Semester Project: Time Varying Graph Convolutional Neural Networks

Manuel Faysse, supervised by Guillermo Ortiz Jiménez

ABSTRACT

This report focuses on the Time-Variant Graph Convolutional Neural Network introduced by Guillermo Ortiz Jiménez in his master thesis [9]. The network is first tested on synthetic epidemics propagation signals and its performances are evaluated. A graph representation of traffic flows in New York city is then created and paired with Uber pickup data to create a complex real dataset of population flows. A fully convolutional variant of the TV-GCNN is then introduced, used for time series prediction and compared against baselines. The performance gap between the original network and its fully convolutional variant leads to a reevaluation of the importance of vertex filtering and of the impact of the fully connected output layer. It is theorized that the final fully connected layer may learn the relationships between vertices during the training process and by doing so reduce the impact of vertex convolutions.

I. INTRODUCTION

The rise of Deep Learning in recent years has completely revolutionized many computer science application domains with Deep Learning based computer vision often at the movement's spearhead. Techniques and architectures were developed to optimize the learning process and leverage the ordered structure of the data to improve model performances. One notable example are convolutional layers, introduced by LeCun[6] in 1995. Convolutional layers work by convolving a fixed size kernel over one or several dimensions of the input data. This allows to apply local filters to the data that are learned during training, and that are used to extract relevant features or patterns. The same convolution operation is repeated by translating the kernel across the different dimensions of the data. In the case of an image for example, the kernel is shifted horizontally and vertically. The operation is thus known as being shift-invariant. Convolutional layers are applied to structured, Euclidean data, and only take into account local neighborhoods to each input signal to compute the output. The sense of "neighborhood" derives directly from the Euclidean data structure (neighboring pixels in an image for example). This differs from dense fully connected layers whose neurons are given all neuron outputs from the previous layer. They therefore require a lot more weights to train and lose the information that arises from the data structure that local filtering leverages.

A lot of data types can be represented on Euclidean grids and are thus susceptible to work with convolution based neural networks (CNNs). In one dimension, Euclidean data include sound or any time-series, in 2D, images, and examples of 3D Euclidean data include videos and hyper-spectral images. Graph data however does not present the same trivial relationships between nodes that neighboring pixels on an image or in a raw sound sample have. Graphs are defined by a set of nodes (vertices) and weighted or unweighted edges. These edges represent the connections between the nodes and can be based on node similarity, spatial distance, existing relationships, etc... Non-euclidean data that can be represented through graphs include social networks, communication and transportation networks (Internet, roads, public transports, electricity), biological networks (brain connections, molecules).

In order to train Deep Learning models on graph data, convolution layers need to be extended to function on irregular data. The challenge is that nodes from a graph may have different numbers of neighbors and irregular structures, which prohibit a fixed size kernel from applying convolution operations normally in a way that is relevant to the data structure. Two main approaches exist to circumvent this issue, spatial-based approaches and spectral based approaches. Spatial-based methods define graph convolutions based on the node's spatial relations. A node's neighbors are used to update the representation of the central node, and node information is essentially shared when nodes are linked by an edge. Spectral methods on the other hand are based on Graph Signal Processing theory[11]. The Fourier transform is extended to Graph signals (GFT) and allows to project the irregularly structured input signal to the orthogonal space formed by the eigenvectors of the graph's Laplacian. Convolution operations can be computed in the frequency domain by element-wise multiplication between the representation of the graph signal and a filter that is also represented by its Graph Fourier transform in the frequency domain. The inverse Graph Fourier finally transforms the filtered signal back to the vertex space.

Training a deep learning model with spectral convolutions can be achieved by considering the filter weights to be learnable parameters (Bruna, 2013[2]). However, this method has a huge computational complexity due to the eigenvalue decomposition of the Laplacian needed for the GFT. Furthermore, it is not local in the vertex domain which is one of the expected properties of a convolutional layer. These problems can be adressed by approximating the filters with Chebyshev polynomials of the graph Laplacian, which allows to avoid the expensive eigenvalue decomposition and

to filter in the vertex domain (1D-FIR filter). Furthermore, filters are localized in space which means local features can be extracted through filtering (ChebNet, Defferard 2016 [4]).

Another deep learning domain of application is the study of time-series. One-dimensional convolution operations can be applied to time series in an attempt to detect patterns in the structured input data. Other architectures exist, such as Long-Short Term Memory networks (LSTM, Hochreiter 1997 [5]) or Gated Reccurent Units (GRU, Cho 2014 [3]) that keep track of an internal hidden state updated at each timestep. This allows them to detect long-term dependencies in the input sequence. More recently, the WaveNet (van den Oord, 2016 [8]) introduced the use of a successive layers of time convolutions with various dilation factors that allow the receptive field of the output neuron to grow exponentially with the number of layers (Dilated Causal time Convolutions). This allows the network to detect patterns at various frequencies and revolutionized the state of the art in speech synthesis.

Spatial-Temporal Graph Neural Networks (STGNNs) combine Graph Convolutional Neural Networks with time-series networks in an attempt to capture the dynamicity of spatio-temporal graphs. An example of such a graph could be a city's traffic information with node values representing temporal information obtained by the traffic sensors and the street network being represented by the graph structure. The aim of STGCNNs is to detect patterns in node inputs that can vary over time while still benefitting from the insight about the graph structure defined in the graph's edges. The Graph Convolutional Recurrent Network (CGRN) combines the ChebNet architecture for the vertex filtering with a LSTM network to be used in the time domain[10]. Many networks that pair a Graph Convolutional Network with a time-series network were developped; the CGCN (Yu, 2018 [13]) interleaves 1D time convolutions with ChebNet based vertex convolutions, the graph WaveNet (Wu,2019 [12]) uses a spatial-based graph convolution method with the dilated causal time convolutions from the WaveNet, etc... In most cases, time convolutions and vertex convolutions are done sequentially. Joint time and vertex convolutions may allow to detect cross-domain correlations and are at the foundation of the Time-Variant Graph Convolutional Neural Network (TV-GCNN, Ortiz Jiménez, 2018 [9]).

II. TV-GCNN MODEL

The TV-GCNN architecture was developped by Guillermo Ortiz Jiménez during his master thesis[9]. In the thesis, Ortiz Jiménez defines a mathematical framework that allows to perform joint convolutions on data from multiple irregular domains, thus extending Defferard's spectrum-free graph convolutional layers to multiple dimensions (2D-FIR graph filter). The convolution operation is extended to any product graph signal and the benefits of Defferard's 1D-FIR filters [4] used in the ChebNet are maintained in the proposed 2D-FIR filter (local filtering in both domains, fixed number of training parameters regardless of the number of nodes, low computational complexity). In the case one of the two domains is time, a spatio-temporal convolutional graph layer is defined (TV-FIR layer). The TV-GCNN is constructed with a series of time-vertex convolution layers, time and vertex pooling layers and a final dense fully connected layer. The joint convolutions allow the detection of complex patterns that can be correlated across both dimensions (Figure 1). This is not possible in STGCNNs that sequentially convolve over one domain after the other.

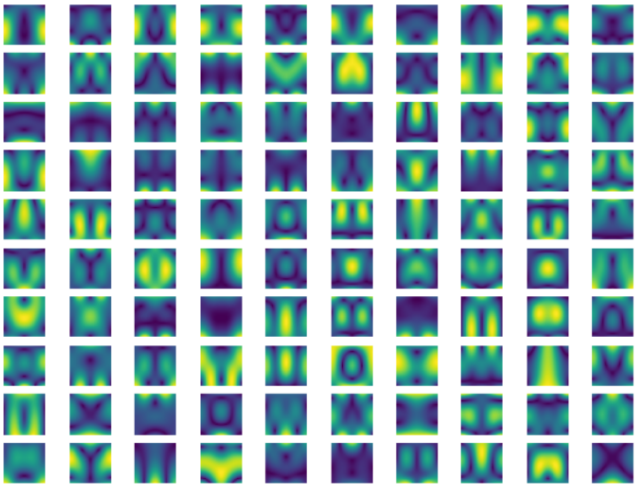


Fig. 1: Collection of spectral time-vertex responses of 2D-FIR graph filters with random coefficients. The horizontal direction represents the symmetric (real coefficients) temporal frequencies, and the vertical direction the graph frequencies. Figure and caption taken from Ortiz Jiménez's thesis (3.1)

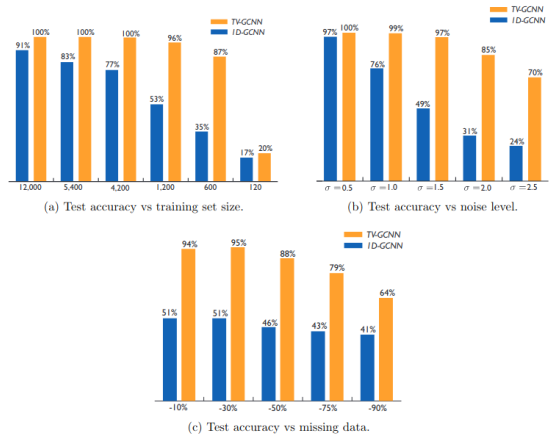
A. Thesis Experiments

In order to validate the performances of the 2D-FIR layer and its TV-GCNN associated network, Ortiz Jiménez implemented the network in TensorFlow v1, along with the ChebNet network to use as a baseline. Both networks were constructed identically wih the sole difference of not having the same convolutional layer. In the thesis experiments as well as in most of the experiments of this report, the network architecture consists of 3 TV-FIR layers (2D-FIR with time and vertex domains) with Max-Pooling in both the time and vertex domain following each TV-FIR layer. The final layer is fully connected with a SoftMax activation function when the network is used for classification purposes.

Ortiz shows that on synthetic signal classification tasks, the TV-GCNN performs better than its Deep-Cheb counterpart without the temporal convolutions. It is especially performant in low training data contexts and is more robust to noise. The thesis concludes that exploiting the joint correlations between space and time in a structured manner leads to a much better generalization ability.

B. Code availability and Framework

The original TV-GCNN code is written using TensorFlow V1 and is available on Github (`https://github.`



(a) Test accuracy vs training set size.

(b) Test accuracy vs noise level.

(c) Test accuracy vs missing data.

Fig. 2: Comparison of the generalization ability of the ChebNet and the TV-GCNN for different training set sizes and noise levels. Figure and caption taken from Ortiz Jiménez's thesis (4.4)

com/gortizji/tv-graph-cnn). The experiments and datasets that were designed over the course of this project are also made available on Github (https://github.com/ManuelFay/tvgcnn-semester) through a series of Python scripts and Jupyter Notebooks. Code related to the TV-GCNN is coded using the TensorFlow V1 framework while work on the time series baselines is done using PyTorch. Several Python librairies described in the GitHub repository ReadMe should be installed for the code to properly run.

III. Epidemiology

The experiments conducted by Guillermo Ortiz Jiménez used synthetic data to evaluate the performances of the model (subsection II-A). The logical next step to validate the TV-GCNN network is to perform experiments on more realistic data with real applications.

A. Context

The TV-GCNN could theoretically be applied to any unstructured graph data that present temporal patterns and be used for classification and regression. It was first decided to attempt using the TV-GCNN with signals propagating on social network graphs (interactions and connections between human beings). Two applications that come to mind are the propagation of epidemics and the spread of fake news through online social networks. While seemingly unrelated , Monti [7] showed in 2019 that the two processes are actually governed by the same underlying patterns and that fake news can be detected using exclusively information about the way they spread through a network. The TV-GCNN could thus be used to tell fake news apart from real news, or more generally identify diseases by the way they propagate.

B. Experimental setups

Creating real fake news or epidemics datasets that satisfied the TV-GCNN requirements (sufficient number of data, fixed graph structure) was an arduous task that was outside the scope of this project. It was thus decided to use epidemiology models to simulate epidemics spreading through a network given different parameters. Different models exist but the SIS model was finally retained for its adequacy to the problem at hand and its simplicity.

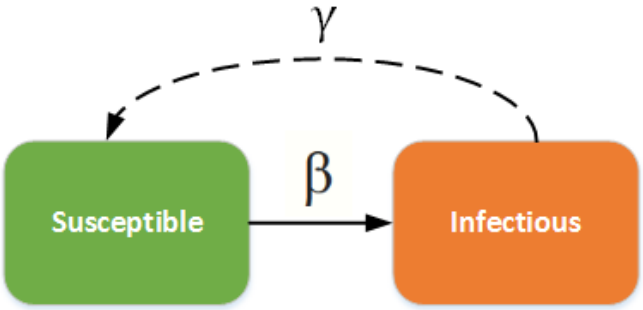


Fig. 3: SIS model schematics

The SIS model considers each node to be in one of two states, a healthy "susceptible" state and an infected state (Figure 3). Parameters β and γ represent the probability of a state transition at each timestep. Infectious nodes can infect neighboring susceptible nodes in the graph (directly connected with an edge) with probability γ and can recover to a susceptible state with probability β. The experiment's idea was to determine whether or not it was possible to classify different types of diseases (modeled by a different pair of parameters) using a few timesteps of propagation information.

The social network structure chosen was a community graph in which the nodes are grouped into different communities that have more intra-community connections than nodes from two distinct communities. This model better allows to model social networks realistically than some of the most common random graph models because they provide a realistic graph structure (as opposed to the Barabasi-Albert model) and degree distribution (as opposed to the Erdos-Renyi model). Two experiments were attempted. In the first one, two classes of signals were modeled by sampling the β and λ parameters from two narrow Gaussian distributions. The intervals from which the Gaussian distribution could be centered were carefully chosen as to stay within realistic propagation patterns and thus avoid having ambiguous signals (all initial infected nodes heal immediately in both cases for example) as well as signals that are so different they are trivial to classify. The difficulty of the experiment was progressively increased by reducing the L2 distance between the Gaussian means and thus augmenting the signal similarity. The second experiment used the same type of Gaussian distribution for the positive class signal (Figure 4) but sampled the second signal from a uniform distribution within the previously defined acceptable range. The idea was to identify a signal

from all the others, that is, to extract it from the background.

Fig. 4: Input data for one type of infectious signal. The horizontal axis represents time, while the vertical axis represent nodes. There is no relationship between the proximity of two nodes on this figure and the presence of an edge between them. Black represents an infected node at the given timestep.

All experiments were conducted with 100 nodes over a period of 128 time frames. Batch size was chosen to be 100 samples, the learning rate $1e-4$ and models were trained on 5000 samples that were regenerated every two runs. Performances were compared to the ChebNet model, both models with hyper-parameters optimized for performance.

C. Results

Results		
Model	TV-GCNN	ChebNet
2 Gaussian Signals (0.003)	94.3 %	92.1%
2 Gaussian Signals (0.0015)	80.3 %	77.1%
Gaussian vs Background	87.0%	81.8%

Fig. 5: Accuracies of the 2 models on the different tasks (6 runs). In parenthesis is the L2 distance between the centers of the Gaussian distribution.

The experiments showed that the TV-GCNN seemed to outperform the tested baseline on this signal classification task. It was logically shown that classifying two distinct signals was an easier task than detecting a signal between other random signals, but that when the two distinct signals were too similar, they were harder to tell apart (Figure 6).

The unavailability of more interesting datasets prevented pursuing this epidemiology topic in more concrete ways. Attempts were also made to try to regress on the model parameters were inconclusive. Given the rest of the project focuses on regression, it was decided to shift the focus to real data experiments and to treat the Epidemiology part as a familiarization with the code base.

IV. Real Data experiments

The initial project idea was to apply the TV-GCNN network to real data with real applications. Since the network

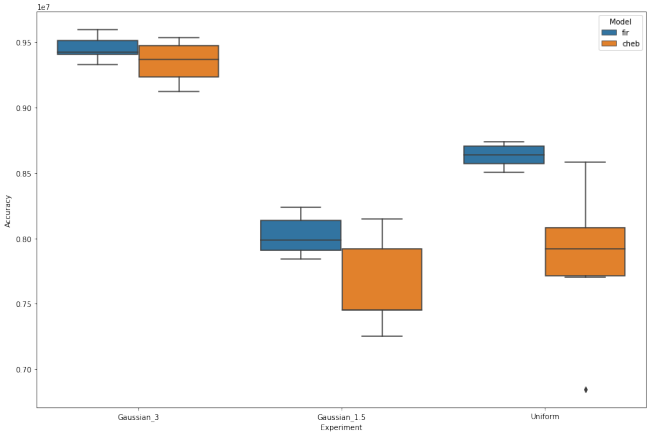


Fig. 6: Boxplots of the model performances. Two gaussians with means separated by a distance of 0.003 (right), Two gaussians with means separated by a distance of 0.0015 (middle),Gaussian and random distribution (right)

leverages information that resides on both the vertex and the temporal domain, it could be applied to any dataset that is comprised of spatio-temporal data such as traffic data or more generally population trends. An interesting domain of application that was elected was thus to adapt the network for time-series forecasting abilities (regression) in order to predict population trends in future timesteps. The point of the experiments was to determine whether adding information about the graph structure (spatial information) could help outperform networks that relied on node-only temporal information for time-series prediction.

A. Dataset and graph creation

The first step in the experimental process was to create structured data to form and populate the spatio-temporal graph structure that the TV-GCNN leverages. It was decided to study population trends, more specifically the case of Uber rides in New York City, New York between January and June of 2015. The data consists of 14.3 million Uber pickups that includes, amongst less relevant information, the city zone and the date precise to the minute. It is interesting because it is a good representation of activity in a city and population flows without being as causal as a traffic dataset, that is, there is no immediate relationship between the activity in any two zones. This data was obtained by the FiveThirtyEight website from the NYC Taxi & Limousine Commission by submitting a Freedom of Information Law request. City zones in the dataset are neighborhood identifiers (Alphabet City, Arden Heights, etc) that the dataset uses to divide the city into 265 unique zones.

These zones will serve as the vertices (nodes) of the graph. In order to establish the presence and the weight of an edge between these nodes, it was first important to geoencode all the zones to get precise spatial coordinates for each zone. This was done by querying the Google Maps API through a Python script and verifying the validity of

the encoding by plotting each zone as a marker on a New York city map (Figure 7).

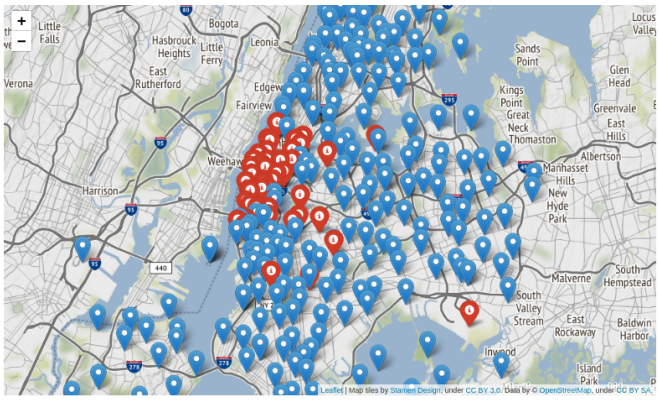


Fig. 7: NY map. Markers represent the GPS coordinates of the zones, red markers represent the 50 most active nodes in terms of Uber rides in the given period of time.

After a bit of data cleaning (3 unknown zones), the next step was to provide a distance metric between nodes. While euclidean (L2) distance would be a simple approach, it does not truly capture the real distance one would have to travel in the city to get from one point to the other. The idea behind this experiment being to forecast population trends while leveraging the spatial relationships between the zones, it is more accurate to use as a distance metric the real distance a Uber vehicle would have to drive between two points (accounting for roads, one-way or walk-only streets, etc) or even the the time it would take a car on average between two nodes (in traffic or not). The Google Maps API can once again be used for these tasks (distance and time) with a limitation however: queries are quadratic with respect to the number of nodes. For 262 zones, this would represent $\frac{262^2}{2} - 262 = 34060$ queries which is too large for Google to allow for free. The workaround that was developed was to divide the zones into bigger macro-zones that made geographical sense (separated by bridges, highways, etc). Queries would be made between each zone within the macro-zones, and between "bridge nodes". Bridge nodes refer to zones that a car would have to pass through to quickly go from a macro-zone to another. It is to note that they may not always be located on the edge of a macro-zone (eg: the quickest way out of a macro-zone may be to take the highway and thus to first drive towards a zone situated in the center of your initial macro-zone). These bridge zones are handpicked to make geographical sense (zones that include bridges, or highway entries and exits, main streets). Finally some long-range random edges are queried to add a bit more information to the graph before full reconstruction. This technique allows the number of queries to fall below 10000.

A Dijkstra shortest path algorithm is then applied to reconstruct the full distance matrix between all nodes given the queried partial information. It is verified that the reconstruction always yielded times that were similar but a bit superior to the Google determined optimal time on test edges that were queried through the API but not initially used for the full distance matrix reconstruction. The time and distance distribution between each edge is represented in Figure 8.

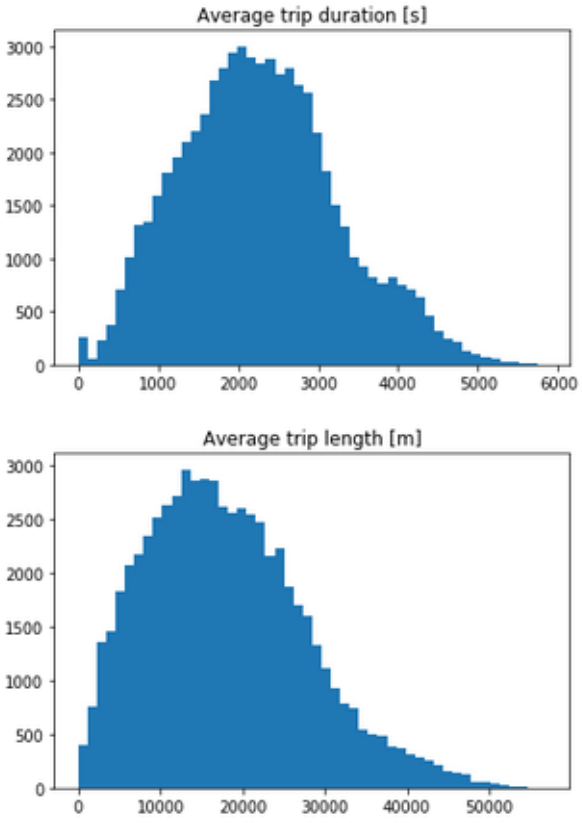


Fig. 8: Time and distance edge distribution in the reconstructed graph.

In order to sparse the graph and link the edge distance (d_{ij} with the weights, an RBF function is applied to each edge distance.

$$rbf(d_{ij}) = e^{d_{ij}^2/2(1.5\sigma)^2} \quad (1)$$

In equation 1, σ is the edge standard deviation. The edges are then thresholded and removed under a certain weight value, and finally, all remaining edge values are MinMax scaled to have values between 0 and 1. At the end of the process, about 7000 edges remain and the fully connected graph component is kept. The nodes that do not belong to the fully connected components represent less than 100 rides over the course of 6 months and are not relevant for the experimental purposes. The final graph is represented in Figure 9.

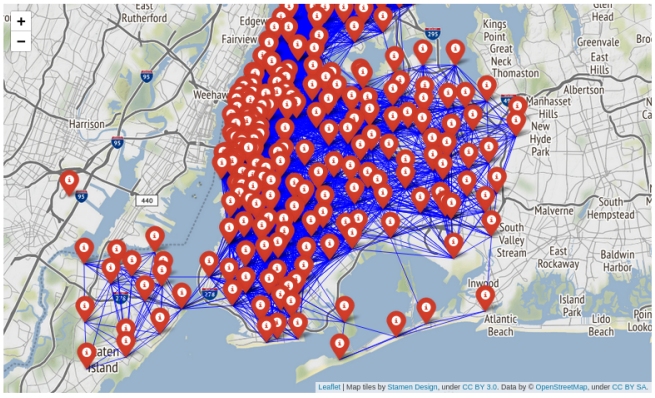


Fig. 9: Final graph. Blue lines represent the remaining edges between nodes.

To populate the graph with the Uber data, rides are aggregated by zone and by time interval. That is, given a time interval (5 minutes, an hour, etc), the number of rides are counted and their sum represents the node value during that interval. A sample is defined as a series of n_a + n_b consecutive time intervals. n_a intervals serve as the model input, and are followed by n_b time intervals that will be used as the prediction. In most experiments, n_a is set to 12 and n_b is set to 7 in order to get 19 total consecutive time intervals. Creating samples that are 19 intervals long present the advantage of separating the total Uber dataset into aperiodic time frames for most time intervals, meaning that samples will not start off at the same time of the day everyday and this helps diversify the training and testing data. Samples are then randomly split into training, validation and testing datasets. Experiments are run different time intervals for aggregation.

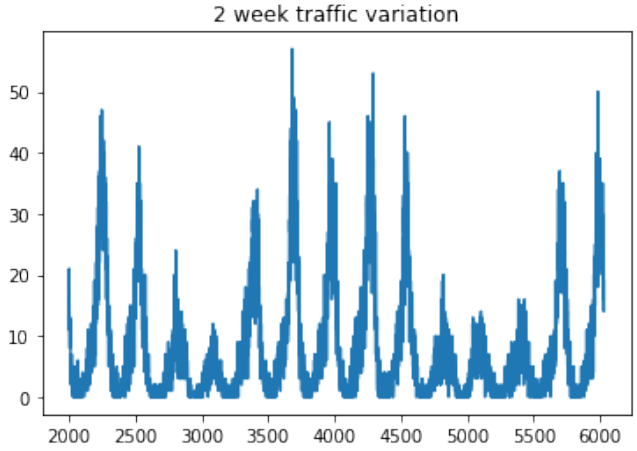


Fig. 10: Time series representing two weeks worth of Uber rides in a frequented node with time aggregated by intervals of 5 minutes.

B. Baseline networks

Evaluating the forecasting abilities of the TV-GCNN is relevant when compared to more classic time-series prediction baselines that do not leverage spatial relationships between the nodes. The interest the TV-GCNN has lies in determining whether the spatial information will improve regression performance. The baselines that are tested on the same task as the TV-GCNN (but with node-wise time series like in Figure 10) include linear regression methods, as well as LSTM networks and dilated time convolution networks described in the introduction. These networks are thus trained by providing training samples that span over a certain amount of time and then evaluating with mean squared error their predictive performance. All data is standardized by node.

1) Linear Regression: The most basic baseline attempted is a simple linear regression on the n_a input time steps. Data is augmented to allow the models to fit the data with polynomial curves (up to degree 4), and to reduce possible overfit, ridge regression is used ($alpha$=0.1).

2) LSTM: The second baseline model is a Long Short Term Memory (LSTM) network comprised of two LSTM cells of 51 channels. The outputs of the second cell are passed through a fully connected dense output layer to output a scalar prediction at each timestep. To predict several timesteps in the future, the output from the first predicted timestep is passed through the LSTM network thus generating the output for the second timestep and so on.

3) Dilated Causal Time Convolutions: The third baseline model is a Dilated Causal Time Convolution network. It works by using successive 1D time convolution layers with increasing dilation rates. This allows the final output node to receive information from a big number of input neurons (large receptive field). The network gains the ability to detect patterns of different frequencies (Figure 11). The same strategy is used than the LSTM to generate multi-timestep predictions although the model is only trained with the loss measured from the first timestep.

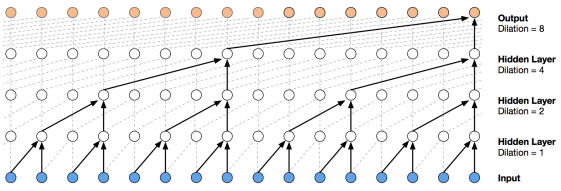


Figure 3: Visualization of a stack of *dilated* causal convolutional layers.

Fig. 11: Dilated Causal Time Convolution. Figure from the WaveNet paper.

C. Baseline results

Baseline results depend greatly on the time intervals that are chosen for the data aggregation. For long intervals, in which we observe long and complex patterns, the more

complex models (LSTM, Dilated Causal Time convolutions) work best (Figure 12). For short time intervals, the signals are very noisy (high variance) and the simple linear regression model often yields very satisfying results. Another factor that has a huge importance in model performance is whether or not the zone is busy or not. Busy nodes have more samples per time interval and thus less variance and more stability. This makes them much easier to predict.

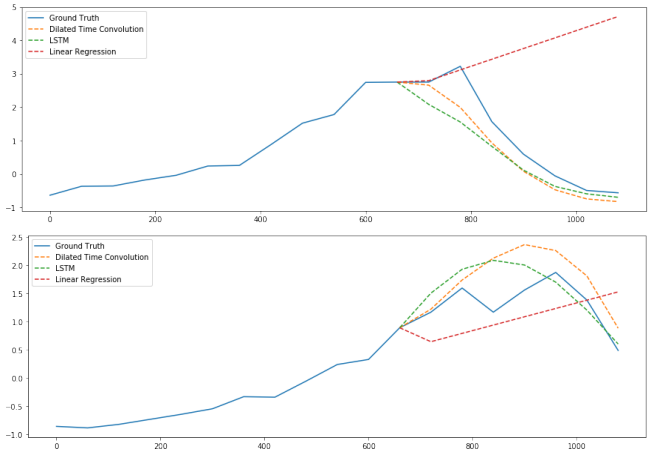


Fig. 12: Baseline network predictions on the busiest graph node with data aggregated by 60 minute time intervals.

Results			
Model	Time inter-val	**Average MSE (1)**	**Average MSE (2)**
Ridge Regression	60	3.50	1.88
LSTM	60	**0.23**	1.05
DC TimeConv	60	0.27	**1.04**
Ridge Regression	20	0.56	1.54
LSTM	20	0.14	**0.85**
DC TimeConv	20	**0.13**	0.93
Ridge Regression	5	0.2	1.58
LSTM	5	0.15	**0.93**.
DC TimeConv	5	**0.14**	0.98

Fig. 13: MSE Loss of the 3 baseline models predicting 7 time-steps in the future, averaged over 6 runs. The (1) represents experiments which were run on the graph's busiest node and (2) on the node with the median amount of rides. It is to note 60 minute aggregations represent 227 unique 12+7 time steps samples, 20 minute aggregations represent 682 and 5 minute aggregations represent 2734.

From the table in Figure 13, it is noticeable that regression models get comparatively better than the two other models when the time interval diminishes. This is due to the fact the best option to optimize the loss when the data is scarce (thus noisy and unpredictable) is to stay centered around the mean. This is supported by the plot in Figure 14 which illustrates that the LSTM adopts this behavior when confronted to scarce data. It was often observed that the LSTM network predicted values of 0, the mean in standardized data, when trained on the less busy nodes or very short time intervals. The loss in these cases is slightly inferior to 1, which is therefore only slightly better than the naive prediction (predicting 0 for all time frames). The Dilated Causal Time Convolution Network, though it produces equivalent performances to the LSTM in most cases, has a tendency to replicate the high frequency oscillations from the input signals in scarce data inputs. This is possible thanks to its design which allows it to convolve over different signal periods and thus find patterns at multiple frequencies and is observable clearly on both plots of Figure 14.

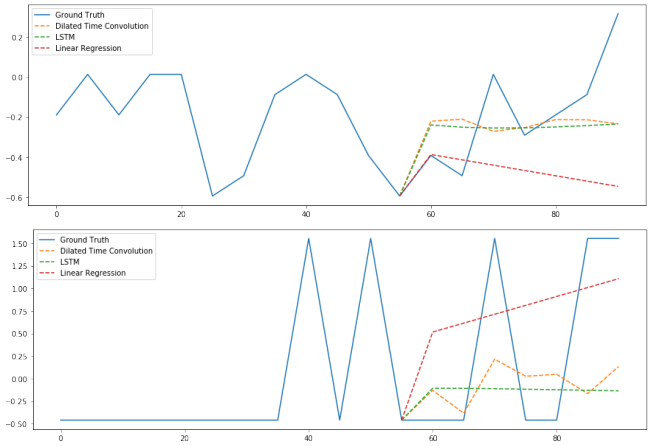


Fig. 14: Baseline network predictions on the busiest graph node (top) and node with median frequentation (bottom) with data aggregated by 5 minute time intervals.

Running the Dilated Time Causal Convolution Network and the LSTM on all nodes individually yields the loss distribution in Figure 15. There is a clear relationship between node activity and predictive abilities, and it seems under a certain level of stable activity, predictions are meaningless. It is calculated that there is a Pearson correlation of -0.91 between the Loss from the LSTM models and the number of Uber rides in a given zone which is almost a total negative linear correlation and -0.88 for the DC TimeConv.

Overall, what these baselines indicate is that time series prediction can be precise for stable input data that present clear patterns but are a near impossible task (for a machine as well as a human) on noisy, unstable, scarce data. This complicates the task for the TV-GCNN on one hand, but on the other hand, this may allow the network to leverage spatial relationships and correlations between node time-series to improve predictive performances.

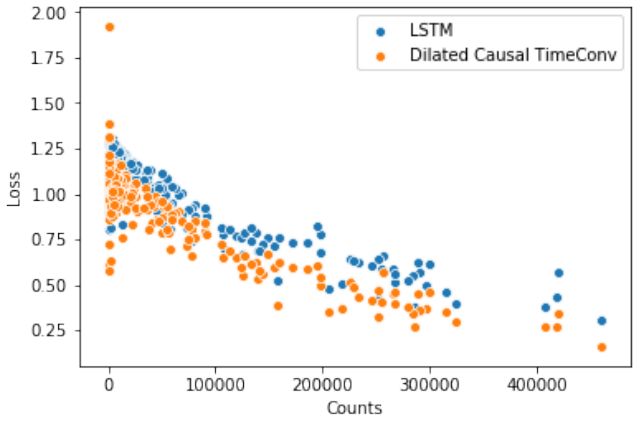


Fig. 15: Scatterplot of the MSE Loss versus the number of Uber rides per node for the baseline networks.

D. Fully Convolutional TV-GCNN

The forecasting application of the TV-GCNN imply the need for node-wise predictions over multiple time-steps. While the current network architecture can be adapted to yield $250 * 7$ outputs (250 nodes x 7 time steps to predict) by augmenting the dense final layer output size, this causes a very large model with a lot of weights and that is slow to train. Furthermore, the dense final layer shares the information between each node at the end of the model, which goes against the idea of testing the performances of the TV-FIR layer that is based partly on vertex based convolutions.

A fully convolutional (FC) version of the TV-GCNN network is thus proposed. This network is composed of the TV-GCNN network backbone without the vertex pooling layers nor the final dense layer. Two final 1D time convolutional layers are added at the end, the first kernel has dimensions that when convolved with the signal reduce the time dimension to a size of 1, and the second convolution adapts the number of filters to equal the number of steps we want to forecast. This technique for forecasting several steps in advance, recommended by Brownlee [1], is verified by comparing the MSE Loss with one forecasting time-step with the the loss from multiple forecasting timesteps and observing similar values.

E. FC TV-GCNN results

The first change that is noticed between the fully convolutional network and the original version are the training time per epoch. On an Intel i7 CPU with 8GB of RAM, training over an epoch takes on average 82 seconds for the original network with the full graph while the FC version trains in 14 seconds. This 590% time difference is caused by the much greater number of parameters in the original version, the main difference residing in the presence of the fully connected layer at the end.

Results				
Model	**Time interval**	**Epochs**	**MSE** (1)	**MSE** (2)
Nodewise LSTM	5	20	1.03	0.62
Nodewise Dilated Time convolutions	5	30	0.89	0.47
TV-GCNN	5	5	0.92	0.72
TV-GCNN	5	10	0.69	0.60
TV-GCNN	5	15	0.58	0.53
TV-GCNN	5	20	0.53	0.50
TV-GCNN	5	40	0.43	0.41
TV-GCNN	60	40	$\star$	0.31
FC TV-GCNN	5	5	0.82	0.71
FC TV-GCNN	5	10	0.77	0.68
FC TV-GCNN	5	15	0.74	0.67
FC TV-GCNN	5	20	0.74	0.67
FC TV-GCNN	60	40	$\star$	0.78

Fig. 16: Epochs are calibrated until no more improvements on the test loss are noted during several epochs. Tests are run 6 times and the average loss value is retained. The nodewise baselines were only run once but have reduced variance due to the fact they are retrained for every node. (1) corresponds to tests on the full graph and (2) on the subgraph of the 50 busiest nodes.

The results indicate that while the fully convolutional version is quicker and trains in less epochs, it is eventually significantly outperformed by the network that contains a dense fully connected layer before the output. Plotting these predictions leads to the realization that the network understood the noisy spikes in the data and learned to replicate the training data to an extent (Figure 17).

However, it is more surprising to see that the predictions that the TV-GCNN network makes for low frequentation nodes are based on the same high variation patterns as for the rest of the nodes (Figure 18). The network does not seem to differentiate the nodes individually when making a prediction but is rather more geared towards recreating common patterns in certain situations.

On the other hand the FC version of the network behaves similarly to the low data context LSTM, that is, predicted values tend to be constant and close to the input time series mean or the global mean (0). By analyzing the differences in predictions between both networks, it could be interpreted

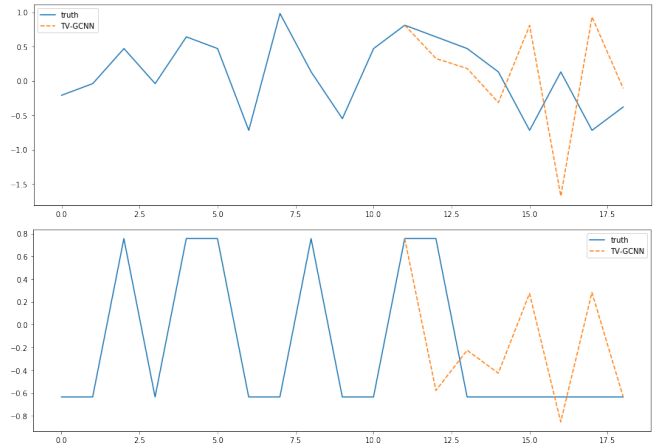


Fig. 17: Original TV-GCNN predictions for nodes with median (bottom) and high (top) frequentations.

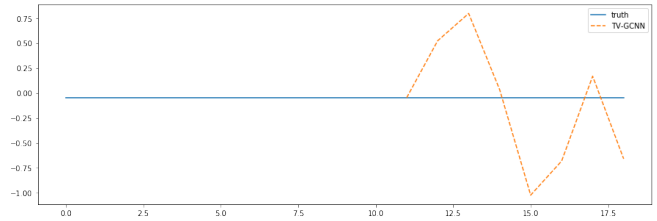


Fig. 18: Original TV-GCNN version on nodes with low frequentations.

that the nodes in the FCN do not interact as much and that removing the fully connected network at the end isolates the nodes from one another more than in the original version of the network.

In both cases, it seems the shared training information helped optimize the global performance when compared to the nodewise baseline networks, although it hindered local performance in the busiest nodes.

Experiments were also run by reducing the original graph to the 50 busiest nodes in order to evaluate how the TV-GCNN would react to a more consistent source of information (Table in Figure 16). As expected, both versions of the network performed better but the network that really benefited from cleaner data across all nodes was the version with the fully connected layer at the end (Figure 20). Evaluating how much of the performance is due to the joint time/vertex pooling from the TV-FIR layers rather than the information sharing and subsequent training that stems from the network architecture (dense final layer) becomes an interesting research question.

Different network architectures and settings were tested. This included changing the number of layers, the order of the Chebyshev interpolation, the number of filters, the size of the time convolution kernels and the pooling ratio. No significant improvements were observed by varying from the

Fig. 19: FC-TV-GCNN version on nodes with high (top), median (middle) and low frequentations.

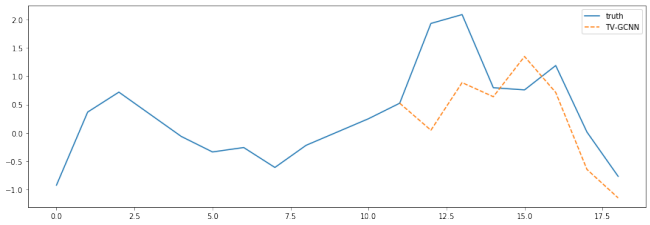


Fig. 20: TV-GCNN predictions for a node in the top 50 graph

default settings.

V. Measuring the impact of the TV-FIR layers

A. Context

In order to measure the impact of the TV-FIR layer which implements the joint time-vertex convolutions and is thus at the foundation of the TV-GCNN network, it is interesting to evaluate the improvements of such a network over a baseline network with vertex convolutions only, like the ChebNet (cf. thesis experiments [9]). It is also relevant to observe the differences over a baseline network that only implements time convolutions.

This allows to better gauge the impact of the vertex-filtering with respect to the time-filtering, and better understand the importance of the network architecture, notably the presence of the dense final fully connected layer before the network's output.

B. Experimental setup

In order to do so, the experiments from the thesis were reproduced with the addition of a new network, the

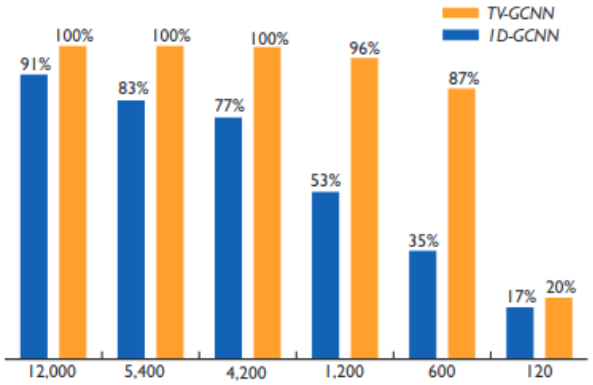


Fig. 21: Experimental plots from Ortiz Jiménez's thesis[9]

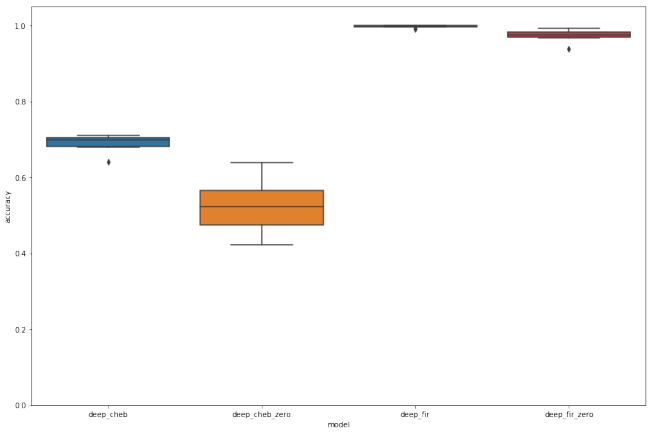


Fig. 22: Boxplot of the accuracies of the 4 networks over 15 runs for 2400 training samples. There is low variation in the accuracies over runs.

"TV-GCNN Zero". This network is a copy of the original TV-GCNN network in which the vertex convolutions are removed and the graph Laplacian is never used. An alternative version of the Deep-Cheb network, the "Deep-Cheb Zero" is also constructed by setting the graph Laplacian to 0 for all matrix values and thus simulating a graph with no edges. The original settings and code are provided on GitHub and were used in these experiments. The number of testing samples was set to 2400, the graph had 100 nodes and data spanned over 128 time frames. A 100 sample batch size was used along with a 10e-4 learning rate. Finally, all four networks were designed to have 3 intermediate layers that each implemented the convolution operations followed by the vertex and the time poolings. A final dense fully connected layer is added at the end of each network. The experiments that were conducted consisted in classifying a synthetic signals in one of 6 classes. The models were tested for their accuracies using a different number of training samples. Experiments were run 12 times for each model and number of training samples combination, and were found to be very consistent between runs, especially for high numbers of training samples (Figure 22).

C. Results

The first thing that is observed (Figure 23) is that the original experimental results from the thesis are reproduced with exactitude. The measured accuracies for both the TV-GCNN and the Deep-Cheb networks are similar to the ones described in the thesis, with the TV-GCNN network performing much better than the Deep-Cheb in low data settings but their accuracy difference progressively reduced upon augmenting the number of training samples.

The Deep-Cheb Zero network performs significantly worse than its original Deep-Cheb counterpart in which vertex convolutions occur. It seems that above 4200 training samples, the performance gap between both networks stays more or less constant regardless of the number of training samples that are added. We could thus quantify in accuracy percentages the benefit vertex convolutions have on classification in this case, which represents about 7.2 %. The improving model performances correlated with increasing training data can probably be explained by the progressive weight refinement of the final fully connected layer that equally benefit both networks.

The TV-GCNN Zero architecture yields very similar results to the ones measured in the TV-GCNN version. A small but clear performance gap exists when the models are trained with low amounts of training samples, but this gap progressively diminishes with more data until it stabilizes at about 0.7 %. We could in this case also attempt to attribute these performance improvements to the presence of vertex convolutions. It nevertheless seems, given the huge performance gap between the Deep-Cheb and the TV-GCNN Zero network, that time convolutions greatly outweight vertex convolutions in the impact they have on the classification accuracies. Without considering that jointly convolving on both the vertex and temporal domain allows to capture correlations between the two domains and may explain part of the performance improvements in the TV-GCNN network, experiments show that removing vertex convolutions from the TV-GCNN network is roughly 20 times less harmful to the metrics than removing temporal convolutions.

The impact difference between spatial and temporal filtering could be explained by the nature of the data. Time series are 1D Euclidean data that convolution operations are known to perform very well on. Removing the ability for local pattern recognition in the time domain is bound to worsen the signal classification abilities of the network. Furthermore, in the TV-GCNN Zero network, information sharing between nodes still exists through the final fully connected dense layer. The network can thus leverage information from multiple time-series (one per node)

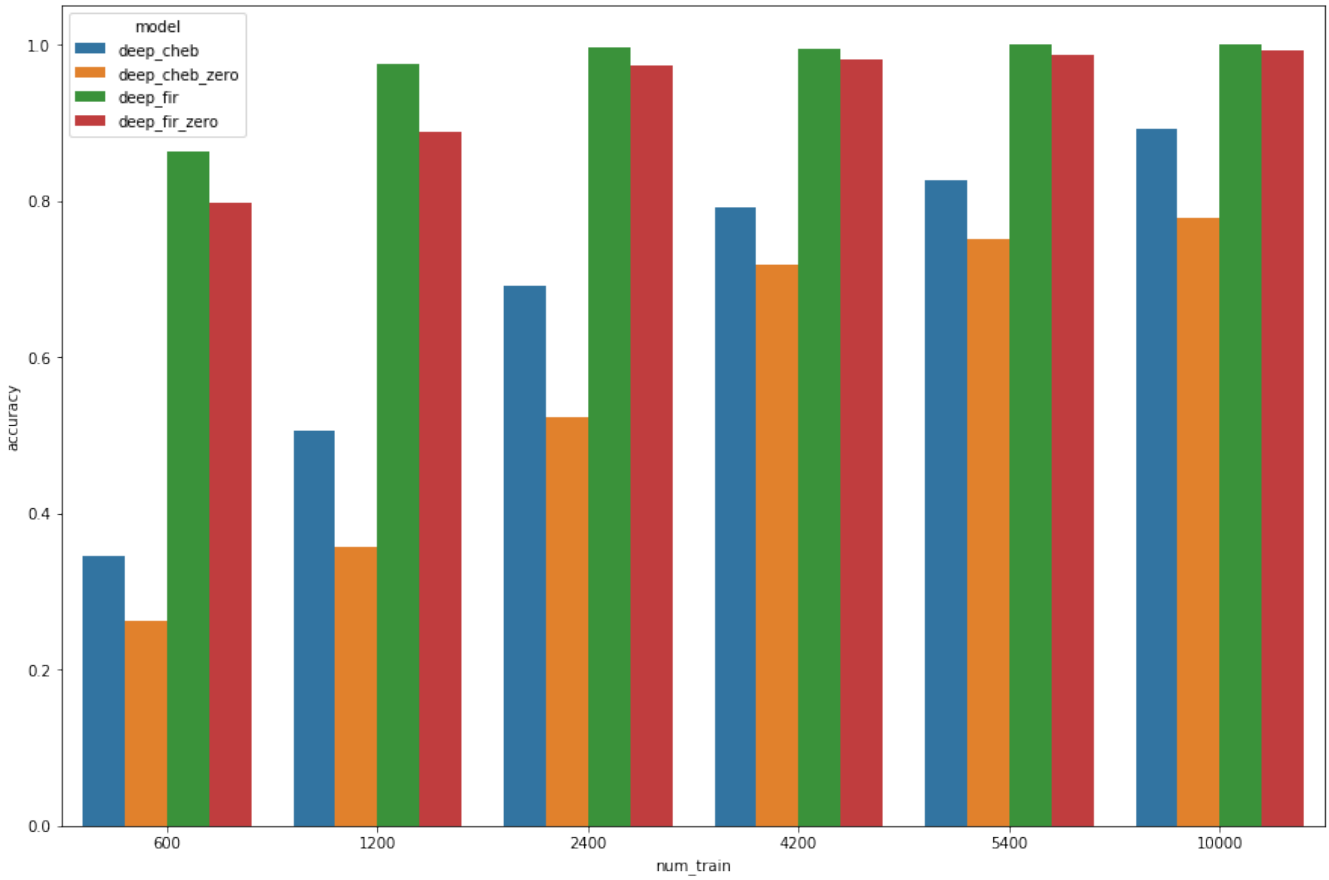


Fig. 23: Reproduced experiments from the thesis with additional baselines.

without necessarily relying on the pre-defined spatial relationships between them that are used during vertex filtering. It is probable however, that spatial relationships between the nodes still exist, but are learned by the fully connected layer during the training process. This theory would help explain the small performance gap between the TV-GCNN and the TV-GCNN Zero network in high training data contexts in the sense that learned spatial relationships during training can be almost as relevant as the pre-defined relationships defined by edge weights, at least in the case of the data used during the experiments.

What remains to be explained is the reason the performance gap between the TV-GCNN and the TV-GCNN Zero network is more important when trained with less data. The only difference between both networks resides in the vertex convolutions that are used to leverage the relationships between vertices defined by the graph structure. The fact the performance gap diminishes with more training data indicates vertex filtering has more impact on the signal classification in low training contexts. This supports the theory that states that vertex relationships are progressively learned during training by the fully connected final layer. The pre-defined relationships that vertex convolutions leverage become less and less important as weights in the fully connected layers are refined to better describe the correlations between nodes.

Performances are improved with more training data because of either an increased data diversity, or an increased number of training steps. In the previous experiments, models trained for 10 epochs regardless of the size of the training data, thus training for a smaller total number of processed samples when faced with less training data. Experiments were run to compare models that were trained with a similar number of total batches but with varying data diversity. This is done by training a model with n_t training samples for n_e epochs, and another model with $2n_t$ training samples for $\frac{n_t}{2}$ epochs. In both cases, the number of total processed samples is equal, and if n_t is a multiple of the batch size, the number of processed batches and therefore of weight update steps are identical. It is shown (Figure 24) that training with less training data but proportionally more epochs yields almost constant performances.

Similarly, training both the TV-GCNN and the TV-GCNN Zero network until the testing accuracies stop improving for both networks leads as expected to a reduced performance gap (Figure 25). These results indicate that some of the network weights need training time to be fine-tuned. Linking

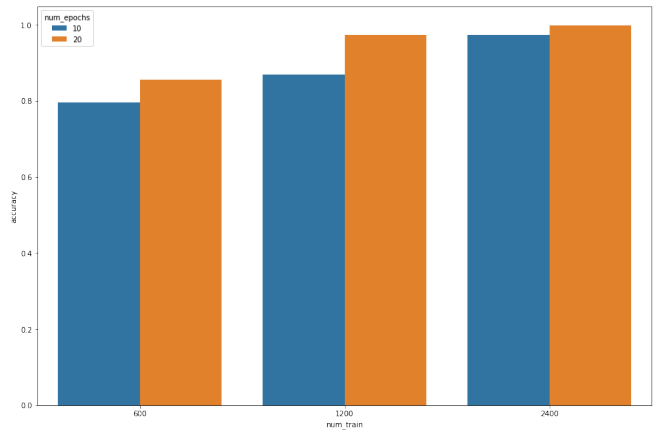


Fig. 24: Test accuracies of the TV-GCNN Zero network with regards to training data size and epoch number. Training with less data but proportionally more epochs yields similar results (orange bars are about the same size thab blue bars to their right).

the results to the ones obtained in Section IV-E, in which it is shown the fully convolutional TV-GCNN network stops improving upon its accuracy in less epochs than its TV-GCNN counterpart, it becomes logical to infer that it is the dense layer that necessitates the most time to optimize its weights. This further supports the idea this final layer is essential to the training process and when sufficiently trained, replaces most of the need for vertex filtering.

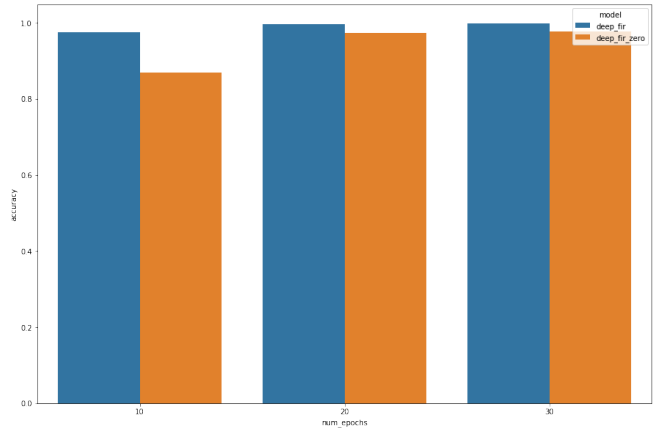


Fig. 25: Test accuracies of the TV-GCNN Zero and TV-GCNN network for 1200 training samples with regards to epoch number.

The theory of the learned vertex relationships in the fully connected final layer is based on the idea the time-series information is shared between nodes in the final layer. While this is exclusively true in the TV-GCNN Zero network, information sharing between nodes also occur during vertex filtering in the TV-GCNN network and may help explain the initial performance gap. In the proposed theory, the better performances in low training situations of the TV-GCNN are due to the information sharing that occurs in the vertex convolutions. This is verified by constructing two versions of the network based on the Laplacians of random graphs that have a different connectivity (Figure 26). Both are then trained and tested on synthetic data. It is shown that the TV-GCNN version based on the graph with the highest connectivity, and thus the smallest graph degree (maximum distance between two nodes) performs best. This can be explained by the greater ease of information propagation in a more connected graph. Vertex convolutions are thus undoubtedly useful, but refined weights in the final fully connected layer reduces their impact in situations where more training is allowed.

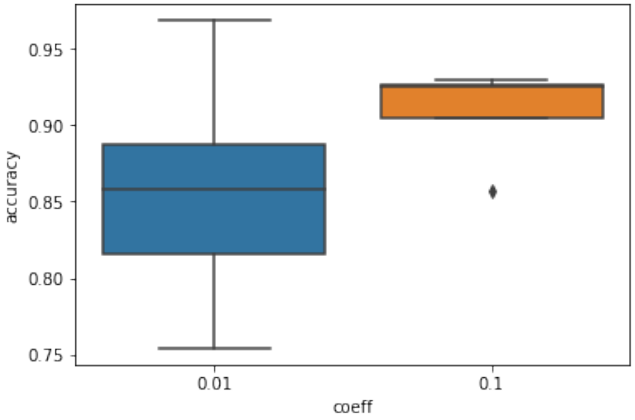


Fig. 26: Two TV-GCNNs that are constructed with the Laplacians of random graphs with different connectivity. The coeff represents the probability of a random edge between two nodes.(10 runs)

VI. Conclusion

Throughout this project, it is shown that the TV-GCNN performs better than temporal and vertex convolutions alone. However, temporal convolutions have much more impact on the performances than vertex filtering. The proposed theory is that the final fully connected layer progressively learns the correlations between vertices during the training processand removing this layer in the FC TV-GCNN leads to bad results.

REFERENCES

[1] Jason Brownlee. Developing convolutional neural network models for time series forecasting. Machine Learning Mastery Blog Post, 2018.

[2] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

[3] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[4] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pages 3844–3852, 2016.

[5] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[6] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[7] Federico Monti, Fabrizio Frasca, Davide Eynard, Damon Mannion, and Michael M Bronstein. Fake news detection on social media using geometric deep learning. *arXiv preprint arXiv:1902.06673*, 2019.

[8] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[9] G. Ortiz-Jimenez. Multidomain graph signal processing: Learning and sampling, Aug 2018.

[10] Youngjoo Seo, Michaël Defferrard, Pierre Vandergheynst, and Xavier Bresson. Structured sequence modeling with graph convolutional recurrent networks. In *International Conference on Neural Information Processing*, pages 362–373. Springer, 2018.

[11] David I Shuman, Sunil K Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE signal processing magazine*, 30(3):83–98, 2013.

[12] Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, and Chengqi Zhang. Graph wavenet for deep spatial-temporal graph modeling. *arXiv preprint arXiv:1906.00121*, 2019.

[13] Bing Yu, Haoteng Yin, and Zhanxing Zhu. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. *arXiv preprint arXiv:1709.04875*, 2017.