UNIVERSITY OF BERGEN
DEPARTMENT OF INFORMATICS

Prediction of Hyper-Congestion in Transport Systems

Author:
Senai Askale

Supervisor:
Prof. Ana Ozaki

UNIVERSITY OF BERGEN
Faculty of Mathematics and Natural Sciences

November 30, 2022

Contents

List of Figures

List of Tables

Abstract

Due to the progressive increase in the population and the complexity of their mobility needs, the evolution of transportation systems to solve advanced mobility problems has been necessary. In recent years, the use of machine learning within transport systems has increased because of a growing amount of data produced. This data needs analyzing, and there exist many standard machine learning methods to do so. A traffic prediction of a volume of traffic flow is usually to manage vehicle movement, reduce congestion, and generate the optimal route. A combination of different machine learning techniques seems to be very promising specially to manage and analyze the massive amount of data generated in a transportation system. However, many of these methods do have problems when it comes to interpretability. Therefore, it is essential to find a method that provides good predictive performance as well as interpretability in high-stake decision domains. The Tsetlin Machine is a new approach within machine learning based on propositional logic. The Tsetlin Machine has shown up as a new and promising candidate regarding performance and interpretability. The main aim of this thesis is to consider whether the Tsetlin Machine is an applicable model to a transportation system and whether it is comparable with neural networks for detecting traffic flow. We also report results from an experiment to evaluate how well the model performs in terms of accuracy. Our results indicate that the model can be used to predict the capacity of traffic flow, but we conclude that more experiments are needed to strengthen this.

Acknowledgements

Frist, my sincere thanks go to my supervisor, Prof. Ana Ozaki, for her patience and insightful contributions at every stage of the work. Thank you for always keeping an open door and making me feel comfortable. I would like also to thank to Prof. Ole-Christoffer Granmo and Associate Prof. K. Darshana Abeyrathna for the documentation that give me great input on the Tsetlin Machine.

Finally, I would like to thank my family and friends for always encouraging me to pursue my dreams and for all their love and support.

Chapter 1

Introduction

1.1 Motivation

The acceleration of urbanization and the rapid growth of the urban population bring great pressure on urban traffic management. An intelligent transportation system is an indispensable part of a smart city, and traffic prediction is an important component of an intelligent transportation system. Accurate traffic prediction is essential to many real-world applications. For example, traffic flow prediction can help a city alleviate congestion. The growing traffic-related datasets provide us with potential new perspectives to explore this problem [1].

Navigating tools like Google maps show us the time needed for our trip, calculate our estimated time of arrival, create the most optimal route based on road conditions and predicted traffic[1]. Traffic prediction is forecasting the volume and density of traffic flow, usually to manage vehicle movement, reduce congestion, and generate the optimal route. Traffic prediction is mainly important for national or local authorities. Many cities adopted intelligent transportation systems that support urban transportation network planning and traffic management. These systems use current traffic information as well as generated predictions to improve transport efficiency and safety by informing users of current road conditions and adjusting road infrastructure, for example, street lights. Furthermore, another area of importance of traffic prediction is the logistics industry. Transportation, delivery, field service, and other businesses must accurately schedule their operations and

[1]https://blog.google/products/maps/google-maps-101-how-ai-helps-predict-traffic-and-determine-routes/

create the most efficient routes. Often, this is not only related to current trips but also activities in the future. Precise forecasts of road and traffic conditions to avoid congestion are crucial for such companies planning and performance. The traffic information must include both historical and current traffic-related data such as the number of vehicles passing at a certain point, their speed, and type such as trucks, and light vehicles. A Devices used to collect this data can be cameras, weigh-in-motion sensors, radars, or other sensor technologies. Some companies use real-time traffic flows such as Google maps platform, Waze, TomTom, HERE, INRIX[2], OTONOMO[3] and PTV that improve real-time traffic flows across road system.

1.2 Research questions

The primary aim of this thesis is to consider whether The Tsetlin Machine approach is applicable within the field of transportation systems. More specifically, the Tsetlin Machines can compete with neural networks to predict accurate traffic flow forecasts. The goal is to answer the following questions.

- Which data features are the most important in making traffic forecasts?
- Interpretability of the Tsetlin Machines
- Compare the Tsetlin Machines approach with neural network
- Which approach for producing traffic forecasts works the best?

1.3 Related work

Much works are already exist on time series forecasting, including in the context of road traffic predictions using machine learning. We mention very few portions of the existing work on the subject. Most of the work consisted of learning about general neural networks and machine learning concepts as well as traffic forecasting.

In the paper Intelligent Transport by J Zhang, F Chen, Y Guo, X Li 2020 [2] investigating short-term passenger flow forecasting is a crucial task

[2]https://inrix.com/products/ai-traffic/
[3]https://otonomo.io/use-cases/traffic-management-data/

for urban rail transit operations. Emerging deep-learning technologies have become effective methods used to overcome this problem. In this study, the authors propose a deep-learning architecture called Conv-GCN that combines a graph convolutional network (GCN) and a three-dimensional (3D) convolutional neural network (3D CNN). First, they introduce a multi-graph GCN to deal with three inflow and outflow patterns (recent, daily, and weekly) separately. Multi-graph GCN networks can capture spatiotemporal correlations and topological information within the entire network.

In the study of Reliable traffic prediction by S. Guo, Y. Lin, S. Li, Z. Chen, and H. Wan [3] is critical to improving the safety, stability, and efficiency of Intelligent Transportation Systems. However, traffic prediction is a particularly challenging problem because traffic data are a typical type of Spatio-temporal data which simultaneously shows correlation and heterogeneity both in space and time. In this paper, they propose a novel end-to-end deep learning model, called ST-3DNet, for traffic raster data prediction. ST-3DNet introduces 3D convolutions to automatically capture the correlations of traffic data in both spatial and temporal dimensions.

The work by Schimbinschi et al. (2015) [4] investigated traffic forecasting in complex urban networks using big data and machine learning. As perhaps expected, they concluded that more data results in better predictions for the ML models. They also concluded that spatial dependencies between road segments are a better predictor compared to temporal patterns. They mention that the accuracy could be further improved if the biggest source of invariance in the data is removed. Finally, they say that ARIMA-based models have trouble forecasting based on Spatio-temporal data and are unable to capture complex dynamics. This would make it a bad candidate for traffic forecasting and a motivator for instead utilizing machine learning.

In the study of investigated traffic flow prediction with big data using deep learning by Lv et al. (2015) [5]. They mention that statistical methods such as linear regression and ARIMA perform well during normal traffic conditions. However, when abnormal traffic patterns appear they do not respond well. They utilized a deep learning technique called stacked autoencoders and concluded that this method was able to capture the nonlinear spatial and temporal correlations of the traffic data.

Yang et al. (2010) [6] is investigated short-term traffic flow predictions using an FFNN while considering weather parameters as features. They concluded that predictions based on weather parameters are more accurate than those without.

Chapter 2

Machine learning

This chapter provides a brief overview of supervised machine learning where machines are trained on labeled datasets and enabled to predict outputs based on the provided training and introduce prediction as a general concept. The section on Recurrent Neural Networks which briefly introduces RNNs is well-known to work well for learning tasks where the input data is sequential. Finally, the section on the Tsetlin machine introduces the architecture of the TM and the function of each layer.

Machine learning

Machine learning offers a new way to solve problems and answer complex questions. In basic terms, ML is a process of training a piece of software, called a model, to make useful predictions from data. An ML model represents the mathematical relationship between the elements of data that an ML system uses to make predictions. ML algorithms are molded on a training dataset to create a model. As new input data is introduced to a trained ML algorithm, it uses the developed model to make a prediction. The prediction is checked for accuracy. Based on its accuracy, the ML algorithm is either deployed or trained repeatedly with an augmented training dataset until the desired accuracy is achieved. Machine learning algorithms can be trained as supervised or unsupervised machine learning.

Supervised machine learning

Supervised machine learning,this type of ML involves supervision, where machines are trained on labeled datasets and enabled to predict outputs based on the provided training. The labeled dataset specifies that some input and output parameters are already mapped. Hence, the machine is trained with the input and corresponding output. A device is made to predict the outcome using the test dataset in subsequent phases. This is like a student learning new material by studying old exams that contain both questions and answers. Once the student has trained on enough old exams, the student is well prepared to take a new exam. These ML systems are "supervised" in the sense that a human gives the ML system data with the known correct results. The primary objective of the supervised learning technique is to map the input variable (a) with the output variable (b). Supervised machine learning is further classified into two broad categories[7].

Regression

Regression algorithms handle regression problems where a regression model predicts a numeric value. These are known to predict continuous output variables. For example, a traffic model that predicts the number of vehicles passing a junction or a model that predicts the future of house prices.

Classification

These refer to algorithms that address classification problems where the output variable is categorical. Classification models predict the likelihood that something belongs to a category. Unlike regression models, whose output is a number, classification models output a value that states whether or not something belongs to a particular category. For example, classification models are used to predict if a photo contains a cat or a dog. Classification models are divided into two groups binary classification and multiclass classification. Binary classification models output a value from a class that contains only two values, for example, a model that outputs either rain or no rain. Multiclass classification models output a value from a class that contains more than two values, for example, a model that can output either rain, hail, snow, or sleet[7].

2.0.1 Prediction

In the modern world, prediction has many areas of application. Some examples include stock market forecasting, weather forecasting, earthquake prediction, and of course traffic prediction. When no historic data features exist, forecasting is usually done subjectively through intuition, logic, and experience. This is performed by experts in each field and is referred to as qualitative forecasting [8]. Conversely, quantitative forecasting models are used to forecast future data based on existing historic data. This data made it possible for mathematicians to develop various mathematical models that could potentially produce more accurate predictions. However, these forecasting techniques are usually made for short to medium term predictions. The reason being that long-term predictions are harder to model. It is especially hard if the variable being predicted depends on many random events, and if the historic data is limited in quantity [8]. Instead, one usually uses qualitative forecasting if this is the case. The historic data corresponds to several features that arrive in a time sequence. This type of data is referred to as a time series, and analysis of this data format is a well-studied area [8].

2.0.2 Time series

A time series is a sequence x of measurements of some observable variable x_t at successive points in time with an equal time interval between every point [8]. Equation (2.1) mathematically describes a time series with T time steps. The subscript of each element represents the time step at which the variable was measured.

$$x = \{x_1, x_2, x_3, \dots, x_T\} \tag{2.1}$$

This time series is a vector with dimensions $T \times 1$. The order of the elements is of importance because it defines the temporal structure of the data points. Furthermore, when only one feature is measured in each time step, it is called a univariate time series. Now, as already mentioned, prediction are more commonly generated based on multiple historic features. In that case a multivariate time series is necessary. This is essentially just multiple univariate time series concatenated. This would give a matrix, which is described in equation (2.2).

$$X = \begin{bmatrix} X11 & X12 & . & . & X1T \\ X21 & X22 & . & . & X2T \\ . & . & & & . \\ . & . & & & . \\ XN1 & XN2 & . & . & XNT \end{bmatrix} \tag{2.2}$$

Each column of this matrix contains all the features of each time step in the time series. Therefore, the column vectors of the matrix correspond to feature vectors. There are N features and T time-steps which means that the dimension of X is $T \times N$. The matrix X can alternatively be described as follows,

$$X = \{x_1, x_2, x_3, \dots, x_T\}, x_t \in \mathbb{R}^N \tag{2.3}$$

where each element is a feature vector of size N.

A time series that can be predicted must consist of repeating temporal patterns that can be modelled. There are three important components of a time series that often needs to be considered. Namely, the seasonality, trend, and noise [8]. The trend describes the overall increase and decrease of the measured variable x_t. The seasonality describes the repeating short-term cycle in the series. The noise corresponds to random variation in the time series.

Stationary Time Series

The observations in a stationary time series are not dependent on time. Time series is stationary if they do not have a trend or seasonal effects. Summary statistics calculated on the time series are consistent over time, like the mean or the variance of the observations.

Non-Stationary Time Series

Observations from a non-stationary time series show seasonal effects, trends, and other structures that depend on the time index. Summary statistics like the mean and variance do change over time, providing a drift in the concepts a model may try to capture. Classical time series analysis and forecasting methods are concerned with making non-stationary time series data stationary by identifying and removing trends and removing seasonal effects. Differencing is a popular and widely used data transform for making time series data stationary machinelearning.

Difference Transform

Differencing is a method of transforming a time series dataset. It can be used to remove the series dependence on time, the so-called temporal dependence. This includes structures like trends and seasonality. Differencing can help stabilize the mean of the time series by removing changes in the level of a time series, and so eliminating or reducing trend and seasonality. Differencing is performed by subtracting the previous observation from the current observation [9].

Augmented Dickey-Fuller

The Augmented Dickey-Fuller test is a type of statistical test called a unit root test. The intuition behind a unit root test is that it determines how strongly a time series is defined by a trend. There are several unit roots tests and the Augmented Dickey-Fuller may be one of the more widely used. It uses an autoregressive model and optimizes an information criterion across multiple different lag values. The null hypothesis of the test is that the time series can be represented by a unit root, that it is not stationary (has some time-dependent structure). The alternate hypothesis (rejecting the null hypothesis) is that the time series is stationary [9].

Null Hypothesis (H0): Fail to reject, it suggests the time series has a unit root, meaning it is non-stationary. It has some time dependent structure. Alternate Hypothesis (H1): The null hypothesis is rejected; it suggests the time series does not have a unit root, meaning it is stationary. It does not have time-dependent structure. We interpret this result using the p-value from the test. A p-value below a threshold (such as 5% or 1%) suggests we reject the null hypothesis (stationary), otherwise a p-value above the threshold suggests we fail to reject the null hypothesis (non-stationary).

p-value ≤ 0.05 : Fail to reject the null hypothesis (H0); the data has a unit root and is non stationary.
p-value ≥ 0.05 : Reject the null hypothesis (H0); the data does not have a unit root and is stationary.

2.0.3 Time Series Prediction

A given time series of historic data features $\{x_1, x_2, \ldots, x_T\}$, the idea is to produce a prediction of some feature δt time steps into the future, δt is called the forecasting horizon. The prediction is denoted $y_{T+\delta t}$, and the generation of this can be described as shown in the equation (2.4) and the function f is predicting models.

$$\hat{y}_{T+\delta t} = f(x1, x2, x3, ..., xT), f : \mathbb{R}^{TxN} \to \mathbb{R}^1 \tag{2.4}$$

2.0.4 Evaluating a predicting model

Accuracy is evaluated based on comparing the predicted value $\hat{y}$ with its actual observed value y. To make forecasts in a time series with T time steps, one could make a prediction for the last value in the time series based on all the previous ones as shown below in (2.5).

$$\hat{y}_T = f(x_1, x_2, x_3, \ldots, x_{T-\delta t}) \tag{2.5}$$

In that case the predicted value is $\hat{y}_T$, which is at a time step within the limits of the time series. The actual value at this time step is accessible from the time series as x_T and is denoted y_T. However, to get a good estimation of how well a model is performing, many forecasts are necessary. This is done by moving through the entire time series and iteratively produce forecasts. There are two methods that accomplish this. The first method consists of an expanding window of time steps that each forecast is based on [10]. The second method is a sliding window method of a constant size that each forecast. The use of prior time steps to predict the next time step is called the sliding window method. For short, it may be called the window method in some literature. In statistics and time series analysis, this is called a lag or lag method. The number of previous time steps is called the window width or size of the lag. Out of the two methods, neither is superior in general. Both come with a few pros and cons, which was discussed by Clark et al. [10]. In short, selecting the optimal window size is not easy. It depends on the overall structure of the available data. If the entire time series follows a similar pattern, then the bigger the window size the better. Conversely, if old time steps are of very little relevance in predicting more recent time steps, then a shorter window size may be preferable. Also, if the time series used is very large, using an expanding window will lead to an infeasible computational complexity. Consequently, fitting the forecasting model as well as generating forecasts will be very time consuming.

Accuracy Metrics

When a large number of forecasts has been generated the next step is to evaluate the accuracy. The accuracy metric we used is Mean Absolute Error (MAE) and is defined as shown in equation (2.6).

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{2.6}$$

MAE measures the average magnitude of the errors across all predictions made. $\hat{y}_i$ corresponds to the i^{th} forecast, and y_i is its actual value. An error of zero means that all forecasts were equal to the actual value, which is the best-case scenario.

2.0.5 Overfitting and Underfitting

When a learning algorithm learns unique features to the training set, and so cannot generalize well to the test set and then the model is said to overfit the training data [11]. Various ML models use different regularization techniques to avoid overfitting. Conversely, when the model cannot learn from the training set at all, it is instead underfitting [11]. In this case, the model will perform poorly on both the training set as well as the test set. In the case of avoiding underfitting, the quality of the training data must be considered. For example, the data must contain relevant features for the learning objective, and various noise and outliers should be removed.

Hyperparameters

The weights and thresholds change over time as the model is trained. There are a few additional parameters that must be chosen. These conversely remain constant throughout the learning phase and are called hyperparameters. There are no general values to assign the hyperparameters, as it is highly situational. Instead, one must use a trial-and-error approach in deciding the optimal values. Some of the important hyperparameters associated with NNs are the number of hidden layers, number of neurons, number of epochs, and learning rate. We have elaborate on each hyperparameter more below.

Many hidden layers adding more hidden layers to the model may allow the network to recognize more complex mappings between the input and output layers. A network with two or more hidden layers is often considered deep learning [12] and is currently the most successful technique in machine learning. Adding many layers could improve the accuracy of the model to a certain degree. Adding too many layers might make the network too complex and consequently overfit the training data. Too many layers may also make the training time too long [13].

Number of neurons in each hidden is in a similar fashion dependent on how complex the given problem is. Too few neurons make the network unable to capture all the information. Moreover, the more training data and input/output mappings the network must learn, the more neurons are required. However, too many neurons make the network more prone to learning unique features of the training set, and may thus start overfitting.

Number of epochs, one epoch correspond to one iteration through all pairs of input/output mappings. This is rarely enough for minimizing the cost function. The network may therefore need many epochs of training before all knowledge has been learned. Too many epochs, however, may lead to overfitting [13].

Learning rate, during the learning phase the weights and thresholds are iteratively updated in order to minimize the cost function. The learning rate decides how quickly these parameters are to be updated, by limiting the amount of change by a certain factor. A learning rate that is too low makes the training very slow. A learning rate that is too high makes the algorithm overshoot, which increases the risk of missing the global minima of the cost function.

2.1 Recurrent Neural Network

A Recurrent Neural Network is a type of neural network which accepts variable-length input and produces variable-length output. It is used to develop various applications such as text-to-speech, chat-bots, language modeling, sentimental analysis, time series stocks forecasting, and machine translation [14]. A recurrent neural network is a recurrent neural network whose current output not only depends on its present value but also past inputs, whereas for a feed-forward network current output only depends on the current input.

In a neural network, inputs and outputs are considered independent of each other. As the sequential pattern exists in time series data, such a neural network does not give efficient results for time series forecasting. As an alternative network, the recurrent neural network is more effective to learn the dependency between observations. In the below diagram Figure 2.2, A chunk of the neural network, A, looks at some input X_t and outputs a value h_t. A loop allows information to be passed from one step of the network to the next.

Recurrent neural networks are well-known to work well for learning tasks where the input data is sequential. Sharing weights between hidden units across each time step, the RNN architecture is a natural way to model time series data, where each time step of the input depends on those in the past. Furthermore, RNNs can handle variable-length inputs, eliminating the need for padding inputs. In contrast, multilayer perceptions or convolutional neural networks are constrained to fixed-size input fixed-size outputs, and several layers and computational steps.

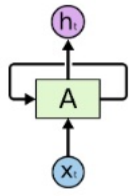


Figure 2.1: Recurrent neural network

The simple RNN is a network with loops that allows persisting information to be passed from one step of the network to the next. A loop allows information to be passed from one step of the network to the next. A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor. RNNs are fit and make predictions over many time steps. As the number of time steps increases, the simple diagram with a recurrent connection begins to lose all meaning. We can simplify the model by unfolding or unrolling the RNN graph over the input sequence. This looping process can be unrolled as described in

Figure 2.2 The process is illustrated for the time-steps from $0, 1, 2 uptotimet$ for $X_0, X_1, X_2, \ldots, X_t$ are the inputs, and $y_0, y_1, y_2, \ldots, y_t$ are respectively the outputs. At the hidden state is an activation function that takes its input from the hidden state of the previous step H_1 and the output of the current step X_t.

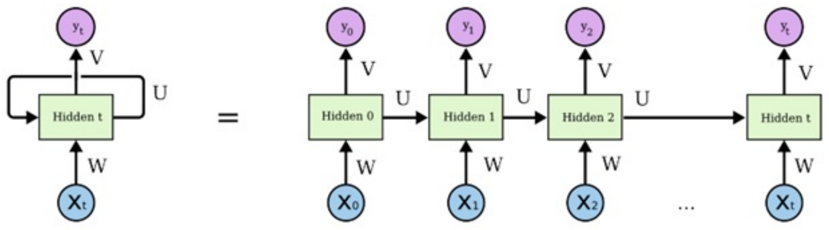


Figure 2.2: Recurrent neural network model folded and unfolded states. Figure from https://colah.github.io/posts/2015- 08-Understanding-LSTMs/

From Figure 2.2, the Section illustrates the folded and unfolded state of the RNN network. The unfolded state of RNN simply illustrates the network of the complete sequence. From the unfolded section, we can see that it is a $t - hidden$ layer neural network. And this can be referred to as deep learning neural network because it has more than one hidden layer. U, V and W are representing weight of neurons. X_t is the input at time step t and Y_t is output at time step t.

RNNs consists of an input layer, an output layer, and a recurrent layer, as depicted in 2.7. They comprised a series of weight matrices and activation functions. Explicitly, the set of equations that maps a set of inputs x to predicted outputs y is.

$$\begin{aligned} h_1 &= f(Wx + b_h) \\ h_i &= f(Wx + Uh_{i-}1 + b_h) \\ y &= softmax(Vh_i) \end{aligned} \tag{2.7}$$

Where b are bias vectors, and W, U, and V are weight matrices shared across time steps. Over the course of training, the model learns which setting of weight matrices W, U, V will minimize an overall loss function. To update these weights, RNNs use backpropagation through time (BPTT) to optimize weights during training. BPTT uses the chain rule to go back from the latest time step to the previous steps and the gradients tend to get smaller and smaller while moving backward in the network [15].

2.1.1 Vanishing and Exploding Gradient

When training RNNs with multiple layers a problem of the Vanishing Gradient can arise. The problem is as the name suggests when the gradient vanishes meaning that it is close to zero[Wang(2019)]. The problem arises if many gradients are close to zero because the gradient is a product of previous gradients. The problem usually occurs in the earlier layers (gradients computed using backpropagation). That is a problem because the weight updates are the subtraction of the gradients multiplied by the learning rate, from the weights themselves. That means that if the gradient is close to zero, the weight updates will have little to no effect at all. That also alleviates the exploding gradient problem, which is the opposite of the vanishing gradient problem, when the gradient reaches an astronomically high value because the gradient can never be higher than one [16].

2.1.2 Long Short-Term Memory

The Long Short-Term Memory (LSTM) network is different from a classical multilayer perceptron. Like a multilayer perceptron, the network is comprised of layers of neurons. Input data is propagated through the network to make a prediction. Like RNNs, the LSTMs have recurrent connections so that the state from previous activations of the neuron from the previous time step is used as context for formulating an output. But unlike other RNNs, the LSTM has a unique formulation that allows it to avoid problems that prevent the training and scaling of other RNNs. The impressive results that were achieved are the reason for the popularity of the technique [16].

The Long Short-Term Memory was introduced by Sepp Hochreicher and Jürgen Schmidhuber in 1997 (Hochreicher & Schmidhuber, 1997)[1]. Over the years it has become one of the most famous RNNs and is still a popular choice for tasks such as handwriting recognition (Graves Schmidhuber, 2009), music composition, traffic forecast (Zhao, et al., 2017), or other sequential problems. In a paper written by researchers from Google (Jozefowicz, et al., 2015) the LSTM is forwarded as an extremely powerful and applicable framework for a broad variety of machine learning tasks. The

[1]https://ieeexplore.ieee.org/abstract/document/6795963

reason for becoming so popular is attributed to the resilience to the exploding and especially the vanishing gradient problem. In 2015 large tech giants revealed that LSTM-based networks were used in their technology. For example, Google started using LSTM for speech recognition in 2015 on Google Voice, and in 2016 they started using it for their translation system reducing translation errors by 50% (Highfield, 2015). Similarly, in 2016 Apple and Amazon started using LSTM-based networks for translations, Quick typing, and text-to-speech technology in their devices.

The key technical historical challenge faced by RNNs is how to train them effectively. Experiments show how difficult this was where the weight update procedure resulted in weight changes that quickly became so small as to have no effect, vanishing gradients, or so large as to result in a very large change or even overflow, exploding gradients. LSTMs overcome this challenge by design. "The Long Short Term Memory architecture was motivated by an analysis of error flow in existing RNNs which found that long-time lags were inaccessible to existing architectures because backpropagated error either blows up or decays exponentially. An LSTM layer consists of a set of recurrently connected blocks, known as memory blocks. These blocks can be thought of as a differentiable version of the memory chips in a digital computer. Each one contains one or more recurrently connected memory cells and three multiplicative units the input, output, and forget gates that provide continuous analogs of write, read and reset operations for the cells. The net can only interact with the cells via the gates [17].

2.1.3 LSTM architecture

LSTM resolves the deficiency of conventional RNNs by being able to learn long-term dependencies. Another difference is that conventional RNNs have only a single neural network layer. The drawing below figure shows the LSTM architecture unrolled over time. The cell to the left is the LSTM cell at the previous time step while the one to the right is the cell at one step into the future. The current time step is in the middle. Three lines go into the cell. In the bottom left corner, it receives the input xt and the output from the previous timestep (the output from the previous layer is in RNNs called the hidden state abbreviated to h_{t-1}. The input x_t and the hidden state h_{t-1} concatenated before it runs into the four gates marked as yellow boxes in the drawing. The third input the cell receives from the previous cell runs as a straight arrow through the upper part of the cells. That is the cell state and enables the LSTM to remember long-term dependencies with a considerably smaller chance for the vanishing and exploding gradient problems seen in RNN.

LSTMs are very impressive. The design of the network overcomes the technical challenges of RNNs to deliver on the promise of sequence prediction with neural networks. The applications of LSTMs achieve impressive results on a range of complex sequence prediction problems. Furthermore, the key benefits of LSTM are that it can overcome the technical problems of training an RNN, namely vanishing and exploding gradients. Possesses memory to overcome the issues of long-term temporal dependency with input sequences. Process input sequences and output sequences time step by time step, allowing variable length inputs and outputs. The forget gate and input gate are used in the updating of the internal state. The output gate is a final limiter on what the cell outputs. It is these gates, and the consistent data flow called the constant error carrousel that keep each cell stable (neither exploding nor vanishing).

Each memory cell's internal architecture guarantees constant error within its constant error carrousel CEC. That represents the basis for bridging very long time lags. Two gate units learn to open and close access to error flow within each memory cell's CEC. The multiplicative input gate affords protection of the CEC from perturbation by irrelevant inputs. Likewise, the multiplicative output gate protects other units from perturbation by currently irrelevant memory contents [18].

2.1.4 LSTM weights

A memory cell has weight parameters for the input, and output, as well as an internal state that is built up through exposure to input time steps. Input Weights are used to weigh input for the current time step. Output Weights are used to weigh the output from the last time step. The internal state is used in the calculation of the output for this time step.

2.1.5 LSTM Gates

The key to the memory cell is the gates. These too are weighted functions that further govern the information flow in the cell. There are three gates and Forget Gate decides what information to discard from the cell. Input Gate decides which values from the input to update the memory state. Output Gate decides what to output based on input and the memory of the cell.

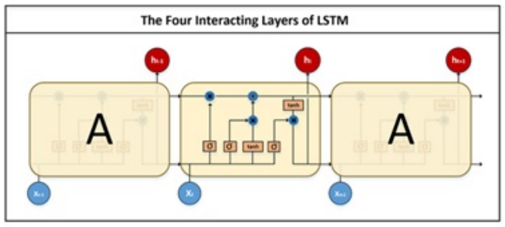


Figure 2.3: Overview of the LSTM network (Olah, 2015)

The cell state is not more than a vector in a mathematical sense the cell state can be thought of as a highway for information that runs through the whole chain of cells with only some linear interactions. It might be one of the most central components and it allows the LSTM to remember long-term input dependencies. It is possible to read write and delete information from this internal memory. The key to solving the vanishing gradient problem is that the new information is added not multiplied to the cell state. Addition distributes gradients equally and the chain rule does not apply within the back-propagation [18].

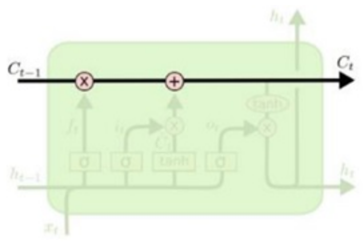


Figure 2.4: cell state of the LSTM network

Within the LSTM cell, four neural network layers have their special function. The sigmoid function that the three of these layers are nested within, outputs matrices with values between 1 and 0. Sigmoid function for gates because we want a gate to give only positive values and should be able to give us a clear-cut answer whether we need to keep a particular feature or we need to discard that feature. “0” means the gates are blocking everything.

"1" means gates are allowing everything to pass through them.

Erasing information from the cell state, the first gate is the forget gate. It considers the current inputs X_t and the output from the previous timestep h_{t-1}. The product of the current input and the weights (a neural layer) is squashed into the sigmoid function that transforms this layer into a matrix with values between 1 and 0. From here the cell state from the previous cell is multiplied element-wise with the forget gate. One may think of the forget gate as a filter, that erases or decreases values that we want to delete or degrade from the previous cell state.

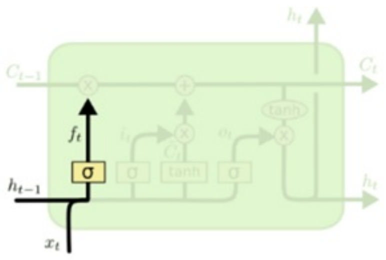


Figure 2.5: forget gate state

The forget gate produces a matrix with values between 0 and 1:

$$f(t) = \sigma(W_i \times \lceil h_{t1}, x_t \rceil + bf) \tag{2.8}$$

Then this is passed up to update the Cell state, the cross above the arrow. Let's say the state from the previous timestep is $C_{t-1} = [2, 4, 6]$ and the above computation gives $f(t) = [1, 0, 1]$ then by element-wise multiplication the number 4 is deleted from the memory. Adding new information to the cell state, the next step includes the gate with the sigmoid and the tanh gate. Namely the first input and the second input gates. Similarly, to the forget gate an input gate act as a filter on the tanh layer. It ranges between 0 and 1 and decides how large proportions of that information should be stored.

$$i_t = \sigma(W_i \times \lceil h_{t1}, x_t \rceil + b_i f) \tag{2.9}$$

The other gate with the *tanh* activation function is creating candidates for the new cell state values (the internal memory) it uses the hyperbolic tangent that ranges between -1 and 1. Apparently because when new candidate values are added to the cell state it can both add and subtract some information. The equation for the new candidate values C^t is given below. A closer look reveals that this equation is equal to the core of the simple recurrent neural network drawn previously.

$$C^t = \tanh(W_i \times [h_{t1}, x_t] + bC) \tag{2.10}$$

Further, one calculates the element-wise multiplication of the possible candidate values with the above input filter.

$$C_{t-i} = C_t \times i_t \tag{2.11}$$

In total the modifications to the cell state is:

$$C_t = f_t \times C_{t-1} + i_t \times C_t \tag{2.12}$$

Then finally, we decide what to output from the cell, this is either the final output or the next hidden state to the next cell depending on the time steps. The inputs are the previous hidden state and the inputs go into an ordinary neural layer. A sigmoid activation is used before this output it is point-wise multiplied with the cell state that has been squashed in a tanh layer to give a vector of values between -1 and 1. In this operation, the cell state filters the output. It seems therefore that the previous output and the current input are most important but that the cell state may modify the final output by multiplying the output with either positive or negative values.

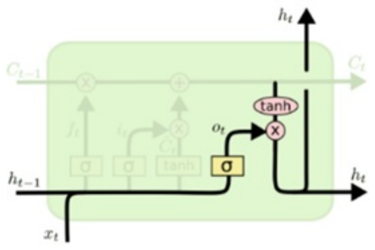


Figure 2.6: output operation

2.1.6 Gated Recurrent Unit

Gated Recurrent Unit, GRU, is a more recent and simpler architecture that is based on the principles of the LSTM with gated units. GRU is an RNN architecture that controls the flow of information by using the reset and update gates. The reset gate decides which information to use and which information to throw away and it can be considered a mix of the LSTM forget and input gates. The Gated recurrent unit,Cho et al., 2014a[2], is a slightly more streamlined variant that often offers comparable performance and is significantly faster to compute [19].

2.2 Tsetlin machine

The Tsetlin Machine is a pattern recognition algorithm introduced by Ole-Christoffer Granmo in his 2018 paper [20]. The paper explains how the Tsetlin Machine can perform complex pattern recognition using a collective of Tsetlin Automata. The Tsetlin Automaton is a method of solving the multi-armed bandit problem from game theory. It identifies these patterns using propositional logic [20], which can also be used for interpretation. It is structured in sub-sections to make each part of the Tsetlin Machine more understandable, starting with the necessary basics and progressing to the more complicated components.

The TM, introduced in 2018 by Granmo, uses the TA as a building block to solve complex pattern recognition tasks. The TM operates as follows. Firstly, propositional formulas in disjunctive normal form are used to represent patterns. The TM is thus a general function approximation. The propositional formulas are learned through training on labeled data by employing a collective of TAs organized in a game. As a result, the architecture of the TM is relatively simple, facilitating transparency and interpretation of both learning and classification. Additionally, the TM is designed for bitwise operation. That is, it takes bits as input and uses fast bit manipulation operators for both learning and classification. This gives the TM an inherent computational advantage. Experimental results show that TM outperforms Support Vector Machines, Random Forests, and Logistic Regression in diverse benchmarks These promising properties and results make the TM an interesting target for further research [3].

[2]https://arxiv.org/pdf/1406.1078.pdf
[3]https://arxiv.org/pdf/1905.04206

2.2.1 Tsetlin Automaton

The Tsetlin Machine is a recent pattern classification method that manipulates expressions in propositional logic based on a team of Tsetlin Automata (TAs) [20]. The basic building block of the Tsetlin Machine is the Tsetlin Automaton. A Tsetlin Automaton (TA) is a fixed structure deterministic automaton that learns the optimal action among the set of actions offered by an environment. Figure 2.7 shows a two-action TA with $2N$ states. The action that the TA performs next is decided by the present state of the TA. States from 1 to N map to Action 1, while states from $N + 1$ to $2N$ map to Action 2. The TA iteratively interacts with its environment. In each iteration, the TA performs the action associated with its current state.

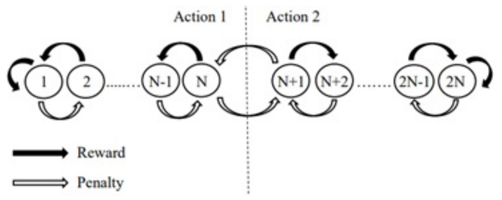


Figure 2.7: A tsetlin automaton for two-action environments from O.C Granmo(2018)

There are two possible actions for Tsetlin Automaton and acting will result in either reward or penalty. The agent of the Automaton will have a state in the range 1 to $2N$. If the agent is in the state range of 1 to N, it will act as 1. Consequently, it will act 2 when in the range $N + 1$ to $2N$, as seen in Figure 2.7. It will then check if the action performed was correct concerning the ground truth. If it was, it gets a reward, and the agent will update its state such that it moves further towards state 1 if the state is in the range 1 to N or towards 2N if the state is in the range $N+1$ to $2N$. This reward and penalty system will show how strongly the automata prefer one action. The further toward one side the agent is, the more secure it is that the given action will give the overall best yield. When transitioning from action 1 to action 2 or reverse, it is always the result of a penalty. This is the way Tsetlin Automata tries to solve explore and exploit the problem.

TM structure

Consider an input feature vector $X = (x_k) \in \{0, 1\}^{\circ}$ consisting of $\circ$ propositional variables x_k with domain $\{0, 1\}$. The TM considers both the features

x_k themselves as well as their negations $\neg x_k$, jointly referred to as literals when forming the clauses. Each clause C_j takes the following form.

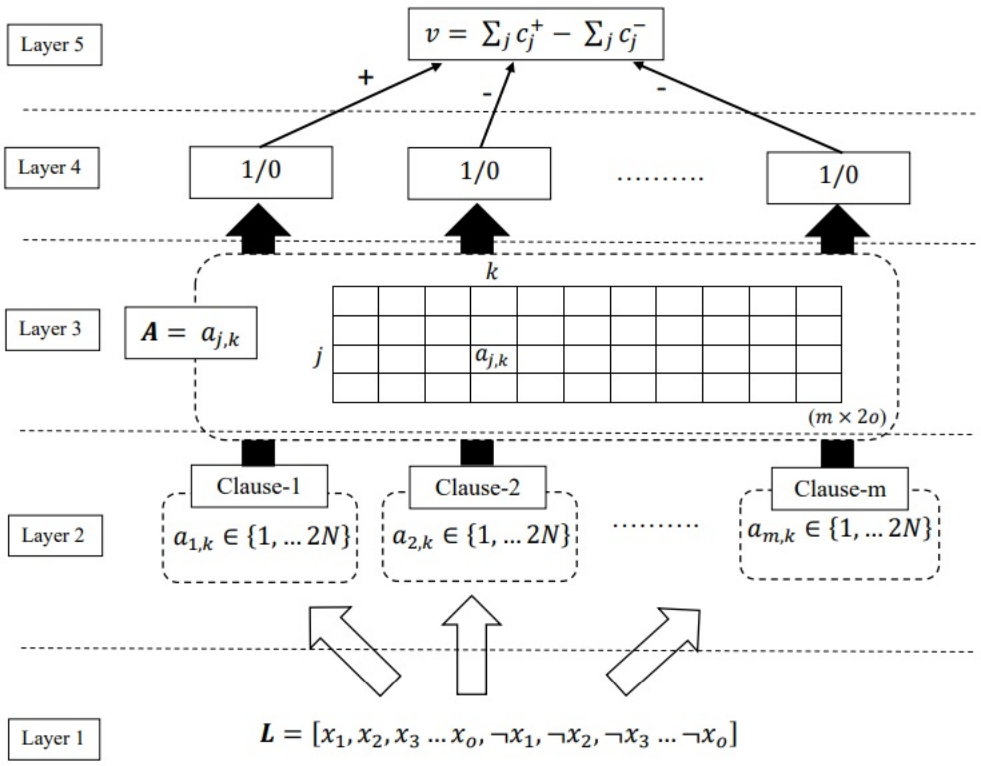

Layer 5
$v = \sum_j c_j^+ - \sum_j c_j^-$
+
-
-
Layer 4
1/0
1/0
1/0
Layer 3
$A = a_{j,k}$
k
j
$a_{j,k}$
$(m \times 2o)$
Layer 2
Clause-1
Clause-2
Clause-m
$a_{1,k} \in \{1, \ldots 2N\}$
$a_{2,k} \in \{1, \ldots 2N\}$
$a_{m,k} \in \{1, \ldots 2N\}$
Layer 1
$L = [x_1, x_2, x_3 \ldots x_o, \neg x_1, \neg x_2, \neg x_3 \ldots \neg x_o]$

Figure 2.8: Tsetlin Mahcine structure, source: [21]

Input data

The Tsetlin Machine will use some given binary data $X = [x_1, x_2, \ldots, x_n], x_p \in \{0, 1\}$. The data will be used to create propositional patterns and thus must be binary so that it is equivalent to a vector of propositional variables. The counterpart to each of the propositional variables forms together the literal set $L = [l_1, l_2, \ldots, l_{2n}] = [x_1, x_2, \ldots, x_n, \neg x_1, \neg x_2, \ldots, \neg x_n]$ meaning L is double the length of X. The Tsetlin Machine learns by creating patterns through ANDing a subset of the literals $L_j \subseteq L$ into conjunctive clauses denoted as C_j, with j as the index of the clause.

Clause construction

A clause construction layer and the sub-patterns associated with class 1 and class 0 are captured by m conjunctive clauses. The value m is set by the user where more complex problems might demand a large m. All clauses receive the same augmented feature set formulated at the input layer, L. However, to perform the conjunction, only a fraction of the literals are utilized. The TM employs two-action TAs in Figure 2.8 to decide which literals are included in which clauses. Since we found a number $2\circ$ of literals in L, the same number of TAs one per literal k is needed by a clause to decide the included literals in the clause. When the index set of the included literals in clause j is given in I_j, the conjunction of the clause can be performed as follows [22].

$$C_j = \wedge_{k \in I_j} l_k \tag{2.13}$$

Notice how the composition of a clause varies from another clause depending on the indexes of the included literals in the set $Ij \subseteq 1, \ldots, 2\circ$. For the special case of $Ij = \emptyset$ an empty clause we have that is, during learning, empty clauses output 1, and during classification, they output 0.

$$C_j = \begin{cases} 1 & duringlearning \\ 0 & otherwise \end{cases}$$

Storing states of TA of Clauses

The TA states on the left-hand side of the automaton (states from 1 to N) ask to *exclude* the corresponding literal from the clause while the states on the right-hand side of the automaton (states from $N+1$ to $2N$) ask to *include* the literal in the clause. The systematic storage of states of TAs in the matrix, A: $A = (a_{j,k} \in \{1, \ldots 2N\}^{mx2\circ})$, with j referring to the clause and k to the literal, allows us to find the index set of the included literals in clause j, I_j as $I_j = \{K | a_{j,k} > N, 1 \leq k \leq 2\circ\}$.

Clause output

Once the TA decisions are available, the clause output can be easily computed. Since the clauses are conjunctive, a single literal of value 0 is enough to turn the clause output to 0 if its corresponding TA has decided to *include* it in the clause.

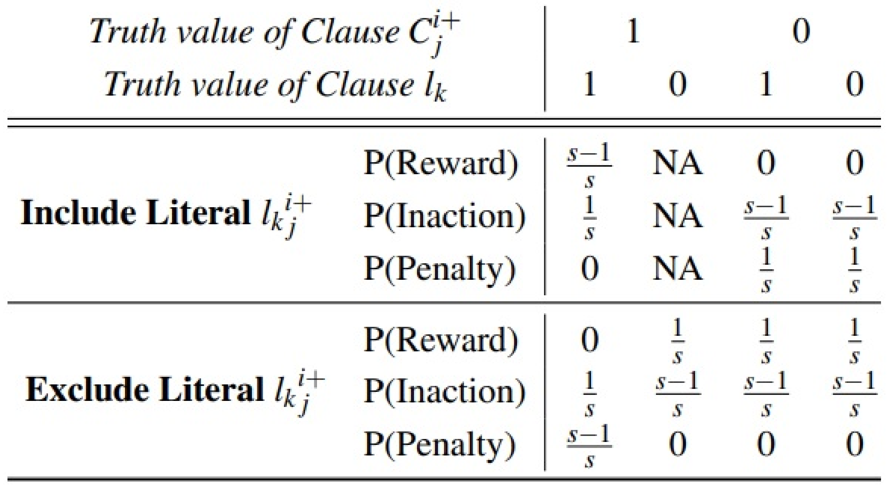
Classification

The classification stage is used to classify data into two classes. Hence, sub-patterns associated with each class have to be separately learned. For this purpose, the clauses are divided into two groups, where one group learns the sub-pattern of class 1 while the other learns the sub-patterns of class 0. For simplicity, clauses with an odd index are assigned a positive polarity (C_j^+) and they are used to capture sub-patterns of output $y = 1$. Clauses with an even index, on the other hand, are assigned negative polarity (C_j^-) and they seek the sub-patterns of output $y = 0$. The summation operator at the end sums the votes for each class separately and considers the difference to decide the final output. $v = \sum_j c_j^+ - \sum_j c_j^-$ then the final output is decided.

$$y = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{if } v < 0 \end{cases}$$

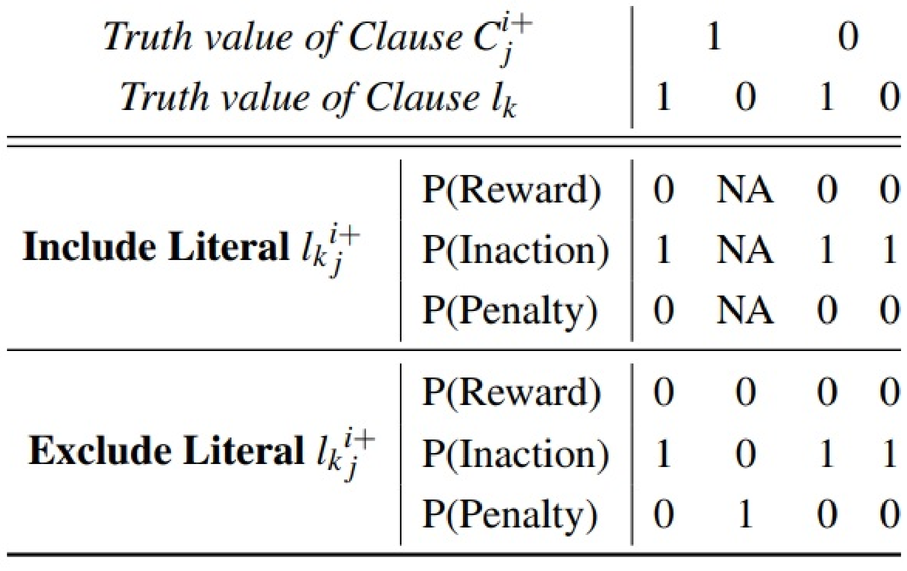
Learning procedure

TAs adjust their states based on feedback from the game (environment), encompassing rewards, penalties, and inaction feedback. To achieve the learning goal, the probabilities of receiving this different feedback have been designed to account for critical factors, namely, the actual output, the clause outputs, the literal values, and the current state of the TAs. In contrast to gradient-based learning, learning in TM naturally combats false positives and false negatives. As a consequence, it eludes the issues attached to gradient-based algorithms such as vanishing/exploding gradients. In the case of categorization problems with two classes, the basic idea is to penalize voters when they vote to procure a false positive or false negative and to reward voters when they vote to procure a true positive or true negative. In the TM, this is done by two types of feedback Type I and Type II [23]. An overview of the transition probability definitions in the feedback mechanism is shown in Tables 2.2.1 and 2.2.1.



Truth value of Clause C_j^{i+}		1		0	
Truth value of Clause l_k		1	0	1	0
Include Literal $l_{k_j}^{i+}$	P(Reward)	$\frac{s-1}{s}$	NA	0	0
	P(Inaction)	$\frac{1}{s}$	NA	$\frac{s-1}{s}$	$\frac{s-1}{s}$
	P(Penalty)	0	NA	$\frac{1}{s}$	$\frac{1}{s}$
Exclude Literal $l_{k_j}^{i+}$	P(Reward)	0	$\frac{1}{s}$	$\frac{1}{s}$	$\frac{1}{s}$
	P(Inaction)	$\frac{1}{s}$	$\frac{s-1}{s}$	$\frac{s-1}{s}$	$\frac{s-1}{s}$
	P(Penalty)	$\frac{s-1}{s}$	0	0	0

Table 2.1: Type I Feedback, as designed for the Classifying Tsetlin Machine Game [20]



Truth value of Clause C_j^{i+}		1		0	
Truth value of Clause l_k		1	0	1	0
Include Literal $l_{k_j}^{i+}$	P(Reward)	0	NA	0	0
	P(Inaction)	1	NA	1	1
	P(Penalty)	0	NA	0	0
Exclude Literal $l_{k_j}^{i+}$	P(Reward)	0	0	0	0
	P(Inaction)	1	0	1	1
	P(Penalty)	0	1	0	0

Table 2.2: Type II Feedback, as designed for the Classifying Tsetlin Machine Game [20]

2.2.2 Tsetlin Machine Classification

For categorization tasks with more classes than two by utilizing an *argmax* function the output of multi-class TM given by this.

$$\hat{y} = argmax_{i=1,\ldots,m} = (\sum_{j=1}^{n} c_j^{i+} - \sum_{j=1} c_j^{i-}) \tag{2.14}$$

where m is the number of distinct classes of objects specified in the problem and the $+$ and symbols indicate the clause polarity,include literals and exclude literals [20].

2.2.3 Tsetlin Machine Regression

To calculate continuous output, the target y is continuously valued, we remove the polarity of clauses since we intend to use the clauses as additive building blocks. That is, we intend to map the total vote count into a single continuous output. As a result, the complexity of the Regression Tsetlin Machine (RTM) is reduced. With merely one type of clause, the summation operator outputs a value between 0 and T, threshold, which is simply the number of clauses that evaluates to 1. This value is then normalized to produce the regression output. Thus, through this simple modification, the TM can now produce continuous output, with a precision that increases with higher T.

The sum of the votes from the clauses $\sum_{j=1}^{m} C_j$ of the TM is normalized to achieve the regression output by dividing by T and multiplying with $\hat{y}_{max}$. The TM output $y_\circ$ is calculated using this.

$$y_\circ = \frac{\sum_{j=1}^{m} C_j(\hat{X}_\circ)X\hat{y}_{max}}{T} \tag{2.15}$$

Feedback, then, is based on comparing the output, $y\circ$ of the TM with the target output $\hat{y}_\circ$. The target value $\hat{y}_\circ$ can be higher or lower than the output value $y\circ$. This is the basis for our new feedback scheme. That is, similarly to other machine learning methods, certain internal operations are needed to minimize the error between the predicted output, $y\circ$, and target output, $\hat{y}_\circ$. In the RTM, this is quite simply achieved by providing Type I and Type II feedback according to the following criteria.

$$Feedback = \begin{cases} TypeI, & \text{if } y_\circ < \hat{y}_\circ \\ TypeII, & \text{if } y_\circ > \hat{y}_\circ \end{cases}$$

The idea here is to increase the number of clauses that output 1 when the predicted output is less than the target output $y_\circ < \hat{y}_\circ$. To achieve this, we then provide Type I feedback. Conversely, Type II feedback is applied to decrease the number of clauses that evaluate to 1 when the predicted output is higher than the target output $y_\circ > \hat{y}_\circ$. And to stabilize learning, an activation probability function makes the probability of giving clause feedback proportional to the difference between the predicted and target output (the error). That is, in the RTM, feedback to clauses is determined stochastically using the following activation probability function P_{act}

$$P_{act} = \begin{cases} \frac{K \times |y_\circ - \hat{y}_\circ|}{\hat{y}_{max}} \end{cases}$$

The constant K is a scaling factor that adjusts the magnitude of the activation function, preventing severe oscillation between predictions.

2.2.4 Continuous input feature

A preprocessing procedure that transforms continuous features into binary variables, while maintaining ranking relationships among the continuous feature values. The preprocessing procedure follows the following steps to convert them into binary form, one feature at a time.

1. First, for each feature, the unique values are identified.
2. The unique values are then sorted from smallest to largest.
3. The sorted unique values are considered as threshold.
4. The original feature values are then compared with identified thresholds, only from their own feature value set. If the feature value is greater than the threshold, set the corresponding Boolean variable to 0, otherwise, set it to 1.
5. The above steps are repeated until all the features are converted into Boolean form.

Chapter 3

Traffic prediction

In this section describes the relationship between the transportation system and Machine learning.

3.1 Traffic flow

Traffic flow is a study of the movement of individual drivers and vehicles between cross-sections and the interactions they make with one another. A better understanding of traffic flow will enable authorities to design roads with an improved level of service, to improve the performance of existing transportation systems for instance operations, and to understand how the system might respond to potential engineering changes.

It is appealing to detect traffic flow information in an accurate and timely manner [24]. Over a years, with the increasing population, traffic flow has also increased. That leads to road accidents and delays in arrival time. ML techniques have proved capable of solving traffic flow patterns and have contributed to the development of ITS [1]. ML approaches such as k-Nearest Neighbors and Support Vector Machine Regression is used to address traffic flow detection problems. Other ML approaches like SVM are also used to predict the travel time of the road segment. The SVM algorithm used for regression problems like detecting traffic flow and predicting speed at randomly selected roads [1]. Travel time detection of road segments is also an important contribution when developing an intelligent transportation system. Public transportation such as trains and buses can utilized effectively if there is an efficient system in place to detect traffic flow and travel time estimation. Several ML techniques such as SVM and deep learning methods such as Long Short-Term Memory (LSTM) have been applied in an area of travel time prediction. Deep Learning methods have attracted a lot of attention nowadays as methods are showing tremendous results in the transportation network [1].

Neural networks have obtained astounding success for important pattern recognition tasks, but they suffer from high computational complexity and a lack of interpretability. Furthermore, the recent Tsetlin Machine, TM, attempts to address this lack by using easy-to-interpret to solve complex pattern recognition problems. The TM provides competitive accuracy in several benchmarks while keeping the important property of interpretability. It further facilitates hardware-near implementation since inputs, patterns, and outputs are expressed as bits, while recognition and learning rely on straightforward bit manipulation. “If the next step in the research confirms my findings, this research can be called ‘groundbreaking’.” Professor Ole-Christoffer Granmo[1]. Tsetlin machine is a quicker simple solution and more precise than most neural network vanilla methods. Additionally, as well as it is a completely new tool for artificial language, picture understanding, pattern recognition, reasoning, planning, and diagnostics.

For example, if Google predicts that traffic is likely to become heavy in one direction, they will automatically find us a lower-traffic alternative. They also look at several other factors, like road quality. Is the road paved or unpaved, covered in gravel, dirt, or mud? Elements like these can make a road difficult to drive down, and they are less likely to recommend this road as part of route. They also look at the size and directness of a road driving down a highway is often more efficient than taking a smaller road with multiple stops. Two other sources of information that are important to make sure they recommend the best routes are authoritative data from local governments and real-time feedback from users. Authoritative data lets Google Maps know about speed limits, tolls, or if certain roads are restricted due to things like construction. And incident reports from drivers let Google maps quickly show if a road or lane is closed, if there is construction nearby, or if there is a disabled vehicle or an object on the road. Both sources are also used to help users understand when road conditions change unexpectedly due to mudslides, snowstorms, or other forces of nature [25].

[1]https://arxiv.org/abs/1804.01508

3.2 Transportation domain

AI and ML have shown promising results in the applications of transport system. When applied to the transportation system, these technologies can improve the quality of lifestyle and provide safety along with the ease of quicker transportation services to people [26]. AI and ML techniques have also been used in the overall development of smart cities. Advancement in AI has given a rise to the introduction of self-driving cars for people who can take advantage of this latest technology. Autonomous vehicles or self-driving cars use a combination of the latest technologies like a sensor, camera, radar, and AI techniques to move around from one location to another [27]. Autonomous vehicles require a lot of trust from the people since there is always a question of safety involved with these vehicles. Therefore, a combination of ML, deep learning, and AI techniques need to prove the safety and reliability of such vehicles.

The combined development of different emerging technologies smart sensors, artificial intelligence boost innovations in transportation systems. The increasing pressure on achieving societal goals within the transport sector for example de carbonization, improving traffic safety, reducing congestion will be another driver for the developments in transportation sector.

3.3 Intelligent transport system

Intelligent Transport Systems (ITS) can be defined as holistic, control, information and communication upgrade to classical transport and traffic systems, which enables significant improvement in performance, traffic flows, efficiency of passenger and goods transportation, safety, and security of transport, ensures more comfortable travelling for passengers, reduces pollution, etc. ITS presents a crucial breakthrough by changing approaches and trends in transport and traffic research and technology aiming to solve escalating problems of congestions, pollution, transport efficiency, safety and security of passengers and goods [27]. This will also prove by numerous AI and machine learning projects related to ITS all over the world.

Recently, the European Union has made some significant efforts in the field of ITS deployment trying to find solutions for the escalating transport and traffic problems. A substantial number of activities has been stipulated by different European bodies with the single objective to enforce the practical ITS deployment all over the Union [?]. An explicit objective of the transportation system is to provide a safe environment for travel while continuing to strive to improve the performance of the system. Although undesirable, crashes and fatalities are inevitable occurrences. Several ITS services aim to minimize the risk of crash occurrence. This objective focuses on reducing the number of crashes and reducing the probability of a fatality should a crash occur. Typical measures of effectiveness used to quantify safety performance include the overall crash rate, fatality crash rate and injury crash rate. ITS services should also strive to reduce the crash rate of a facility or system. The need for ITS is rising linearly with the increase in the population. Hence, the transportation system should be safer, faster, reliable, environment friendly, and cost effective [28].

The other major benefits of ITS can be improving mobility and reliability by reducing delay and travel time is a major objective of many ITS components. Delay can be measured in many different ways, depending on the type of transportation system being analysed. Delay of a system is typically measured in seconds or minutes of delay per vehicle. Also, delay for users of the system may be measured in person-hours. Delay for freight shipments could be measured in time past scheduled arrival time of the shipment. Delay can also be measured by observing the number of stops experienced by drivers before and after a project is deployed or implemented. Travel time variability indicates the variability in overall travel time from an origin to a destination in the system, including any modal transfers or end route stops. This measure of effectiveness can readily be applied to intermodal goods movement as well as personal travel. Reducing the variability of travel time improves the reliability of arrival time estimates that travellers or companies use to make planning and scheduling decisions. By improving operations and incident response, and providing information on delays, ITS services can reduce the variability of travel time in transportation networks. For example, traveller information products can be used in trip planning to help re-route commercial drivers around congested areas resulting in less variability in travel time.

Most ITS components seek to optimize the efficiency of existing facilities and use of rights-of-way so that mobility and commerce needs can be met while reducing the need to construct or expand facilities. This is accomplished by increasing the effective capacity of the transportation system. Effective capacity is the maximum potential rate at which persons or vehicles may traverse a link, node or network under a representative composite of roadway conditions, including weather, incidents and variation in traffic demand patterns (McGurrin and Wunderlich, 1999). Capacity, as defined by the Highway Capacity Manual, is the maximum hourly rate at which persons or vehicles can reasonably be expected to traverse a given point or uniform section of a lane or roadway during a given time period under prevailing roadway, traffic and control conditions, (TRB, 2000). The major difference between effective capacity and capacity is that capacity is measured under typical conditions for the facility, such as good weather and pavement conditions, with no incidents affecting the system, while effective capacity can vary depending upon these conditions and the use of management and operational strategies. Throughput is defined as the number of persons, goods or vehicles traversing a roadway section or network per unit time. Increases in throughput are sometimes realizations of increases in effective capacity. Under certain conditions, it may reflect the maximum number of travellers that can be accommodated by a transportation system. Throughput is more easily measured than effective capacity and, therefore, can be used as a surrogate measure when analysing the performance of an ITS project.

The use of vehicles and other transportation means has increased which has led to traffic congestion and road accidents. Hence, there is a demand for intelligent transportation systems in the country that can provide safe and reliable transportation while maintaining environmental conditions such as pollution, CO2 emission, and energy consumption. An application of Artificial intelligence and Machine Learning can be applied to develop an Intelligent Transportation system that can address the issues of traffic congestion and road safety to prevent accidents. Various ML approaches to detect road anomalies for avoiding obstacles, predict real-time traffic flow to achieve smart and efficient transportation, detect and prevent road accidents to ensure safety.

Transportation difficulties can become a major challenge especially when the network and users activities are too difficult to predict and model the patterns in travel. Thus, to overcome the challenges of increasing travel demand, environmental degradation, safety concerns, and CO2 emissions, AI, and ML are deemed to be a perfect fit for transportation systems. In developing countries, the steady growth of urban and rural traffic due to the increasing population is the main cause of these challenges. For instance, in Australia, by 2031, the population is expected to increase to 30 million, and therefore the cost of congestion is expected to reach 53.3 billion [?].

In the twenty-first century, several researchers are attempting to achieve a smarter and reliable transportation system or in other words, Intelligent Transportation Systems (ITS). ITS will have less detrimental effects on the environment, and people will be using AI and ML techniques that are more reliable and cost-effective [29].

Chapter 4

Results and discussion

In this section, we present a result. Predicting the accuracy of each model for the evaluation with MAE as the accuracy metric. From our test the best results for each likelihood plotted in the figures below. The predicted traffic flow plotted against the actual observation traffic flow for contrast. The dataset and code is available in this repository [30].

4.1 Hardware and software

The analysis in this thesis has done on most modern personal computers. The experiments were run on a laptop with Intel(R) Core(TM) $i5 - 7200U$ CPU processor at 2.50GHz. The RAM of the laptop was 8.00GB and running Windows 10 as the operating system. Furthermore, we have used a friend's MacBook Pro, apple M1 new chip 8GB RAM, and 256GB SSD storage.

To run all the necessary programs for the analysis. The thesis used Python version 3.7. The primary development environment used was Jupyter Notebook. That provided a convenient and structured way of implementing the various steps in the working procedure described in the previous section. Several python libraries have been utilized in different ways.

NumPy supports a wide range of hardware and computing platforms, and plays well with distributed, GPU, and sparse array libraries. Fast and versatile, the NumPy vectorization, indexing, and broadcasting concepts are the de-facto standards of array computing today [31].

Pandas used for data analysis and statistics. Pandas is part of the Anaconda distribution and can be installed with Anaconda or Miniconda. Pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with relational or labeled data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real-world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open source data analysis or manipulation tool available in any language [32].

Scikit-learn is an open source machine learning library that supports supervised and unsupervised learning. It also provides various tools for model fitting, data preprocessing, model selection and model evaluation. [33].

pyTsetlinMachine used to implements the Tsetlin Machine, Embedding Tsetlin Machine, Convolutional Tsetlin Machine, Regression Tsetlin Machine, and Weighted Tsetlin Machine, with support for continuous features, multigranularity, clause indexing, and drop clause/literal [34]

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python [35].

Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library. It supported multiple backends, including TensorFlow, Microsoft Cognitive Toolkit and Theano. Keras contains numerous implementations of commonly used neural-network building blocks such as layers, activation functions, optimizers, and a host of tools to make working with image and text data easier to simplify the coding necessary for writing deep neural network [36].

4.2 Data preparation and dataset

In our thesis, we used the dataset to predict the daily traffic flow. The dataset provides the number of hourly traffic flow from January 2016 to October 2018, or just under three years of data. We started with a simple forecast model to provide a baseline of performance for more sophisticated methods to improve upon. Other hyperparameters result in a well-performing model of both LSTM and GRU, but our intention in this thesis is to compare Tsetlin Machine with Neural Network. Furthermore, In the test section, we saved the final five months of the original dataset in a separate file to validate the final model. We used this file and used it to see how well the models are on unseen data.

Feature Selection is one of the core concepts in machine learning which hugely impacts the performance of a model. The data features that we use to train our models have a huge influence on the performance we can achieve. We looked at how certain features affect predictions. Determining which features yield the most predictive power is another crucial step in a model-building process. Moreover, Temperature, holiday, and weather features have a large effect on our model predictions while others have not. Additionally, we have created new features from existing ones. We have done this by simple mathematical operations by aggregations to obtain the mean and allocated *zero* if its value is less than aggregated mean value otherwise we assigned *one*.

There is a mechanism to transform continuous features into binary values while maintaining ranking relationships among the continuous feature values. It is based on sorting the identified unique values from smallest to largest. But, It is significant that the time series has not shuffled, as this will remove its temporal structure.

4.2.1 Time series

Time series datasets may contain trends and seasonality, which may need to be removed before modeling. Trends can result in a varying mean over time, whereas seasonality can result in a changing variance over time, both of which define a time series as being nonstationary. Stationary datasets are those that have a stable mean and variance and are in turn much easier to model. There are many types of seasonality. Some obvious examples include the time of day, daily, weekly, monthly, and annual in our datasets. So, identifying whether there is a seasonality component in our time series problem is subjective. The simplest approach to determining if there is an aspect of seasonality is to plot and review our data, perhaps at different scales and with the addition of trend lines.

Data split

It is common to split the dataset into multiple parts, but we split the time series into a training set 80% and a test set 20%. The training set is used the train the model to learn the temporal patterns throughout the time series and the test set is used to measure how well trained model can perform prediction on previously unseen data. Moreover, it is important that the time series is not shuffled, as this will remove its temporal structure. Also, the test set must arrive after the training set to avoid forecasting the past.

4.3 Tsetlin Machine classification

We used classification methods to determine high-volume of traffic flow and low-volume traffic flow. We used mean values to distinguish between high and low values. We describe the interpretability of the model below subsection.

4.3.1 Interpretability

We present the performance of the Tsetlin Machine on the traffic dataset and its interpretability. The Tsetlin Machine has three hyperparameters needed to be defined by the user and fine-tuned as required. In our case, we use 9000 clauses, C, threshold T of 500, and specificity s of 32. The parameters are responsible for avoiding over-fitting as well as deciding how many literals can be included and excluded from the clause. We have manually assigned these parameters as the main aim is to demonstrate the interpretability of the model. During training, the TM arrives at a set of clauses using the features provided, which together describe the task in general. During testing, each sample can match only a subset of all clauses, and these clauses define the classification problem concerning that particular sample only. In our case, we use high traffic volume as one and low traffic volume as zero. For instance, for the high volume of traffic, it looks like, if $\mathsf{high\_temperature} \land \mathsf{clear} \land \mathsf{not\_holiday} \land \mathsf{not\_rainwithinhours}$, then it can be potential for the high volume of traffic.

The sub-patterns it has learned during training. They are literal forms from the binarized input data, being shown as conjunctive literal. The Tsetlin Machine learns this and utilizes conjunctive clauses to represent the particular facets of each category. For instance, the sub-patterns formed clauses for the class of high traffic volume.

C1: high_temperature $\wedge$ clear $\wedge$ not_holiday $\wedge$ not_rainwithinhours

C2: high_temperature $\wedge$ snow $\wedge$ holiday $\wedge$ rainwithinhours

C3: high_temperature $\wedge$ rain $\wedge$ not_holiday $\wedge$ rainwithinhours

C4: low_temperature $\wedge$ rain $\wedge$ holiday $\wedge$ rainwithinhours

C5: low_temperature $\wedge$ clear $\wedge$ not_holiday $\wedge$ not_rainwithinhours

Similarly, clauses for the class of low volume of traffic,
C1: high_temperature $\wedge$ clear $\wedge$ not_holiday $\wedge$ not_rainwithinhours

C2: low_temperature $\wedge$ snow $\wedge$ holiday $\wedge$ rainwithinhours

C3: high_temperature $\wedge$ rain $\wedge$ not_holiday $\wedge$ not_rainwithinhours

C4: low_temperature $\wedge$ rain $\wedge$ holiday $\wedge$ rainwithinhours

C5: low_temperature $\wedge$ snow $\wedge$ not_holiday $\wedge$ rainwithinhours

and so on up to 9000 number of clauses formed for each of these classes. So,let us evaluate input for the classes of high traffic volume and low traffic volume. For instance, if we consider an input:
input = [2° $\wedge$ Sleet $\wedge$ christmas $\wedge$ 10.41mm] and after the necessary preprocessing the input becomes:

input = [low_temperature $\wedge$ snow $\wedge$ holiday $\wedge$ rainwithinhours]

If we pass input to both clauses it satisfies three clauses for the low volume of traffic and whereas it only satisfies two clauses for the sub-patterns of the high volume of traffic. Moreover, the summation of votes is higher towards the low volume of traffic, and assigning it to class y equals zero which is the low volume of traffic.

Features	RTM	LSTM	GRU
All features	62%	63%	65%
without daily rain	65%	66%	70%
without temperature	70%	80%	87%

Table 4.1: Accuracy results of the experiments

A binarized format can influence the prediction results, but also shows that the more relevant attributes have a greater impact on the experimental results. The experimental results show that staging features in a binarized format for RTM can improve the accuracy of traffic flow prediction.

4.4 Regression Tsetlin Machine

We want to predict the traffic flow in the future, and we cannot easily collect updated data to validate the model. Therefore, we are using the lagged data from t to t-n to predict the target, t+n. So, we used t-1 to test the Tsetlin Machine and Neural Networks models. The baseline prediction for time series forecasting is a naive forecast or persistence. This is where the observation from the previous time step is used as the prediction for the observation at the next time step.

All the models were trained using the same dataset. In the classical Tsetlin Machine and Multiclass Tsetlin Machine, the polarity of clauses is used to classify data into different classes. We remove the polarity of clauses since we intend to use the clauses as additive building blocks that can be used to calculate the continuous output. That is, we intend to map the total vote count into a single continuous output [23].

Hyperparameters:

We investigate the effect the hyper-parameters T and s have on learning the Tsetlin machine. As a strategy for problems where the number of clauses is unknown, and for real-world applications where noise plays a significant role, the RTM can be initialized with a much larger threshold. Then, since the output is a fraction of the threshold, T, the error decreases [23]. For example the occurrence probability of any of the 8-bit patterns is 1/256. However, to capture the pattern of 8-bits according to the TM dynamics [20], 1 /s should be equal to the probability of the considered pattern, which is 128

/256 = 1 /2. Hence, s should be 2. For instance, if we assign s = 4, clauses will start to learn a much finer pattern.

We prepare the necessary hyper-parameters, the number of clauses 9000, Threshold T = 4000, and specificity s = 32. Since we have two outputs high and low traffic capacity classes, each class is assigned 9000 clauses to learn the sub-patterns. So, there are altogether 9000 clauses representing all the two classes. When test data passed to those 9000 clauses, some of the clauses are triggered and the class that have the highest number of clauses giving the value 1 decided to be the predicted class. Each class has a collection of sub-patterns. When the test data satisfy with one of the lists of sub-patterns, it represented with that class.

Parameters used in Neural Network

The number of neurons in each layer is in a similar fashion dependent on how complex the given problem is. Too few neurons make the network unable to capture all the information. Logically, the more training data and input/output mappings the network must learn, the more neurons are required. However, too many neurons make the networks more prone to learning unique features of the training set and may thus start overfitting. A learning rate that is too low makes the training slow. A learning rate that is too high makes the algorithm overshoot, which increases the risk of missing the global minima of the cost function.

LSTM and GRU are more popular recurrent neural networks. Based on this model, it is more realistic and persuasive to analyze the effect of the dataset on prediction results. The experiment conducts predictive analysis on the test dataset. GRU is the same as LSTM with improved algorithm and both have the same input and output shape. The input shape format is several timesteps and we set the lag equal to one as it is a period, representing the value of predicting the next period with data per one day of periods. The number of units defines the cell and hidden states of a dimension.

The input data passed into the first LSTM layer with an input of $(None, 1, 5)$ as $(samples, timesteps, features)$ and the output as $(None, 50)$ and then passed into the second LSTM layer input of $(None, 50)$ and output as $(None, 50)$. And then through Dropout layers, Random loss of a certain proportion of parameters. It can effectively prevent over-fitting during training. And finally, the full connection layer, the output latitude of the fully connected layer is 1. Here the number of output neurons is set to 1. The result of training or prediction is an input of $(None, 50)$, the final output is $(None, 1)$. Similarly, we used GRU the same process with the same parameter as for LSTM.

Evaluation

We evaluated the performance of predictions using the mean absolute error, MAE. That gives us more weight to predictions that are grossly wrong and will have the same units as the original data. Any transforms to the data must be reversed before the MAE calculated and reported to make the performance between Tsetlin Machine and Neural Network directly comparable. We calculated the MAE using the helper function from the scikit-learn library mean absolute error that calculates the mean absolute error between a list of expected values, the test set, and the list of predictions.

In the test set section, we saved twenty percent of the original dataset to test the final model. We can use this data to see how well our model is on unseen data. Load the model and use it to forecast the next traffic flow. A plot of the predictions compared to the validation dataset is also provided. Neural network expectedly gets the best accuracy for short-term forecasts. The traffic flow in the short-term future is most of the time quite like the current traffic flow.

The dataset provides the number of hourly traffic flow from January 2016 to October 2018, or just under three years of data. We started with a simple forecast model to provide a baseline of performance for more sophisticated methods to improve upon. In the section, we saved the final five months of the original dataset in a separate datasets to validate the final model. We used this datasets to see how well the models are on unseen data. A plot of the predictions compared to the validation dataset is also provided we see below.

Features	**RTM**	**LSTM**	**GRU**
All features	53	13.43	9.22
without daily rain	47	10.75	6.70
without temperature	30	0.27	0.19

Table 4.2: MAE of predicting results from experiment

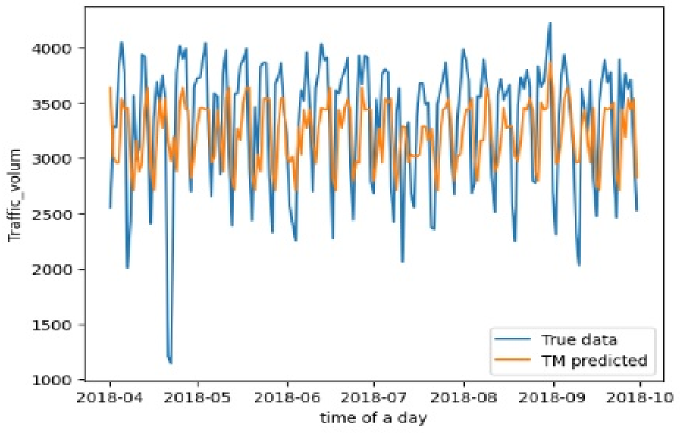


Figure 4.1: RTM: the traffic flow prediction of one day plot against with the actual observed traffic flow

The experimental was performed on the attributes of weather, temperature, amount of rain within one hour, and holiday or regular day data. The experimental results are shown in the Figure 4.1 where the blue curve represents the real value, and the orange curve represents the predicted value. The model under RTM represents the duration between April to October of the test set and curve represents the traffic flow.

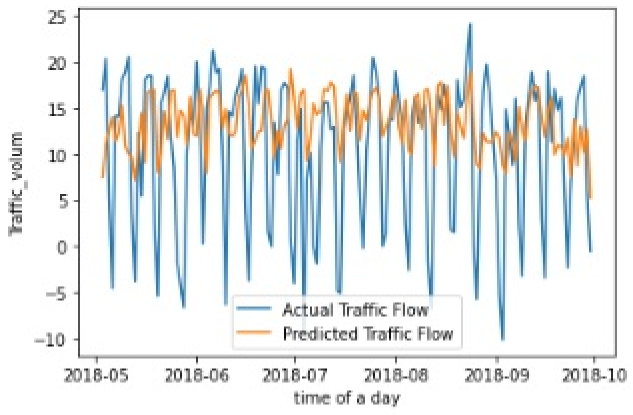


Figure 4.2: GRU: the traffic flow prediction of one day plot against with the actual observed traffic flow

The model under GRU is the same as RTM and represents the duration between April to October of the test set and curve represents the traffic flow.

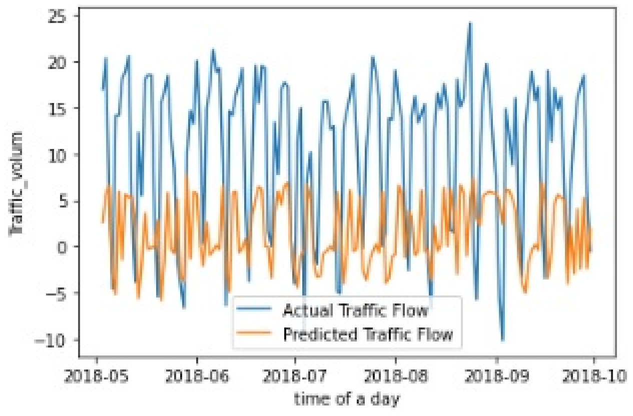


Figure 4.3: LSTM: the traffic flow prediction of one day plot against with the actual observed traffic flow

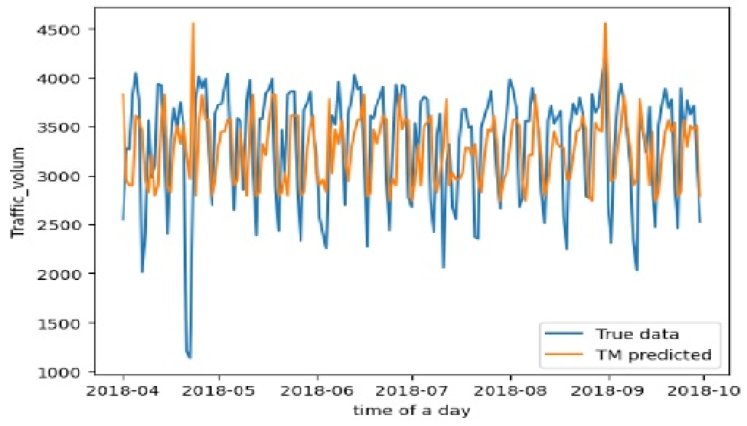


Figure 4.4: RTM: After dropping daily rain,traffic flow prediction with the actual observed traffic flow

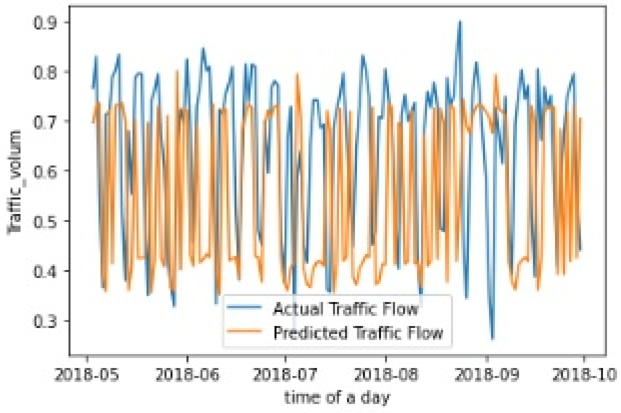


Figure 4.5: GRU: After dropping daily rain,traffic flow prediction with the actual observed traffic flow

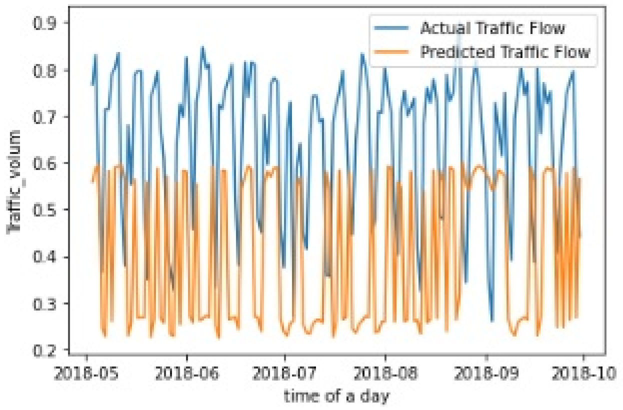


Figure 4.6: LSTM: After dropping daily rain,traffic flow prediction with the actual observed traffic flow

We leave some variables that have fewer impacts on the performance of the model. We can see from the figures both Neural Network models perform well on unseen data. But, RTM performs average because of transforms continuous features into binary while maintaining relationships among the continuous feature values is not optimal. The experiment was performed on the attributes of weather, temperature and holiday or not holiday of test set. The experiment result is shown in the Figure 4.4 where the blue curve represents the real value, and the orange curve represent the predicted value. From the experiment result that the features in a binarized format for RTM can improve the accuracy of traffic flow prediction.

Chapter 5

Conclusions and future work

5.1 Conclusion

The smart transportation system requires a need for a suitable machine-learning algorithm that is lightweight and accurate in capturing and detecting traffic patterns and generating real-time traffic information. AI and ML have shown promising results in the applications of transportation systems. When applied to the transportation system, these technologies can improve the quality of lifestyle and provide safety along with the ease of quicker transportation services to people. Advancement in AI has given a rise to the introduction of self-driving cars for people who can take advantage of this latest technology. Autonomous vehicles require a lot of trust from the people since there is always a question of safety involved with these vehicles. Therefore, a combination of ML, deep learning, and AI techniques need to prove the safety and reliability of such vehicles.

A crucial factor that makes the Tsetlin Machine promising is the interpretability it provides with clauses in the form of conjunction clauses. Potentially this gives an insight into which patterns are important for predicting the target values. That makes Tsetlin machines allow for a degree of transparency in the decision-making process.

5.2 Future work

We did not take some features into account that can significantly affect prediction accuracy. In our study, we involved the intact dataset. We do not split datasets on weekdays and weekends the traffic flows to observe how that affects prediction accuracy. Therefore, future studies can make that to observe if that gives a better performance in further studies.

To increase the performance, one could explore hyperparameter tuning more to see if that can lead to better performance. Additionally, there is an option for weighted clauses and this version of the Tsetlin Machine weighs clauses so that it needs fewer clauses to reinforce patterns, which require different settings of hyperparameters s, T, and the number of clauses. One can also look at different methods for feature selection to see how much impact finding the right features have on the accuracy and can see with a different dataset with more features.

Bibliography

[1] Xueyan Yin et al. “Deep learning on traffic prediction: Methods, analysis and future directions”. In: *IEEE Transactions on Intelligent Transportation Systems* (2021).

[2] Jinlei Zhang et al. “Multi-graph convolutional network for short-term passenger flow forecasting in urban rail transit”. In: *IET Intelligent Transport Systems* 14.10 (2020), pp. 1210–1217.

[3] Shengnan Guo et al. “Deep Spatial–Temporal 3D Convolutional Neural Networks for Traffic Data Forecasting”. In: *IEEE Transactions on Intelligent Transportation Systems* 20.10 (2019), pp. 3913–3926. DOI: `10.1109/TITS.2019.2906365`.

[4] Florin Schimbinschi et al. “Traffic forecasting in complex urban networks: Leveraging big data and machine learning”. In: *2015 IEEE international conference on big data (big data)*. IEEE. 2015, pp. 1019–1024.

[5] Yisheng Lv et al. “Traffic flow prediction with big data: a deep learning approach”. In: *IEEE Transactions on Intelligent Transportation Systems* 16.2 (2014), pp. 865–873.

[6] Jyun-Yan Yang et al. “Prediction of short-term average vehicular velocity considering weather factors in urban VANET environments”. In: *2010 International Conference on Machine Learning and Cybernetics*. Vol. 6. IEEE. 2010, pp. 3039–3043.

[7] *Google Developers,developers.google.com.* `https://developers.google.com/machine-learning`. [Accessed 22-Nov-2022]. 2022.

[8] Rob J Hyndman and George Athanasopoulos. *Forecasting: principles and practice*. OTexts, 2018.

[9] Jason Brownlee. *How to Identify and Remove Seasonality from Time Series Data Machine Learning Mastery*. 2020. URL: `https://machinelearningmastery.com/time-series-seasonality-with-python/`.

[10] Todd E Clark and Michael W McCracken. “Improving forecast accuracy by combining recursive and rolling forecasts”. In: *International Economic Review* 50.2 (2009), pp. 363–395.

[11] Aurélien Géron. “Hands-on machine learning with scikit-learn and tensorflow: Concepts”. In: *Tools, and Techniques to build intelligent systems* (2017).

[12] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. *Learning internal representations by error propagation.* Tech. rep. California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[13] BERNHARD Mehlig. “Machine learning with neural networks”. In: *arXiv preprint arXiv:1901.05639* (2019).

[14] Yonghui Wu et al. “Google’s neural machine translation system: Bridging the gap between human and machine translation”. In: *arXiv preprint arXiv:1609.08144* (2016).

[15] Paul J Werbos. “Generalization of backpropagation with application to a recurrent gas market model”. In: *Neural networks* 1.4 (1988), pp. 339–356.

[16] Sepp Hochreiter. “The vanishing gradient problem during learning recurrent neural nets and problem solutions”. In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6.02 (1998), pp. 107–116.

[17] Alex Graves and Jürgen Schmidhuber. “Framewise phoneme classification with bidirectional LSTM and other neural network architectures”. In: *Neural networks* 18.5-6 (2005), pp. 602–610.

[18] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[19] Junyoung Chung et al. “Empirical evaluation of gated recurrent neural networks on sequence modeling”. In: *arXiv preprint arXiv:1412.3555* (2014).

[20] Ole-Christoffer Granmo. “The Tsetlin Machine–A Game Theoretic Bandit Driven Approach to Optimal Pattern Recognition with Propositional Logic”. In: *arXiv preprint arXiv:1804.01508* (2018).

[21] K Darshana Abeyrathna et al. “The regression Tsetlin machine: a novel approach to interpretable nonlinear regression”. In: *Philosophical Transactions of the Royal Society A* 378.2164 (2020), p. 20190165.

[22] Kuruge Darshana Abeyrathna, Ole-Christoffer Granmo, and Morten Goodwin. "Adaptive Sparse Representation of Continuous Input for Tsetlin Machines Based on Stochastic Searching on the Line". In: *Electronics* 10.17 (2021), p. 2107.

[23] K Darshana Abeyrathna et al. "The regression Tsetlin machine: a Tsetlin machine for continuous output problems". In: *EPIA Conference on Artificial Intelligence*. Springer. 2019, pp. 268–280.

[24] Mohammed Hadi et al. "Guidelines for Evaluation of Ramp Signaling Deployments in a Real-Time Operations Environment". In: (2017).

[25] *How AI helps predict traffic and determine routes-blog.google*. `https://blog.google/products/maps/google-maps-101-how-ai-helps-predict-traffic-and-determine-routes/`. [Accessed 23-Nov-2022]. 2020.

[26] Anton Sysoev et al. "Conceptual scheme of regional module for intelligent transportation and logistics system". In: *International Conference on Traffic and Transport Engineering*. 2018, pp. 139–146.

[27] Loc Dang. "Intelligent Transport System and its application in Ho Chi Minh city". In: (2018).

[28] Mohd Omar and Pradeep Kumar. "Detection of roads potholes using yolov4". In: *2020 International Conference on Information Science and Communications Technologies (ICISCT)*. IEEE. 2020, pp. 1–6.

[29] Roxanne Neufville, Hassan Abdalla, and Ali Abbas. "Potential of Connected Fully Autonomous Vehicles in Reducing Congestion and Associated Carbon Emissions". In: *Sustainability* 14.11 (2022), p. 6910.

[30] SA. *Git Hub - SenaiAG/Prediction-in-transport-system-github.com*. `https://github.com/SenaiAG/Prediction-in-transport-system`. [Accessed 27-Nov-2022]. 2022.

[31] Travis Oliphant. "NumPy: A guide to NumPy; 2006–". In: *URL http://www. numpy. org/.[Last accessed: 2019-05-13]* (2019).

[32] Wes McKinney. "Pandas, python data analysis library". In: *URL http://pandas. pydata. org* (2015).

[33] Oliver Kramer. "Scikit-learn". In: *Machine learning for evolution strategies*. Springer, 2016, pp. 45–53.

[34] Ole-Christoffer Granmo. *pyTsetlinMachine pypi.org*. `https://pypi.org/project/pyTsetlinMachine/`. [Accessed 18-Nov-2022]. 2021.

[35] Valentina Porcu. “Matplotlib”. In: *Python for Data Mining Quick Syntax Reference*. Springer, 2018, pp. 201–234.

[36] François Chollet et al. “Keras [WWW Document]”. In: *GitHub. URL https://github. com/fchollet/keras* (2015).

[37] J Lau. “Google Maps 101: how AI helps predict traffic and determine routes”. In: *Retrieved online from https://blog. google/products/maps/google-maps-101-howai-helps-predict-traffic-and-determine-routes* (2020).

[38] Adrian Wheeldon, Alex Yakovlev, and Rishad Shafik. “Self-Timed Reinforcement Learning using Tsetlin Machine”. In: *27th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC 2021)*. IEEE. 2021. URL: `https://arxiv.org/abs/2109.00846`.

[39] Sondre Glimsdal and Ole-Christoffer Granmo. “Coalesced Multi-Output Tsetlin Machines with Clause Sharing”. In: *arXiv preprint arXiv:2108.07594* (2021). URL: `https://arxiv.org/abs/2108.07594`.

[40] Martın Abadi et al. “Tensorflow: Large-scale machine learning on heterogeneous distributed systems”. In: *arXiv preprint arXiv:1603.04467* (2016).

[41] John D Hunter. “Matplotlib: A 2D graphics environment”. In: *Computing in science & engineering* 9.03 (2007), pp. 90–95.

[42] Fabian Pedregosa et al. “Scikit-learn: Machine learning in Python”. In: *the Journal of machine Learning research* 12 (2011), pp. 2825–2830.

[43] Wes McKinney et al. “Data structures for statistical computing in python”. In: *Proceedings of the 9th Python in Science Conference*. Vol. 445. 1. Austin, TX. 2010, pp. 51–56.

[44] *A guide to NumPy*. Vol. 1. Trelgol Publishing USA, 2006.

[45] Francisco André Barreiros Murçós. “Urban Transport Evaluation Using Knowledge Extracted from Social Media”. In: (2021).

[46] C Nicholson. “AI Wiki A Beginner’s Guide to Important Topics in AI”. In: *Machine Learning, and Deep Learning. https://wiki. pathmind. com/neural-network* (2019).

[47] David B Larson et al. “Performance of a deep-learning neural network model in assessing skeletal maturity on pediatric hand radiographs”. In: *Radiology* 287.1 (2018), pp. 313–322.

[48] Andrej Krenker, Janez Bešter, and Andrej Kos. “Introduction to the artificial neural networks”. In: *Artificial Neural Networks: Methodological Advances and Biomedical Applications. InTech* (2011), pp. 1–18.