OPTIMIZING NEURAL ODES FOR WEB TRAFFIC FORECASTING

Madhav Ramesh (mramesh5), Kevin Lu (klu25), Brian Sutioso (bsutioso), Matthew Meeker (mmeeker1)
Brown University
Providence, RI 02912, USA
{madhav_ramesh, kevin_lu2, brian_sutioso, matthew_meeker}@brown.edu

1 INTRODUCTION

With the exponential growth in data collection, there has been a corresponding explosion in interest regarding deep learning methods for time series forecasting; however real data is often noisy, irregularly sampled, and filled with inconsistent spiking. This class of time series is incredibly difficult and lacking solutions which address all three issues.

In this work we introduce a novel approach by using a Neural ODE combined with a GRU-D encoder. Neural ODEs are a class of deep learning models which encode an ODE from the input data into latent space by parameterizing the derivative of the hidden state of the encoder block. For networks with recurrent structure such as RNNs and LSTMS, the hidden state can be described recursively as

$$h_{t+1} = h_t + f(h_t, \theta_t)$$

where t describes the recurrence step after the input, which is h_0 and θ_t are the trainable parameters at step t.

Notice that if f represents a gradient update to h then this resembles Euler's discretization approach for ODEs. The output of the model is effectively a permutation of $h(t)$, so the solution of the differential equation

$$\begin{cases} h'(t) = f(h(t), t, \theta) \\ h(0) = \text{input layer} \end{cases}$$

The output of the model is constructed from the solution of this latent ODE evaluated as an initial value problem. As a result, it's common for the decoder of the model to be a differentiable ODE solver. It is important for the ODE solver to be differentiable for back propagation. There are a slew of ODE solvers which could be used to solve for h, we discuss our experimental results with several differentiable ODE solvers.

While the NeuralODE theoretically provides a time invariant approach, in practice it is extremely difficult to train on data which has large gaps, or a large percentage of the time series is missing. To account for this we also introduce a novel modification of Neural ODEs, replacing the encoder block, traditionally a GRU or LSTM model with a GRU-D cell instead.

A GRU-D is a modification of the standard GRU cell, where if the input is NULL, we interpolate with the following equation

$$x_t^d \leftarrow m_t^d x_t^d + (1 - m_t^d)\gamma_{\boldsymbol{x}_t}{}^d x_{t'}^d + (1 - m_t^d)(1 - \gamma_{\boldsymbol{x}_t}{}^d)\tilde{x}^d \tag{1}$$

where $x_{t'}^d$ is the last observation of the d-th variable ($t' < t$) and $\tilde{x}^d$ is the empirical mean of the d-th variable. With decay rates described as

$$\boldsymbol{\gamma}_t = \exp\{-\max(\mathbf{0}, \boldsymbol{W}_\gamma \boldsymbol{\delta_t} + \boldsymbol{b}_\gamma)\} \tag{2}$$

where $\boldsymbol{W}_\gamma$ and $\boldsymbol{b}_\gamma$ are model parameters that we train jointly with all the other parameters of the GRU.

$$\boldsymbol{h}_{t-1} \leftarrow \boldsymbol{\gamma}_{\boldsymbol{h}_t} \odot \boldsymbol{h}_{t-1}, \tag{3}$$

Thus the update equation of GRU-D are described as

$$\boldsymbol{z}_t = \sigma\left(\boldsymbol{W}_z \boldsymbol{x}_t + \boldsymbol{U}_z \boldsymbol{h}_{t-1} + \boldsymbol{V}_z \boldsymbol{m}_t + \boldsymbol{b}_z\right) \qquad \boldsymbol{r}_t = \sigma\left(\boldsymbol{W}_r \boldsymbol{x}_t + \boldsymbol{U}_r \boldsymbol{h}_{t-1} + \boldsymbol{V}_r \boldsymbol{m}_t + \boldsymbol{b}_r\right)$$
$$\tilde{\boldsymbol{h}}_t = \tanh\left(\boldsymbol{W} \boldsymbol{x}_t + \boldsymbol{U}(\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}) + \boldsymbol{V} \boldsymbol{m}_t + \boldsymbol{b}\right) \qquad \boldsymbol{h}_t = (\boldsymbol{1} - \boldsymbol{z}_t) \odot \boldsymbol{h}_{t-1} + \boldsymbol{z}_t \odot \tilde{\boldsymbol{h}}_t$$

This update proves to be powerful compared to the standard GRU for our purposes, even for data which doesn't contain that many missing values.

1.1 Exploratory Analysis

Neural ODEs produce a time invariant and input adaptive evaluation method for forecasting which means that for data prone to spikes and gaps, Neural ODEs should theoretically be able to perform well. Recently, Bilos et. al featured, through NeurIPS 2021, a different approach to Neural ODEs compare to the original landmark paper published by Chen et. al. The authors called their approach "Neural flows", similar in concept to the continuous normalizing flows approach. The approach learns to predict solutions rather than an encoding for the ODE itself, and is much less input adaptive as a result. It has been shown to generalize even worse than Neural ODEs, so we chose to use the original approach for our purposes. As a proof of concept we trained a Neural ODE to fit a time series which produces a spiral. The play dataset time series is composed of t observations of (x, y) coordinates for points.

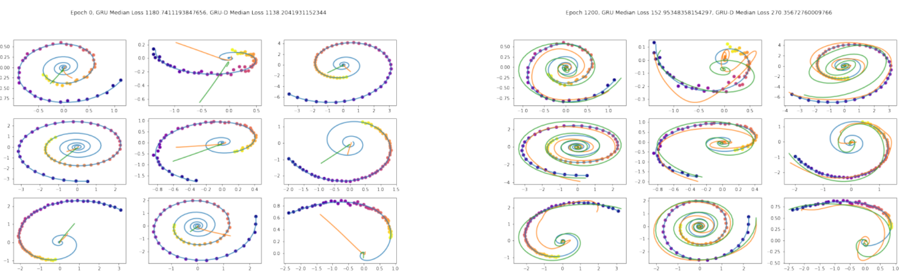
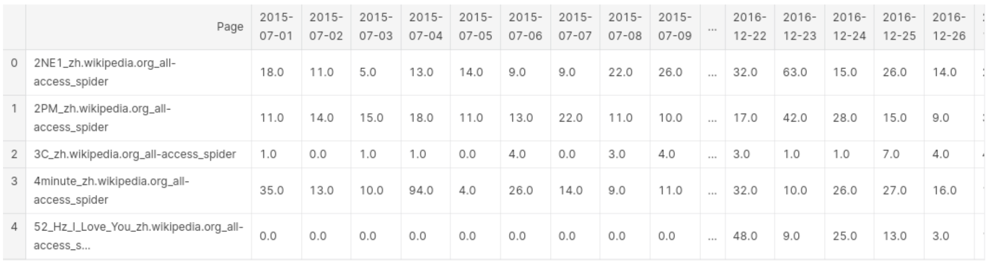


Figure 1: Before (left) and after (right) training for 1200 epochs on the spiral dataset

2 Methodology

2.1 Data Preprocessing

The training data was readily available using the Kaggle API and can be found at `https://www.kaggle.com/c/web-traffic-time-series-forecasting/data`. The data is made up of a page column, containing a concatenation of the article name, type of traffic, and crawling agents used to obtain data, as well as the number of page hits for each day between July 1st, 2015 and December 31st, 2015. Empty values are used to represent missing data (note that missing here can either mean the traffic was zero or that the data was not available for that day). The training data consists of a total of 145,063 websites and 803 time points in total.



	Page	2015-07-01	2015-07-02	2015-07-03	2015-07-04	2015-07-05	2015-07-06	2015-07-07	2015-07-08	2015-07-09	...	2016-12-22	2016-12-23	2016-12-24	2016-12-25	2016-12-26
0	2NE1_zh.wikipedia.org_all-access_spider	18.0	11.0	5.0	13.0	14.0	9.0	9.0	22.0	26.0	...	32.0	63.0	15.0	26.0	14.0
1	2PM_zh.wikipedia.org_all-access_spider	11.0	14.0	15.0	18.0	11.0	13.0	22.0	11.0	10.0	...	17.0	42.0	28.0	15.0	9.0
2	3C_zh.wikipedia.org_all-access_spider	1.0	0.0	1.0	1.0	0.0	4.0	0.0	3.0	4.0	...	3.0	1.0	1.0	7.0	4.0
3	4minute_zh.wikipedia.org_all-access_spider	35.0	13.0	10.0	94.0	4.0	26.0	14.0	9.0	11.0	...	32.0	10.0	26.0	27.0	16.0
4	52_Hz_I_Love_You_zh.wikipedia.org_all-access_s...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	48.0	9.0	25.0	13.0	3.0

Figure 2: First 5 pages of training data

We first preprocess our data by converting it to a PyTorch tensor and using the methods described below to account for missing data. First, we remove any rows where more than 10% of the time points have missing data. This is to shrink the training dataset so that the model only trains on web traffic data with sufficient data points to allow effective fitting of an ODE.

Next, we use various interpolation methods to remove the missing time points in the remaining rows. Among the interpolation methods we used were a zero-based interpolation (each NaN value in the dataset is replaced by a 0), a median-based interpolation (each NaN value in the dataset is replaced by the median of the row), a mean-based interpolation (each NaN value in the dataset is replaced by the mean of the row), and a mean-based interpolation with noise (each NaN value in the dataset is replaced by a value normally distributed around the mean of the row). For models made using a GRU-D, we don't perform any of the above interpolations and directly feed in data containing the NaN values. Among the models using the above interpolation methods, the best-performing model was obtained with a median-based interpolation. We discuss possible reasons for why this may be further below.

2.2 Model Architecture

Our final model architecture consists of a variational autoencoder, where the encoder block is a GRU-D cell and the decoder block is a latent neural ODE. The encoder block generates a latent space representation of an ODE for the time series data and the decoder block essentially integrates over this ODE to obtain a prediction for any time value given an initial time. We use a learning rate of 0.001, train on a random sampling of 200 timesteps, batch size of 1000, and the Adam optimizer.

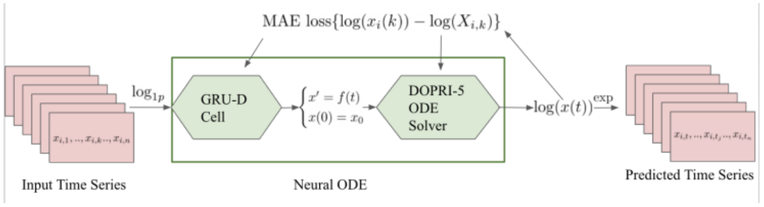
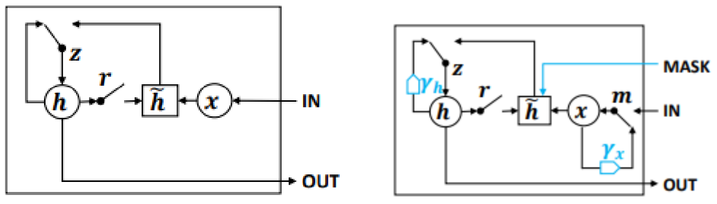

MAE loss$\{\log(x_i(k)) - \log(X_{i,k})\}$
$\log_{1p}$
GRU-D Cell
$x' = f(t)$
$x(0) = x_0$
DOPRI-5 ODE Solver
$\log(x(t))$
exp
$x_{i,1}, \ldots, x_{i,k} \ldots, x_{i,n}$
$x_{i,t_1}, \ldots, x_{i,t_j}, \ldots, x_{i,t_m}$
Input Time Series
Neural ODE
Predicted Time Series

Figure 3: Latent ODE model with GRU-D encoder

The GRU-D cell is implemented from scratch and can be called in exactly the same way as the GRU cell. A PyTorch implementation of the GRU-D cell is given in the publicly available repository since other publicly available implementations are either buggy or hardcoded for time series classification problems rather than time series forecasting problems.

A high-level overview of a GRU vs GRU-D cell is provided below:



(a) GRU (b) GRU-D

Figure 4: Graphical Illustrations of GRU vs. GRU-D models

Essentially, a GRU-D uses a mask to filter out missing input values as they are fed to the network. However, these missing values are replaced by the output of a hidden layer that extrapolates these values based on previous data fed to the network. A GRU-D allows for far more flexibility it interpolation since it dynamically trains a neural network to calculate missing values rather than using a fixed method such as zero, median, or mean interpolation.

The latent ODE decoder determines a trajectory from the initial hidden state z_{t_0} output by the encoder and uses it to predict $z_{t_1}, \ldots, z_{t_n}$. We specifically use a DOPRI 5 solver to integrate the ODE. After experimentation with solvers including the Euler's method, Dopri5, Dopri8, Adams Bashfroth Moulton, Midpoint, and the built-in scipy solver, we found that Dopri5 was the best. This is likely because Dopri5, Dopri8, and the built-in scipy solver are able to account for the greatest stiffness in data and thus, allow for use of higher learning rates with lower batch sizes.

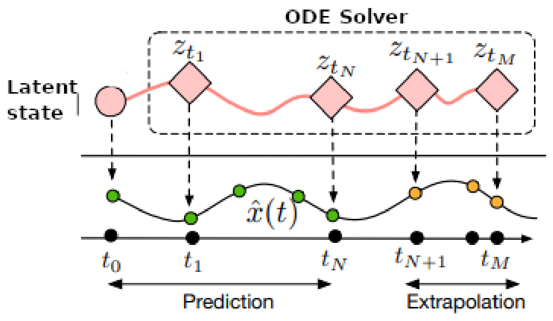


Figure 5: Latent ODE decoder architecture

2.3 Training Method

We train by shuffling the rows in the training data to obtain a random permutation of websites for each epoch. For each row in a batch, we then sample n timepoints starting from a random date between July 1st, 2015 and December 31st, 2015. We experimented with randomly dropping timepoints to create non-continuous data but found that, while this didn't make a substantial difference for the training loss, it lead to inefficient generalization to the testing data. Additionally, it outweighed the benefits of any given interpolation method and as such, we decided to feed in continuous time data after interpolation was performed (for the GRU).

2.4 Loss Methods

Particularly difficult was finding a loss function that could be minimized. Although the test predictions are evaluated using a SMAPE loss, such a loss is difficult to optimize since the value of the function near 0 is undefined. As a result, we experimented with other loss functions including MSE, MAE, a differentiable version of SMAPE, a rounded version of SMAPE, and MAPE. Comparisons between each of these functions are shown below:

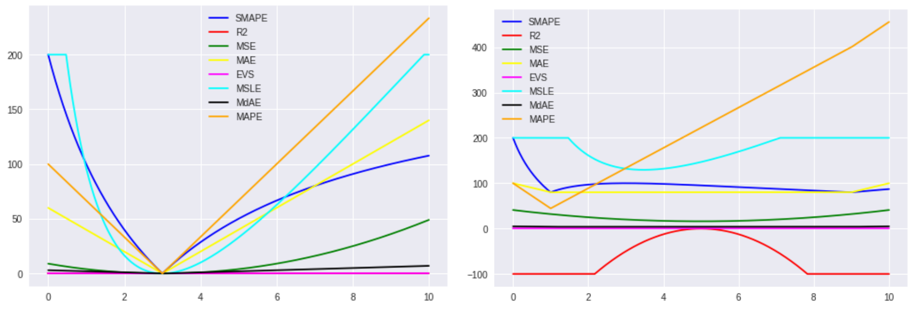


Figure 6: Loss Function Comparison

We found that performing a log1p transformation on the data before applying the MAE loss function allowed us to minimize the loss function optimally. This is because a log1p transformation decreases the magnitude of the data points, allowing for more continual training of the model given data containing sudden spikes in web traffic. The MAE loss function smoothes the results at most data points, providing a similar function. Given that the MAE loss is close to the SMAPE competition loss function, it is an efficient substitute for training.

3 RESULTS

We obtained a final SMAPE score of 41.2. For reference, the top submission on Kaggle which uses a Seq2Seq model achieves a SMAPE score of 35.48, where lower values are more accurate. We train with about half the amount of parameters compared to the top submission (28838 vs 50116) and also with 125 epochs (125 gradient updates per epoch) vs 10,000 epochs (20 gradient updates per epoch). Trained for a greater number of epochs, it is likely that our model could achieve a sub-40 SMAPE score. Given the large difference in the number of parameters and training time, there is significant evidence that neural ODEs are better able to forecast time series data, particularly when significant portions are missing.

Training took roughly 4 hours per model, 10 seconds per batch with GRU, and 12 seconds per batch with GRU-D. Below, we plot the final SMAPE scores for twelve different models, with each model trained on a different combination of encoder architecture (GRU vs. GRU-D) and loss function (MAE, MSE, or differentiable SMAPE).

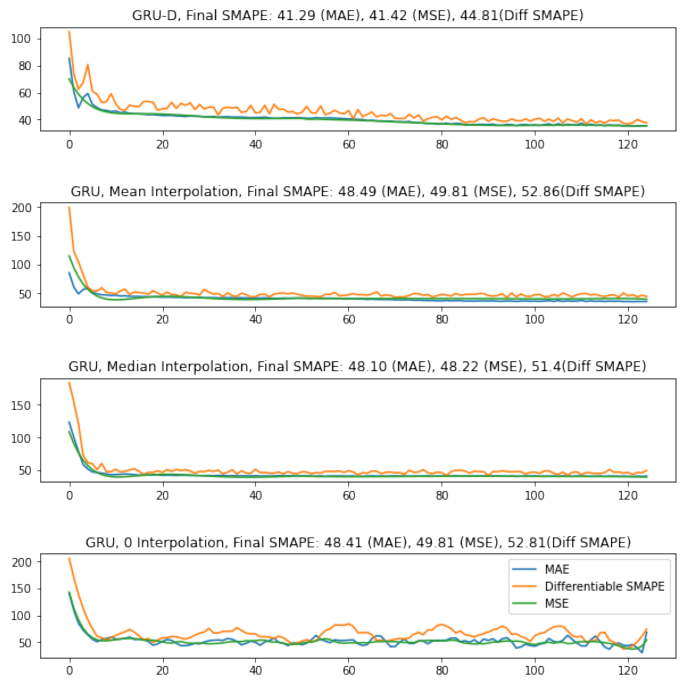


Figure 7: Final SMAPE scores for 12 different models

We also experimented with different ODE solvers for the GRU-D architecture and an MAE loss. Among the solvers we tried were the DOPRI-5, DOPRI-8, Scipy Solver, Adams Bashforth Moulton, Midpoint, and Euler solvers.

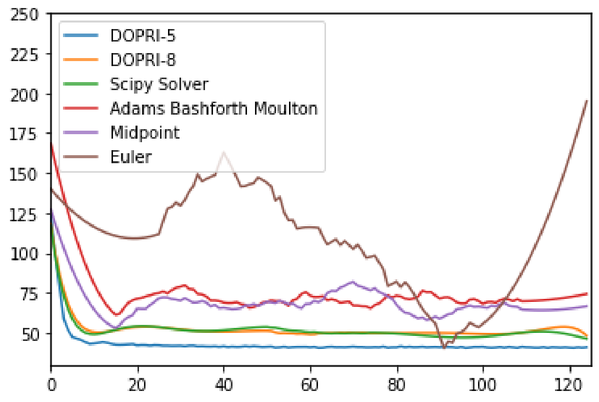


Figure 8: SMAPE scores with different ODE solvers

Below, we display the results of our model predictions versus the actual web traffic values for selected samples. Note that the predictions are typically smoother as a result of the shape of the MAE loss function as well as the ODE solver.

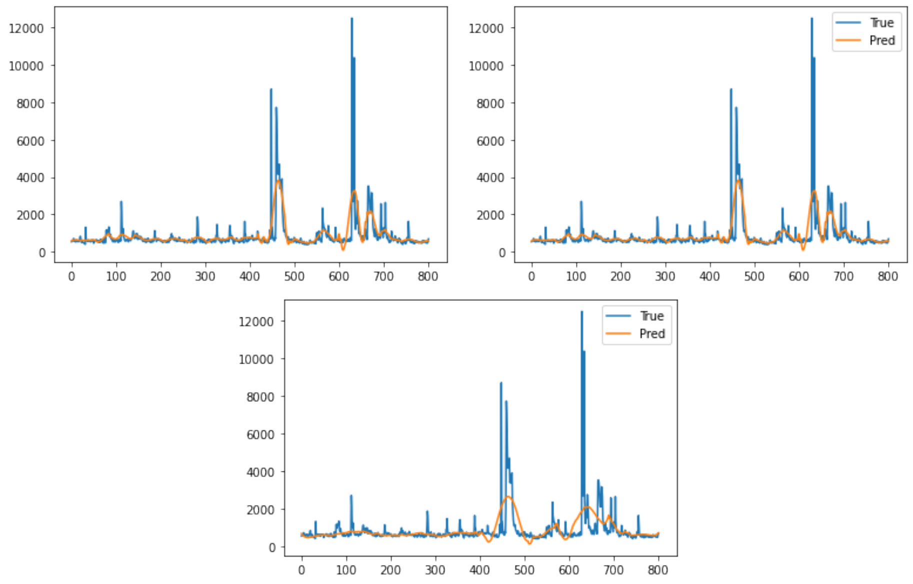


Figure 9: Model predictions vs. actual time series data

4 Challenges

The point of the dataset was to try and develop a method for fitting very noisy, jagged, and irregularly sampled data. This is one of the most challenging characterizations of time series, and as a result we had to research and employ a myriad of techniques to allow our model to effectively fit and predict the data. We split up our challenges and solutions into 3 main categories: Data Processing: Since the data had many large peaks it made training impossible since the model would both attempt to fit the low average traffic and peak traffic. As a result, we initially employed outlier exclusion, but this effectively prevented the model from learning at all. We also employed average window smoothing, but this caused the same issue. As a result we chose to just log transform the input, so our model would train in log space, drastically reducing the euclidean distance between the peak and maximal value. Also, many websites had extremely large gaps in between data entries, roughly

20% of the entire time series would be NaNs in these cases, as a result, we excluded these websites from training overall, and then trained a separate model with its own weights to fit particularly on these websites. Here the GRU-D based models exceptionally outperformed the GRU models since interpolation methods effectively erased any possible trends to be extrapolated from the gaps. Interpolation methods themselves were a pitfall, initially we were not adding any random noise and as a result the models performed exceptionally well in training but had poor extrapolation performance on the test data sets. We tried different distributions, but found uniform distribution led to the best performance. We tested this by removing random points, replacing the missing points with the interpolation with different noises (Gaussian, Xaviar, Exponential, Uniform) and we found uniform produced the lowest KL-divergence.

4.1 Training

The train time was the most exceptional difficulty, since two of our group members could not run the model locally, we were limited to 6 devices counting the free GCP accounts each of us made. This meant we had to be careful about setting up and running experiments since training took roughly 4 hours. Each batch took about 10 seconds to train for GRU models and 12 seconds for GRU-D models, we used batch size = 1000, and there are roughly 144,000 websites. We also had a lot of difficulty setting up Cuda for both the GRU and GRU-D based model, so we were unable to take advantage of GPUs in our training. While the loss profile seemingly indicates convergence, the model actually becomes much more accurate even past 100 epochs. In our literature review, most experiments trained for over 1000-10,000 epochs, but this would take over 2 weeks by our calculations without Cuda, so we settled on a realistic amount of epochs for experiments and trained for a whole day for our final model which we submitted on Kaggle.

4.2 Testing

The model initially had really poor testing performance due to overfitting. Using a 10-fold cross validation to optimize hyperparameters alleviated the issue, but the most significant improvement came from implementing dropout. We suspect applying gentler smoothing techniques than average window such as Savitzky-Golay filtering would improve generalizability too.

5 Reflection

The model worked out the way we expected it to. We were able to apply NeuralODEs to incredibly difficult time series forecasting problems, achieving near state of the art results. We did this using GRU-D's powerful interpolation capabilities, along with methods such as dropout and input transform/normalization. We have achieved a 41.21 SMAPE evaluation score which places us at the higher percentile on the official kaggle public leaderboard. Overall, we are satisfied with the results achieved by the model and are close to achieving our specified target goal.

Our initial approach was using a SMAPE loss as a loss function. Upon realizing that using SMAPE as a loss function causes unstable behavior near zero values, we transitioned over to training the model on the MAE loss function (Mean Absolute Error) by feeding it log transformed data. This produces smoothed results at most data points. If we were to do the project all over again, we would have tried a more complex preprocessing framework that would allow us to account for interpolations at every day of the week instead of simply taking interpolations for each week. This would have produced deeper comparisons against our GRU-D method.

In addition to considerations mentioned above such as Savitzky-Golar filtering, training for longer, and adding Cuda capabilities, we are also very interested in latent PDE generation rather than latent ODE. This is because while our system of ODEs representation does well, it fails to effectively capture relationships between websites. Intuitively it makes sense for related topics or articles in the same language to have somewhat related traffic. We theorize that making each website a dimension of a latent PDE and using PDE solving techniques could also be fruitful.

The biggest takeaway from this project is the importance of cross validation. Since we are dealing with an irregular dataset, we would have to consider other methods of interpolation on nan values. This includes median, mean, and zero interpolation, and finally an implementation of GRU-D which instead accounts for these nan values and trains on them. We have done testing on these methods and found that our intuition on GRU-D is correct and would produce the most accurate results. Another takeaway is the importance of looking at our current solutions from a practical standpoint. For example, is including nan values in training feasible? We did some thinking on the problem at hand and concluded that missing values could be a form of informative missingness (eg. anomalies/inconvenience). This means that by addressing missing values in the time series, we would be able to achieve better results.

6 Related Work

The original paper for this work, Neural Ordinary Differential Equations (https://arxiv.org/pdf/1806.07366.pdf) lays out three main benefits of neural ODES:

- They have a constant memory cost, allowing for training of deeper models.
- Adaptive computation allows for an accuracy vs. time tradeoff
- Applicable to continuously-defined dynamics

In this paper, the authors actually mention that applying traditional neural networks to irregularly-sampled data such as network traffic is difficult but that neural ODEs present a continuous-time, generative approach that may provide significant improvements. However, we have not found any papers since then that lay out the architecture of such a model and evaluate its performance against a test dataset.

Neural ODEs have successfully been applied to other time-series data though. For example, they have been used to successfully predict the weather in Delhi based only on a dataset of climate measurements over the past few years (https://sebastiancallh.github.io/post/neural-ode-weather-forecast/).

Additionally, there are numerous articles discussing potential variations on the basic neural ODE architecture described by Ricky Chen et. al in the original paper such as ODE-RNN hybrids and latent ODEs (https://proceedings.neurips.cc/paper/2019/file/42a6845a557bef704ad8ac9cb4461d43-Paper.pdf) that provide ample opportunity for fine-tuning of model parameters and potentially creation of a novel architecture specific to this problem.

The original paper for GRU-D (https://arxiv.org/pdf/1606.01865.pdf)

In this paper, the authors points out that missing time steps in temporal data would count as a valuable observation. Instead of filling in missing values, masking is used to inform the model on missing data and time intervals are used to observe input patterns. By considering these dependencies and training on all components through back propagation, using GRU-D would greatly improve the model's accuracy.

Given the existing theoretical background as well as effective, albeit limited, papers applying neural ODEs to other time-series data, this presents an exciting opportunity to explore the potential of neural ODEs with a GRU-D encoder for irregularly sampled datasets.