ANEDA: Adaptable Node Embeddings for Shortest Path Distance Approximation

Frank Pacini[1], Yuan Liu[1], Chau Pham[1], Maximilian Katzmann[2], Sarel Cohen[2], Peter Chin[1], Tobias Friedrich[2]

[1] Boston University

[2] Hasso Platner Institute, University of Potsdam

ABSTRACT

Shortest path distance approximation is a crucial aspect of many graph algorithms, in particular the heuristic-based routing algorithms that make fast, scalable map navigation possible. Past literature has introduced deep learning models which try to approximate these distances by training on graph embeddings (i.e. node2vec, Gra100, ProNE, Poincare). We propose a more lightweight technique to approximate graph distances, ANEDA (Adaptable Node Embeddings for Distance Approximation), where we instead train the embeddings directly, using either previous embedding methods or geographic coordinates as a good initialization. Our approach also offers the flexibility to use different distance measures to fit the expected structure of the graph, or generate graph representations with desired properties. Through experiments on several social and geographic networks, we show our model's error reduction of up to 75% against two recent deep learning approaches, and its competitive performance against the larger, state-of-the-art architecture.

1 INTRODUCTION

Given a weighted graph $G = (V, E)$ and an embedding dimension D, the goal of this work is to find a combination of an embedding map $\phi : V \longrightarrow \mathbb{R}^D$ and a distance measure $M : \mathbb{R}^D \times \mathbb{R}^D \longrightarrow \mathbb{R}_+$ such that for nodes $u, v \in V$ and distance between them $d_{u,v}$,

$$\hat{d}_{u,v} = M\big(\phi(u), \phi(v)\big) \approx d_{u,v}.$$

Finding such an embedding map is useful because we can represent the graph by effectively approximating the $|V| \times |V|$ adjacency matrix, which may be infeasible to store or calculate for large graphs, in a $|V| \times D$ embedding matrix containing the vectors $\phi(v)$. Querying for a distance approximation would then take $O(1)$ to retrieve $\phi(v)$ and (for reasonable choices) $O(D)$ to apply M.

1.1 Node Embedding Techniques

General approaches to produce an embedding map ϕ for graphs include various random walk strategies [12], [6], [15], [13], [5], [17] which define node representations based on cooccurence in these walks, and matrix factorization techniques [1], [4], [11], [20] which manipulate the graph adjacency matrix to more explicitly represent multi-level graph structure. We focus our investigation on two well-established techniques utilizing each approach: Node2Vec [6] and GraRep [4]. Node2Vec utilizes the Skip-Gram model to generate embeddings, which encodes pair associations as a function of the dot product. GraRep on the other hand computes and then merges k-step transition probabilities of node pairs to produce their final representations.

Figure 1 from [3] visualizes the difference between pairs of embeddings generated by Node2Vec and GraRep on the Winterthur, Switzerland OSMnx [2] graph, where the shortest path distances have been divided by the graph diameter. In GraRep (with order 100), we can see that the embedding difference approximates the distribution of graph distances fairly well, whereas Node2Vec produces larger embedding distances that may result in faster training when used in our model.

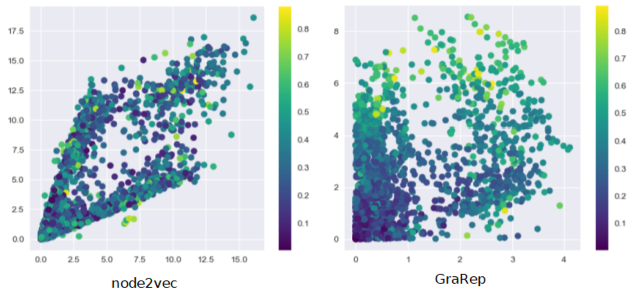

Figure 1: WTUR 2d vector difference with SP distance labels

1.2 Distance Approximation Networks

An important issue when trying to use node embeddings for distance approximation is that most well-known techniques are not explicitly designed for distance preservation, and are instead evaluated on more general downstream tasks such as classification or clustering. In response, several deep learning architectures have been proposed to train on graph embedding inputs as a strategy for distance approximation ([16], [14], [3]). They formulate the distance approximation problem as a machine learning task by simply precomputing the shortest path distances on a subset of node pairs, and using the model to generalize to a full representation matrix. These approaches all use an existing graph embedding technique for ϕ and then try to find an optimal measure M using deep learning, or train both jointly.

1.3 Heuristic-based Routing

An important application for these distance approximation techniques is network routing. Dijkstra's is the well known standard algorithm for finding shortest path routes between a source node u and target v, however in practice it is too slow to be feasible for large scale routing or very large networks. A* [8] is an optimization to Dijkstra's that uses an adjustable heuristic function h to improve

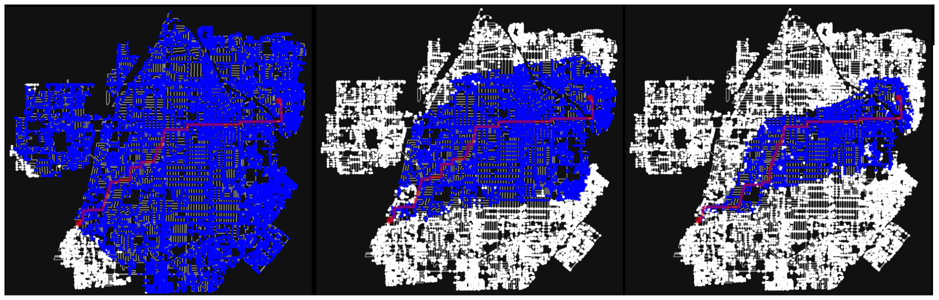

Figure 2: Santa Ana, CA routing visits with different A* heuristics

performance, at the (usually small) sacrifice of finding an approximate shortest path. In order to prioritize visiting nodes that are expected to bring the algorithm closer to the target v, A* queries h for approximations of the distance between each candidate node w and v. A* performance is heavily dependent on the quality of h, so we want to provide A^* with distance approximation functions that have as little prediction error as possible, given the distance data that we can feasible calculate for G.

Figure 2 illustrates the impact of heuristic choices by comparing nodes visited in a street navigation task. In this example and for all other street networks presented in this work, each node is a street corner, and each edge weight gives the travel distance along a particular street between two nodes. Labeled in blue is the nodes visited by Dijkstra's (which is equivalent to A* with $h = 0$), A^* with h as the geographic distance between the nodes (from their lat-long coordinates), and A^* with an embedding model from early in our investigation. A^* using the geographic distances improves upon Dijkstra's by avoiding paths in the entirely wrong direction from the target, but does not sufficiently model network topology to give very good approximations, for instance in the case of indirect routes or minimally connected components. Training on graph distances directly, as in our model, helps to mitigate these issues.

2 ANEDA

As opposed to previous networks, our approach is instead to explicitly choose a function M for the vectors, initialize $\phi^* = \phi$ and then try to find an optimal ϕ^*. Our model is therefore an embedding matrix similar in spirit to Node2Vec, initialized with a graph embedding not explicitly trained for distance approximation. We then tune this initialization in order to get improved shortest path approximations with respect to a chosen distance measure, such as the Euclidean distance or dot product. We focus our research on finding good choices for initialization (**2.2**) and distance measure (**2.3**). For simplicity, in the rest of the paper we also denote $\mathbf{u} = \phi(u)$ when ϕ is our model.

Besides comparable performance to state-of-the-art techniques, we believe the merits of our approach are:

(1) Adaptability. Our model can used to find node representations in a particular space by tweaking the distance measure. This could be useful in order to create distance-preserving graph embeddings in spaces with desired properties, or tailor representation for graphs with known or theorized properties. The model can also be easily exported as a 2d array and queried for downstream tasks.

(2) Size. Our model is an embedding matrix of size $|V| \times D$, whereas approaches such as Vdist2vec [14] contains an embedding layer of the same size as part of its larger model, which is likely to improve train and evaluation time. Our model can optionally be initialized with another embedding of size $|V| \times k$, where $k \leq D$, that can be discarded after initialization. Vdist2vec on the other hand requires a one hot input of size $2|V|$ per example to train.

(3) Simplicity. Our model requires only an embedding matrix and trainer (we use node2vec as our base code), plus a simple distance measure in order to implement the model for a specific task.

2.1 Data Processing

In order to produce node pairs with distance labels for the model to train with, we follow a similar approach to [16], which is to randomly select l nodes from the graph to use as landmarks, and compute the distance between this landmark and all other nodes using Dijkstra's. Given landmark u and other node v, each training example in our dataset is simply $< u, v, d_{u,v} >$. Since we are tuning the embeddings directly, we only need the node indices as inputs. Our embedding is instead queried when evaluating the loss, similar to Node2Vec [6], whereas the approach in [16] may need to store the full vector input since it is a function of the node embeddings for u and v. This greatly reduces the dataset space requirements of our approach, since each example requires storing 3 values instead of $D + 1$ values.

Another notable difference is [16] makes their validation sets include entirely distinct nodes from the test set. We cannot do this for our model since it would prevent embeddings for the validation nodes from ever being trained, so we instead remove train landmarks from the validation sets in order prevent the two sets from having any of the same pairs. As suggested by [3], for weighted graphs we also normalize the edge weights by dividing by the largest across all edges before computing the distance labels.

2.2 Initialization

In order to improve training time and potentially performance, we use two embedding types as initializations for the model.

For any graph, we can train graph embeddings not designed specifically for distance approximation, such as Node2Vec, and then tune our model to produce embeddings which approximate graph distances very well. We test specifically with Node2Vec [7] and GraRep [4] embeddings.

If the graph is geographic, we can also use geographic coordinates of each node as an initialization. We evaluate on several geographic networks with either latitude-longitude coordinates and UTM coordinates of the nodes included. For latitude-longitude coordinates, where $\varphi \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ = latitude and $\lambda \in [-\pi, \pi]$ = longitude, in general we convert to 3 dimensional Cartesian coordinates using the identity

$$x = \cos(\varphi)\cos(\lambda), \quad y = \cos(\varphi)\sin(\lambda), \quad z = \sin(\varphi)$$

This is done since differences in latitude and longitude reflect distances differently depending on location, and the change also results in coordinates on the same scale as the normalized graph distances. No changes are made to UTM coordinates other than normalization since these have a preapplied projection into 2 dimensional Euclidean space.

We make use of k-dimensional coordinate vectors if available of graph embeddings to initialize the first k of D-dimensional embeddings. The remaining $D - k$ dimensions are initialized at random from a normal distribution with a variance of $1/D$. This allows us to train embeddings of a much larger size than the 2 or 3 dimensional geographic coordinates, or add additional dimensions to initial embeddings for tuning. In general, though, we train initial graph embeddings using $D = k$.

The different initialization techniques also offer the opportunity to choose distance measures for training with an expected relationship to the initial embedding. For instance, when using the Euclidean distance for training and latitude-longitude initialization (transformed as mentioned above), the initial distance measure is an underapproximation (although very accurate for small sections of the Earth like cities) of the geographic distance between the nodes. This measure is a reasonable baseline approximation for downstream tasks like routing, as we observe in later experiments. In Figure 3, we visualize how these initial coordinates are projected by the model in a graph where the geographic distance does not perform adequately. This is done by applying the transformation from Lat-Long to Cartesian for the initialization, applying the reverse on the output embeddings and then modifying the graph coordinate attributes. The Santa Ana, CA graph from OSMnx has two "islands" which, except for several bridging edges, are separated by highways that extend outside the city boundaries. The model moves these islands up and farther left and right, while also condensing them. Nodes in regions with many short edges are generally condensed, while holes in the graph are created in areas with fewer, longer streets. This model was trained with $d = k = 3$, so this visualization may not fully reflect the representations learned in embeddings with many additional dimensions.

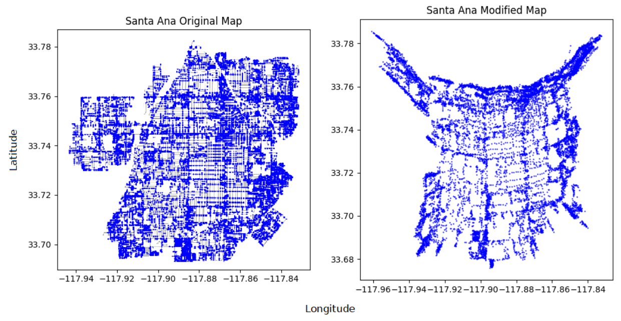

Figure 3: Initial coordinate modification by ANEDA ($D = 2$)

2.3 Distance Measures

We tested several different distance measures to train the embeddings on, including common norms for Euclidean geometry and measures for hyperbolic and elliptical geometry.

2.3.1 p-norm. Using several different p values, we tested the p-norm of the difference between two node embeddings which is given by

$$\hat{d}_{u,v} = \left(\sum_{i=1}^{D} (\mathbf{u}_i - \mathbf{v}_i)^p \right)^{1/p}.$$

$p = 1$ gives the taxicab distance and $p = 2$ gives the common Euclidean distance. Higher p values produce smaller distances and increase the weight of the larger coordinates, which in this case are the elementwise differences between $\mathbf{u}$ and $\mathbf{v}$. The limit of this is the ∞-norm where

$$\hat{d}_{u,v} = \max_i(\mathbf{u}_i - \mathbf{v}_i).$$

2.3.2 Poincare Hyperbolic. We additionally use two measures of distance in hyperbolic geometry based on the Poincare Disk and Minkowski Hyperboloid models for hyperbolic space.

In the 2d Poincare Disk model, points are fixed inside the unit disk, with the edge of the disk representing infinite distance. As mentioned in [18], we can extend this idea to the D-dimensional unit ball and then the distance measure is

$$\hat{d}_{u,v}(u,v) = \operatorname{arcosh}\left(1 + 2\frac{||\mathbf{u} - \mathbf{v}||^2}{(1 - ||\mathbf{u}||^2)(1 - ||\mathbf{v}||^2)}\right).$$

As the model construction and formula itself would suggest, the Poincare model requires that $||\mathbf{u}|| < 1$ for all nodes u. For the embeddings this means we must fix the Euclidean norm of the initial embeddings to some value $c < 1$, where $c \approx 1$ produces very high distances and $c << 1$ produces low distances. Through empirical testing, we found that when using Cartesian coordinates, $c = \sqrt{2} - 1$ produced the best local approximation for the geographic distance on graphs from several parts of the Earth, sometimes with a smaller margin of error than the Euclidean distance. The reason for this optimal value is another area to investigate. The norm setting in geographic graphs allows us to test whether tuning embeddings to hyperbolic geometry is useful in graphs with a standard, physical geometry.

2.3.3 Minkowski Hyperbolic. In the Minkowski Hyperboloid model, points are fixed inside the forward sheet of a two sheeted hyperboloid in $D+1$ dimensional space. All points $\mathbf{x} \in \mathbb{R}^{D+1}$ must satisfy

$$\mathbf{x}_0^2 - \mathbf{x}_1^2 - \cdots - \mathbf{x}_D^2 = 1.$$

Then the distance measure between 2 points $u, v \in \mathbb{R}^{D+1}$ is

$$\hat{d}_{u,v}(u,v) = \operatorname{arccosh}(\mathbf{u}_0\mathbf{v}_0 - \mathbf{u}_1\mathbf{v}_1 - \cdots - \mathbf{u}_D\mathbf{v}_D).$$

In practice, in order to enforce the constraint for each point $\mathbf{x}$, we keep its values in the last D dimensions and set the first dimension, x_0 to:

$$x_0 = \sqrt{||\mathbf{x}||^2 + 1}$$

The measure becomes

$$\hat{d}_{u,v}(u,v) = \cosh^{-1}\left(\sqrt{(||\mathbf{u}||^2+1)(||\mathbf{v}||^2+1)} - \mathbf{u}\cdot\mathbf{v}\right).$$

2.3.4 Elliptic. We also tested using measures in a positive curvature geometry since this is the true geometry of geographic points. Assuming a radius of 1, the elliptical distance is simply the angle between the points in radians, which can be obtained from the dot product:

$$\hat{d}_{u,v} = \cos^{-1}\left(\frac{\mathbf{u}\cdot\mathbf{v}}{||\mathbf{u}||||\mathbf{v}||}\right).$$

A natural initialization to use with the elliptic distance is geographic coordinates (still with the Cartesian transformation), since this gives the geographic distance exactly, and so should in theory provide a better initialization than with the Euclidean distance measure.

Another intuitive pairing is Node2Vec vectors, since they are trained on the dot product. In the Node2Vec loss, the sigmoid is applied so that vectors which are close together produce a prediction of 1, which means they are expected to cooccur in a random walk, and vectors far apart produce 0. The inverse cosine in elliptical distance in a sense reverses this by setting the distance to 0 if the normalized dot product is 1, which means the embeddings overlap, and setting the distance $\frac{\pi}{2}$ if the normalized dot product is -1, which means the embeddings are in opposite directions. A caveat of this is we must ensure the weights are normalized so that the diameter of the graph is $< \frac{\pi}{2}$ or scale the measure, since otherwise the embeddings would not be able to produce large distances.

2.3.5 Inverse Dot. With inspiration from the elliptic distance, we tried several other functions besides inverse cosine which would take the normalized dot product δ and "invert" it by mapping $\delta = -1$ to a large distance and $\delta = 1$ to 0. We found the most effective to be

$$\hat{d}_{u,v} = \left(1 - \frac{\mathbf{u}\cdot\mathbf{v}}{||\mathbf{u}||||\mathbf{v}||}\right) * d_{max}/2$$

where d_{max} is the graph's diameter. In practice, if d_{max} is unknown for a large graph it can estimated as long as it greater than the true diameter, since we just need that the measure is able to predict d_{max} given the constraints on δ.

3 EXPERIMENTS

3.1 Datasets

We ran experiments on several graphs in order to compare our model to previous work. Hartford, Connecticut is a geographic driving network from OSMnx [2], which is shown in 4. Surat, India and Dongguan, China are geographic networks from the CSUN lab[9], and ego-facebook-original is a social network graph from the SNAP dataset [10], and is unweighted as opposed to all of the geographic graphs.

Table 1: Descriptions of datasets used in our experiments

	$\lvert V\rvert$	$\lvert E\rvert$	AVG DEG	d_{max}
HARTFORD (**HF**)	1581	2470	2.75	12 KM
SURAT (**SU**)	2591	3670	2.86	51 KM
EGO-FACEBOOK (**FB**)	4039	88234	2	8
DONGGUAN (**DG**)	8315	11128	2.32	160 KM

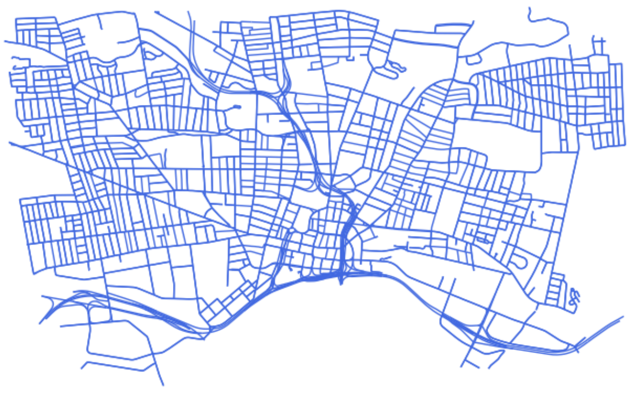

Figure 4: Hartford, Connecticut network from OSMnx [2]

3.2 Baselines

We compare our model against three previous deep learning architectures for distance approximation: Qi et al. [14] which introduces Vdist2vec, Rizi et al. [16] and Brunner [3]. We tailor our experimental setup for comparison purposes. We use $D = k = 128$ embeddings for all experiments except against Qi et al. We train Node2Vec with the recommended settings of walk length = 80, $p = q = 1$, 10 walks per node, 100 epochs, 256 batch size and lr=0.01, however we use a window size of 10 for Brunner and 5 for Rizi et al. since they differed in this respect. For Qi et al. we also use window size of 10. For GraRep, GraRep10 and GraRep100 denote order 10 and 100 embeddings, respectively. We use 10 SVD iterations for all experiments and averaging to merge the k-step representations. Each experiment setup is detailed in the following sections.

3.3 Experiment A: Vdist2vec Comparison

Comparing against Qi et al, we follow the same experiment setup, besides deviating from their learning rate of 0.01 where noted, and compare on **SU** and **DG** graphs. We similarly use $2\% * |V|$ dimensional embeddings, and train on all pairs in the graph for 20 epochs with lr=0.01, batch size = $|V|$ and Mean Relative Error (MRE) loss. When reporting results, we divide their Mean Absolute Error (MAE) by the max graph weight as in other experiments for proper comparison.

3.4 Experiment B: Rizi et al. Comparison

We compare to the original approach of Rizi et al. since they explicitly utilize node2vec for model inputs, and we can compare this to the impact of our embedding initialization with node2vec. They also allow us to analyze our architecture in a non-geographic setting with likely a very different graph topology.We follow a similar setup to Rizi et al for running on **FB**, with some slight deviations. As mentioned previously we cannot completely separate the train and test set nodes, and we do not perform any undersampling or removal of pairs based on distance as they suggest. We also use the same number of landmarks $l = 100$ and have the expected dataset size $|V| * l = 400,039$, however they report a *larger* train set size of $1,022,640$ despite our lack of undersampling. We also use 25 landmarks for testing to produce a similar test set size of ≈100,000. Besides this, we retain the same hyperparameters except where noted, that is, we run for 15 epochs with batch size 1024, lr=0.01 and MSE loss. Since the graph is unweighted, Rizi et. al round their predictions before reporting test error, so we do so as well.

3.5 Experiment C: Brunner Comparison

Brunner [3] uses coordinate initialization for training their GCN models. Our primary change in approach in this respect is to allow the coordinate embeddings to be tuned during training rather than remaining static inputs. In this experiment we assess how our model is able to utilize the combination of coordinate initialization and distance labels in comparison to the Brunner models. Therefore the initialization for all models is set to geographic coordinate.

Using the same **SU** graph from Qi et al, we follow Brunner's experimental setup precisely. We first max-normalize the graph edges and node coordinates during preprocessing. We train with the Adam optimizer and mean relative error for loss. For hyperparameters, we set lr=0.001, batch size = 512, a train ratio of 10% and we test on the remainder. Brunner divides the mean absolute error by the graph diameter in their results, which we reverse for consistency across experiments. We also make no tweaks to parameters per model for this set of results.

3.6 Experiment D: Hartford Routing

Lastly, we assess our methods for distance approximation on a downstream routing task. We chose the Hartford driving graph from OSMnx [2], which is displayed in **Appendix 1**. This graph is small enough that we can run routing on all pairs of nodes and report performance percentiles. The graph also has a simple, but not grid-like layout which should make it a fair baseline for larger cities and geographic networks. Based on a comparison test, we chose a heuristic weighting of $\alpha = 1.5$ as the optimal trade-off between node visits and shortest path distance stretch. For this experiment we used a smaller train ratio of 3% of nodes as landmarks, which would be more realistic for larger graphs. To ensure model convergence, we use full batch gradient descent with lr=0.01 and train for 1000 epochs. This also makes near certain that all models will converge.

To measure our performance on routing we define a basic metric. If $S_{u,v}$ is the unweighted shortest path length between u and v, and $T_{u,v}$ is the number of nodes visited while routing (included repeated visits), then $Q_{u,v}$ is simply

$$Q_{u,v} = 100\% * \left(1 - \frac{S_{u,v}}{T_{u,v}}\right)$$

which is the percent of visited nodes that were not strictly necessary to perform path routing. We report Q-percentiles for each technique, running on all pairs of nodes in the graph. Evaluating with Q makes sure we are accounting for the actual path length, since most paths are short and should require very few visits. However, it is important to note that higher Q values are much more likely to be short paths with inefficiencies, which would ultimately impact routing time less than long paths with the same Q value. For this reason, even though it is not a highly accurate metric, we also report overall time taken to route all pairs in our results.

4 RESULTS

Table 2 summarizes the best performing technique for each approximation experiment vs the comparison model. We produce models which surpass two current approaches to distance approximation on geographic and social networks. Against the state-of-the-art approach for geographic networks, vdist2vec, we achieve a superior error on one of our comparison graphs, and surpass their base model in MAE and MRE on the other graph. Of our models, Inv-Dot performed the best at distance approximation and routing consistently in the geographic networks we tested. However, it failed on the unweighted social network graph and the L6-Norm was most successful, as we discuss later.

4.1 Experiment A: Vdist2vec Results

In table 1 and table 2, we show results against Vdist2vec on two geographic networks: **SU** and **DG** from [14]. We observe that at least in geographic networks, our Inv-Dot model is competitive with Vdist2vec but does not surpass it entirely in error as we will see with the other experiments. On **SU**, Inv-Dot performs between the initial Vdist2vec and Vdist2vec-S in terms of MRE regardless of initialization. When additionally trained on MAE, our model surpasses the base Vdist2vec entirely. On **DG**, our Inv-Dot with node2vec initialization model achieves state-of-the-art in terms of MRE, which occurs whether we train with MAE or MRE as the loss. The results also illustrate a significant difference between Inv-Dot and the 6-norm, which is that Inv-Dot trains quickly enough to converge to a low error regardless of initialization and to some extent the learning rate. The L6 norm needed its learning rate lowered from the experiment default for stability, and showed more variation of results based on initialization.

4.2 Experiment B: Rizi et al. Results

In the non-geographic setting, we show significant performance increase over the Rizi et al. model as evidenced by table 3 showing approximation errors on the ego-facebook-original graph. Two of our models, elliptic and L6 norm with node2vec, show serious performance jumps relative to the Rizi et al. model. We display their result with the best (averaging, denoted $\oslash$) or second best (subtracting, denoted $\ominus$) performing input vector merging scheme for this graph. We should note that the elliptic measure result could be considered more significant since it was achieved after

Table 2: Experiments Summary

Experiment	A - Surat			A - Dongguan			B - Facebook			C - Surat		
	Method	MAE	MRE	Method	MAE	MRE	Method	MAE	MRE	Method	MAE	MRE
Proposed	Inv-Dot n2v	0.0098	0.027	Inv-Dot n2v	0.0196	**0.01**	Elliptic	0.0272	0.0157	L6	0.0291	0.0357
	L6 n2v	0.0194	0.0261				L6 n2v	**0.0267**	**0.0143**	Inv-Dot	**0.0161**	**0.0229**
Baselines	Vdist2vec	0.0113	0.027	Vdist2vec	0.0118	0.015	Rizi ⊘	0.118	0.038	CN	0.0265	0.0378
	Vdist2vec-S	**0.0067**	**0.0114**	Vdist2vec-S	**0.0062**	0.014	Rizi ⊖	0.197	0.071	SCN	0.0238	0.0309

Table 3: Experiment A - Surat

Technique	Initialization	Performance		Params
		MAE	MRE	
L6-Norm	Random	0.0206	0.0266	lr=0.001
	Coord	0.0328	0.034	
	node2vec	0.0194	0.0261	
Inv-Dot	Random	0.0161	0.0197	
	Coord	0.0163	0.0207	
	node2vec	0.0159	0.019	
	GraRep	0.0164	0.0209	
Inv-Dot MAE	node2vec	0.0109	0.0239	
Inv-Dot MSE	node2vec	0.0098	0.027	lr=0.0003
Vdist2vec base	N/A	0.0113	0.027	
Vdist2vec-S		**0.0067**	**0.014**	

Table 4: Experiment A - Dongguan

Technique	Initialization	Performance		Params
		MAE	MRE	
L6-Norm	Random	0.0355	0.0286	lr=0.001
	Coord	0.0679	0.0374	
	node2vec	0.0349	0.0304	
Inv-Dot	Random	0.0207	0.0345	
	Coord	0.021	0.0121	
	node2vec	0.0196	**0.01**	
Inv-Dot MAE	node2vec	0.0178	0.0123	
Inv-Dot MSE	node2vec	0.0211	0.0171	
Vdist2vec base		0.0118	0.015	
Vdist2vec-S		**0.0062**	0.014	

lowering, rather than raising the learning rate from the default for the experiment. Tweaking the learning rate down was helpful for most models, which is perhaps related to the low number of epochs set by Rizi et al. for this experiment.

One notable comparison in our results is between node2vec and GraRep initialization for the L6 norm. We ended up using node2vec in most experiments for our embedding initialization, since we found it performed equally or better to GraRep. This opposes the results from evaluating experiments in [3], which, based on performance of embeddings as inputs to their evaluation model, concluded that node2vec performed worse than GraRep for distance preservation. Our results instead show that node2vec embeddings are more easily tuned to be distance oreserving. It may be that our setup for GraRep is distinct from [3]: we use averaging of the k-step representations whereas they do not specify. In addition to its performance for initialization, node2vec has the added benefit of being a good pairing with measures that apply a transformation dot product such as elliptic and Inv-Dot, since node2vec is trained on the sigmoid of the dot product.

The most curious result from this experiment was the very poor performance of Inv-Dot, as opposed to its domination in other experiments. In this setting, Inv-Dot either showed unstable gradient descent or immediately overfitted even after the learning rate was drastically reduced. This performance is also unintuitive since the highly similar elliptic measure performs second to best on this graph. The primary difference between elliptic and inv-dot is that the former applies the non-linear inverse cosine to the normalized dot product whereas the latter applies a linear map into the same range. Both are scaled so that they produce distances between 0 and the graph diameter. This result refutes our initial hypothesis that the elliptic measure performed poorly due to producing an ineffective gradient. The additional poor performance of distance measures from the Minkowski hyperbolic model and Poincare (which we could not train successfully due to gradient issues) is also unintuitive considering theoretical results suggesting that hyperbolic space is a good embedding space for social networks [19]. To motivate future work, we suggest three possible causes for these results:

(1) The Facebook graph has special characteristics and performance on other social networks should be investigated in future experiments.

(2) Training with specific distance measures does not always result in mapping into the desired space, so improvements to enforce this could result in richer embeddings.

(3) The success of certain distance measures does not relate to graph topology in the ways we have assumed when choosing measures. For instance, a graph fitting a hyperbolic model would not imply a hyberbolic distance measure will result in the best representation.

4.3 Experiment C: Brunner Results

Against Brunner we find that with coordinate initialization several of our measures are competitive with their graph neural network architecture, as shown in table 4. L4, L6 and L8 are all within the same range of performance as these networks, illustrating that rich representations can be learned even when the distance measure isn't an ideal approximation with respect to the provided initial coordinates. The more typical L2 distance also performs in the range in MAE, even though we train on MRE. As expected, hyperbolic

Table 5: Experiment B - Facebook

Technique	Initialization	Performance		Params
		MAE	MRE	
L2-Norm	Random	0.114	0.0586	lr=0.008
L6-Norm	Random	0.0482	0.022	lr=0.001
	node2vec	**0.0267**	**0.0143**	lr=0.03
	GraRep	0.1317	0.056	
Elliptic	Random	0.0272	0.0157	lr=0.0003
Poincare	Random			
Minkowski	Random	0.4068	0.1464	lr=0.005
Inv-Dot	Random	0.6278	0.1637	lr=2e-5
	node2vec	0.5001	0.1365	
Rizi	node2vec ⊘	0.118	0.038	
Rizi	node2vec ⊖	0.197	0.071	

Table 6: Experiment C - Surat

Technique	Performance	
	MAE	MRE
L2-Norm	0.0411	0.0605
L4-Norm	0.0299	0.0385
L6-Norm	0.0291	0.0357
L8-Norm	0.0307	0.0372
Poincare	0.0951	0.0404
Minkowski	0.3525	0.2101
Inv-Dot	**0.0161**	**0.0229**
CN	0.0265	0.0378
SCN	0.0238	0.0309
SECN	0.0433	0.0450
SACN	0.0419	0.0436

distance models do not perform as well as p-norms or dot product, which are more intuitive measures for graphs representing physical geometries. As with the previous experiment on geographic networks, our best performance comes from the Inv-Dot measure, which surpasses all comparison models on this experiment.

Additionally, although we do not report approximation error results on OSMnx graphs, we found that in practice, 3d Cartesian coordinates were more useful for model initialization than the 2d UTM format coordinates from the CSUN graphs (**SU** and **DG**). This could be because the UTM format introduces some projection error that does not aid in the model finding its own projection, or because the Cartesian coordinates offer more dimensions for the model to tune on. This means that our results with coordinate initialization here may be an underevaluation of what is possible with geographic initialization, for particularly for the p-norms which we noted in experiment 1 are more sensitive to initialization.

Since Qi et al. and Brunner both ran experiments on the **SU** graph, we can also get some sense of how the difference in experiment set-up affects performance of our methods. We found the Brunner setup to be much more stable generally and so required no hyperparameters tweaks for specific models. In comparison to the Qi experiment, it used a lower learning rate (0.01 -> 0.001), smaller batch size (2500 $\rightarrow$ 512) and more train epochs (20 $\rightarrow$ 100. Most significantly, Qi et al. trains on all node pairs, whereas the Brunner experiments use 10% train ratio and tested on the remainder. We can see that between these two experiments L6 and Inv-Dot set ups performed similarly. This is good news since it suggests that the models achieve good representations without simply memorizing distances for the provided node pairs. In the future, experiments with lower train ratios and larger graphs may be more useful to evaluate performance, since this line of research in distance approximations with embeddings or neural networks is only useful when $|V|$ is too large to be able to generate shortest path distances for all node pairs in a feasible amount of time with modest space available.

4.4 Experiment D: Hartford Routing Results

Finally, we evaluate our methods on an A^* routing task on the Hartford network, and show results in table 5, in comparison to a baseline distance heuristic using the geographic coordinates. From our results we can see that in comparison to the baseline geographic distance heuristic, all methods produce better average Q, but not to a very significant degree in most cases.

They also differ greatly in the performance distribution by which their average error is achieved. For instance, the Poincare Hyperbolic and Elliptic measures produce the second and third best 99th percentile Q, but second and third worst median Q. It is possible they produce generally well distributed embeddings but are unable to fit in error very closely. For this reason we think it is worth investigating how they could be redesigned so that they produce embeddings in hyperbolic and elliptic geometry, respectively, but with improved performance in model training. On the other hand, the L6 norm performed the second best in 50th and the worst in 99th percentile Q. Considering the intuition for performance of higher p-norms given in section 5.1, this may be because the model is attempting to represent the graph adjacency matrix in the embedding space by ignoring some parts of the network, at least moreso than the other methods. This would allow for better median performance at the expense of edge case performance.

Our most significant finding here was that routing performance did not exactly follow the distance approximation error that we investigated in the previous experiments. For instance, L2 distance performed better than the L6 norm. This suggests that the ideal p may be lower than indicated by our other experiments, even though L6 produces the lower average error.

Comparing directly on median Q, we observe that Inv-Dot with coordinate initialization performed poorly compared to the other methods. This is not intuitively surprising since the dot product of coordinates does not accurate reflect the geographic distance, so the initialization would be poor. However it is opposite to our findings in Experiment C. This could be due to the different experimental setup, but it is also possible the mean errors reported earlier do not accurately indicate the routing performance possible with each method. Routing is dependent heavily on not only the average approximation error but also the distribution of errors. For instance if, looking back to Experiment C, our Inv-Dot embeddings produced an absolute error of exactly 0.0161 for each distance, considering the average true distance is 1.2, this would probably result in quite good routing performance. However if this error was all concentrated on specific node pairs, particularly if they are

Table 7: Experiment D - Hartford Routing

Technique	Initialization	Q-Average	Q-50th	Q-90th	Q-99th	Routing Time
L2-Norm	Coord	14.45	8.7	37.5	86.06	**18:25**
L6-Norm	Coord	20.09	8.33	67.86	97.14	22:21
	node2vec	17.12	7.55	54.05	96.16	21:58
Inv-Dot	Coord	22.55	13.89	60.98	89.77	20:27
	node2vec	**12.53**	**7.14**	**32.56**	**82.59**	18:44
Poincare	Coord	15.86	9.76	40.32	86.23	18:32
Spherical	Coord	15.33	9.43	38.36	85.71	19:01
Geographic Dist		23.86	8.33	54.29	96.81	22:54

frequent heuristic queries, the routing performance would be very poor. The routing performance also takes into account more greatly the representation of commonly visited nodes in the network, and helps to indicate whether the magnitude of approximation error is significant or not using a consistently scaled metric. For these reasons, we recommend heuristic-based routing as an evaluation task for distance approximation models, since it helps to verify that the embedding representations fit both the network topology as well as the representation constraints from the chosen measure.

In addition, we found during our experiments that worst case performance for routing generally occurs because approximations for specific pairs are far too high, usually in small loci that have very few connections. This results in the algorithm prioritizing alternative paths throughout the network. In practice, more consistent routing performance can be obtained with an ensemble method, for instance by using a simpler heuristic or Dijkstra's itself if the initial distance approximation from u to v is sufficiently low.

5 DISCUSSION

In general, we found on the tested graphs that higher norms had better output performance. To see potentially why this could be, suppose we have a 2 dimensional embedding matrix, MAE loss and that $|\mathbf{u}_1 - \mathbf{v}_1| = d_{u,v}$ for a particular training pair (u, v). Then we have the loss for this example as

$$L = \left| \left((\mathbf{u}_1 - \mathbf{v}_1)^p + d_{u,v}^p \right)^{1/p} \right|$$

which is minimized for a larger range of $\mathbf{u}_1 - \mathbf{v}_1$ around 0 when p is large. This means if the distance is already well approximated by some coordinate, the gradient update for the other coordinates will be small as long as the difference is not excessively large, so they can instead be tuned for other examples. Given this, the theoretical optimal embedding for the norm measure would be using the ∞-norm with d as the maximum degree of G, and would have that for each pair (u, v) there is a corresponding coordinate i^* in the embedding such that $i^* = \text{argmax}_i |\mathbf{u}_i - \mathbf{v}_i| = d_{u,v}$. Such an embedding would be similar to an adjacency matrix for G. It is not obvious whether such an embedding is possible generally or for any G, so this question is something to pursue in future work. However, we can say that higher p-norms are most likely better able to approximate the adjacency matrix in the lower dimensional embedding space $|V| \times D$.

However, the model can only train for a certain number of iterations, and higher p values result in lower initial prediction values. Given the experimental setups we use, it appears empirically that the L6-Norm is the optimal point in this tradeoff between poor initialization and better optima.

5.1 Inv-Dot

As mentioned above, the choice of Inv-Dot as a measure was motivated by empirical issues with using the Elliptic measure on geographic networks. Ignoring scaling into the proper range $x \in [-1, 1]$ and $y \in [0, d_{max}]$ performed for both measures, the change is simply to replace $\cos^{-1}(x)$ with $-x$, which is a much simpler gradient and is constant as long as the input $x \in (-1, 1)$. Since this change is small, we hypothesize the performance in geographic networks is mainly due to the change in gradient which allows faster training. Further investigation is necessary to understand the interesting results on Ego Facebook and whether these results generalize to other social networks.

6 CONCLUSION

We've proposed a simple embedding approach, ANEDA, to learn distance preserving graph embeddings in explicit representation spaces, and demonstrated comparable or better performance against state-of-the-art deep learning architectures.

For future investigation, we consider the following areas of extension or improvement of our work:

(1) Verify whether using specific measures is successful in most cases at embedding into the respective mathematical space (within some margin of error).
(2) Modify measures for certain geometries (e.g. hyperbolic, elliptic) to produce better gradients.
(3) Introduce a loss or training constraint to better enforce the triangle inequality so that our model defines a metric space.
(4) Design a custom loss to target worst case distance approximation, particularly for use in downstream routing tasks.
(5) Apply the technique in an online learning setting, where embeddings must be updated as changes to the graph occur.

REFERENCES

[1] Amr Ahmed, Nino Shervashidze, Shravan Narayanamurthy, Vanja Josifovski, and Alexander J. Smola. 2013. Distributed Large-Scale Natural Graph Factorization. In *Proceedings of the 22nd International Conference on World Wide Web* (Rio de Janeiro, Brazil) *(WWW '13)*. Association for Computing Machinery, New York, NY, USA, 37–48. https://doi.org/10.1145/2488388.2488393

[2] Geoff Boeing. 2017. Osmnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, Vol. 65. 126–139.

[3] Dustin Brunner. 2021. Distance Preserving Graph Embedding.

[4] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2015. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM international on conference on information and knowledge management*. 891–900.

[5] Haochen Chen, Bryan Perozzi, Yifan Hu, and Steven Skiena. 2017. HARP: Hierarchical Representation Learning for Networks. arXiv:1706.07845 [cs.SI]

[6] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. arXiv:1607.00653 [cs.SI]

[7] Aditya Grover and Jure Leskovec. 2016. Node2vec: Scalable Feature Learning for Networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Francisco, California, USA) *(KDD '16)*. Association for Computing Machinery, New York, NY, USA, 855–864. https://doi.org/10.1145/2939672.2939754

[8] Peter E Hart, Nils J Nilsson, and Bertram Raphael. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics* 4, 2 (1968), 100–107.

[9] Alireza Karduni, Amirhassan Kermanshah, and Sybil Derrible. 2016. A protocol to convert spatial polyline data to network formats and applications to world urban road networks, Vol. 3. Scientific Data. https://doi.org/10.6084/m9.figshare.2061897.v1

[10] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford large network dataset collection.

[11] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. 2016. Asymmetric Transitivity Preserving Graph Embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Francisco, California, USA) *(KDD '16)*. Association for Computing Machinery, New York, NY, USA, 1105–1114. https://doi.org/10.1145/2939672.2939751

[12] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 701–710.

[13] Bryan Perozzi, Vivek Kulkarni, Haochen Chen, and Steven Skiena. 2017. Don't Walk, Skip! Online Learning of Multi-scale Network Embeddings. arXiv:1605.02115 [cs.SI]

[14] Jianzhong Qi, Wei Wang, Rui Zhang, and Zhuowei Zhao. 2020. A Learning Based Approach to Predict Shortest-Path Distances. In *EDBT*.

[15] Leonardo F.R. Ribeiro, Pedro H.P. Saverese, and Daniel R. Figueiredo. 2017. struc2vec. *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Aug 2017). https://doi.org/10.1145/3097983.3098061

[16] Fatemeh Salehi Rizi, Joerg Schloetterer, and Michael Granitzer. 2018. Shortest path distance approximation using deep learning techniques. In *2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE, 1007–1014.

[17] Jörg Schlötterer, Martin Wehking, Fatemeh Salehi Rizi, and Michael Granitzer. 2019. Investigating Extensions to Random Walk Based Graph Embedding. *2019 IEEE International Conference on Cognitive Computing (ICCC)* (2019), 81–89.

[18] Puoya Tabaghi and Ivan Dokmanić. 2020. Hyperbolic distance matrices. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1728–1738.

[19] Kevin Verbeek and Subhash Suri. 2014. Metric Embedding, Hyperbolic Space, and Social Networks. In *Proceedings of the Thirtieth Annual Symposium on Computational Geometry* (Kyoto, Japan) *(SOCG'14)*. Association for Computing Machinery, New York, NY, USA, 501–510. https://doi.org/10.1145/2582112.2582139

[20] Jie Zhang, Yuxiao Dong, Yan Wang, Jie Tang, and Ming Ding. 2019. ProNE: Fast and Scalable Network Representation Learning. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, 4278–4284. https://doi.org/10.24963/ijcai.2019/594