Using Message Passing Networks to Predict Train Arrival Times

Daniel Melcer
melcer.d@northeastern.edu

Jonathan Corzo
corzo.j@northeastern.edu

Samuel Steiner
steiner.s@northeastern.edu

March 2021

1 Introduction

Every week, hundreds of thousands of riders rely on the the timely operation of the MBTA's rapid transit network, one of the busiest systems in North America. A single train can carry hundreds of people, meaning that even small delays can affect everyone. This led us to analyze the data that the MBTA publishes to discover if we could predict train arrival times and delays accurately. Our objective for this project is twofold: First we aim to create a message passing neural network (MPNN), which we believe can accurately predict the delay, if any, of a train. Second, we wish to compare our MPNN to an Ordinary Least Squares (OLS) model. We believe that the MPNN will outperform OLS, as the MPNN will have a greater ability to handle the spatial-temporal nature of the data.

2 Background

2.1 Predicting Train Delays

Several studies have been conducted in an attempt to correlate common delay factors with actual train arrival times. Wang and Zhang used weather records, historical train delay records, and train schedule data across 75 stations along the Beijing–Guangzhou Chinese railway line, and applied gradient-boosted regression tree analysis to predict delays [1]. While their model was able to somewhat accurately predict which trains would be extremely delayed by 80 minutes or more, it did not produce accurate results for shorter and moderate delays.

Nilsson and Hanning compared several methods of predicting train delays in Stockholm [2]. They tested a neural network, as well as several variants of decision tree models. The neural network models outperformed all other prediction methods by a nontrivial margin. We plan to emulate these comparison methods by comparing our MPNN model with OLS regression, using the approach outlined in section 3.3.

2.2 Graph Neural Networks and Message Passing

The idea of a Message Passing Neural Network first originated in the context of graphs. It is difficult to represent an entire graph as the input to a neural network. One approach proposed to tackle this problem was to store a hidden state for each node. Then, repeatedly run an algorithm: at each time step, messages are passed between graph nodes until the values of the hidden states converge [3].

More specifically, a Graph Neural Network (GNN) algorithm typically consists of three operations. First, each graph node may generate a "message" vector. The contents of this vector are generated by a neural network which accepts the hidden state of the graph node. Second, the node's neighbors receive this message. A second neural network takes as input the receiver's hidden state, and the contents of the message vector. It outputs a new hidden state for the receiver. Lastly, after some convergence criterion is met, the values of each node's hidden state are fed through a third neural network (a "readout function") to obtain the final result for the classification or regression task. Recently, GNNs have been used to model queuing communication networks, obtaining accurate delay predictions for both synthetic and real-life networks [4].

Graph Neural Networks have continued to evolve. For example, the Gated Recurrent Unit (GRU) update function was developed to improve performance on Recurrent Neural Networks [5]. Li et al. responded by creating Gated Graph Geural Networks (GG-NNs), which incorporate GRUs into the update function upon receiving a message [6].

However, GNNs still only took as input a static graph, and resulted in a single prediction. Battaglia et al. generalized the message passing functionality of GNNs to allow for node-level effects at different times, and complex interactions between different graph nodes [7]. Gilmer et al. compared several variations of this idea to predict the properties of molecules in quantum chemistry [8]. They refer to the general idea as Message Passing Neural Networks (MPNN), and generalize such networks further to allow for some message and readout functions that were not previously explored. We use this concept of a MPNN as a starting point for our train delay prediction system.

3 Proposed Methodology

3.1 Datasets

For this project we will be using data published by the MBTA, containing information about train arrival and departure [9]. We will focus specifically on the Orange line northbound route, from Forest Hills to Oak Grove. The data was collected between 2016-2019. Each year of data contains approximately 47,000 trips, totalling approximately 1.7 million individual arrival and departure events. Additionally, the MBTA publishes the number of fare gate entries at each station, with 30 minute granularity. We chose to exclude data from 2020, due to irregularities in ridership caused by the COVID-19 pandemic. We plan to also augment the dataset with the local weather conditions, collected by the weather station at Boston Logan International Airport through the Meteostat API [10]. This data includes the temperature and general weather state each hour.

3.2 Message Passing Neural Network

We will have two main components in our prediction pipeline, the "State Vector System" (SVS) and the "Prediction Vector System" (PVS).

The following definitions are used in this section:

- Difference from mean (DFM): for a given feature, we normalize the data so that the mean of this feature is zero.
- Train timestamp: The number of seconds that have passed since a given train left Forest Hills.
- Dwell time: The number of seconds between a train's arrival at and departure from a station.
- Headway: For a given departure, the number of seconds that have passed since the last train's departure from the same station.

3.2.1 State Vector System

SVS contains two main entities: Trains, and Stations. A train is a vector in $\mathbb{R}^t$, and a station is a vector in $\mathbb{R}^s$.

We define the vector $\theta_T \in \mathbb{R}^t$ to represent the initial train vector. For all stations s, we define $\theta_{S,s} \in \mathbb{R}^s$ to be the initial station vectors for a given day. Lastly, we define θ_{depart} and $\theta_{entries\_and\_weather}$ to be neural network weights.

SVS uses a number of functions to build up state vectors:

- $M_{depart}(t : \mathbb{R}^t, s : \mathbb{R}^s, ts : \mathbb{R}, dwl : \mathbb{R}, hd : \mathbb{R} | \theta_{depart}) \to \mathbb{R}^t \times \mathbb{R}^s$. When a train departs from the station, M_{depart} is called with the current train vector, current station vector, the train timestamp (DFM for this station), the train's dwell time (DFM for this station), and the headway (also DFM). This outputs a new train vector, and a new station vector. The implementation of M_{depart} will be a neural network, with a GRU layer.
- $M_{entries\_and\_weather}(s : \mathbb{R}^s, p : \mathbb{R}, w : \{\text{“nice”}, \text{“rain”}, \text{“snow”}\} | \theta_{entries\_and\_weather}) \to \mathbb{R}^s$. Every 30 minutes, $M_{entries\_and\_weather}$ is called with the current station weather, the number of fare gate entries (DFM), and the approximate weather conditions at the station. It returns a new station entry, and is also implemented as a neural network with a GRU.

For a given simulated day, all station vectors are initialized to $\theta_{S,s}$. Throughout the day, $M_{entries\_and\_weather}$ is called with the current weather conditions and the number of gate entries. The station vector is replaced with the output of this function.

When a new train is dispatched at the beginning of the line, it is initialized with θ_T. As it encounters each station, the M_{depart} function is called for this train and each successive station. Both the train and the station vectors are updated to the output of this function.

Thus, at a given point in time, we have a station vector for every station, and a train vector for each currently moving train. However, these vectors are semantically meaningless without PVS.

3.2.2 Prediction Vector System

A prediction vector is defined as a member of $\mathbb{R}^p$. We introduce three new neural network weights, θ_{p0}, θ_{ps}, and θ_{pt}.

PVS consists of three functions, all implemented as neural networks:

- $P_{init}(t : \mathbb{R}^t | \theta_{p_0}) \to \mathbb{R}^p$. It maps a train to a prediction vector.
- $P_{station}(p : \mathbb{R}^p, s : \mathbb{R}^s | \theta_{ps}) \to \mathbb{R}^p$. Given a prediction vector and a station, update the prediction vector to include this station. This is implemented with a GRU layer, where the station is treated as the “message”.
- $P_{predict}(p : \mathbb{R}^p | \theta_{pt}) \to \mathbb{R}$. Given a prediction vector, output the total travel time (DFM).

If a train is at Ruggles, and we want to predict its time to get to both Mass Ave and Back Bay, we would use the following pseudocode:

```
input: train vector t0, station vectors s_mass_ave and s_back_bay,
       neural network weights theta_p0, theta_ps, and theta_pt
pred0 = P_init(t0, theta_p0)
pred1 = P_station(pred0, s_mass_ave, theta_ps)
pred2 = P_station(pred1, s_back_bay, theta_ps)
P_predict(pred0, theta_pt) // Should tend towards 0
P_predict(pred1, theta_pt) // (Estimated time from ruggles to mass ave
                              - average time from ruggles to mass ave)
P_predict(pred2, theta_pt) // (Estimated time from ruggles to back bay
                              - average time from ruggles to back bay)
```

3.2.3 Loss Function and Training

There are 20 stations. Thus, for a given train, there are $\frac{20(20-1)}{2} = 190$ possible predictions. When a train leaves Forest Hills, we can predict the delay time to get to Forest Hills, the delay to get to Green Street, delay to Stony Brook, etc. Then, when the train leaves Green Street, we can predict the delay to Green Street, delay to Stony Brook, delay to Roxbury Crossing, and so on. Note that the expected Forest Hills to Forest Hills delay will always be 0, causing $P_{predict}(P_{init}(t|\theta_{p0})|\theta_{pt})$ to tend towards zero.

Let $y_{t,s,d}$ represent the delay time for train t between station s and station d, and $\hat{y}_{t,s,d}$ represent the predicted delay, calculated through the process described above. If $\mathcal{T}$ represents all trains in a batch, and $\mathcal{S}$ represents all pairs of stations (where the second station is north of the first station), the loss function that we are trying to minimize is

$$L = \sum_{t \in \mathcal{T}} \sum_{(s,d) \in \mathcal{S}} (\hat{y}_{t,s,d} - y_{t,s,d})^2$$

All parameters $\theta_T, \theta_{S,s}, \theta_{depart}, \theta_{entries\_and\_weather}, \theta_{p0}, \theta_{ps}$, and θ_{pt} are jointly optimized via gradient descent. An automatic differentiation library will be used to track the gradient throughout the computation.

3.3 Expected Outcomes

Several components must be implemented for our system. We must implement data preprocessing, construct the neural network components of SVS and PVS, perform the operations required to obtain train and station vectors, use the train and station vectors to obtain time predictions, use these predictions to obtain a loss function, and perform gradient descent on this loss function.

We plan to compare the results of this architecture on the testing dataset to the results obtained by OLS regression. Because the magnitudes of the predicted delays matter, we will use the MSE statistic to evaluate success for the regression task. We believe that our message passing architecture will achieve better accuracy than OLS because it can model delays that accumulate over time due to congestion.

We may also compare different loss functions for gradient descent. In particular, it may be beneficial to weight the error of a prediction for stations that are far from a given train vector more or less relative to the error of a prediction for a nearby station. Additionally, we may add a regularization term if necessary.

4 Individual Tasks

Jonathan will be responsible for preprocessing the data, and for the final analysis of the results. Daniel will implement the training loop and the PVS portion of the network, and Samuel will implement the SVS portion. Samuel will also implement and run the OLS model, and he will train the neural network on his local machine.

References

[1] P. Wang and Q.-p. Zhang, "Train delay analysis and prediction based on big data fusion," *Transportation Safety and Environment*, vol. 1, no. 1, pp. 79–88, Jul. 2019, ISSN: 2631-4428. DOI: `10.1093/tse/tdy001`. [Online]. Available: `https://doi.org/10.1093/tse/tdy001` (visited on 03/10/2021).

[2] R. Nilsson and K. Henning, *Predictions of train delays using machine learning*, eng. 2018. [Online]. Available: `http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-230224` (visited on 03/10/2021).

[3] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The Graph Neural Network Model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, Jan. 2009, Conference Name: IEEE Transactions on Neural Networks, ISSN: 1941-0093. DOI: `10.1109/TNN.2008.2005605`.

[4] K. Rusek and P. Chołda, "Message-Passing Neural Networks Learn Little's Law," *IEEE Communications Letters*, vol. 23, no. 2, pp. 274–277, Feb. 2019, ISSN: 1558-2558. DOI: `10.1109/LCOMM.2018.2886259`.

[5] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," en, *arXiv:1406.1078 [cs, stat]*, Sep. 2014, arXiv: 1406.1078. [Online]. Available: `http://arxiv.org/abs/1406.1078` (visited on 03/12/2021).

[6] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated Graph Sequence Neural Networks," *arXiv:1511.05493 [cs, stat]*, Sep. 2017, arXiv: 1511.05493. [Online]. Available: `http://arxiv.org/abs/1511.05493` (visited on 03/12/2021).

[7] P. W. Battaglia, R. Pascanu, M. Lai, D. Rezende, and K. Kavukcuoglu, *Interaction networks for learning about objects, relations and physics*, 2016. arXiv: `1612.00222 [cs.AI]`.

[8] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural Message Passing for Quantum Chemistry," en, *arXiv:1704.01212 [cs]*, Jun. 2017, arXiv: 1704.01212. [Online]. Available: `http://arxiv.org/abs/1704.01212` (visited on 03/10/2021).

[9] *MBTA Blue Book Open Data Portal*, en. [Online]. Available: `https://mbta-massdot.opendata.arcgis.com/datasets/` (visited on 03/12/2021).

[10] *Meteostat Developers*. [Online]. Available: `https://dev.meteostat.net/` (visited on 03/12/2021).