CS230

Exploring Graph CNNs for Smartphone Positioning

Adyasha Mohanty
Department of AeroAstro
Stanford University
madyasha@stanford.edu

Abstract

A majority of GNSS receivers today are installed in smartphones. These receivers however provide location accuracy of up to a few meters only due to noisy measurements, lower signal level, and multi-path effects in urban environments. Traditional model-based processing methods make assumptions about the noise models and require manual tuning of parameters such as covariances based on the operating environment. In contrast, learning-based approaches can model environment-specific errors and improve positioning accuracy in dense urban environments. In this work, we propose a Graph Convolution Neural Network based approach to learn positioning corrections from smartphone GNSS measurements. Our graph can model multi-constellation and multi-frequency signals as well as learn context-specific features to provide meter-level accuracies on diverse urban datasets, outperforming both model-based and learning-based baselines.

1 Introduction

Improving smartphone positioning can enable autonomous lane-keeping, advanced augmented reality technologies, and precise mapping applications. Due to constraints on GNSS chipset, size, and hardware cost, smartphone GNSS receivers can only provide meter-level accuracy in urban positioning. The accuracy is degraded further in the presence of buildings and tree canopy due to reflections and blocked signals.

Significant efforts have been underway towards improving the quality of positioning given the hardware constraints. For example, a novel Android GNSS application was released in 2016 (Humphreys et al. (2016)) which provided users access to raw GNSS measurements directly from the application. Additionally, Google released open datasets in 2020 and 2021 (Fu et al. (2020)) which were collected in multiple cities to promote novel positioning solutions. Traditional methods of processing smartphone GNSS measurements include factor graph optimization, Kalman filter and Weighted-least-squares (WLS). However, recently, there has been an emergence of learning-based techniques due to their ability in modeling environment and context-specific errors while making no assumptions on the underlying noise models.

In this work, we propose a Graph Convolution Neural Network (GCNN) that s a position correction given an initial receiver position. Given the success of GCNNs in operating on non-Euclidean data structures, we use them to learn a graph using satellite positions as nodes and preconditioned input features from a Kalman filter. Our GCNN can handle varying satellite visibility in urban environment, retains permutation-invariance properties via an unordered structure of nodes and can model measurements from multiple constellations and multiple signal frequencies. The GCNN takes the following inputs: a) a feature matrix that is constructed using satellite geometry and signal characteristics such as signal-to-noise ratio and measurement residuals and c) an adjacency matrix

CS230: Deep Learning, Autumn 2022, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

that describes the connections among the different nodes in the graph. Given the graph structure, the GCNN infers a 3D position correction in an end-to-end manner. The correction is then applied to the coarse position to obtain the final receiver position.

2 Related work

In Siemuri et al. (2021), the authors trained three machine learning algorithms such as Linear Regression, Bayesian Ridge Regression and a Bayesian Neural Network to predict the positioning correction. The results showed that the weighted combination approach outperformed all three algorithms in terms of positioning accuracy. Another line of work (Kanhere et al. (2021)) used a Set Transformer (Lee et al. (2019)) approach to learn the correction using pseudorange measurement residuals and satellite geometry. This approach outperformed WLS method on real-world data but provided limited positioning accuracy due to reliance on a single type of range measurements. In (Han et al. (2021)), the authors utilized reinforcement learning to tune measurement noise covariances in a Kalman filter that combines GNSS-IMU measurements but the filter required tuning the reward function for the measurement noise covariances. Lastly, our prior work (Mohanty and Gao (2022)) proposed a hybrid approach using a simple Graph CNN and a Kalman filter to learn a positioning correction. However, this work did not explore a variety of GCNN architectures and edge connections to improve the positioning performance.

3 Dataset and Features

We use the Google Smartphone Decimeter Challenge (GSDC) 2021 datasets (Fu et al. (2020)) GSDC Dataset that contain trajectories from different phones and different cities. We use the following observables: ground truth from a high-grade GNSS-inertial system containing ECEF X,Y and Z positions of the phone receivers, GNSS range measurements from multiple constellations and signal frequencies, signal-carrier noise measurements and uncertainties associated with each GNSS measurement. We split the datasets into 74 training datasets and 12 test datasets.

Preprocessing: We first estimate an initial receiver position using a Kalman filter on the GNSS range measurements. We then group all the measurements into different constellations and signal types, followed by eliminating inter-system and inter-frequency biases, clock biases, tropospheric and ionospheric errors. Using the satellite groups and range measurements, we construct feature vectors that contain the measurement residual, LOS vector, range uncertainty and signal-to-noise ratio. The range uncertainty and the signal-to-noise ratio values are used directly from the raw measurements. Sample pictures showing the distribution of values for some of the above features are shown in Fig. 1.

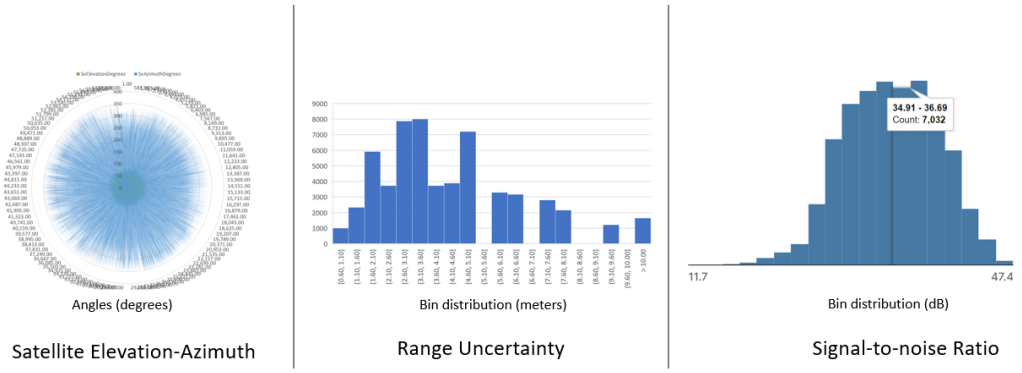

Figure 1: Analysis of a sample training dataset from San Jose which shows the distribution of the satellites in a skyplot, range uncertainties and signal-to-noise ratio values.

4 Methods

Graph Structure: In the GCNN, each node is represented using the satellite position p_t^j. Each node in the graph uses a 6 1 feature vector which is constructed during the preprocessing phase.

We establish edges in the graph if the nodes satisfy the following conditions: a) they belong to the same GNSS constellation b) difference in measurement residuals per satellite (node) $< 5\ m$ and c) pseudorange uncertainty $< 5\ m$. The thresholds for making the edge connections were determined by analyzing the mean and standard deviation of the measurements and identifying data points beyond the 1 value.

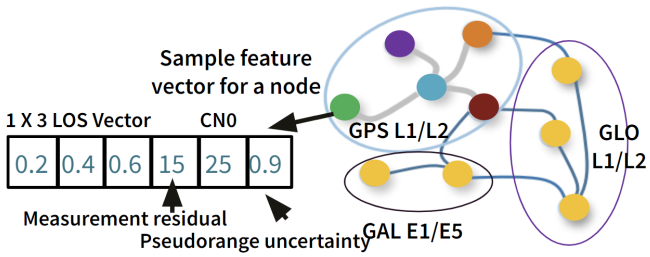

Figure 2: Sample structure of the graph showing various nodes depicting satellite positions and node groups that are classified according to constellation type.

Convolution Layers and Prediction: The GCNN has two convolution layers which are modeled using the GINConvolution operator Xu et al. (2019). GIN operators satisfy the WL graph isomorphism test and achieve maximum discriminative power among other choices of convolution layers. The GIN operator updates the node representation in the graph as:

$$h_v^k = MLP^K(1 + \ ^k)h_v^{k-1} + \sum_{u \in \mathcal{N}(v)} h_u^{k-1} \tag{1}$$

where MLP denotes a multi-layer perceptron that can represent the composition of several functions, is a fixed scalar, h_v^k is the feature vector of node v as the kth layer and $N(v)$ is the set of nodes adjacent to the node v.

We perform mean pooling across all the graph nodes which instantiates message passing among different nodes of the network, allowing the neighboring nodes to update their features and weights concurrently, and also reduces the spatial resolution of the graph for subsequent layers. This operation is shown as:

$$h_v^k = ReLU(W \quad mean(h_u^{k-1}; 8u\ 2\ N(v)\ [\ v)) \tag{2}$$

where $ReLU$ is the activation function and W is a learnable matrix containing weights for the previous layer. Post-aggregation, we pass the updated features at every node through a series of fully connected linear layers which increase graph expressivity and improve our final prediction. The GCNN is trained with a standard Euclidean loss function given by

$$Loss = \sum_{i=1}^{N} jj\ \ x_t \quad \Delta x_t jj_2 \tag{3}$$

where the first term is the predicted position correction from the GCNN and the second term is the true correction calculated from the Kalman filter position and the true receiver position. We use ReLU layers for activations, layer norms for faster convergence, and Dropout to prevent overfitting on small datasets.

5 Experiments/Results/Discussion

Baselines: We compare our proposed algorithm against two baselines: a temporal method which is a Kalman filter solution and a fully connected network. The Kalman filter uses raw GNSS measurements as inputs and has been tuned for maximal performance using Bayesian hyperparameter optimization. The fully connected network takes as inputs the same features as our proposed GCNN

and consists of 5 linear layers with ReLU activations. The hidden layers have 1000, 800, 500 and 250 neurons, respectively. We also apply Dropout with a probability of 0.5 in the last layer.

Hyperparameters: In our graph CNN, we select 128 as the size of the hidden layer and dropout probability of 0.5 to prevent overfitting considering the small size of the training datasets. We also experiment with two different types of egde connections. A loose edge connection indicates that the edges are connected only if they belong to the same constellation or have similar measurement residuals with a threshold set to 5 m. A dense connection represents that the nodes may be connected if they also have similar uncertainties in the pseudorange measurements, with a threshold value set to 15 m for both the measurement residual and the pseudorange uncertainty.

We use Pytorch Geometric (Fey and Lenssen (2019)) to train the GCNN for 10 epochs using the MSE loss function described previously, with a learning rate of 0.01, weight decay of 5e 4, and Adam optimizer (Kingma and Ba (2015)). From preliminary experiments, we found that performing stochastic gradient descent resulted in the best positioning performance compared to mini-batch gradient descent. We performed training with freely available GPUs on Kaggle, which is a Google platform for ML research. The network architecture (best perofmance) is described below.

Table 1: Parameters of the GCNN architecture

Module	Layer	Parameter
GINConv(1)	Linear	6 128
	ReLU	-
	Linear	4 32
GINConv(2)	Linear	32 32
	ReLU	-
	Linear	32 32
	LayerNorm	32
Post-Message Passing	Linear	32 32
	Dropout	p = 0.25
	Linear	32 3

We use quantitative metrics such as the mean, median, maximum and minimum horizontal positioning error across all 12 test datasets. The horizontal positioning error is described by the loss function given previously and calculates the difference in the horizontal (2D) positioning correction between the prediction from the graph CNN and the true correction from ground truth. For qualitative results, we compare the predicted trajectory from all the algorithms w.r.t. true trajectory.

We first analyze the horizontal positioning error across all test datasets. As shown in Table 4, our GCNN outperforms both baselines for each metric. Even though the number of available training datasets is small, by using Dropout and GIN Convolution layers, and early stopping, we are able to prevent overfitting of the network on the test datasets. However, the fully connected network overfits on the test datasets as observed from the large maximum positioning and mean errors.

Table 2: Summary of positioning error on test datasets. Our best performing GCNN outperforms all baselines across 12 test datasets.

Error Metric (meter)	Kalman Filter	Fully Connected	Our Approach
Mean	4.5	6.1	3.1
Minimum	2.4	1.9	1.3
Maximum	8.4	11.2	4.5

We show qualitative results via trajectory plots in Fig. 3 that are generated by plotting the position predictions from our GCNN, the Kalman filter baseline and the fully-connected network baseline for a Los Angeles dataset. The figure indicates that our algorithm is able to follow the ground truth trajectory more closely compared to the baselines. Although the predictions from the baselines show some deviations, the GCNN is able to fine tune the correction and compensate for the deviations leading to improved results from our algorithm.