small vans can be used for very different purposes, ranging from the handling of heavy loads to passenger cars for field service operators, and this greatly affects the kind of behavior that is captured in the GPS data. To address this ambiguity, finer-grained GPS data could be beneficial, as well as additional complementary sources of information, *e.g.* axle load sensors or RPMs data.

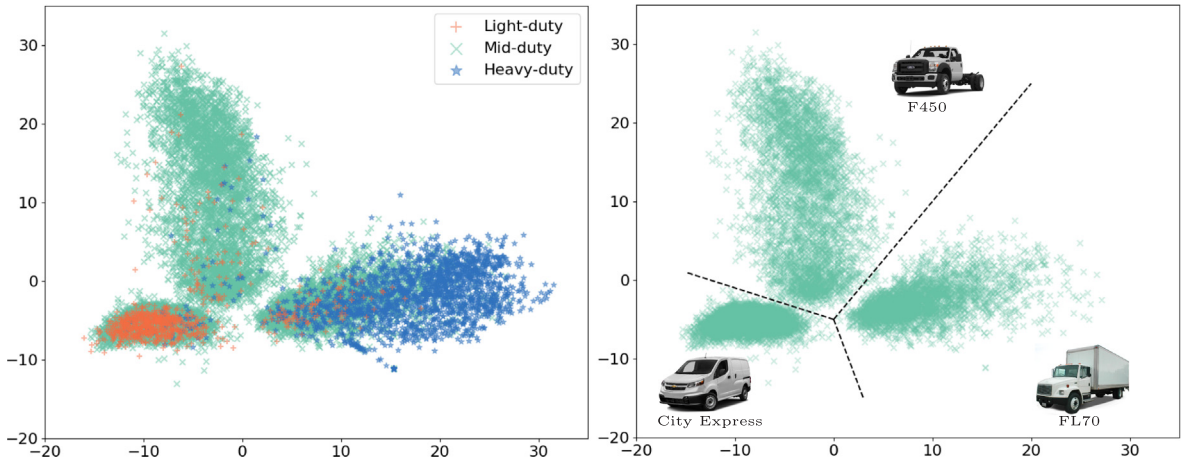
The ambiguity is confirmed by the Principal Component Analysis reported in Fig. 8a. This analysis shows that the hierarchy of feature extractors learned by the network is able to embed a sequence in a feature space where light-duty vehicles (leftmost cluster) and heavy-duty vehicles (rightmost cluster) form two well separated clusters along the main principal direction. Interestingly, mid-duty vehicles are clearly divided into three clusters in the space of the first two principals directions: two clusters correspond to light-duty and heavy-duty vehicles, while the third cluster spreads along the second principal component, again well separated from the others. If we select the most frequent mid-duty vehicle from each cluster, as reported in Fig. 8b, we can see that they correspond to the semantics of the clusters we identified: in the cluster overlapping with light-duty vehicles (left), the most frequent mid-duty vehicle is the Chevrolet City Express, a small van; in the cluster overlapping with heavy-duty vehicles (right), the most frequent mid-duty vehicle is the Freightliner FL70, a small truck; in the third cluster (top), the most common mid-duty vehicle is the FORD F450 chassis, an average pickup. The network has automatically learned to segment the mid-duty vehicles into sub-classes and to infer the similarities with the other classes: this is a remarkable result and shows that deep learning holds the potential to discriminate between finer-grained classes as well.

4. Conclusions

In this work we have investigated the effectiveness of modern neural network architectures used with low-frequency GPS data for the purpose of vehicle classification. When applied to the problem of recognizing whether the vehicle generating the GPS track is a light-duty or a heavy-duty vehicle, deep recurrent architectures are shown to outperform the existing state-of-the-art based on hand-crafted features and shallow classification algorithms like SVMs. The proposed architectures seamlessly generalize to multi-class classification, and we have used them to set a reference performance for the yet unexplored problem of classifying vehicles into three categories from low-frequency GPS data. All our tests have been carried out on one among the largest datasets of GPS sequences used

Table 4
Confusion matrix for the best model on the multi-class problem when aggregating all the sequences of a vehicle.

		predicted		
		light	mid	heavy
real	light	**85%** (709)	13% (112)	2% (19)
	mid	41% (4848)	**48%** (5648)	11% (1220)
	heavy	1% (22)	6% (104)	**93%** (1770)



(a) All classes. (b) Only mid-duty vehicles.

Fig. 8. Principal Component Analysis embedding along the first two directions of the output state of the last layer before the output layer of the best model for the multi-class case. To reduce the number of points and study how the network has learned to map the different classes, we encode, for each vehicle, the sequence classified with highest confidence by the network. All sequences are sampled from the test set. In figure (b), we report also the most frequent mid-duty vehicles in the three clusters defined by the dotted lines.

to date.

In particular, we have investigated recurrent neural networks with Long-Short Term Memory neurons, and have experimentally shown that feed-forward input-to-recurrent and recurrent-to-output layers can greatly enhance their performance. A variant of this architecture, where the recurrent-to-output layers operate on max- or average-pooled outputs of the LSTM layer across time, is shown to have similar performance to the architecture without pooling gates in the binary case, and slightly superior results in the multi-class case.

The results that can be obtained with data from other types of hardware sensors, as we mentioned in the introduction, are typically still superior to what can be achieved with low-frequency GPS data only. However, the kind of data used in this study is being increasingly collected by individuals, by public institutions, and by companies operating in the growing market of IoT and fleet intelligence solutions. Therefore, we expect the interest in leveraging these data assets to grow in the coming years and we think that other companies and research labs experimenting on GPS data with deep learning technologies will benefit from the findings of our study.

As far as our research is concerned, we would like to assess the improvements in classification accuracy achievable by using several complementary sources of information that could be collected from our trackers, like GPS altitude, RPMs, and accelerometer data. Moreover, architectures based on alternative recurrent units (such as the Gated Recurrent Units Chung et al., 2014), attention mechanisms (Vaswani et al., 2017), or one-dimensional convolutional neural networks, could be evaluated, and different techniques to cope with the class imbalance could be explored, such as undersampling/oversampling methods.

Another potential avenue for research would be to extend and generalize our approach to deal with high-frequency data obtained from smartphone sensors (Vlahogianni and Barmpounakis, 2017), applying recurrent neural network architectures not only to vehicle classification, but also to other related problems, such as transportation mode detection (Xiao et al., 2015) or driving behavior identification (Johnson and Trivedi, 2011; Paefgen et al., 2012). For the mentioned problems, our proposed architecture could be adapted by removing the pooling gates, so as to obtain a sequence-to-sequence classification model, rather than a classifier with a single output.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X., 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org. URL http://tensorflow.org/.

Benekohal, R., Girianna, M., 2003. Technologies for truck classification and methodologies for estimating truck vehicle miles traveled. Transp. Res. Record: J. Transport. Res. Board (1855), 1–13.

Bengio, Y., Simard, P., Frasconi, P., 1994. Learning long-term dependencies with gradient descent is difficult. IEEE Trans. Neural Networks 5 (2), 157–166.

Bolbol, A., Cheng, T., Tsapakis, I., Haworth, J., 2012. Inferring hybrid transportation modes from sparse GPS data using a moving window SVM classification. Comput. Environ. Urban Syst. 36 (6), 526–537 special Issue: Advances in Geocomputation.

Bottou, L., Curtis, F.E., Nocedal, J., 2016. Optimization methods for large-scale machine learning. http://arxiv.org/abs/1606.04838.

Breiman, L., 2001. Random forests. Machine Learning 45 (1), 5–32. http://dx.doi.org/10.1023/A:1010933404324.

Brodersen, K. H., Ong, C. S., Stephan, K. E., Buhmann, J. M., 2010. The balanced accuracy and its posterior distribution. In: Pattern recognition (ICPR), 2010 20th international conference on. IEEE, pp. 3121–3124.

Chen, J., Deng, L., 2014. A primal-dual method for training recurrent neural networks constrained by the echo-state property. In: ICLR. URL http://arxiv.org/abs/1311.6091.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y., 2014. Learning phrase representations using RNN encoder-decoder for

statistical machine translation. http://arxiv.org/abs/1406.1078.
Chollet, F., 2015. Keras. https://github.com/fchollet/keras.
Chung, J., Gulcehre, C., Cho, K., Bengio, Y., 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555.
Deng, L., Hinton, G., Kingsbury, B., 2013. New types of deep neural network learning for speech recognition and related applications: An overview. In: Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on. IEEE, pp. 8599–8603.
Dong, W., Li, J., Yao, R., Li, C., Yuan, T., Wang, L., 2016. Characterizing driving styles with deep learning. arXiv preprint arXiv:1607.03611. URL http://arxiv.org/abs/1607.03611.
Duchi, J., Hazan, E., Singer, Y., 2011. Adaptive subgradient methods for online learning and stochastic optimization. J. Mach. Learn. Res. 12, 2121–2159. URL http://dl.acm.org/citation.cfm?id=1953048.2021068 .
Eren, H., Makinist, S., Akin, E., Yilmaz, A., 2012. Estimating driving behavior by a smartphone. In: Intelligent Vehicles Symposium (IV), 2012 IEEE. IEEE, pp. 234–239.
Gebru, T., Krause, J., Wang, Y., Chen, D., Deng, J., Fei-Fei, L., 2017. Fine-grained car detection for visual census estimation. In: AAAI. pp. 4502–4508.
Gers, F.A., Schmidhuber, J., Cummins, F., 2000. Learning to forget: continual prediction with LSTM. Neural Comput. 12 (10), 2451–2471.
Glorot, X., Bengio, Y., 2010. Understanding the difficulty of training deep feedforward neural networks. In: JMLR W&CP: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010). Vol. 9. pp. 249–256.
Gonzalez, P., Weinstein, J., Barbeau, S., Labrador, M., Winters, P., Georggi, N.L., Perez, R., 2008. Automating mode detection using neural networks and assisted GPS data collected using GPS-enabled mobile phones. In: 15th World congress on intelligent transportation systems.
Goodfellow, I., Bengio, Y., Courville, A., 2016. Deep Learning. MIT Press. URL http://www.deeplearningbook.org.
Graves, A., 2012. Supervised Sequence Labelling with Recurrent Neural Networks. Springer.
Graves, A., Schmidhuber, J., 2009. Offline handwriting recognition with multidimensional recurrent neural networks. In: Advances in Neural Information Processing Systems. pp. 545–552.
Gupte, S., Masoud, O., Martin, R.F., Papanikolopoulos, N.P., 2002. Detection and classification of vehicles. IEEE Trans. Intell. Transport. Syst. 3 (1), 37–47.
Guyon, I., Elisseeff, A., 2003. An introduction to variable and feature selection. J. Mach. Learn. Res. 3 (Mar), 1157–1182.
Hallenbeck, M.E., Selezneva, O.I., Quinley, R., 2014. Verification, refinement, and applicability of long-term pavement performance vehicle classification rules. Tech. Rep.
Harlow, C., Peng, S., 2001. Automatic vehicle classification system with range sensors. Transport. Res. Part C: Emerg. Technol. 9 (4), 231–247.
Hastie, T., Tibshirani, R., Friedman, J., 2009. The Elements of Statistical Learning. vol. 1. Springer series in statistics Springer, Berlin.
He, K., Zhang, X., Ren, S., Sun, J., Dec 2015. Delving deep into rectifiers: surpassing human-level performance on imagenet classification. In: 2015 IEEE International Conference on Computer Vision (ICCV). pp. 1026–1034.
Hochreiter, S., Schmidhuber, J., 1997. Long short-term memory. Neural Comput. 9 (8), 1735–1780.
Ioffe, S., Szegedy, C., 2015. Batch normalization: accelerating deep network training by reducing internal covariate shift. In: International Conference on Machine Learning. pp. 448–456.
Johnson, D.A., Trivedi, M.M., 2011. Driving style recognition using a smartphone as a sensor platform. In: Intelligent Transportation Systems (ITSC), 2011 14th International IEEE Conference on. IEEE, pp. 1609–1615.
Kadous, M.W., 2002. Temporal classification: Extending the classification paradigm to multivariate time series. Ph.D. thesis, The University of New South Wales.
Kafai, M., Bhanu, B., 2012. Dynamic Bayesian networks for vehicle classification in video. IEEE Trans. Indus. Infor. 8 (1), 100–109.
Kingma, D., Ba, J., 2015. Adam: a method for stochastic optimization. In: International Conference on Learning Representations (ICLR2015). CBLS. URL http://arxiv.org/abs/1412.6980.
Leduc, G., 2008. Road traffic data: collection methods and applications. Working Papers on Energy, Transport and Climate Change (1).
Le, Q.V., Jaitly, N., Hinton, G.E., 2015. A simple way to initialize recurrent networks of rectified linear units. arXiv preprint arXiv:1504.00941. URL http://arxiv.org/abs/1504.00941.
Ma, X., Grimson, W.E.L., 2005. Edge-based rich representation for vehicle classification. In: Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1. Vol. 2. IEEE, pp. 1185–1192.
Nair, V., Hinton, G.E., 2010. Rectified linear units improve restricted Boltzmann machines. In: Frnkranz, J., Joachims, T. (Eds.), 27th International Conference on Machine Learning (ICML-10). Omnipress, pp. 807–814.
Paefgen, J., Kehr, F., Zhai, Y., Michahelles, F., 2012. Driving behavior analysis with smartphones: insights from a controlled field study. In: Proceedings of the 11th International Conference on mobile and ubiquitous multimedia. ACM, p. 36.
Pascanu, R., Gulcehre, C., Cho, K., Bengio, Y., 2013a. How to construct deep recurrent neural networks. arXiv preprint arXiv:1312.6026.
Pascanu, R., Mikolov, T., Bengio, Y., 2013b. On the difficulty of training recurrent neural networks. ICML (3 2), 1310–1318.
Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E., 2011. Scikit-learn: machine learning in python. J. Mach. Learn. Res. 12, 2825–2830.
Rumelhart, D.E., Hinton, G.E., Williams, R.J., 1986. Learning representations by back-propagating errors. Nature 323 (6088), 533–536.
Saxe, A.M., McClelland, J.L., Ganguli, S., 2013. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. CoRR abs/1312.6120. URL http://arxiv.org/abs/1312.6120.
Simoncini, M., Sambo, F., Taccari, L., Bravi, L., Salti, S., Lori, A., 2016. Vehicle classification from low frequency GPS data. In: 2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW). pp. 1159–1166.
Song, X., Kanasugi, H., Shibasaki, R., 2016. Deeptransport: prediction and simulation of human mobility and transportation mode at a citywide level. In: Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9–15 July 2016. pp. 2618–2624. URL http://www.ijcai.org/Abstract/16/372.
Srivastava, N., Hinton, G.E., Krizhevsky, A., Sutskever, I., Salakhutdinov, R., 2014. Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. 15 (1), 1929–1958.
Sun, Z., Ban, X., 2013. Vehicle classification using GPS data. Transp. Res. Part C: Emerg. Technol. 37, 102–117.
Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I., 2017. Attention is all you need. In: Advances in Neural Information Processing Systems. pp. 6000–6010.
Vlahogianni, E.I., Barmpounakis, E.N., 2017. Driving analytics using smartphones: algorithms, comparisons and challenges. Transport. Res. Part C: Emerg. Technol. 79, 196–206.
Wu, Y., Schuster, M., Chen, Z., Le, Q.V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, L., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M., Dean, J., 2016. Google's neural machine translation system: bridging the gap between human and machine translation. CoRR abs/1609.08144. URL http://arxiv.org/abs/1609.08144.
Wyman, J.H., Braley, G.A., Stevens, R.I., 1985. Field evaluation of FHWA vehicle classification categories. Maine Department of Transportation, Bureau of Highways, Materials and Research Division.
Xiao, G., Juan, Z., Zhang, C., 2015. Travel mode detection based on GPS track data and Bayesian networks. Comput. Environ. Urban Syst. 54, 14–22.
Yang, Y., 1999. An evaluation of statistical approaches to text categorization. Inform. Ret. 1 (1-2), 69–90.
Zeiler, M.D., 2012. ADADELTA: an adaptive learning rate method. CoRR abs/1212.5701. URL http://arxiv.org/abs/1212.5701.
Zhang, X., Zhao, J., LeCun, Y., 2015. Character-level convolutional networks for text classification. In: Advances in Neural Information Processing Systems. pp. 649–657.
Zheng, Y., Liu, L., Wang, L., Xie, X., 2008. Learning transportation mode from raw GPS data for geographic applications on the web. In: Proceedings of the 17th international conference on World Wide Web. ACM, pp. 247–256.
Zhou, Y., Nejati, H., Do, T.-T., Cheung, N.-M., Cheah, L., 2016. Image-based vehicle analysis using deep neural network: a systematic study. In: Digital Signal Processing (DSP), 2016 IEEE International Conference on. IEEE, pp. 276–280.