REVIEW PAPER

A Review of Location Encoding for GeoAI: Methods and Applications

Gengchen Mai[a,b,c], Krzysztof Janowicz[a,b], Yingjie Hu[d], Song Gao[e], Bo Yan[a,b], Rui Zhu[a,b], Ling Cai[a,b], and Ni Lao[f]

[a]STKO Lab, Department of Geography, UC Santa Barbara, CA, USA, 93106;
[b]Center for Spatial Studies, UC Santa Barbara, CA, USA, 93106;
[c]Department of Computer Science, Stanford University, CA, USA, 94305;
[d]GeoAI Lab, Department of Geography, University at Buffalo, NY, USA, 14260;
[e]GeoDS Lab, Department of Geography, University of Wisconsin-Madison, WI, USA, 53706;
[f]Mosaix.ai, Palo Alto, CA, USA, 94303

ARTICLE HISTORY
Compiled November 8, 2021

ABSTRACT
A common need for artificial intelligence models in the broader geoscience is to represent and encode various types of spatial data, such as points (e.g., points of interest), polylines (e.g., trajectories), polygons (e.g., administrative regions), graphs (e.g., transportation networks), or rasters (e.g., remote sensing images), in a hidden embedding space so that they can be readily incorporated into deep learning models. One fundamental step is to encode a single point location into an embedding space, such that this embedding is learning-friendly for downstream machine learning models such as support vector machines and neural networks. We call this process *location encoding*. However, there lacks a systematic review on the concept of location encoding, its potential applications, and key challenges that need to be addressed. This paper aims to fill this gap. We first provide a formal definition of location encoding, and discuss the necessity of location encoding for GeoAI research from a machine learning perspective. Next, we provide a comprehensive survey and discussion about the current landscape of location encoding research. We classify location encoding models into different categories based on their inputs and encoding methods, and compare them based on whether they are parametric, multi-scale, distance preserving, and direction aware. We demonstrate that existing location encoding models can be *unified* under a shared formulation framework. We also discuss the application of location encoding for different types of spatial data. Finally, we point out several challenges in location encoding research that need to be solved in the future.

KEYWORDS
Location encoding; GeoAI; representation learning; spatially-explicit machine learning

1. Introduction and Motivation

The rapid development of novel deep learning and representation learning techniques and the increasing availability of diverse, large-scale geospatial data have fueled substantial progress in geospatial artificial intelligence (GeoAI) research (Smith 1984, Couclelis 1986, Openshaw and Openshaw 1997, Janowicz *et al.* 2020). This includes progress on a wide spectrum of challenging tasks such as terrain feature detection and extraction

CONTACT Gengchen Mai. Email: gengchen_mai@geog.ucsb.edu

(Li and Hsu 2020), land use classification (Zhong *et al.* 2019), navigation in the urban environment (Mirowski *et al.* 2018), image geolocalization (Weyand *et al.* 2016, Izbicki *et al.* 2019), toponym recognition and disambiguation (DeLozier *et al.* 2015, Wang *et al.* 2020), geographic knowledge graph completion and summarization (Qiu *et al.* 2019, Yan *et al.* 2019), traffic forecasting (Li *et al.* 2018a), to name a few.

Despite the fact that these models are very different in design, they share a common characteristic - they need to *represent* (or *encode*) different types of spatial data, such as points (e.g., points of interest (POIs)), polylines (e.g., trajectories), polygons (e.g., administrative regions), graphs/networks (e.g., transportation networks), or raster (e.g., satellite images), in a hidden embedding space so that they can be utilized by machine learning models such as deep neural nets (NN). For raster data, this encoding process is straightforward since the regular grid structures can be directly handled by existing deep learning models such as convolutional neural networks (CNN) (Krizhevsky *et al.* 2012). The representation problem gets more complicated for vector data such as point sets, polylines, polygons, and networks, which have more irregular spatial organization formats, because the concepts of location, distance, and direction among others do not have straightforward counterparts in existing NN and it is not trivial to design NN operations (e.g., convolution) for irregularly structured data (Valsesia *et al.* 2019).

Early efforts perform data transformation operations to *convert* the underlying spatial data into a format which can be handled by existing NN modules (Wang *et al.* 2019). However, this conversion process often leads to information loss. For example, many early research about point cloud classification and segmentation first *converted* 3D point clouds into volumetric representations (e.g., voxelized shapes) (Maturana and Scherer 2015, Qi *et al.* 2016) or rendered them into 2D images (Su *et al.* 2015, Qi *et al.* 2016). Then they applied 3D or 2D CNN on these converted data representations for the classification or segmentation tasks. These practices have a major limitation – choosing an appropriate spatial resolution for a volumetric representation is challenging (Qi *et al.* 2017a). A finer spatial resolution leads to data sparsity and higher computation cost while a coarser spatial resolution provides poor prediction results.

The reason for performing such data conversions is a lack of means to directly handle vector data in deep neural nets. An alternative approach is to encode these spatial data models directly. The first step towards such goal is to encode a point location into an embedding space such that these location embeddings can be easily used in the downstream NN modules. This is the idea of *location encoding*.

Location encoding (Mac Aodha *et al.* 2019, Mai *et al.* 2020b, Zhong *et al.* 2020, Mai *et al.* 2020a, Gao *et al.* 2019, Xu *et al.* 2018, Chu *et al.* 2019) refers to a NN-based encoding process which represents a point/location into a high dimensional vector/embedding such that this embedding can preserve different spatial information (e.g., distance, direction) and, at the same time, be *learning-friendly*

for downstream machine learning (ML) models such as neural nets and support vector machines (SVM). By *learning friendly* we mean that the downstream model does not need to be very complex and does not require lots of training data to prevent model overfitting. The encoding results are called location embeddings. And the corresponding NN architecture is called *location encoder*, which is a general-purpose model that can be incorporated into different GeoAI models for different downstream tasks.

Figure 1 is an illustration of location encoding. Here, we use location-based species classification as an example of the downstream tasks which aims at predicting species y based on a given location $\mathbf{x}$. The training objective is to learn the conditional distribution $P(y|\mathbf{x})$, i.e., the probability of observing y given $\mathbf{x}$, which is highly non-linear. The idea of location encoding can be understood as a feature decomposition process which

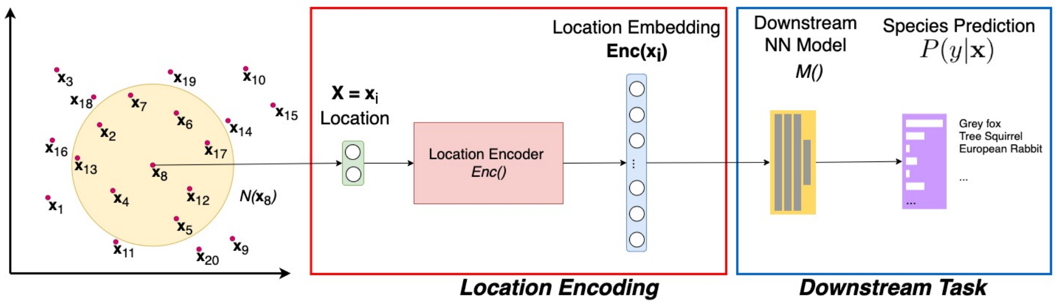


Figure 1.: An illustration of location encoding. Here, we use *location-based species classification* as an example of the downstream tasks. Those 20 points in 2D space represent species occurrence records. Each occurrence can be written as $p_i = (\mathbf{x}_i, y_i)$ where $\mathbf{x}_i$ indicates the 2D locations and y_i indicates the corresponding species type, i.e., the ground truth label. $\mathcal{N}(\mathbf{x}_i)$ indicates the spatial neighborhood of $\mathbf{x}_i$. A location encoder $Enc(\cdot)$ takes 2D location $\mathbf{x}_i$ as its input and outputs a location embedding as a high dimensional vector. This embedding is further fed into a downstream NN model $M()$ for species prediction. The whole model architecture can be trained end-to-end in a supervised learning manner.

decomposes location $\mathbf{x}$ (e.g., a two-dimensional vector of latitude and longitude) into a learning-friendly high dimensional vector (e.g., a vector with 100 dimensions), such that the highly non-linear distribution $P(y|\mathbf{x})$ can be learned with a relatively simple learner such as a linear SVM or a shallow NN model $M()$. The key benefits of such an architecture are to require less training data with simpler learners, and the possibility to leverage unsupervised training to better learn the location representations.

Recently, the effectiveness of location encoding has been demonstrated in multiple GeoAI tasks including geo-aware image classification (Yin *et al.* 2019, Chu *et al.* 2019, Mac Aodha *et al.* 2019, Mai *et al.* 2020b), POI classification (Mai *et al.* 2020b), place annotation (Yin *et al.* 2019), trajectory prediction (Xu *et al.* 2018, Yin *et al.* 2019), location privacy protection (Rao *et al.* 2020), geographic question answering (Mai *et al.* 2020a), 3D protein distribution reconstruction (Zhong *et al.* 2020), point cloud classification and segmentation (Qi *et al.* 2017a,b, Li *et al.* 2018b), and so on. Despite these successful stories, there is still a lack of a systematic review on such a topic. This paper fills this gap by providing a comparative survey on different location encoding models. We give a general conceptual formulation framework which unifies almost all existing location encoding methods.

It is worth mentioning that the location encoding discussed in this work is different from the traditional location encoding systems (i.e., geocoding systems)[1] which convert geographic coordinates into codes using an encoding scheme such as Geohash or codes for partition tiles such as Open Location Code and what3words. These traditional encoding systems are designed to support navigation and spatial indexing, while the neural location encoders we present here are used to support downstream ML models.

The contributions of our work are as follows:

(1) Although there are multiple existing works on location encoding, the necessity to design such a model is not clear. In this work, we formally define the location encoding problem and discuss the necessity from a machine learning perspective.
(2) We conduct a systematic review on existing location encoding research. A detailed classification system for location encoders is provided and all models

[1] `https://gogeomatics.ca/location-encoding-systems-could-geographic-coordinates-be-replaced-and-at-what-cost/`

are reformulated under a unified framework. This allows us to identify the commonalities and differences among different location encoding models. As far as we know, this is the first review on such a topic.
(3) We extend the idea of location encoding to the broader topic of encoding different types of spatial data (e.g., polylines, polygons, graphs, and rasters). The possible solutions and challenges are discussed.
(4) To emphasize the general applicability of location encoding, we discuss its potential applications in different geoscience domains. We hope these discussions can open up new areas of research.

The rest of this paper is structured as follows. We first introduce a formal definition of location encoding in Section 2. Then, in Section 3, we discuss the necessity of location encoding. Next, we provide a general framework for understanding the current landscape of location encoding research and survey a collection of representative work in Section 4. In Section 5, we discuss how to apply location encoding for different types of spatial data. Finally, we conclude our work and discuss future research directions in Section 6.

2. Definitions

Definition 2.1 (Location Encoding). Given a set of points $\mathcal{P} = \{p_i\}$, e.g., the locations of sensors, species occurrences, and so on, where each point (e.g., an air quality sensor) $p_i = (\mathbf{x}_i, \mathbf{v}_i)$ is associated with a location $\mathbf{x}_i \in \mathbb{R}^L$ (e.g., a sensor's location) in L-D space ($L = 2, 3$) and attributes $\mathbf{v}_i \in \mathbb{R}^E$ (e.g., air quality measurements). We define the location encoder as a function $Enc^{(\mathcal{P},\theta)}(\mathbf{x}) : \mathbb{R}^L \rightarrow \mathbb{R}^d$ ($L \ll d$), which is parameterized by θ and maps any coordinate $\mathbf{x}$ in space to a vector representation of d dimension. This process is called *location encoding* and the results are called *location embeddings*.

Here, $Enc^{(\mathcal{P},\theta)}(\mathbf{x})$ indicates that the encoding result of $\mathbf{x}$ may also depend on other locations in $\mathcal{P}$. When $Enc^{(\mathcal{P},\theta)}(\mathbf{x})$ is independent of other points in $\mathcal{P}$, we can simplify it to $Enc^{(\theta)}(\mathbf{x})$. Note that sometimes, the input of the location encoder can be both locations and attributes, i.e., $Enc^{(\mathcal{P},\theta)}(\mathbf{x}, \mathbf{v}) : \mathbb{R}^{L+E} \rightarrow \mathbb{R}^d$.

Figure 1 illustrates the idea of location encoding in Definition 2.1. The 20 2D points serve as an example of $\mathcal{P} = \{p_i\}$ with $L = 2$ and $n = |\mathcal{P}| = 20$. Note that $Enc^{(\mathcal{P},\theta)}(\mathbf{x})$ can not only be used to encode global location $\mathbf{x}$, but also be utilized to encode the spatial relation between two locations, i.e., the spatial affinity vector $\Delta_{AB} = \mathbf{x}_A - \mathbf{x}_B$.

One question we may ask is *whether a location encoder can preserve spatial information such as distance and direction information after the encoding process.* From a spatial information preservation perspective, there are two properties we expect a location encoder to have: distance preservation and direction awareness.

Property 2.1 (Distance Preservation). The distance preservation property requires *two nearby locations to have similar location embeddings.* More concretely, given any pair of location $(\mathbf{x}_A, \mathbf{x}_B)$, the inner product/similarity between their resulting location embeddings, i.e, $\langle Enc^{(\mathcal{P},\theta)}(\mathbf{x}_A), Enc^{(\mathcal{P},\theta)}(\mathbf{x}_B)\rangle$ monotonically decreases when the distance[2] between $\mathbf{x}_A$ and $\mathbf{x}_B$, i.e., $\parallel \mathbf{x}_A - \mathbf{x}_B \parallel$, increases.

Property 2.1 can be seen as a reflection of Tobler's First Law of Geography (TFL) (Tobler 1970) in location encoding. The requirement of distance preservation has been adopted by multiple existing location encoding works. For example, Gao *et al.*

[2]This can be Euclidean distance, manhattan distance, geodesic distance, great circle distance, and so on.

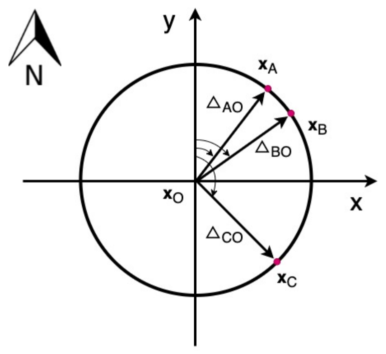


Figure 2.: An illustration of the direction preservation property of location encoding.

(2019) proposed a learnable location representation model $v(\mathbf{x})$ which consists of three sub-models: vector matrix multiplication, magnified local isometry, and global adjacency kernel. The global adjacency kernel sub-model assumes $\langle v(\mathbf{x}_a), v(\mathbf{x}_b)\rangle = (Kd)f(\| \mathbf{x}_a - \mathbf{x}_b \|)$, where K, d are predefined constants and $f(r)$ is the adjacency kernel that decreases monotonically as r increases. It can be seen that this sub-model directly satisfies Property 2.1. Mai *et al.* (2020b) also showed that their proposed multi-scale location encoder has a similar distance preservation property.

Property 2.2 (Direction Awareness)**.** *Locations that point into similar directions have more similar (relative) location embeddings than those who point into very different directions.* More concretely, as shown in Figure 2, given $\mathbf{x}_O$ as the reference point and y axis as the global north direction, $\mathbf{x}_A$, $\mathbf{x}_B$, and $\mathbf{x}_C$ are on the same circle centered at $\mathbf{x}_O$ and therefore share the same distance to $\mathbf{x}_O$. The relative spatial relation between $\mathbf{x}_A$ and $\mathbf{x}_O$ is defined as $\Delta_{AO} = \mathbf{x}_A - \mathbf{x}_O$. The direction of Δ_{AO}, i.e., $\angle_{AO}$ is defined as the clockwise angle between y axis and Δ_{AO}. The same logic applies to $\mathbf{x}_B$ and $\mathbf{x}_C$. We say a location encoder is direction aware if it satisfies the following property: the inner product $\langle Enc^{(\mathcal{P},\theta)}(\Delta_{AO}), Enc^{(\mathcal{P},\theta)}(\Delta_{BO})\rangle > \langle Enc^{(\mathcal{P},\theta)}(\Delta_{AO}), Enc^{(\mathcal{P},\theta)}(\Delta_{CO})\rangle$ if $|\angle_{AO} - \angle_{BO}| < |\angle_{AO} - \angle_{CO}|$.

Property 2.2 is a reflection of the Generalized First Law of Geography (Zhu *et al.* 2019b) which includes direction into the consideration of similarities. In this paper, we call a location encoder an *isotropic* location encoder if it only preserves the spatial proximity but ignores the variance of location embeddings when direction changes. We need to develop direction-aware, so-called anisotropic location encoders when the isotropicity assumption can not be held anymore. In fact, in normal spatial analysis, isotropicity is the "default" assumption in most of the time. Although anisotropic versions of many geospatial analysis techniques have been developed such as directional kriging (Te Stroet and Snepvangers 2005), anisotropic clustering (Mai *et al.* 2018), direction remains on the level of an afterthought (Zhu *et al.* 2019b). A similar situation can be seen in the current location encoding research, or GeoAI research in general. Compared with studies on distance preserved location encoders, there has been much less work on how to make a location encoder direction aware. Mai *et al.* (2020b) empirically showed that their multi-scale location encoder as well as many baseline models are direction-aware. However, this is just a by-product from their visualization analysis of the response maps of these location encoders. No theoretical proof is provided. As far as we know, there is no existing research that aims at developing a direction-aware location encoder deliberately.

Besides the above two spatial information preservation properties, Location Encoder

$Enc^{(\mathcal{P},\theta)}()$ should also satisfy the following two properties to ensure its generalizability.

Property 2.3 (Inductive Learning Method)**.** Location Encoder $Enc^{(\mathcal{P},\theta)}()$ is an inductive learning method, i.e., the pretrained location encoder can be utilized to encode any location without retraining even if it does not appear in the training set.

Property 2.3 makes $Enc^{(\mathcal{P},\theta)}()$ differ from many existing transductive-learning-based location representation learning methods such as Location2Vec (Zhu *et al.* 2019a), POI2Vec (Feng *et al.* 2017), and Kejriwal and Szekely (2017). For example, Kejriwal and Szekely (2017) converted a set of GeoNames locations into a k-th nearest neighbor graph in which locations (i.e., nodes) are linked to nearby locations by distance-weighted edges. A random-walk-based graph embedding method is used to learn an embedding for each location. This method is essentially a transductive learning model: when new locations are added to the training set, the graph is modified and the whole model has to be retrained to obtain the embeddings of new locations.

Property 2.4 (Task Independence)**.** Location Encoder $Enc^{(\mathcal{P},\theta)}()$ should be task-independent or so called task-agnostic, i.e., the same model architecture can be used in different downstream tasks without any modification.

Property 2.4 also differentiate $Enc^{(\mathcal{P},\theta)}()$ from some existing task-dependent location representation approaches. For instance, both Location2Vec (Zhu *et al.* 2019a) and POI2Vec (Feng *et al.* 2017) learned the embeddings of locations (e.g., cell stations, POIs) based on trajectories by adopting a Word2Vec-style training objective. This kind of location representation cannot be easily transferred to other tasks beyond human mobility. Similarly, Gao *et al.* (2020) discretized the study area into $N \times N$ lattice and learned the embedding of each location by letting a Long Short Term Memory (LSTM) based artificial agent navigate through the study area. The learned location embeddings are used for simulation purpose but not for other geospatial tasks.

Property 2.5 (Parametric Model)**.** A *parametric model* is a learning model with a finite set of parameters θ.

A parametric model is not very flexible, but the model complexity is bounded. In contrast, a *non-parametric model* assumes that the data distribution cannot be defined with a finite set of parameters θ and the size of parameters θ can grow as the amount of data grows (Russell and Norvig 2015). So a non-parametric model is more flexible while the model complexity is unbounded.

Although Property 2.1, 2.2, and 2.5 are expected for location encoders, not all models we will discuss in Section 4 have these properties. See Table 1 for the detailed comparison. However, all location encoders discussed in Section 4 satisfy Property 2.3 and 2.4. So we will not discuss these two properties separately for each model.

For simplicity, we will use $Enc()$ and $Enc^{(\mathcal{P})}()$ to indicate $Enc^{(\theta)}()$ and $Enc^{(\mathcal{P},\theta)}()$.

3. The necessity of location encoding for GeoAI

In this section we motivate the need to embed a location $\mathbf{x} \in \mathbb{R}^L$ $(L = 2, 3)$ into a high dimensional vector $Enc^{(\mathcal{P},\theta)}(\mathbf{x}) \in \mathbb{R}^d$, which may seem counter-intuitive at first. We mainly address this issue from a machine learning perspective.

A key concept in statistical machine learning is bias-variance trade-off (Hastie *et al.* 2009). On the one hand, when a learning system is required to pick one hypothesis out of a large hypothesis space (e.g., deciding the parameters of a large 24 layer neural networks), it is flexible enough to approximate almost any non-linear distribution (low bias). However, it needs a lot of training data to prevent overfitting. This is called the low bias high variance situation. On the other hand, when the hypothesis space is restricted (e.g., linear regression or single layer neural nets) the system has little chance to over fit, but might be ill-suited to model the underlying distribution and result in low performance in both training and testing sets (high bias). This situation is called low variance high bias. For many applications the data distribution is complex and highly non-linear. We may not have enough domain knowledge to design good models with low variance (the effective model complexity) and low bias (the model data mismatch) at the same time. Moreover, we might want to avoid adopting too much domain knowledge into the model design which will make the resulting model task specific. For example, the distribution of plant species (such as $P(y|\mathbf{x})$ in Figure 1) may be highly irregular influenced by several geospatial factors and interactions among species (Mac Aodha *et al.* 2019). Kernel (smoothing) methods (e.g., Radial Basis Function (RBF)) and neural networks (e.g., feed-forward nets) are two types of most successful models which require very little domain knowledge for model design. They both have well established ways of controlling the effective model complexity. The kernel methods are more suited to low dimensional input data – modeling highly non-linear distributions with little model complexity. However, they need to store the kernels during inference time which is not memory efficient. Neural networks have more representation power which means a deep network can approximate very complex functions with no bias, while requiring more domain knowledge for model design to achieve lower variance and bias.

From a statistical machine learning perspective, the main purposes of location encoding is to produce *learning friendly representations of geographic locations* for downstream models such as SVM and neural networks. By learning friendly we mean that the downstream model does not need to be very complex and require large training samples. For example, the location encoding process may perform a feature decomposition ($\mathbf{x} \in \mathbb{R}^L \rightarrow Enc^{(\mathcal{P},\theta)}(\mathbf{x}) \in \mathbb{R}^d$, where $L < d$) so that the distribution we want to model such as $P(y|\mathbf{x})$ in Figure 1 becomes linear in the decomposed feature space, and a simple linear model can be applied. Figure 3 illustrates this idea by using a simple binary classification task. If we use the original geographic coordinates $\mathbf{x}$ as the input features to train the binary classifier, the resulting classifier M_1 will be a complex and nonlinear function which is prone to overfitting as shown in the left of Figure 3. After the location encoding process, the geographic coordinates feature is decomposed so that a simple linear model M_2 can be used as the binary classifier[3].

4. A review of the current landscape of location encoding

In this section, we provide a comprehensive review of the existing location encoding techniques. Instead of enumerating every existing location encoding approach we organize our discussion in a top-down manner. We first classify location encoding models into different groups according to the input of location encoders and how they manipulate the spatial features. Firstly, according to the input, we can classify the existing location encoders into two categories: *single point location encoder* $Enc(\mathbf{x})$ and *aggregation location encoder* $Enc^{(\mathcal{P})}(\mathbf{x})$. $Enc(\mathbf{x})$ only considers the current point's location while $Enc^{(\mathcal{P})}(\mathbf{x})$ additionally considers points in its neighborhood $\mathcal{N}(\mathbf{x}) \subseteq \mathcal{P}$.

[3]The dimensionality of the location embedding space will be larger (e.g., 32 or 128); we use 3D here for ease of illustration.

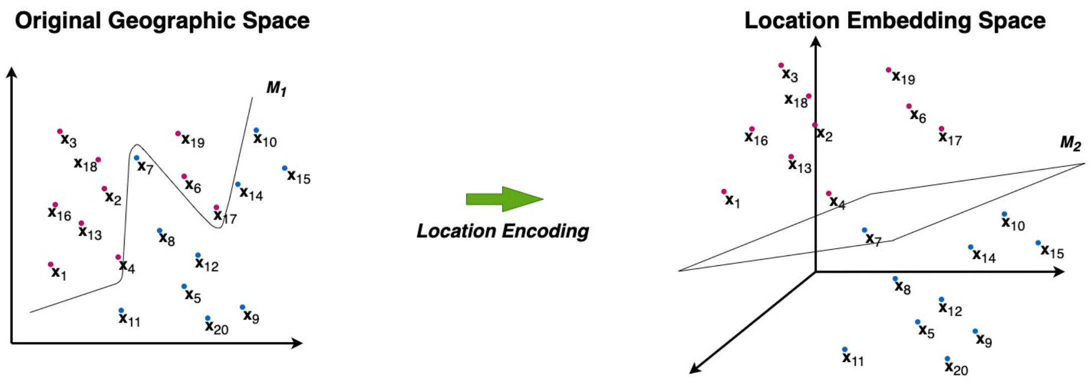

Original Geographic Space
M_1
x_3 x_{19} x_{10} x_{18} x_7 x_{15} x_6 x_2 x_{14} x_{16} x_{17} x_{13} x_8 x_{12} x_1 x_4 x_5 x_{11} x_{20} x_9
Location Encoding
Location Embedding Space
M_2
x_3 x_{19} x_{18} x_6 x_{16} x_2 x_{17} x_{13} x_1 x_4 x_{10} x_7 x_{14} x_{15} x_8 x_{12} x_5 x_9 x_{11} x_{20}

Figure 3.: An illustration of how location encoding can help to produce learning-friendly representations of geographic locations for downstream models. We use the same 20 points in Figure 1 as an example of $\mathcal{P} = \{p_i\}$. The red and blue points indicate they belong to two different classes. M_1 and M_2 are the illustrations of the trained binary classifiers in the original geographic space and the location embedding space.

Table 1.: Overview of location encoding approaches. Single point location encoders $Enc(\mathbf{x})$ and aggregation location encoders $Enc^{(\mathcal{P})}(\mathbf{x})$ are further classified based on either $PE(\mathbf{x})$ or $\mathcal{N}(\mathbf{x})$ (see Figure 1). (M) indicates multi-scale representation. * indicates a generalized version of the original model cited. We consider multiple criteria of location encoders: 1) L: The spatial dimension of $\mathcal{P}$; 2) Parametric: Is the location encoder a parametric model (Yes) or non-parametric model (No)? 3) Mul.S.: Does the location encoding adopt a multi-scale approach? 4) Dist.P.: Does this location encoder preserve distance (Property 2.1)? 5) Dir.A.: Is this location encoder aware of direction (Property 2.2)? For Dist.P. and Dir.A. "Yes" or "No" indicates whether the property can be proved empirically (for example by using the response maps of trained location encoders Mai *et al.* (2020b)). "-" indicates that the property is unknown. "Yes+" indicates that the property was shown both theoretically and empirically.

Encoder	$PE(\mathbf{x})$	Model	L	Parametric	Mul.S.	Dist.P.	Dir.A.
$Enc(\mathbf{x})$	Discretization	onehot(Tang *et al.* 2015)	2,3	Yes	No	No	-
		tile(Mai *et al.* 2020b)	2,3	Yes	No	No	-
	Direct	direct(Xu *et al.* 2018, Chu *et al.* 2019, Rao *et al.* 2020)	2,3	Yes	No	Yes	Yes
	Sinusoidal	wrap(Mac Aodha *et al.* 2019)	2	Yes	No	Yes	-
	Sinusoidal (M)	TF(Zhong *et al.* 2020)	2,3	Yes	Yes	-	-
		theory/Space2Vec (Mai *et al.* 2020b)	2	Yes	Yes	Yes+	Yes
		grid/Space2Vec (Mai *et al.* 2020b)	2,3	Yes	Yes	Yes	-
Encoder	$\mathcal{N}(\mathbf{x})$	Model	L	Parametric	Mul.S.	Dist.Pr.	Dir.A.
$Enc^{(\mathcal{P})}(\mathbf{x})$	Kernel	GPS2Vec(Yin *et al.* 2019)	2	Yes	No	-	-
		rbf(Mai *et al.* 2020b)	2,3	Yes/No	No	Yes	No
		Adapted kernel* (Berg *et al.* 2014)	2,3	Yes/No	No	-	-
	Global	PointNet (Qi *et al.* 2017a)	2,3	Yes	No	-	-
	Local	VoxelNet (Zhou and Tuzel 2018)	2,3	Yes	No	-	-
		SAGAT (Mai *et al.* 2020b)	2,3	Yes/No	Yes/No	Yes/No	Yes/No
		DGCNN (Wang *et al.* 2019)	2,3	Yes	No	-	-
	Hierarchical	PointNet++ (Qi *et al.* 2017b)	2,3	Yes	Yes	-	-
		PointCNN (Li *et al.* 2018b)	2,3	Yes	Yes	-	-
		GraphCNN (Valsesia *et al.* 2019)	2,3	Yes	Yes	-	-

Then, in Section 4.1, $Enc(\mathbf{x})$ is further classified into sub-categories based on the type of positional encoder $PE(\mathbf{x})$. Next, in Section 4.2, $Enc^{(\mathcal{P})}(\mathbf{x})$ is classified based on the used neighborhood $\mathcal{N}(\mathbf{x})$. The survey result is summarized in Table 1. Finally, the comparison among different models will be discussed in Section 4.3.

4.1. *Single point location encoder* $\boldsymbol{Enc}(\mathbf{x})$

Interestingly, most existing single point location encoders, i.e., $Enc(\mathbf{x})$, (Tang *et al.* 2015, Gao *et al.* 2019, Xu *et al.* 2018, Chu *et al.* 2019, Mac Aodha *et al.* 2019, Mai *et al.* 2020b, Zhong *et al.* 2020, Rao *et al.* 2020) share a similar structure:

$$Enc(\mathbf{x}) = \mathbf{NN}(PE(\mathbf{x})), \tag{1}$$

Here, $\mathbf{NN}(\cdot) : \mathbb{R}^W \to \mathbb{R}^d$ is a learnable neural network component which maps the input position embedding $PE(\mathbf{x}) \in \mathbb{R}^W$ into the location embedding $Enc(\mathbf{x}) \in \mathbb{R}^d$. A common practice is to define $\mathbf{NN}(\cdot)$ as a multi-layer perceptron, while Mac Aodha *et al.* (2019) adopted a more complex $\mathbf{NN}(\cdot)$ which includes an initial fully connected layer, followed by a series of residual blocks. The purpose of $\mathbf{NN}(\cdot)$ is to provide a learnable component for the location encoder, which captures the complex interaction between input locations and target labels.

$PE(\cdot)$ is the most important component which distinguishes different $Enc(\mathbf{x})$. Usually, $PE(\cdot)$ is a *deterministic* function which transforms location $\mathbf{x}$ into a W-dimension vector, so-called position embedding. The purpose of $PE(\cdot)$ is to do location feature normalization (Chu *et al.* 2019, Mac Aodha *et al.* 2019, Rao *et al.* 2020) and/or feature decomposition (Mai *et al.* 2020b, Zhong *et al.* 2020) so that the output $PE(\mathbf{x})$ is more learning-friendly for $\mathbf{NN}(\cdot)$. In Table 1 we further classify different $Enc(\mathbf{x})$ into four sub-categories based on their $PE(\cdot)$: discretization-based, direct, sinusoidal, and sinusoidal multi-scale location encoder. Each of them will be discussed in detail below.

4.1.1. Discretization-based location encoder

The early pioneers (Tang *et al.* 2015) argued that GPS coordinates are rather precise location indicators which are difficult to use by a classifier. So instead of using the coordinates, they discretized the whole study area into grid tiles and indicates each point by the corresponding grid that it falls into.

Definition 4.1 (Discretization-based Location Encoder)**.** A discretization-based location encoder divides the study area (e.g., the earth surface) into regular area units such as grids, hexagons, or triangles - $Enc_{discretize}(\mathbf{x}) = \mathbf{NN}(PE_{discretize}(\mathbf{x}))$ where $PE_{discretize}(\cdot)$ is usually a tile lookup function which maps $\mathbf{x}$ to a one hot vector that indicates the corresponding tile id it falls into.

onehot. Early work in location encoding does not really have learnable component specific to the location encoder. For example, Tang *et al.* (2015) divided the study area (the contiguous United States) into M rectangle grids. Given a location $\mathbf{x}$ (the geotag of an image), $PE_{onehot}(\mathbf{x}) \in \mathbb{R}^M$ is a one hot vector to indicate which grid $\mathbf{x}$ falls into and $\mathbf{NN}(\cdot)$ as an identity function, i.e., $Enc_{onehot}(\mathbf{x}) = \mathbf{NN}(PE_{onehot}(\mathbf{x})) = PE_{onehot}(\mathbf{x})$.

tile. Later Mai *et al.* (2020b) introduced *tile* as one of $Enc_{discretize}(\mathbf{x})$ which uses a trainable location embedding matrix as $\mathbf{NN}(\cdot)$. This makes it possible for the model to benefit from unsupervised training.

Although $Enc_{discretize}(\mathbf{x})$ shows promising results on tasks such as geo-aware image classification, it has several inherent limitations: 1) Each tile embedding are trained separately and spatial dependency is ignored, i.e., they do not have the distance preservation property; 2) They have only one fixed spatial scale which can not effectively handle points with varied density; 3) Choosing the correct discretization is very challenging (Openshaw 1981, Fotheringham and Wong 1991), and incorrect choices will significantly affect the model's performance and efficiency (Lechner *et al.* 2012).

There are some possible solutions for these problems. For problem 1 we can add a regularization term in the loss function to make nearby tile embeddings have higher cosine similarity. For problem 2 and 3 one can adopt an adaptive discretization (Weyand *et al.* 2016) or multi-level discretization strategy as Kulkarni *et al.* (2020) did which uses deeper levels (smaller tiles) for higher point density areas and shallower levels (larger tiles) for sparse areas. However, finer spatial resolution or multi-level discretization means more tiles and more learnable parameters which can easily lead to overfitting.

4.1.2. Direct location encoder

Recently, researchers adopted a rather simple approach by directly applying neural networks to (normalized) coordinates (Xu *et al.* 2018, Chu *et al.* 2019, Rao *et al.* 2020).

Definition 4.2 (Direct Location Encoder)**.** A direct location encoder is defined as $Enc_{direct}(\mathbf{x}) = \mathbf{NN}(PE_{direct}(\mathbf{x}))$ where $PE_{direct}(\mathbf{x})$ is usually a function to normalize or standardize the input location feature $\mathbf{x}$ and $\mathbf{NN}(\cdot)$ is a multi-layer perceptron.

direct. There are many slight variations of $Enc_{direct}(\mathbf{x})$ models. Chu *et al.* (2019) took the input longitude and latitude (i.e., $\mathbf{x} = [\lambda, \phi]^T$ of a image) and normalized them to range $[-1, 1)$ by dividing them with constant values. Similarly, in trajectory synthesis study, Rao *et al.* (2020) deployed $PE_{direct}(\mathbf{x})$ which standardized each trajectory point $\mathbf{x} = [\lambda, \phi]^T$ by using the centroid of all trajectory points. In order to perform pedestrian trajectory prediction, Xu *et al.* (2018) also designed a simple location encoder whose $PE_{direct}(\mathbf{x})$ normalizes $\mathbf{x}$ to $[0, 1]$. Without a feature decomposition step, these models often fail to capture the fine details of data distributions, and have worse prediction accuracy than *tile* on specific tasks.

4.1.3. Sinusoidal location encoder

Definition 4.3 (Sinusoidal Location Encoder)**.** A sinusoidal location encoder is defined as $Enc_{sinu}(\mathbf{x}) = \mathbf{NN}(PE_{sinu}(\mathbf{x}))$ where $PE_{sinu}(\mathbf{x})$ is a deterministic function which processes $\mathbf{x}$ with sinusoidal functions, e.g., sin(), after location feature normalization.

wrap. Mac Aodha *et al.* (2019) developed $Enc_{wrap}(\mathbf{x}) = \mathbf{NN}(PE_{sinu}(\mathbf{x}))$ which uses sinusoidal functions to wrap the geographic coordinates. In Equation 2, longitude λ and latitude ϕ are first normalized into range $[-1, 1]$ by dividing by 180° and 90° accordingly and then are fed into $\sin(\pi x)$ and $\cos(\pi x)$ functions.

$$PE_{wrap}(\mathbf{x}) = [\sin(\pi\frac{\lambda}{180^\circ}), \cos(\pi\frac{\lambda}{180^\circ}), \sin(\pi\frac{\phi}{90^\circ}), \cos(\pi\frac{\phi}{90^\circ})],\ where\ \mathbf{x} = (\lambda, \phi) \quad (2)$$

The purpose to use sinusoidal functions is to wrap geographic coordinates around the world Mac Aodha *et al.* (2019). This ensures that longitude $\lambda_1 = -180^\circ$ and $\lambda_2 = 180^\circ$ have the same encoding results. However, applying this encoding strategy to latitudes is problematic. As for $\phi_1 = -90^\circ$ and $\phi_2 = 90^\circ$, i.e., the South pole and North pole, they would have identical encoding results which is problematic. Moreover, even if we fix this problem, *wrap* is still not a spherical distance-preserved location encoder.

4.1.4. Sinusoidal multi-scale location encoder

One limitation of all location encoders we discussed so far is that they can not handle non-uniform point density (Qi *et al.* 2017a) or mixtures of distributions with very different characteristics (Mai *et al.* 2020b). For example, given a set of POIs, some types tend to cluster together such as night clubs, women's clothing, restaurants while other POI types are rather evenly distributed such as post offices, schools, and fire stations. Similarly, as for species occurrences, some species herd together such as wildebeests and zebras while the individuals of other species tend to walk alone and protect their own territory such as tigers and bears. This will result in different spatial distribution patterns of these species occurrences. In order to jointly model these spatial distributions, we need an encoding method which supports multi-scale representations.

Inspired by the position encoding architecture in Transformer (Vaswani *et al.* 2017), researchers developed multi-scale location encoders by using sinusoidal functions with different frequencies (Mai *et al.* 2020b, Zhong *et al.* 2020).

Definition 4.4 (Sinusoidal Multi-Scale Location Encoder)**.** The sinusoidal multi-scale location encoder is defined as $Enc_{sinmul}(\mathbf{x}) = \mathbf{NN}(PE_{sinmul}(\mathbf{x}))$ where $PE_{sinmul}(\mathbf{x})$ decomposes $\mathbf{x}$ into a multi-scale representation based on different sinusoidal functions with different frequencies:

$$PE_{sinmul}(\mathbf{x}) = [PE_0^{(S)}(\mathbf{x}); ...; PE_s^{(S)}(\mathbf{x}); ...; PE_{S-1}^{(S)}(\mathbf{x})]. \tag{3}$$

Here S is the total number of scales. $PE_s^{(S)}(\mathbf{x})$ processes the location features with different sinusoidal functions whose frequency is determined by the scale s.

TF. Zhong *et al.* (2020) slightly changed the position encoding architecture in Transformer (Vaswani *et al.* 2017) and applied them in high dimension data points such as 3D Cartesian coordinates.

Definition 4.5 (Transformer-based Location Encoder)**.** The transformer-based location encoder $Enc_{TF}(\mathbf{x}) = \mathbf{NN}(PE_{TF}(\mathbf{x}))$ is following Definition 4.4. For each scale $s \in \{0, 1, ..., S-1\}$, $PE_s^{(S)}(\mathbf{x}) = PE_s^{TF}(\mathbf{x})$ is defined by Equation 4. Here, $\mathbf{x}^{[l]}$ indicates the lth dimension of $\mathbf{x}$.

$$PE_s^{TF}(\mathbf{x}) = [PE_{s,1}^{TF}(\mathbf{x}); ...; PE_{s,l}^{TF}(\mathbf{x}); ...; PE_{s,L}^{TF}(\mathbf{x})], \tag{4a}$$

$$where\ PE_{s,l}^{TF}(\mathbf{x}) = [\cos(\frac{2\pi S \mathbf{x}^{[l]}}{S^{(s+1)/S}}); \sin(\frac{2\pi S \mathbf{x}^{[l]}}{S^{(s+1)/S}})],\ \forall l = 1, 2, ..., L. \tag{4b}$$

Zhong *et al.* (2020) showed that $Enc_{TF}(\mathbf{x})$ works well for noiseless data, but for noisy data they need to exclude the top 10% highest frequency components (the smallest several s) in Equation 3. This indicates the necessity of another parameter to control the smallest scale we consider in sinusoidal functions. That is the usage of λ_{min} in *theory* and *grid* which we will discussed below.

theory. Space2Vec (Mai *et al.* 2020b) introduced *theory* as a 2D multi-scale location encoder by using sinusoidal functions with different frequencies.

Definition 4.6 (Theory Location Encoder)**.** Let $\mathbf{a}_1 = [1, 0]^T, \mathbf{a}_2 = [-1/2, \sqrt{3}/2]^T, \mathbf{a}_3 = [-1/2, -\sqrt{3}/2]^T \in \mathbb{R}^2$ be three unit vectors which are oriented 120° apart from each other. $\lambda_{min}, \lambda_{max}$ are the minimum and maximum grid scale, and $g = \frac{\lambda_{max}}{\lambda_{min}}$. $Enc_{theory}(\mathbf{x})$ is following Definition 4.4 where in each scale $s \in \{0, 1, ..., S-1\}$, $PE_s^{(S)}(\mathbf{x}) = PE_s^{theory}(\mathbf{x})$ is defined in Equation 5. Here, $\langle \cdot, \cdot \rangle$ indicates vector inner product.

$$PE_s^{theory}(\mathbf{x}) = [PE_{s,1}^{theory}(\mathbf{x}); PE_{s,2}^{theory}(\mathbf{x}); PE_{s,3}^{theory}(\mathbf{x})], \tag{5a}$$

$$where\ PE_{s,j}^{theory}(\mathbf{x}) = [\cos(\frac{\langle \mathbf{x}, \mathbf{a}_j \rangle}{\lambda_{min} \cdot g^{s/(S-1)}}); \sin(\frac{\langle \mathbf{x}, \mathbf{a}_j \rangle}{\lambda_{min} \cdot g^{s/(S-1)}})],\ \forall j = 1, 2, 3. \tag{5b}$$

Both $Enc_{theory}(\cdot)$ and Gao *et al.* (2019) are inspired by the grid cell research from neuroscience field (Hafting *et al.* 2005, Blair *et al.* 2007, Killian *et al.* 2012, Agarwal *et al.* 2015). In fact, Gao *et al.* (2019) inspired and laid the theoretical foundation of $Enc_{theory}(\mathbf{x})$. As we discussed in Section 2, Gao *et al.* (2019) proposed a location representation model $v(\mathbf{x})$ which consists of three sub-models. They also proposed a complex-value-based location encoder $\mathbf{\Psi}(\mathbf{x})$ as an analytical solution for $v(\mathbf{x})$ which inspired $Enc_{theory}(\cdot)$. More specifically, given two location $\mathbf{x}_a$, $\mathbf{x}_b$, Gao *et al.* (2019) proved that $\langle \mathbf{\Psi}(\mathbf{x}_a), \mathbf{\Psi}(\mathbf{x}_b) \rangle = 3(1 - \frac{\beta}{4} \parallel \mathbf{x}_b - \mathbf{x}_a \parallel^2)$ where $\beta = \parallel \mathbf{a}_j \parallel_2^2 = 1$. That means the inner products between their location embeddings increase when $\parallel \mathbf{x}_b - \mathbf{x}_a \parallel^2$ decrease, which satisfies Property 2.1. Mai *et al.* (2020b) showed that $Enc_{theory}(\cdot)$ also satisfies Property 2.1 both theoretically and empirically.

grid. *grid* is another type of $Enc_{sinmul}(\mathbf{x})$ proposed by Space2Vec (Mai *et al.* 2020b).

Definition 4.7 (Grid Location Encoder). $Enc_{grid}(\mathbf{x})$ follows Definition 4.4 where at each scale $s \in \{0, 1, ..., S-1\}$, $PE_s^{(S)}(\mathbf{x}) = PE_s^{grid}(\mathbf{x})$ is defined by Equation 6. Here, $\lambda_{min}, \lambda_{max}$ and g follow the same definition as Definition 4.6.

$$PE_s^{grid}(\mathbf{x}) = [PE_{s,1}^{grid}(\mathbf{x}); ...; PE_{s,l}^{grid}(\mathbf{x}); ...; PE_{s,L}^{grid}(\mathbf{x})], \tag{6a}$$

$$where\ PE_{s,l}^{grid}(\mathbf{x}) = [\cos(\frac{\mathbf{x}^{[l]}}{\lambda_{min} \cdot g^{s/(S-1)}}); \sin(\frac{\mathbf{x}^{[l]}}{\lambda_{min} \cdot g^{s/(S-1)}})],\ \forall l = 1, 2, .., L. \tag{6b}$$

Mai *et al.* (2020b) shows that for both *theory* and *grid*, λ_{max} can be directly determined based on the size of the study area while λ_{min} is the critical parameter which decides the highest spatial resolution $Enc(\mathbf{x})$ can handle.

4.1.5. Comparison among different $Enc(\mathbf{x})$

Compared with *discretize* which yields identical embeddings for $\mathbf{x}$ that fall into the same tile, *direct* can distinguish nearby locations, i.e., $Enc_{direct}(\mathbf{x}_a) \neq Enc_{direct}(\mathbf{x}_b)$, if $\mathbf{x}_a \neq \mathbf{x}_b$. Mai *et al.* (2020b) compared them in different tasks and found out that without an appropriate location feature normalization $PE_{direct}(\mathbf{x})$, *direct* will show lower performance than *tile*. This indicates the importance of $PE_{direct}(\mathbf{x})$.

One advantage of *direct* is its simple architecture with fewer hyperparameters. However, compared with $PE_{sinu}(\mathbf{x})$ and $PE_{sinmul}(\mathbf{x})$, $PE_{direct}(\mathbf{x})$ is rather hard for $\mathbf{NN}(\cdot)$ to learn from and may produce over-generalized distributions.

Compared with TF and *grid*, *theory* has a theoretical foundation to ensure Property 2.1. However, *theory* can only be applied to point sets in 2D space. In contrast, TF and *grid* lack a theoretical guarantee for Property 2.1 while they can be utilized for points in any L-D space. TF and *grid* follow similar idea while *grid* has an additional parameter λ_{min}, which is more flexible for data sets with different characteristics.

4.2. *Aggregation location encoder* $\boldsymbol{Enc}^{(\mathcal{P})}(\mathbf{x})$

Definition 4.8 (Aggregation Location Encoder)**.** The aggregation location encoder $Enc^{(\mathcal{P})}(\mathbf{x})$ jointly considers the location feature $\mathbf{x}$ and the aggregated features from the neighborhood of $\mathbf{x}$, denoted as $\mathcal{N}(\mathbf{x})$. Inspired by the Graph Neural Network (GNN) framework (Xu *et al.* 2019), a generic model setup of $Enc^{(\mathcal{P})}(\mathbf{x})$ can be defined as

$$\mathbf{h}_{\mathbf{x}}^{(0)} = Enc(\mathbf{x}), \tag{7a}$$

$$\mathbf{g}_{\mathbf{x}}^{(m)} = Agg_{\mathbf{x}_i \in \mathcal{N}(\mathbf{x})}\{\mathbf{h}_{\mathbf{x}_i}^{(m-1)}\}, \tag{7b}$$

$$\mathbf{h}_{\mathbf{x}}^{(m)} = Cmb(\mathbf{h}_{\mathbf{x}}^{(m-1)}, \mathbf{g}_{\mathbf{x}}^{(m)}), \tag{7c}$$

$$Enc^{(\mathcal{P})}(\mathbf{x}) = Rdt(\mathbf{h}_{\mathbf{x}}^{(M)}). \tag{7d}$$

It consists of M Location Encoding Aggregation (LEA) layers, which iteratively update the location representations. In Equation 7a, the initial location embedding $\mathbf{h}_{\mathbf{x}}^{(0)}$ can be computed based on any single point location encoder discussed in Section 4.1. Each LEA layer constitutes one neighborhood aggregation operation $Agg_{\mathbf{x}_i \in \mathcal{N}(\mathbf{x})}\{\cdot\}$ (Equation 7b) and a feature combination operation $Cmb(\cdot,\cdot)$ (Equation 7c). $Agg_{\mathbf{x}_i \in \mathcal{N}(\mathbf{x})}\{\cdot\}$ aggregates the feature $\mathbf{h}_{\mathbf{x}_i}^{(m-1)}$ of $\mathbf{x}_i$ in $\mathcal{N}(\mathbf{x})$ from the previous LEA layer which can be seen as an analogy of the convolution operation of CNN on point sets. $Agg_{\mathbf{x}_i \in \mathcal{N}(\mathbf{x})}\{\cdot\}$ can be element-wise max/min/mean pooling, sum, or any permutation invariant architectures (Zaheer *et al.* 2017, Qi *et al.* 2017a, Veličković *et al.* 2018, Mai *et al.* 2019a). $Cmb(\cdot,\cdot)$ combines the point-wise feature $\mathbf{h}_{\mathbf{x}}^{(m-1)}$ with the aggregated features $\mathbf{g}_{\mathbf{x}}^{(m)}$ which can be vector concatenation, element-wise max, min or mean. In the last layer, a readout function $Rdt(\cdot)$, which can be an identity function or a multi-layer perceptron, produces the final aggregated location embedding for $\mathbf{x}$ (See Equation 7d).

Equation 7 can be treated as an analogy of the GNN framework (Battaglia *et al.* 2018, Xu *et al.* 2019, Wu *et al.* 2020b) which is a general framework for different neural network architectures applied on graphs such as GCN (Kipf and Welling 2017), GraphSAGE (Hamilton *et al.* 2017), GG-NN (Li *et al.* 2016), GAN (Veličković *et al.* 2018), MPNN (Gilmer *et al.* 2017), R-GCN (Schlichtkrull *et al.* 2018), CGA (Mai *et al.* 2019a), TransGCN (Cai *et al.* 2019), and so on.

$\mathcal{N}(\mathbf{x})$ can be defined in different ways such as the top kth nearest locations (Appleby *et al.* 2020, Mai *et al.* 2020b), locations within a buffer radius (Qi *et al.* 2017b), or locations within the same voxel as $\mathbf{x}$ (Zhou and Tuzel 2018). According to the definition of $\mathcal{N}(\mathbf{x})$, we classify $Enc^{(\mathcal{P})}(\mathbf{x})$ into different categories: kernel, global, local neighborhood, and hierarchical neighborhood aggregation location encoder as summarized in Table 1. We will discuss each in detail in the following section.

4.2.1. Kernel-based location encoder

A kernel-based location encoder needs two components: a predefined kernel function $k(\cdot,\cdot)$ and a set of kernel center points $\mathcal{Q} = \{p_j\}$. The selection of $k(\cdot,\cdot)$ depends on the nature of the dataset. The popular options are RBF kernels and Mercer kernels (Vapnik 2013). $\mathcal{Q}$ can be equal to or a subset of the training point set, i.e., $\mathcal{Q} \subseteq \mathcal{P}$ or can be a predefined point set such as the centers of regular grids (Yin *et al.* 2019).

Definition 4.9 (Kernel-Based Location Encoder)**.** Given a kernel function $k(\cdot,\cdot)$ and the kernel center point set $\mathcal{Q} = \{p_j\}$, we define the kernel-based location encoder as

Equation 8 by following Definition 4.8 where $M = 1$.

$$\mathbf{h}_{\mathbf{x}}^{(0)} = \mathbf{x}, \tag{8a}$$

$$\mathbf{g}_{\mathbf{x}}^{(1)} = Agg_{p_i \in \mathcal{N}(\mathbf{x})}^{(kernel)}\{\mathbf{h}_{\mathbf{x}_i}^{(0)}\} = \Psi(\mathbf{x}) = [k(\mathbf{x}, \mathbf{x}_1); ...; k(\mathbf{x}, \mathbf{x}_{|\mathcal{Q}|})], \tag{8b}$$

$$\mathbf{h}_{\mathbf{x}}^{(1)} = Cmb(\mathbf{h}_{\mathbf{x}}^{(0)}, \mathbf{g}_{\mathbf{x}}^{(1)}) = \mathbf{g}_{\mathbf{x}}^{(1)}, \tag{8c}$$

$$Enc_{kernel}^{(\mathcal{P})}(\mathbf{x}) = Rdt(\mathbf{h}_{\mathbf{x}}^{(1)}) = \mathbf{NN}(\mathbf{h}_{\mathbf{x}}^{(1)}). \tag{8d}$$

Here, each $\mathbf{x}$ has a fixed "neighborhood" - $\mathcal{Q}$, i.e., $\mathcal{N}(\mathbf{x}) = \mathcal{Q} = \{p_j = (\mathbf{x}_i)\}$. $[\cdot;\cdot]$ indicates vector concatenation. $|\mathcal{Q}|$ indicates the total number of kernels and $\mathbf{NN}(\cdot)$ is a multi-layer perceptron.

From a location feature decomposition perspective, $Enc_{kernel}^{(\mathcal{P})}(\mathbf{x})$ can be seen as using kernel trick to decompose $\mathbf{x}$ into multiple kernel features (see Section 3). From a location encoding aggregation perspective, it can be seen as aggregating the kernel features derived from the interaction between $\mathbf{x}$ and each kernel. Since each location $\mathbf{x} \in \mathcal{P}$ shares the same $\mathcal{N}(\mathbf{x}) = \mathcal{Q}$, $Agg_{p_i \in \mathcal{N}(\mathbf{x})}^{(kernel)}\{\cdot\}$ does not need to be a permutation invariant function. Here we use a concatenation of the $|\mathcal{Q}|$ kernel features of $\mathbf{x}$. Examples of $Enc_{kernel}^{(\mathcal{P})}(\mathbf{x})$ include GPS2Vec (Yin *et al.* 2019) and the rbf baseline used by Space2Vec (Mai *et al.* 2020b). The main difference among them are the definitions of $\mathcal{Q}$ and $k(\cdot,\cdot)$.

GPS2Vec. GPS2Vec (Yin *et al.* 2019) divided the Earth into multiple UTM zones and trained different $Enc_{kernel}^{(\mathcal{P})}(\mathbf{x})$ separately. Each zone is further divided into q grids whose centers are used as $\mathcal{Q}$ and $k(\mathbf{x}, \mathbf{x}_j) = \exp(-\frac{\| \mathbf{x} - \mathbf{x}_j \|_2}{\zeta})$. Here $\| \cdot \|_2$ indicates L2 norm and ζ is a constant attenuation coefficient. We denote this model as $Enc_{GPS2Vec}^{(\mathcal{P})}(\cdot)$.

rbf. Mai *et al.* (2020b) proposed a RBF kernel based location encoder as one of their baselines, called rbf. $\mathcal{Q}$ are randomly sampled q points from $\mathcal{P}$, i.e., $\mathcal{Q} \subseteq \mathcal{P}$. The kernel function $k(\mathbf{x}, \mathbf{x}_j) = \exp\big(-\frac{\| \mathbf{x} - \mathbf{x}_j \|_2^2}{2\sigma^2}\big)$ is a RBF kernel. We denote it as $Enc_{rbf}^{(\mathcal{P})}(\cdot)$.

Adapted kernel* Instead of using constant kernel bandwidth, Berg *et al.* (2014) also proposed an idea of adaptive kernels to model the spatio-temporal distribution prior of bird species. They precomputed this prior as a fixed scale kernel density estimation (KDE) map. Here, instead of making a precomputed KDE map, we adopt this idea to design an adaptive kernel based location encoder $Enc_{ak}(\mathbf{x})$ in which $k(\mathbf{x}, \mathbf{x}_j) = \exp(-\frac{\| \mathbf{x} - \mathbf{x}_j \|_2^2}{2h_\sigma(\mathbf{x})^2})$. $h_\sigma(\mathbf{x})$ is a kernel bandwidth function of location $\mathbf{x}$, e.g., defining $h_\sigma(\mathbf{x})$ as the half of the distance from $\mathbf{x}$ to its Kth nearest neighbor. We use * to differentiate $Enc_{ak}(\mathbf{x})$ from the original model proposed by Berg *et al.* (2014).

Despite its simple design and the ability to handle non-linear distributions, $Enc_{kernel}(\mathbf{x})$ has several shortcomings. Firstly, $Enc_{kernel}(\mathbf{x})$ has to memorize $\mathcal{Q}$ at testing time which will affect memory efficiency. Secondly, since $\mathbf{h}_{\mathbf{x}}^{(1)} \in \mathbb{R}^{|\mathcal{Q}|}$, the size of $\mathcal{Q}$ directly decides the number of learnable parameters in $\mathbf{NN}(\cdot)$. This creates a performance-efficiency trade-off problem. When $|\mathcal{Q}|$ is small, $Enc_{kernel}(\mathbf{x})$ has less memory burden. $\mathbf{NN}(\cdot)$ also has fewer learnable parameters which need less training data and is less likely to over fit. However, $\mathcal{Q}$ is rather distributed sparsely over the study area which affects the quality of the encoding results. When $|\mathcal{Q}|$ is rather large, the encoding results are more accurate, but we need a huge amount of memory to store $\mathcal{Q}$ and $\mathbf{NN}(\cdot)$ needs more learning parameters. Moreover, the prediction of $Enc_{kernel}(\mathbf{x})$

also depends on the distribution of $\mathcal{Q}$ since it performs poorly on data sparse regions.

4.2.2. Global aggregation location encoder

The global aggregation location encoder $Enc^{(\mathcal{P})}_{global}(\mathbf{x})$ defines $\mathcal{N}(\mathbf{x})$ as all locations in $\mathcal{P}$, i.e., $\mathcal{N}_{global}(\mathbf{x}) = \mathcal{P}$.

PointNet. PointNet (Qi *et al.* 2017a) was originally proposed for 3D point cloud classification and segmentation. Its point cloud segmentation architecture can be formulated as a global aggregation location encoder as shown in Equation 9. Equation 9a first embeds each point into an initial embedding with a *direct*-like single point location encoder $Enc_{pnet}(\mathbf{x})$. It normalizes the input 3D point feature $\mathbf{x} = [x, y, z]^T \in \mathbb{R}^3$ on to a unit sphere, denoted as $PE_{pnet}(\cdot)$, before feeding them into $\mathbf{NN}(\cdot)$. $\mathbf{NN}(\cdot)$ consists of two affine transformations (parameterized as t-nets) $TN_1(\cdot)$ and $TN_2(\cdot)$, which are separated by a multi-layer perceptron $MLP_1(\cdot)$. These affine transformations help to make the semantic labeling of a point cloud invariant to geometric transformations (Qi *et al.* 2017a). Note that PointNet only has one LEA layer, i.e., $M = 1$. $Agg^{(pnet)}_{\mathbf{x}_i \in \mathcal{N}(\mathbf{x})}\{\cdot\}$ is a multi-layer perceptron $MLP_2(\cdot)$ for each $\mathbf{x}_i \in \mathcal{N}_{pnet}(\mathbf{x}) = \mathcal{P}$ followed by an element-wise max pooling $MaxPool_{\mathbf{x}_i \in \mathcal{N}_{pnet}(\mathbf{x})}\{\cdot\}$ (See Equation 9b). $Cmb(\cdot, \cdot)$ is a vector concatenation operation followed by a multi-layer perceptron $MLP_3(\cdot)$ and the readout function $Rdt(\cdot)$ is an identity function as shown in Equation 9c, 9d.

$$\mathbf{h}^{(0)}_{\mathbf{x}} = Enc_{pnet}(\mathbf{x}) = \mathbf{NN}(PE_{pnet}(\mathbf{x})) = TN_2(MLP_1(TN_1(PE_{pnet}(\mathbf{x})))), \quad (9a)$$

$$\mathbf{g}^{(1)}_{\mathbf{x}} = Agg^{(pnet)}_{\mathbf{x}_i \in \mathcal{N}_{pnet}(\mathbf{x})}\{\mathbf{h}^{(0)}_{\mathbf{x}_i}\} = MaxPool_{\mathbf{x}_i \in \mathcal{N}_{pnet}(\mathbf{x})}\{MLP_2(\mathbf{h}^{(0)}_{\mathbf{x}_i})\}, \quad (9b)$$

$$\mathbf{h}^{(1)}_{\mathbf{x}} = Cmb^{(pnet)}(\mathbf{h}^{(0)}_{\mathbf{x}}, \mathbf{g}^{(1)}_{\mathbf{x}}) = MLP_3([\mathbf{h}^{(0)}_{\mathbf{x}}; \mathbf{g}^{(1)}_{\mathbf{x}}]), \quad (9c)$$

$$Enc^{(\mathcal{P})}_{pnet}(\mathbf{x}) = Rdt^{(pnet)}(\mathbf{h}^{(1)}_{\mathbf{x}}) = \mathbf{h}^{(1)}_{\mathbf{x}}. \quad (9d)$$

4.2.3. Local aggregation location encoder

The local aggregation location encoder $Enc^{(\mathcal{P})}_{local}(\mathbf{x})$ considers a local neighborhood $\mathcal{N}(\mathbf{x})$ such as all locations within a buffer of $\mathbf{x}$ with radius r, i.e., $\mathcal{N}_{buffer}(\mathbf{x}) = \{\mathbf{x}_i | \parallel \mathbf{x} - \mathbf{x}_i \parallel_2 \leqslant r, \ \forall \mathbf{x}_i \in \mathcal{P} \wedge \mathbf{x}_i \neq \mathbf{x}\}$.

VoxelNet. Zhou and Tuzel (2018) discretizes the 3D space into unit voxels. For any location $\mathbf{x}$, its neighborhood is defined as $\mathcal{N}_{vox}(\mathbf{x}) = \{\mathbf{x}_i | \iota(\mathbf{x}) = \iota(\mathbf{x}_i), \ \forall \mathbf{x}_i \in \mathcal{P}\}$, where $\iota(\mathbf{x})$ is a voxel lookup function which returns the ID of the voxel in which $\mathbf{x}$ falls into.

$$\mathbf{h}^{(0)}_{\mathbf{x}} = Enc_{vox}(\mathbf{x}) = \mathbf{NN}(PE_{vox}(\mathbf{x})) = PE_{vox}(\mathbf{x}), \quad (10a)$$

$$\mathbf{g}^{(m)}_{\mathbf{x}} = Agg^{(vox)}_{\mathbf{x}_i \in \mathcal{N}_{vox}(\mathbf{x})}\{\mathbf{h}^{(m-1)}_{\mathbf{x}_i}\} = MaxPool_{\mathbf{x}_i \in \mathcal{N}_{vox}(\mathbf{x})}\{FCN^{(m)}(\mathbf{h}^{(m-1)}_{\mathbf{x}_i})\}, \quad (10b)$$

$$\mathbf{h}^{(m)}_{\mathbf{x}} = Cmb^{(vox)}(\mathbf{h}^{(m-1)}_{\mathbf{x}}, \mathbf{g}^{(m)}_{\mathbf{x}}) = [\mathbf{h}^{(m-1)}_{\mathbf{x}}; \mathbf{g}^{(m)}_{\mathbf{x}}], \quad (10c)$$

$$Enc^{(\mathcal{P})}_{vox}(\mathbf{x}) = Rdt^{(vox)}(\mathbf{h}^{(M)}_{\mathbf{x}}) = \mathbf{h}^{(M)}_{\mathbf{x}}. \quad (10d)$$

First, a *direct*-like location encoder $Enc_{vox}(\cdot)$ (Equation 10a) encodes $\mathbf{x} = [x, y, z]^T$ into $PE_{vox}(\mathbf{x}) = [x, y, z, x - \bar{x}, y - \bar{y}, z - \bar{z}]^T$. Here $\mathbf{NN}(\cdot)$ is an identity function[4], and

[4]In Zhou and Tuzel (2018), they encode each 3D point as $[x, y, z, v, x - \bar{x}, y - \bar{y}, z - \bar{z}]^T$ where v is an attribute

$(\bar{x}, \bar{y}, \bar{z})$ is the centroid of all points in $\mathcal{N}_{vox}(\mathbf{x})$. Then aggregation (Equation 10b) is done by a pointwise fully connected layer $FCN^{(m)}(\cdot)$ followed by an element-wise max pooling $MaxPool_{\mathbf{x}_i \in \mathcal{N}_{vox}(\mathbf{x})}\{\cdot\}$. $\mathbf{g}_{\mathbf{x}}^{(m)}$ encodes the shape contained within the current voxel $\mathcal{N}_{vox}(\mathbf{x})$. $Cmb(\cdot, \cdot)$ is simply a vector concatenation operation (See Equation 10c), and $Rdt_{vox}(\cdot)$ is a identity function (See Equation 10d).

SAGAT. Mai *et al.* (2020b) proposed a modified graph attention network (GAT) (Veličković *et al.* 2018) to model the spatial interactions among nearby locations (e.g., POIs). $\mathcal{N}_{knn}(\mathbf{x})$ is defined as all k nearest neighbors of location $\mathbf{x}$. The original model focuses on encoding feature information $\mathbf{v}$ for each location such as POI type, but here we generalize it as a generic local aggregation location encoder (Spatial-Aware Graph Attention Network, in short, **SAGAT**):

$$\mathbf{h}_{\mathbf{x}}^{(0)} = Enc_{sagat}(\mathbf{x}), \tag{11a}$$

$$\mathbf{g}_{\mathbf{x}}^{(m)} = Agg^{(sagat)}_{\mathbf{x}_i \in \mathcal{N}_{knn}(\mathbf{x})}\{\mathbf{h}_{\mathbf{x}_i}^{(m-1)}\} = \sigma\Big(\frac{1}{U}\sum_{u=1}^{U} \sum_{\mathbf{x}_i \in \mathcal{N}_{knn}(\mathbf{x})} \alpha_{iu}^{(m)} \mathbf{h}_{\mathbf{x}_i}^{(m-1)}\Big), \tag{11b}$$

$$\mathbf{h}_{\mathbf{x}}^{(m)} = Cmb^{(sagat)}(\mathbf{h}_{\mathbf{x}}^{(m-1)}, \mathbf{g}_{\mathbf{x}}^{(m)}) = \mathbf{g}_{\mathbf{x}}^{(m)}, \tag{11c}$$

$$Enc_{sagat}^{(\mathcal{P})}(\mathbf{x}) = Rdt^{(sagat)}(\mathbf{h}_{\mathbf{x}}^{(M)}) = \mathbf{h}_{\mathbf{x}}^{(M)}, \tag{11d}$$

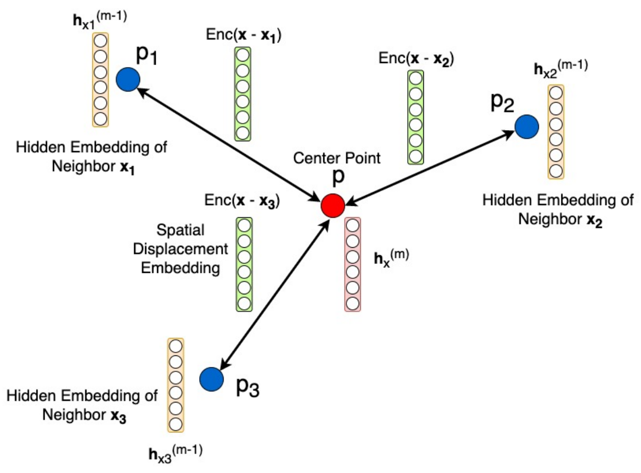
$$where\ \alpha_{iu}^{(m)} = \frac{exp(\sigma_{iu}^{(m)})}{\sum_{\mathbf{x}_o \in \mathcal{N}_{knn}(\mathbf{x})} exp(\sigma_{ou}^{(m)})}, \tag{11e}$$

$$\sigma_{iu}^{(m)} = LeakyReLU((\mathbf{a}_u^{(m)})^T [\mathbf{h}_{\mathbf{x}}^{(m-1)}; \mathbf{h}_{\mathbf{x}_i}^{(m-1)}; Enc(\mathbf{x} - \mathbf{x}_i)]). \tag{11f}$$

In Equation 11a, SAGAT first uses $Enc_{sagat}(\mathbf{x})$ to lift each location into the embedding space. Originally, Mai *et al.* (2020b) used a feature encoder as $Enc_{sagat}(\mathbf{x})$ to represent $\mathbf{v}$ into a feature embedding. However, here, we define $Enc_{sagat}(\mathbf{x})$ to be any single point location encoder discussed in Section 4.1. Equation 11b is the most important neighborhood aggregation step. Multi-head attention is adopted to aggregate the neighboring location embedding $\mathbf{h}_{\mathbf{x}_i}^{(m-1)}$ of $\mathbf{x}_i \in \mathcal{N}_{knn}(\mathbf{x})$ from the previous layer. U is the total number of attention heads. $\alpha_{iu}^{(m)}$ is the attention coefficient of $\mathbf{h}_{\mathbf{x}_i}^{(m-1)}$ in the uth attention head within the mth layer which is normalized across $\mathcal{N}_{knn}(\mathbf{x})$ based on Equation 11e. $\sigma_{iu}^{(m)}$ is its non-normalized counterpart. In Equation 11f, $\mathbf{h}_{\mathbf{x}}^{(m-1)}$ and $\mathbf{h}_{\mathbf{x}_i}^{(m-1)}$ are the representations of the center location $\mathbf{x}$ and its neighbor $\mathbf{x}_i$ in the $m-1$-th layer. Compared with GAT (Veličković *et al.* 2018), $Enc(\mathbf{x}-\mathbf{x}_i)$ is added in the attention score computation so the spatial affinity $\Delta\mathbf{x}_i = \mathbf{x} - \mathbf{x}_i$ is considered in the aggregation step. This practice makes SAGAT "spatial-aware". $\mathbf{a}_u^{(m)}$ is a learnable attention vector for the uth attention head and $LeakyReLU(\cdot)$ is the LeakyReLU activation function. $Enc(\cdot)$ is an encoder for spatial relation, which can be any single-point location encoder such as *tile*, *rbf*, *direct*, *theory*, *grid*, *TF*, and so on. Mai *et al.* (2020b) only utilized one aggregation layer, i.e., $M = 1$. Here, we generalize it into multiple layers, i.e., $M \geqslant 1$. The combination and readout function are both identity functions.

Figure 4 illustrates the mth aggregation layer of SAGAT. In this example, we consider three neighbors. Three yellow vectors $\mathbf{h}_{\mathbf{x}_i}^{(m-1)}$ $(i = 1, 2, 3)$ are the hidden embeddings of three neighbors. Three green vectors $Enc(\mathbf{x} - \mathbf{x}_i)$ are the spatial displacement

of each point, e.g., received reflectance. Here we skip v to focus on the location information. However, as we said in Definition 2.1, it is very easy to extend $Enc^{(\mathcal{P},\theta)}(\mathbf{x})$ to $Enc^{(\mathcal{P},\theta)}(\mathbf{x}, \mathbf{v})$.



$h_{x1}^{(m-1)}$
p1
Enc(x - x1)
Enc(x - x2)
$h_{x2}^{(m-1)}$
p2
Hidden Embedding of Neighbor x1
Center Point
p
Hidden Embedding of Neighbor x2
Enc(x - x3)
Spatial Displacement Embedding
$h_x^{(m)}$
Hidden Embedding of Neighbor x3
p3
$h_{x3}^{(m-1)}$

Figure 4.: An illustration of the mth aggregation layer of SAGAT.

embeddings which are used in the spatial-aware graph attention attention. The red vector are the hidden embedding of center location $\mathbf{x}$ in the next layer.

DGCNN. Dynamic Graph CNNs (in short, DGCNN) (Wang *et al.* 2019) is proposed to jointly consider global shape structure and local neighborhood information when learning on point clouds. DGCNN also has M LEA layers. Compared with other location encoders, DGCNN has two crucial distinctions: 1) dynamic graph neighborhood $\mathcal{N}^{(m)}_{dgcnn}(\mathbf{x})$ and 2) an edge convolution module $EdgeConv^{(m)}_{\mathbf{x}_i \in \mathcal{N}^{(m)}_{dgcnn}(\mathbf{x})}\{\cdot\}$.

$$\mathbf{g}^{(m)}_{\mathbf{x}} = Agg^{(dgcnn)}_{\mathbf{x}_i \in \mathcal{N}^{(m)}_{dgcnn}(\mathbf{x})}\{\mathbf{h}^{(m-1)}_{\mathbf{x}_i}\} = EdgeConv^{(m)}_{\mathbf{x}_i \in \mathcal{N}^{(m)}_{dgcnn}(\mathbf{x})}\{\mathbf{h}^{(m-1)}_{\mathbf{x}_i}\} \tag{12a}$$

$$= MaxPool_{\mathbf{x}_i \in \mathcal{N}^{(m)}_{dgcnn}(\mathbf{x})}\{MLP^{(m)}_e([\mathbf{h}^{(m-1)}_{\mathbf{x}}; \mathbf{h}^{(m-1)}_{\mathbf{x}_i} - \mathbf{h}^{(m-1)}_{\mathbf{x}}])\}, \tag{12b}$$

$$\mathbf{h}^{(m)}_{\mathbf{x}} = Cmb^{(dgcnn)}(\mathbf{h}^{(m-1)}_{\mathbf{x}}, \mathbf{g}^{(m)}_{\mathbf{x}}) = \mathbf{g}^{(m)}_{\mathbf{x}}. \tag{12c}$$

The former means that instead of using a fixed neighborhood as VoxelNet and SAGAT does, DGCNN recomputes the neighborhood of $\mathbf{x}$ for each LEA layer. In the mth layer, given the embeddings of all locations from the previous layer $\mathcal{H}^{(m-1)} = \{\mathbf{h}^{(m-1)}_{\mathbf{x}_j} | \forall \mathbf{x}_j \in \mathcal{P}\}$, $\mathcal{N}^{(m)}_{dgcnn}(\mathbf{x})$ is defined as the top K nearest neighbors of $\mathbf{h}^{(m-1)}_{\mathbf{x}}$ from $\mathcal{H}^{(m-1)}$ in the embedding space (not Euclidean space). Since $\mathcal{H}^{(m-1)}$ is updated after each LEA layer, $\mathcal{N}^{(m)}_{dgcnn}(\mathbf{x})$ is called a dynamic neighborhood. As for the edge convolution, $EdgeConv^{(m)}_{\mathbf{x}_i \in \mathcal{N}^{(m)}_{dgcnn}(\mathbf{x})}\{\cdot\}$ is an aggregation operator which captures local geometric structure while maintaining permutation invariance. Equation 12 mainly shows how the edge convolution works while skipping other steps such as location embedding initialization and the readout function. $EdgeConv^{(m)}_{\mathbf{x}_i \in \mathcal{N}^{(m)}_{dgcnn}(\mathbf{x})}\{\cdot\}$ concatenates $\mathbf{h}^{(m-1)}_{\mathbf{x}}$ and its affinity to its neighbor $\mathbf{h}^{(m-1)}_{\mathbf{x}_i} - \mathbf{h}^{(m-1)}_{\mathbf{x}}$. The result is fed into a multi-layer perceptron $MLP^{(m)}_e(\cdot)$ followed by a pointwise max pooling. The combination operator is an identity function (See Equation 12c). Although $\mathcal{N}^{(m)}_{dgcnn}(\mathbf{x})$ is dynamic, we only consider single-scale neighborhoods for each $\mathbf{x}$. So we classify DGCNN can a local aggregation location encoder.

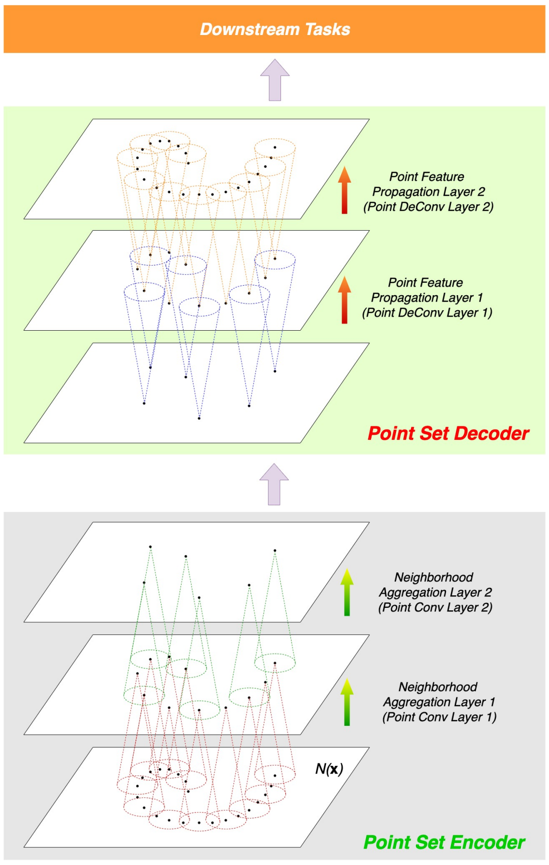

Downstream Tasks
Point Feature Propagation Layer 2 (Point DeConv Layer 2)
Point Feature Propagation Layer 1 (Point DeConv Layer 1)
Point Set Decoder
Neighborhood Aggregation Layer 2 (Point Conv Layer 2)
Neighborhood Aggregation Layer 1 (Point Conv Layer 1)
N(x)
Point Set Encoder

Figure 5.: An illustration of the Conv-DeConv architecture (Noh *et al.* 2015) for $Enc^{(\mathcal{P})}_{hieagg}(\mathbf{x})$ such as PointNet++ (Qi *et al.* 2017b), PointCNN (Li *et al.* 2018b), and Graph-Conv GAN (Valsesia *et al.* 2019).

4.2.4. Hierachical aggregation location encoder

Instead of defining one neighborhood per location, the hierachical aggregation location encoder $Enc^{(\mathcal{P})}_{hieagg}(\mathbf{x})$ defines a multi-scale neighborhood for location $\mathbf{x}$, i.e., $\mathcal{N}_{hieagg}(\mathbf{x}) = \{\mathcal{N}^{(1)}_{hieagg}(\mathbf{x}), \mathcal{N}^{(2)}_{hieagg}(\mathbf{x}), ..., \mathcal{N}^{(m)}_{hieagg}(\mathbf{x}), ..., \mathcal{N}^{(M)}_{hieagg}(\mathbf{x})\}$ which can be aggregated in a hierarchical manner as illustrated in Figure 5.

Definition 4.10 (Hierachical Aggregation Location Encoder)**.** $Enc^{(\mathcal{P})}_{hieagg}(\mathbf{x})$ first hierarchically aggregates location features from these neighborhoods and then propagates the aggregated features back to each location. $Enc^{(\mathcal{P})}_{hieagg}(\mathbf{x})$ usually adopts a Conv-DeConv (Noh *et al.* 2015) like architecture which consists of two modules: 1) **Point Set Encoder** $PTConv^{(\mathcal{P})}_{hieagg}(\cdot)$: a stack of M neighborhood aggregation layers (or called Point Conv layers (Li *et al.* 2018b)) in which each location embedding in the m layer is aggregated from its neighborhood $\mathcal{N}^{(m-1)}_{hieagg}(\mathbf{x})$ in the previous layer; 2) **Point Set Decoder** $PTDeConv^{(\mathcal{P})}_{hieagg}(\cdot)$: a stack of M point feature propagation layers (or called Point DeConv layers) in which each DeConv-like architecture propagates the location embeddings from the previous layers to a denser set of locations. This whole architecture follows the U-Net design (Ronneberger *et al.* 2015):

$$Enc^{(\mathcal{P})}_{hieagg}(\mathbf{x}) = PTDeConv^{(\mathcal{P})}_{hieagg}(PTConv^{(\mathcal{P})}_{hieagg}(\mathbf{x})) \tag{13}$$

Here, $PTConv^{(\mathcal{P})}_{hieagg}(\cdot)$ follows the model setup in Equation 7. Sometimes, $PTDeConv^{(\mathcal{P})}_{hieagg}(\cdot)$ also follows Equation 7 such as PointCNN (Li *et al.* 2018b).

Figure 5 shows an illustration of this Conv-DeConv architecture of $Enc^{(\mathcal{P})}_{hieagg}(\mathbf{x})$.

PointNet++. As an extension of PointNet, PointNet++ (Qi *et al.* 2017b) follows the Conv-DeConv like architecture in Figure 5 to achieve hierarchical feature aggregation and propagation. The point set encoder $PTConv^{(\mathcal{P})}_{pnet+}(\mathbf{x})$ consists of M Point Conv Layers (so-called set abstraction levels (SAL) in Qi *et al.* (2017b)) each of which is composed of three key layers:

(1) **The sampling layer** at the mth SAL samples a point subset $\mathcal{P}^{(m)}$ from the previous SAL - $\mathcal{P}^{(m-1)}$, and, therefore, $\mathcal{P}^{(m)} \subset \mathcal{P}^{(m-1)}$ and $\mathcal{P}^{(0)} = \mathcal{P}$.
(2) **The grouping layer** at the mth SAL groups points in $\mathcal{P}^{(m-1)}$ into the neighborhood $\mathcal{N}^{(m)}_{pnet+}(\mathbf{x}) \subseteq \mathcal{P}^{(m-1)}$ of each point $\mathbf{x} \in \mathcal{P}^{(m)}$. $\mathcal{N}^{(m)}_{pnet+}(\mathbf{x})$ is defined as all points $\mathbf{x}_i \in \mathcal{P}^{(m-1)}$ within radius $r^{(m)}$.
(3) **The aggregation layer** at the mth SAL produces new location embedding $\mathbf{h}^{(m)}_{\mathbf{x}}$ for each $\mathbf{x} \in \mathcal{P}^{(m)}$ by aggregating $\mathbf{h}^{(m-1)}_{\mathbf{x}_i}$ of $\mathbf{x}_i$ in $\mathbf{x}$'s neighborhood $\mathcal{N}^{(m)}_{pnet+}(\mathbf{x})$.

$$\begin{aligned}
\mathbf{h}^{(0)}_{\mathbf{x}} &= Enc_{direct}(\mathbf{x}), && (14a)\\
\mathbf{g}^{(m)}_{\mathbf{x}} &= Agg^{(ssg)}_{\mathbf{x}_i \in \mathcal{N}^{(m)}_{pnet+}(\mathbf{x})}\{\mathbf{h}^{(m-1)}_{\mathbf{x}_i}\} && (14b)\\
&= MaxPool_{\mathbf{x}_i \in \mathcal{N}^{(m)}_{pnet+}(\mathbf{x})}\{MLP^{(m)}(\mathbf{h}^{(m-1)}_{\mathbf{x}_i})\}, && (14c)\\
\mathbf{h}^{(m)}_{\mathbf{x}} &= Cmb^{(ssg)}(\mathbf{h}^{(m-1)}_{\mathbf{x}}, \mathbf{g}^{(m)}_{\mathbf{x}}) = \mathbf{g}^{(m)}_{\mathbf{x}}, && (14d)\\
PTConv^{(\mathcal{P})}_{ssg}(\mathbf{x}) &= Rdt^{(ssg)}(\mathbf{h}^{(M)}_{\mathbf{x}}) = \mathbf{h}^{(M)}_{\mathbf{x}}. && (14e)
\end{aligned}$$

Qi *et al.* (2017b) proposed several versions of PointNet++: **SSG** (ablated PointNet++ with single scale grouping in each level), **MSG** (multi-scale grouping PointNet++), and **MRG** (multi resolution grouping PointNet++). All of them are $Enc^{(\mathcal{P})}_{hieagg}(\mathbf{x})$ according to Definition 4.10. Equation 14 shows how the point set encoder of **SSG** works. In the mth SAL, the PointNet layer aggregates features in $\mathcal{N}^{(m)}_{pnet+}(\mathbf{x})$ by using a PointNet-like aggregation function $Agg^{(ssg)}_{\mathbf{x}_i \in \mathcal{N}^{(m)}_{pnet+}(\mathbf{x})}\{\cdot\}$ - $MLP^{(m)}(\cdot)$ followed by a max pooling function (See Equation 14c).

Because $\mathcal{P}^{(M)} \subset \mathcal{P}^{(M-1)} \subset ... \subset \mathcal{P}^{(0)} = \mathcal{P}$, $\mathcal{P}^{(M)}$ is a small subset of $\mathcal{P}$ which can be seen as the skeleton points of $\mathcal{P}$. $PTConv^{(\mathcal{P})}_{ssg}(\cdot)$ can only produce embeddings for $\mathbf{x} \in \mathcal{P}^{(M)}$. To obtain embeddings for each $\mathbf{x} \in \mathcal{P}$, $PTDeConv^{(\mathcal{P})}_{ssg}(\cdot)$ is used to propagate location features back to each $\mathbf{x} \in \mathcal{P}$ by using an inverse distance weighted interpolation method. This can be seen as a reverse process of $PTConv^{(\mathcal{P})}_{ssg}(\cdot)$. The sampled point sets $\{\mathcal{P}^{(0)} = \mathcal{P}, \mathcal{P}^{(1)}, \mathcal{P}^{(2)}, ..., \mathcal{P}^{(m)}, ..., \mathcal{P}^{(M)}\}$ are used in a reverse manner to progressively interpolate the location features back to a larger sampled point set until we get location embeddings for all $\mathbf{x} \in \mathcal{P}$. This idea follows the Conv-DeConv idea in Equation 13.

Compared with **SSG**, **MSG** changes the point set encoder by concatenating location embeddings of the same points obtained from neighborhoods with different spatial scales. **MSG** is rather computationally expensive since it aggregates features in larger scale neighborhoods for each centroid point. **MRG** solves this by concatenating location embeddings obtained from different **SSG** point set encoders with varied numbers of SAL layers. Please refer to Qi *et al.* (2017b) for detailed description.

PointCNN. Similar to PointNet++, PointCNN (Li *et al.* 2018b) also utilizes a point set encoder with M Point Conv layers (which we call PointCNN Layers here). Each layer also consists of three key steps: sampling layer, grouping layer, aggregation layer (PointNet layer). The differences from **SSG** are mainly in grouping and aggregation layer. The mth PointCNN grouping layer defines the neighborhood $\mathcal{N}^{(m)}_{ptcnn}(\mathbf{x}) \subset \mathcal{P}^{(m-1)}$ as a set of K points uniformly sampled from the $K\mathcal{D}$ nearest neighbors of $\mathbf{x}$ obtained from $\mathcal{P}^{(m-1)}$. This can be seen as an analogy of the dilated convolution idea from the traditional CNN models and $\mathcal{D}$ is the dilation rate. The PointCNN aggregation layer defines a convolution operation, $Conv^{(ptcnn)}(\kappa^{(m)}, \cdot)$, over $\mathcal{N}^{(m)}_{ptcnn}(\mathbf{x})$ as an analogy to the convolution operation over images. $\kappa^{(m)}$ is the convolution kernels in the mth layer.

$$\mathbf{h}^{(m)}_{\mathbf{x}_i,\delta} = MLP^{(m)}_{\delta}(\mathbf{x}_i - \mathbf{x}), \forall \mathbf{x}_i \in \mathcal{N}^{(m)}_{ptcnn}(\mathbf{x}), \tag{15a}$$

$$\mathcal{X}^{(m)} = MLP^{(m)}(\Gamma_{\mathcal{N}^{(m)}_{ptcnn}(\mathbf{x})}(\mathbf{x}_i - \mathbf{x})), \tag{15b}$$

$$\mathbf{g}^{(m)}_{\mathbf{x}} = Agg^{(ptcnn)}_{\mathbf{x}_i \in \mathcal{N}^{(m)}_{ptcnn}(\mathbf{x})}\{\mathbf{h}^{(m-1)}_{\mathbf{x}_i}\} \tag{15c}$$

$$= Conv^{(ptcnn)}(\kappa^{(m)}, \mathcal{X}^{(m)} \times \Gamma_{\mathcal{N}^{(m)}_{ptcnn}(\mathbf{x})}([\mathbf{h}^{(m)}_{\mathbf{x}_i,\delta}; \mathbf{h}^{(m-1)}_{\mathbf{x}_i}])), \tag{15d}$$

$$\mathbf{h}^{(m)}_{\mathbf{x}} = Cmb^{(ptcnn)}(\mathbf{h}^{(m-1)}_{\mathbf{x}}, \mathbf{g}^{(m)}_{\mathbf{x}}) = \mathbf{g}^{(m)}_{\mathbf{x}}, \tag{15e}$$

$$PTConv^{(\mathcal{P})}_{ptcnn}(\mathbf{x}) = Rdt^{(ptcnn)}(\mathbf{h}^{(M)}_{\mathbf{x}}) = [\mathbf{h}^{(M)}_{\mathbf{x}}; MLP_g(\mathbf{x})], \tag{15f}$$

Equation 15 describes how the point set encoder of PointCNN works. In Equation 15a, a multiple-layer perceptron $MLP^{(m)}_{\delta}(\cdot)$ individually lifts the spatial affinity $\mathbf{x}_i - \mathbf{x}$ for each neighbor $\mathbf{x}_i \in \mathcal{N}^{(m)}_{ptcnn}(\mathbf{x})$ into $\mathbf{h}^{(m)}_{\mathbf{x}_i,\delta} \in \mathbb{R}^{C_\delta}$, a C_δ dimensional embedding. Then in Equation 15b, $\Gamma_{\mathcal{N}^{(m)}_{ptcnn}(\mathbf{x})}(\mathbf{x}_i - \mathbf{x}) \in \mathbb{R}^{K \times L}$ represents a stack of the spatial affinity vector $\mathbf{x}_i - \mathbf{x} \in \mathbb{R}^L$ of all $\mathbf{x}_i \in \mathcal{N}^{(m)}_{ptcnn}(\mathbf{x})$ which results in a $K \times L$ matrix. The multi-layer perceptron $MLP^{(m)}(\cdot)$ converts this matrix into a $K \times K$ $\mathcal{X}$-transformation matrix - $\mathcal{X}^{(m)}$. Next, in Equation 15d, a point convolution operator $Conv^{(ptcnn)}(\kappa^{(m)}, \cdot)$ aggregates the concatenation $[\mathbf{h}^{(m)}_{\mathbf{x}_i,\delta}; \mathbf{h}^{(m-1)}_{\mathbf{x}_i}] \in \mathbb{R}^{C_\delta + d^{(m-1)}}$. $\mathcal{X}^{(m)}$ is used here to permute this $K \times (C_\delta + d^{(m-1)})$ matrix and has to be aware of the order of all $\mathbf{x}_i \in \mathcal{N}^{(m)}_{ptcnn}(\mathbf{x})$. In the final readout function (Equation 15f) another $MLP_g(\cdot)$ is used to directly lift $\mathbf{x}$ into a high dimensional embedding which is concatenated with $\mathbf{h}^{(M)}_{\mathbf{x}} \in \mathbb{R}^{d^{(M)}}$. This can be seen as an analogy of the skip-connection in traditional CNN. In Equation 15d, when $m = 1$, $\mathbf{h}^{(m-1)}_{\mathbf{x}_i} = \mathbf{h}^{(0)}_{\mathbf{x}_i}$ needs to be initialized. Given $p_i = (\mathbf{x}_i, \mathbf{v}_i)$ (See Definition 2.1), Li *et al.* (2018b) made $\mathbf{h}^{(0)}_{\mathbf{x}_i} = \mathbf{v}_i$. We also can choose $\mathbf{h}^{(0)}_{\mathbf{x}_i} = \mathbf{x}_i$.

Similar PointCNN layers are deployed in the point set decoder $PTDeConv^{(\mathcal{P})}_{ptcnn}(\mathbf{x})$ to form a Conv-DeConv like architecture. Similar $Conv^{(ptcnn)}(\kappa^{(m)}, \cdot)$ operator is used while the only difference is $PTDeConv^{(\mathcal{P})}_{ptcnn}(\mathbf{x})$ has more points but less feature channels in its output vs. its input, and $\{\mathcal{P}^{(m)}\}$ are forwarded from $PTConv^{(\mathcal{P})}_{ptcnn}(\mathbf{x})$.

Graph-Conv GAN. Valsesia *et al.* (2019) presented a graph convolution based

$Enc^{(\mathcal{P})}_{hieagg}(\mathbf{x})$ for 3D point cloud generation based on Generative Adversarial Network (GAN) (Goodfellow *et al.* 2014). We denote the location encoder of the model as $Enc^{(\mathcal{P})}_{gcgan}(\mathbf{x})$. The same Conv-DeConv idea is used here (See Definition 4.10) and $PTConv^{(\mathcal{P})}_{gcgan}(\mathbf{x})$ consists of M Point Conv layers each of which is composed of a sampling, a grouping, and an aggregation layer:

$$\mathbf{h}^{(0)}_{\mathbf{x}} = Enc(\mathbf{x}), \tag{16a}$$

$$\mathbf{g}^{(m)}_{\mathbf{x}} = Agg^{(gcgan)}_{\mathbf{x}_i \in \mathcal{N}^{(m)}_{knn}(\mathbf{x})}\{\mathbf{h}^{(m-1)}_{\mathbf{x}_i}\} = \sum_{\mathbf{x}_i \in \mathcal{N}^{(m)}_{knn}(\mathbf{x})} \frac{F^{(m)}(\mathbf{h}^{(m-1)}_{\mathbf{x}_i} - \mathbf{h}^{(m-1)}_{\mathbf{x}})\mathbf{h}^{(m-1)}_{\mathbf{x}_i}}{|\mathcal{N}^{(m)}_{knn}(\mathbf{x})|}, \tag{16b}$$

$$\mathbf{h}^{(m)}_{\mathbf{x}} = Cmb^{(gcgan)}(\mathbf{h}^{(m-1)}_{\mathbf{x}}, \mathbf{g}^{(m)}_{\mathbf{x}}) = \sigma\Big(\mathbf{g}^{(m)}_{\mathbf{x}} + \mathbf{W}^{(m)}\mathbf{h}^{(m-1)}_{\mathbf{x}} + \mathbf{b}^{(m)}\Big), \tag{16c}$$

$$PTConv^{(\mathcal{P})}_{gcgan}(\mathbf{x}) = Rdt^{(gcgan)}(\mathbf{h}^{(M)}_{\mathbf{x}}) = \mathbf{h}^{(M)}_{\mathbf{x}}. \tag{16d}$$

We denote the location embedding of $\mathbf{x}$ at the mth layer as $\mathbf{h}^{(m)}_{\mathbf{x}} \in \mathbb{R}^{d^{(m)}}$. $Enc(\mathbf{x})$ in Equation 16a can be any single point location encoder. $Agg^{(gcgan)}_{\mathbf{x}_i \in \mathcal{N}^{(m)}_{knn}(\mathbf{x})}\{\mathbf{h}^{(m-1)}_{\mathbf{x}_i}\} \in \mathbb{R}^{d^{(m)}}$ in Equation 16b uses a graph convolution operator to aggregate the neighborhood $\mathcal{N}^{(m)}_{knn}(\mathbf{x})$ at the mth layer. $\mathcal{N}^{(m)}_{knn}(\mathbf{x})$ is defined as $\mathbf{x}$'s K-th nearest neighbors - $\mathcal{N}^{(m)}_{knn}(\mathbf{x}) = KNN(\mathbf{x}, K, \mathcal{P}^{(m-1)})$. $\mathbf{h}^{(m-1)}_{\mathbf{x}_i} - \mathbf{h}^{(m-1)}_{\mathbf{x}} \in \mathbb{R}^{d^{(m-1)}}$ capture the spatial affinity between neighboring location $\mathbf{x}_i$ and $\mathbf{x}$. $F^{(m)}(\cdot)$ denotes a fully-connected network which regresses $\mathbf{h}^{(m-1)}_{\mathbf{x}_i} - \mathbf{h}^{(m-1)}_{\mathbf{x}}$ into a matrix $F^{(m)}(\mathbf{h}^{(m-1)}_{\mathbf{x}_i} - \mathbf{h}^{(m-1)}_{\mathbf{x}}) \in \mathbb{R}^{d^{(m)} \times d^{(m-1)}}$. Equation 16c shows how to combine the neighborhood feature $\mathbf{g}^{(m)}_{\mathbf{x}} \in \mathbb{R}^{d^{(m)}}$ with $\mathbf{x}$'s own feature from the previous layer $\mathbf{h}^{(m-1)}_{\mathbf{x}} \in \mathbb{R}^{d^{(m-1)}}$. $\mathbf{W}^{(m)} \in \mathbb{R}^{d^{(m)} \times d^{(m-1)}}$ and $\mathbf{b}^{(m)} \in \mathbb{R}^{d^{(m)}}$ are a learnable matrix and a bias vector respectively.

It is worth mentioning that the main difference among SSG, PointCNN, and GraphConv GAN is different aggregation layers used in their point set encoders (See Equation 14, 15, and 16). Figure 5 shows their shared Conv-DeConv architecture.

4.3. *Comparison among different models*

After introducing $Enc(\mathbf{x})$ and $Enc^{(\mathcal{P})}(\mathbf{x})$, it is worth to compare them from different aspects. First, we provide a general comparison between $Enc(\mathbf{x})$ and $Enc^{(\mathcal{P})}(\mathbf{x})$:

(1) $Enc(\mathbf{x})$ encodes $\mathbf{x}$ independently without considering its spatial context. In contrast, $Enc^{(\mathcal{P})}(\mathbf{x})$ jointly consider $\mathbf{x}$ and its neighbor $\mathcal{N}(\mathbf{x})$. $Enc^{(\mathcal{P})}(\mathbf{x})$ can be seen as a generalizaton of $Enc(\mathbf{x})$ in which any $Enc(\mathbf{x})$ can be used to compute $\mathbf{h}^{(0)}_{\mathbf{x}}$ (See Equation 7).

(2) When a new location $\mathbf{x}'$ is added to $\mathcal{P}$, $Enc(\mathbf{x}_i)$ is unaffected for all $\mathbf{x}_i \in \mathcal{P}$. In contrast, as for $Enc^{(\mathcal{P})}(\mathbf{x})$, for all $\mathbf{x}_i \in \{\mathbf{x}_i | \mathbf{x}_i \in \mathcal{P} \wedge \mathbf{x}' \in \mathcal{N}(\mathbf{x}_i)\}$, $Enc^{(\mathcal{P})}(\mathbf{x}_i)$ will be updated since $\mathcal{N}(\mathbf{x}_i)$ is modified. However, $Enc^{(\mathcal{P})}_{kernel}(\mathbf{x})$ is unaffected since $\mathcal{N}(\mathbf{x}) = \mathcal{Q}$ is the same for all $\mathbf{x} \in \mathcal{P}$ which is unchanged.

(3) $Enc(\mathbf{x})$ has a rather high inference speed, while the aggregation operator in $Enc^{(\mathcal{P})}(\mathbf{x})$ is time-consuming.

(4) Since $Enc^{(\mathcal{P})}(\mathbf{x})$ additionally considers $\mathcal{N}(\mathbf{x})$, it has richer features for model prediction and has a potential higher performance compared to $Enc(\mathbf{x})$.

We can see that both $Enc(\mathbf{x})$ and $Enc^{(\mathcal{P})}(\mathbf{x})$ have advantages and disadvantages. Although both of them are task-agnostic, they excel at different tasks and the model selection should depend on the current task:

(1) The first criterion is the input for a single model prediction - one location $\mathbf{x}$ (e.g., geo-aware image classification) or a whole point set $\mathcal{P}$ (e.g., point cloud segmentation). The former setup prefers $Enc(\mathbf{x})$ given its fast inference speed. The latter one prefers $Enc^{(\mathcal{P})}(\mathbf{x})$ since it can additionally captures spatial context information. Moreover, some tasks require producing a single embedding for the whole point set $\mathcal{P}$ or parts of it such as point cloud classification and objection recognition. For these tasks, $Enc^{(\mathcal{P})}(\mathbf{x})$ is the only choice.
(2) Another criterion is the preference between faster inference speed or higher prediction accuracy. A mobile application may prefer a faster inference speed in which $Enc(\mathbf{x})$ excels. An enterprise application might prefer higher prediction accuracy where $Enc^{(\mathcal{P})}(\mathbf{x})$ is preferred.

Next, we compare different sub-categories of location encoders from five different perspectives. The results shown in Table 1 will be discussed in detail as follows.

(1) L: In terms of the spatial dimension of $\mathbf{x}$ a location encoder can handle, almost all models can handle different L, e.g., $L = 2, 3$. Exceptions are GPS2Vec, *wrap*, and *theory*. GPS2Vec and *wrap* are specifically designed for GPS coordinates which can be uniquely identified by ϕ and λ. *theory* is designed only for 2D coordinates since it is inspired by neuroscience research about grid cells which are critical for self-motion integration and navigation of mammals in a 2D space.
(2) Parametric: As shown in Table 1, all models except *rbf* and adaptive kernel* are parametric models, which means their learnable parameters have a fixed size. *rbf* and adaptive kernel* can be either parametric or non-parametric models. Since $\mathbf{h}_{\mathbf{x}}^{(1)} \in \mathbb{R}^{|\mathcal{Q}|}$ (See Equation 8b, 8c), the size of the kernel center set $\mathcal{Q}$ decides the number of learnable parameters in $\mathbf{NN}(\cdot)$ of $Enc_{kernel}^{(\mathcal{P})}(\mathbf{x})$. If $\mathcal{Q} = \mathcal{P}$, then both *rbf* and adaptive kernel* become non-parametric models. Otherwise, if $\mathcal{Q}$ has a fixed size, both of them are parametric model.
(3) Multi-scale: Both $Enc_{sinmul}(\mathbf{x})$ and $Enc_{hieagg}^{(\mathcal{P})}(\mathbf{x})$ utilize multi-scale approaches. So they are better at capturing locations with non-uniform density or a mixture of distributions with different characteristics. However, they adopt different multi-scale approaches. The former designs multi-scale representations based on $PE(\mathbf{x})$ which uses sinusoidal functions with different frequencies, while the latter aggregates the neighborhood of $\mathbf{x}$ in a hierarchical manner. From a practical perspective, many studies have shown that multi-scale location encoders can outperform single-scale models and models without scale related parameters on various tasks. For example, Qi *et al.* (2017b) showed that PointNet++ can outperform PointNet on both point cloud classification and segmentation task. Mai *et al.* (2020b) showed that multi-scale models (*theory* and *grid*) can outperform *tile*, *direct*, *wrap*, and *rbf* on both POI type classification task and geo-aware image classification task. Mai *et al.* (2020a) showed that *theory* can outperform *direct* on geographic question answering task.
(4) Distance Prevervation: In terms of the question whether a location encoder is distance preserved (Property 2.1), we mainly consider them from a empirical perspective, e.g., whether the response map of a location encoder shows a spatial continuity pattern or a spatial heterogeneity pattern. The former implies that this model is distance preserved, while the latter indicates otherwise. For example, Mai *et al.* (2020b) systematically compared the response maps of different location encoders such as *tile*, *direct*, *wrap*, *theory*, *grid*, and *rbf*, after training on the

POI type classification task (See Figure 2 in Mai *et al.* (2020b)). The results show that *direct*, *wrap*, *theory*, and *rbf* are distance preserved while *tile* and *direct* are not. Moreover, as we discussed in Section 4.1.4, *theory* also have a theoretical proof for distance preservation. So we put "Yes$^+$" in Table 1. In terms of other location encoders, since no experiment or theoretical proof has been done, whether they have this property is unknown.

(5) Direction Awareness: A similar logic is used here. The response maps produced by Mai *et al.* (2020b) showed that *direct* and *theory* is aware of direction information while *rbf* is not. No conclusion can be drawn for other models.

(6) SAGAT is a bit different. Its properties also depend on the property of the used $Enc(\cdot)$ in Equation 11f. SAGAT can be a parametric or non-parametric (e.g., use *rbf* as $Enc(\cdot)$) model. It becomes a multi-scale approach if $Enc(\cdot)$ uses a multi-scale representation. Whether SAGAT has the distance preservation and direction awareness property also depends on the used $Enc(\cdot)$.

5. Applying location encoding to different types of spatial data

Location encoders can be directly utilized on multiple point set-based GeoAI tasks such as geo-aware image classification, POI type classification, and point cloud segmentation. However, there are many other tasks that are defined on other types of spatial data such as polylines, polygons, and graphs (networks). This section discusses the potential of location encoders to model these types of spatial data.

5.1. *Polyline*

The location-to-polyline relation can be seen as an analogy of the word-to-sentence relation. In NLP, a sentence, as an ordered sequence of words, can be encoded by different sequential neural nets such as different recurrent neural networks (RNN) (Hochreiter and Schmidhuber 1997, Cho *et al.* 2014) and Transformer (Vaswani *et al.* 2017). Their idea is to feed the embedding of each word token into a sequential model at each time step to encode the whole word sequence as one single hidden state or a sequence of hidden states.

Similarly, we can encode a polyline as an ordered sequence of locations, by using these sequential neural network models. At each time step, we will encode the current location into a location embedding and feed it into the sequential model. In fact, several recent work about human mobility directly follow this idea. Xu *et al.* (2018) utilized a *direct* location encoder to represent each trajectory point into a location embedding. Then a trajectory, represented as a sequence of location embeddings, are encoded by an LSTM for pedestrian trajectory prediction. Similarly, Rao *et al.* (2020) proposed an LSTM-TrajGAN framework to generate privacy-preserving synthetic trajectory data in which each trajectory point was encoded by a *direct* location encoder.

5.2. *Polygon*

Encoding polygon geometries into the embedding space is a logical next step. It is very useful for several geospatial tasks which require comparing polygon geometries such as geographic entity alignment (Trisedya *et al.* 2019), spatial topological reasoning (Regalia *et al.* 2019), and geographic question answering (Mai *et al.* 2019b, 2020a, 2021).

However, unlike a polyline, which can be represented by an ordered sequence of locations, a polygon should be represented by all locations within it. The topological relationships between any location $\mathbf{x}$ and a polygon should be preserved after the polygon encoding process. In other words, a polygon encoder should be topology aware. As far as we know, polygon encoding is still an ongoing research problem that does not have satisfactory solutions. Mai *et al.* (2020a) presented a geographic entity bounding box encoding model as the first step towards polygon encoding by uniformly sampling a location from within the bounding box of a geographic entity and feeding it to a location encoder. Despite its innovativeness, this model still cannot handle fine-grained polygon geometries. Yan *et al.* (2021) proposed a graph convolutional autoencoder (GCAE) which can encode simple polygons, i.e., polygons that do not intersect themselves and have no holes. GCAE converts the exterior of a simple polygon into a graph and then encodes this graph into an embedding space. The shortcoming of GCAE is that it cannot handle polygons with holes and multipolygons. Moreover, it cannot preserve the topology information. So one interesting future research direction is developing a topology-aware polygon encoder which can handle both simple and complex polygons.

5.3. *Graph*

Graph (or network) is also an important spatial data format used in multiple geospatial data sets such as transportation networks (Li *et al.* 2018a, Cai *et al.* 2020), spatial social networks (Andris 2016), and geographic knowledge graph (GeoKG) (Mai *et al.* 2020a). A graph can be defined as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ and $\mathcal{E}$ are the set of nodes and edges in this graph. In the geospatial domain, each node $e \in \mathcal{V}$ or a subset of nodes in $\mathcal{V}$ is associated with a location $\mathbf{x}$ [5] such as the sensor locations in a sensor network, users' locations in a spatial social network, or locations of geographic entities in a GeoKG. We further call this kind of graph a *spatially embedded graph*.

The early practice to encode spatially embedded graphs is to treat them as normal non-spatial graphs and use some existing GNN models or (knowledge) graph embedding models (Grover and Leskovec 2016, Bordes *et al.* 2013, Trouillon *et al.* 2017). In order to add the spatial information as additional features without significantly modifying the existing architectures, we can modify the node encoder by using one $Enc(\mathbf{x})$ in Table 1 as the node encoder or one component of it while keeping other components unchanged. The model can be trained with the same loss function. Mai *et al.* (2020a) adopted exactly this practice and developed a spatially-explicit knowledge graph embedding model. Similar ideas can be applied to other spatially embedded graphs.

Interestingly, other than the normal graph data, many pioneer research applied GNN models to a point set through a *point-set-to-graph* conversion. They first converted point set $\mathcal{P}$ into a graph based on spatial relations, e.g., a k-th nearest neighbor spatial graph, in which nodes indicate points while edges are associated with pairwise distance based weights. After this conversion, a GNN model is applied on this graph so that node attribute prediction can be done based on not only the nodes' own features but also their spatial context. Many GeoAI research has adopted this practice to tackle different tasks including air quality forecasting (Lin *et al.* 2018), place characteristics prediction (Zhu *et al.* 2020), GeoNames entity embedding learning (Kejriwal and Szekely 2017), and different spatial interpolation problems (Appleby *et al.* 2020, Wu *et al.* 2020a). We argue that *this kind of distance weighted graph method is insufficient to capture the relative spatial relations*, since it necessarily forfeits information about the spatial layout

[5]Each node can associate with more complex geometries, and one single location is a simplification.

of points. Some important spatial information is lost such as the direction relations which are important for certain tasks when isotropic assumption is not held. Instead, we advocate the idea of using any $Enc^{(\mathcal{P})}(\mathbf{x})$ discussed in Section 4.2 for these tasks since $Enc^{(\mathcal{P})}(\mathbf{x})$ is better at capturing spatial relations among locations.

5.4. *Raster*

Convolutional Nerual Networks (CNNs) (Lecun and Bengio 1995) are at the core of many highly successful models in manipulating raster data such as image classification, image generation, and image understanding. This great success is due to the ability of the convolution operation to exploit the principles of locality, stationarity, and compositionality. Locality is due to the local connectivity, stationarity is owed to shift-invariance, and compositionality stems from the multi-resolution structure of the raster data (Bronstein *et al.* 2017). The number of learnable parameters is greatly reduced because of its feature locality and weight sharing across the data domain (Valsesia *et al.* 2019). Due to the success of location encoding on vector data, it is particularly interesting to think about the questions *how we can apply location encoding techniques on rasters* and *what the benefits are.*

Interestingly, with increasing popularity of the Transformer (Vaswani *et al.* 2017) architecture, several efforts have been made to replace CNN with a Transformer-like architecture for raster-based tasks. The idea is that instead of using CNN kernels, we first encode the pixel features as well as pixel locations into the embedding space with a location-encoder-like architecture, so-called *pixel position encoding*, and then a self-attention is applied on top of these pixel embeddings for different vision tasks. However, one problem with this approach is that this per-pixel based self-attention has a very high computational cost. One solution, among others, proposed by Vision Transformer (ViT) (Dosovitskiy *et al.* 2021) uses a per-image-patch (instead of per-pixel) self-attention which significantly lowers the computational cost. Dosovitskiy *et al.* (2021) showed that ViT can outperform traditional CNN-based models on several image classification benchmarks. However, the patch size becomes an important hyperparameter which will significantly affect model performance. Applying location encoders on raster data is a very new research direction. Existing work mainly focuses on encoding the positions of pixels on an image. When a pixel represents an area on the earth's surface (e.g., pixels in a satellite image), it is potentially very beneficial to encode pixel's geographic locations rather than its image positions. The geo-locations can serve as a channel, which transfers knowledge learnt from large quantities of unlabeled data (geographic data, geo-tagged image, or geo-tagged text) to the supervised learning tasks.

6. Conclusion and Vision for Future Work

In this work, we formulate location encoding as an inductive learning based, task agnostic encoding technique for geographic locations. A formal definition of location encoding is provided, and two expected properties – distance preservation and direction awareness – are discussed from the perspective of GIScience. We illustrate the necessity of location encoding for GeoAI from a statistical machine learning perspective. A general classification framework has been provided to understand the current landscape of location encoding research (See Table 1). We classify the existing location encoders into two categories: single point location encoder $Enc(\mathbf{x})$ and aggregation location encoder $Enc^{(\mathcal{P})}(\mathbf{x})$. For each category, we unify the location encoders into the same formulation framework (See Equations 1 and 7). Different location encoders are also compared based on various characteristics. Finally, we demonstrate the possible usage of location encoding for different types of spatial data.

There are several interesting future research directions of location encoding:

(1) **Region representation learning**: As we discussed in Section 5.2, there is no satisfactory solution for polygon encoding (so called region representation learning), which will be very useful in various tasks such as geographic entity alignment and topological relation reasoning. How to design a topology-aware polygon encoder which can handle simple polygons, polygon with holes, and multipolygons simultaneously is an interesting future research direction.

(2) **Spatiotemporal point encoding**: All the methods discussed so far are focused on location information whereas the temporal aspect of geospatial data is also very important. Several important related questions are: 1) How to utilize temporal information in GeoAI models? 2) Can we encode temporal information in a similar manner as spatial information? 3) What are the important properties we need to preserve when doing temporal encoding? 4) How to combine temporal encoding and location encoding in a single framework? As for event sequences that happen synchronously (Kazemi *et al.* 2019), i.e., sampled at regular intervals, the temporal information can be modeled implicitly by RNNs, or fed in RNNs as another input dimension after transforming time into handcrafted features (Du *et al.* 2016, Li *et al.* 2017, Rao *et al.* 2020). Instead of using handcrafted temporal features, recent work proposed to encode time as learnable vector representations such as Time2Vec (Kazemi *et al.* 2019) and Cai *et al.* (2020). These temporal encoders are expected to preserve important properties such as periodicity, temporal continuity, invariance to time rescaling, and so on. However, there are no systematic comparison studies among these temporal encoding approaches. As for combining location and temporal encoding, one obvious way is to add temporal information as an additional dimension of the location features. Mac Aodha *et al.* (2019) adopted this practice by adding time as an additional feature of $PE_{wrap}(\mathbf{x})$ in Equation 2. This leaded to a small performance improvement (0.25%-1.37%). However, they failed to consider those important properties of time mentioned above. Future research is needed to study the pros and cons of different temporal encoding approaches and how to combine it with location encoding.

(3) **Spherical location encoding**: As we discussed in Section 5.4, currently, there are no existing location encoders which can preserve spherical surface distance. When we are dealing with large-scale geospatial data sets (e.g., global SST data, species occurrences all over the world) in which the map distortion problem is no longer negligible, a spherical-aware location encoder is required which enable us to directly *calculate on a round planet* (Chrisman 2017).

(4) **Unsupervised learning for location encoding**: Most of the location encoders listed in Table 1 are trained in a supervised learning fashion which prohibits the application of the trained location embedding on other tasks. In contrast, text encoding methods, e.g., BERT, are trained in an unsupervised manner from numerous unlabeled data, and the pretrained model can be utilized in different downstream tasks (Devlin *et al.* 2018). How to design an unsupervised learning framework for location encoding is a very attractive research direction. Recently, multiple point cloud generative models have been proposed such as r-GAN/l-GAN (Achlioptas *et al.* 2018), Graph-Conv GAN (Valsesia *et al.* 2019), tree-GAN (Shu *et al.* 2019), PointFlow (Yang *et al.* 2019), and Generative PointNet (Xie *et al.*

2021). Their objective is to reconstruct given point clouds. This presents one possible unsupervised learning framework of location encoding for unmarked points (points without attributes). Another interesting idea is unsupervised learning of the spatial distribution of marked points (points with attributes).

Data and Codes Availability Statement
There is no code implementation for this review paper.

Acknowledgements

We would like to thank Dr. Fei Du for his comments on the differences between location encoding and geohash. We also want to Thank Prof. Stefano Ermon for his suggestions on unsupervised learning for location encoding. We would like to thank the three anonymous reviewers for their thoughtful comments and suggestions.

This work is mainly funded by the National Science Foundation under Grant No. 2033521 A1 – KnowWhereGraph: Enriching and Linking Cross-Domain Knowledge Graphs using Spatially-Explicit AI Technologies. Gengchen Mai acknowledges the support by UCSB Schmidt Summer Research Accelerator Award, Microsoft AI for Earth Grant, and IARPA SMART 2020-0072. Yingjie Hu acknowledges support by the National Science Foundation under Grant No. BCS-2117771. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Notes on contributors

Gengchen Mai is a Postdoctoral Scholar at the Stanford AI Lab, Department of Computer Science, Stanford University. He is also affiliated with the Stanford Sustainability and AI Lab. He obtained his Ph.D. degree in Geography from the Department of Geography, University of California, Santa Barbara in 2021 and B.S. degree in GIS from Wuhan University in 2015. His research mainly focuses on Spatially-Explicit Machine Learning, Geospatial Knowledge Graph, and Geographic Question Answering. The first draft of this manuscript was submitted when Gengchen was a Ph.D. student at the Space and Time for Knowledge Organization Lab, Department of Geography, UC Santa Barbara.

Krzysztof Janowicz is a Professor for Geoinformatics at the University of California, Santa Barbara and director of the Center for Spatial Studies. His research focuses on how humans conceptualize the space around them based on their behavior, focusing particularly on regional and cultural differences with the ultimate goal of assisting machines to better understand the information needs of an increasingly diverse user base. Janowicz's expertise is in knowledge representation and reasoning as they apply to spatial and geographic data, e.g., in the form of knowledge graphs.

Yingjie Hu is an Assistant Professor in GIScience at the Department of Geography, University at Buffalo, The State University of New York. He holds a Ph.D. in Geography from the University of California, Santa Barbara. His main research interests include Geospatial Artificial Intelligence and Spatial Data Science.

Song Gao is an Assistant Professor in GIScience at the Department of Geography, University of Wisconsin-Madison. He holds a Ph.D. in Geography at the University of California, Santa Barbara. His main research interests include Place-Based GIS, Geospatial Data Science, and GeoAI approaches to Human Mobility and Social Sensing.

Bo Yan received his bachelor's degree in GIS from Wuhan University and his PhD in Geography from the University of California Santa Barbara. Dr Yan's research area includes Geospatial Knowledge Graphs and machine learning in the geospatial domain.

Rui Zhu is a Postdoctoral Scholar at the Center for Spatial Studies, University of California, Santa Barbara. He obtained his Ph.D. in Geography from the same university in 2020; his M.S. degree was in Information Science from the University of Pittsburgh. Zhu's research interests include geospatial semantics, spatial statistics, as well as their broader interactions in geospatial artificial intelligence (GeoAI).

Ling Cai is a fourth-year PhD student at the Space and Time for Knowledge Organization Lab, Department of Geography, University of California, Santa Barbara. She obtained her M.S. degree in Geographical Information Science from Chinese Academy of Sciences and B.S. degree from Wuhan University. Her research interests include qualitative spatial temporal reasoning, temporal knowledge graph, neuro-symbolic AI, and urban computing.

Ni Lao is currently a research scientist at Google. He holds a Ph.D. degree in Computer Science from the Language Technologies Institute, School of Computer Science at Carnegie Mellon University. He was the Chief scientist and co-founder of Mosaix.ai, a voice search AI startup. He is an expert in Machine Learning (ML), Knowledge Graph (KG) and Natural Language Understanding (NLU). He is known for his work on large scale inference and learning on KGs. This paper was done when Ni worked at Mosaix.ai.

References

Achlioptas, P., *et al.*, 2018. Learning representations and generative models for 3d point clouds. *In*: *International Conference on Machine Learning.* 40–49.

Agarwal, A., *et al.*, 2015. A boon in studies of cognitive functions: brain and grid cells. *Current Science*, 108 (12), 2142–2144.

Andris, C., 2016. Integrating social network data into gisystems. *International Journal of Geographical Information Science*, 30 (10), 2009–2031.

Appleby, G., Liu, L., and Liu, L.P., 2020. Kriging convolutional networks. *In*: *AAAI 2020.*

Battaglia, P.W., *et al.*, 2018. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261.*

Berg, T., *et al.*, 2014. Birdsnap: Large-scale fine-grained visual categorization of birds. *In*: *CVPR 2014.* 2011–2018.

Blair, H.T., Welday, A.C., and Zhang, K., 2007. Scale-invariant memory representations emerge from moire interference between grid fields that produce theta oscillations: a computational model. *Journal of Neuroscience*, 27 (12), 3211–3229.

Bordes, A., *et al.*, 2013. Translating embeddings for modeling multi-relational data. *In*: *Advances in neural information processing systems.* 2787–2795.

Bronstein, M.M., *et al.*, 2017. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34 (4), 18–42.

Cai, L., *et al.*, 2020. Traffic transformer: Capturing the continuity and periodicity of time series for traffic forecasting. *Transactions in GIS.*

Cai, L., *et al.*, 2019. TransGCN: Coupling transformation assumptions with graph convolutional networks for link prediction. *In*: *ACM K-CAP 2019.* 131–138.

Cho, K., *et al.*, 2014. Learning phrase representations using rnn encoder–decoder for statistical machine translation. *In*: *EMNLP 2014.* 1724–1734.

Chrisman, N.R., 2017. Calculating on a round planet. *International Journal of Geographical Information Science*, 31 (4), 637–657.

Chu, G., *et al.*, 2019. Geo-aware networks for fine-grained recognition. *In*: *Proceedings of the*

IEEE International Conference on Computer Vision Workshops. 0–0.

Couclelis, H., 1986. Artificial intelligence in geography: Conjectures on the shape of things to come. *The professional geographer*, 38 (1), 1–11.

DeLozier, G., Baldridge, J., and London, L., 2015. Gazetteer-independent toponym resolution using geographic word profiles. *In*: *AAAI 2015*. 2382–2388.

Devlin, J., *et al.*, 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dosovitskiy, A., *et al.*, 2021. An image is worth 16x16 words: Transformers for image recognition at scale. *In*: *ICLR 2021*.

Du, N., *et al.*, 2016. Recurrent marked temporal point processes: Embedding event history to vector. *In*: *ACM SIGKDD 2016*. 1555–1564.

Feng, S., *et al.*, 2017. POI2Vec: Geographical latent representation for predicting future visitors. *In*: *AAAI 2017*.

Fotheringham, A.S. and Wong, D.W., 1991. The modifiable areal unit problem in multivariate statistical analysis. *Environment and planning A*, 23 (7), 1025–1044.

Gao, R., *et al.*, 2019. Learning grid cells as vector representation of self-position coupled with matrix representation of self-motion. *In*: *ICLR 2019*.

Gao, R., *et al.*, 2020. A representational model of grid cells based on matrix lie algebras. *arXiv preprint arXiv:2006.10259*.

Gilmer, J., *et al.*, 2017. Neural message passing for quantum chemistry. *In*: *ICML 2017*.

Goodfellow, I., *et al.*, 2014. Generative adversarial nets. *In*: *Advances in Neural Information Processing Systems*. 2672–2680.

Grover, A. and Leskovec, J., 2016. node2vec: Scalable feature learning for networks. *In*: *ACM SIGKDD 2016*. 855–864.

Hafting, T., *et al.*, 2005. Microstructure of a spatial map in the entorhinal cortex. *Nature*, 436 (7052), 801–806.

Hamilton, W., Ying, Z., and Leskovec, J., 2017. Inductive representation learning on large graphs. *In*: *Advances in neural information processing systems*. 1024–1034.

Hastie, T., Tibshirani, R., and Friedman, J., 2009. *The elements of statistical learning: Data mining, inference, and prediction*. Springer.

Hochreiter, S. and Schmidhuber, J., 1997. Long short-term memory. *Neural computation*, 9 (8), 1735–1780.

Izbicki, M., Papalexakis, E.E., and Tsotras, V.J., 2019. Exploiting the earth's spherical geometry to geolocate images. *In*: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 3–19.

Janowicz, K., *et al.*, 2020. GeoAI: Spatially explicit artificial intelligence techniques for geographic knowledge discovery and beyond. *International Journal of Geographic Information Science*.

Kazemi, S.M., *et al.*, 2019. Time2vec: Learning a vector representation of time. *arXiv preprint arXiv:1907.05321*.

Kejriwal, M. and Szekely, P., 2017. Neural embeddings for populated geonames locations. *In*: *International Semantic Web Conference*. Springer, 139–146.

Killian, N.J., Jutras, M.J., and Buffalo, E.A., 2012. A map of visual space in the primate entorhinal cortex. *Nature*, 491 (7426), 761–764.

Kipf, T.N. and Welling, M., 2017. Semi-supervised classification with graph convolutional networks. *In*: *ICLR 2017*.

Krizhevsky, A., Sutskever, I., and Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. *In*: *Advances in neural information processing systems*. 1097–1105.

Kulkarni, S., *et al.*, 2020. Spatial language representation with multi-level geocoding. *arXiv preprint arXiv:2008.09236*.

Lechner, A.M., *et al.*, 2012. Investigating species–environment relationships at multiple scales: Differentiating between intrinsic scale and the modifiable areal unit problem. *Ecological Complexity*, 11, 91–102.

Lecun, Y. and Bengio, Y., 1995. *Convolutional networks for images, speech and time series.* The MIT Press, 255–258.

Li, W. and Hsu, C.Y., 2020. Automated terrain feature identification from remote sensing imagery: a deep learning approach. *International Journal of Geographical Information Science*, 34 (4), 637–660.

Li, Y., *et al.*, 2018a. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. *In*: *ICLR 2018.*

Li, Y., Du, N., and Bengio, S., 2017. Time-dependent representation for neural event sequence prediction. *arXiv preprint arXiv:1708.00065.*

Li, Y., *et al.*, 2018b. PointCNN: Convolution on x-transformed points. *In*: *Advances in neural information processing systems.* 820–830.

Li, Y., *et al.*, 2016. Gated graph sequence neural networks. *In*: *ICLR 2016.*

Lin, Y., *et al.*, 2018. Exploiting spatiotemporal patterns for accurate air quality forecasting using deep learning. *In*: *ACM SIGSPATIAL 2018.* 359–368.

Mac Aodha, O., Cole, E., and Perona, P., 2019. Presence-only geographical priors for fine-grained image classification. *In*: *ICCV 2019.* 9596–9606.

Mai, G., *et al.*, 2020a. SE-KGE: A location-aware knowledge graph embedding model for geographic question answering and spatial semantic lifting. *Transactions in GIS*, 24, 623–655.

Mai, G., *et al.*, 2018. ADCN: An anisotropic density-based clustering algorithm for discovering spatial point patterns with noise. *Transactions in GIS*, 22 (1), 348–369.

Mai, G., *et al.*, 2019a. Contextual graph attention for answering logical queries over incomplete knowledge graphs. *In*: *ACM K-CAP 2019.* 171–178.

Mai, G., *et al.*, 2020b. Multi-scale representation learning for spatial feature distributions using grid cells. *In*: *ICLR 2020.*

Mai, G., *et al.*, 2021. Geographic Question Answering: Challenges, Uniqueness, Classification, and Future Directions. *AGILE 2021: GIScience Series*, 2, 1–21.

Mai, G., *et al.*, 2019b. Relaxing unanswerable geographic questions using a spatially explicit knowledge graph embedding model. *In*: *AGILE 2019.* Springer, 21–39.

Maturana, D. and Scherer, S., 2015. Voxnet: A 3d convolutional neural network for real-time object recognition. *In*: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* IEEE, 922–928.

Mirowski, P., *et al.*, 2018. Learning to navigate in cities without a map. *In*: *Advances in Neural Information Processing Systems.* 2419–2430.

Noh, H., Hong, S., and Han, B., 2015. Learning deconvolution network for semantic segmentation. *In*: *Proceedings of the IEEE international conference on computer vision.* 1520–1528.

Openshaw, S., 1981. The modifiable areal unit problem. *Quantitative geography: A British view*, 60–69.

Openshaw, S. and Openshaw, C., 1997. *Artificial intelligence in geography.* John Wiley & Sons, Inc.

Qi, C.R., *et al.*, 2017a. PointNet: Deep learning on point sets for 3d classification and segmentation. *In*: *CVPR 2017.* 652–660.

Qi, C.R., *et al.*, 2016. Volumetric and multi-view cnns for object classification on 3d data. *In*: *CVPR 2016.* 5648–5656.

Qi, C.R., *et al.*, 2017b. PointNet++: Deep hierarchical feature learning on point sets in a metric space. *In*: *Advances in neural information processing systems.* 5099–5108.

Qiu, P., *et al.*, 2019. Knowledge embedding with geospatial distance restriction for geographic knowledge graph completion. *ISPRS International Journal of Geo-Information*, 8 (6), 254.

Rao, J., *et al.*, 2020. LSTM-TrajGAN: A deep learning approach to trajectory privacy protection. *In*: *GIScience 2020.* 12:1–12:17.

Regalia, B., Janowicz, K., and McKenzie, G., 2019. Computing and querying strict, approximate, and metrically refined topological relations in linked geographic data. *Transactions in GIS*, 23 (3), 601–619.

Ronneberger, O., Fischer, P., and Brox, T., 2015. U-net: Convolutional networks for biomedical image segmentation. *In*: *International Conference on Medical image computing and computer-*

assisted intervention. Springer, 234–241.

Russell, S. and Norvig, P., 2015. *Artificial Intelligence: a modern approach.* Pearson.

Schlichtkrull, M., *et al.*, 2018. Modeling relational data with graph convolutional networks. *In*: *European Semantic Web Conference.* Springer, 593–607.

Shu, D.W., Park, S.W., and Kwon, J., 2019. 3d point cloud generative adversarial network based on tree structured graph convolutions. *In*: *ICCV 2019.* 3859–3868.

Smith, T.R., 1984. Artificial intelligence and its applicability to geographical problem solving. *The Professional Geographer*, 36 (2), 147–158.

Su, H., *et al.*, 2015. Multi-view convolutional neural networks for 3d shape recognition. *In*: *ICCV 2015.* 945–953.

Tang, K., *et al.*, 2015. Improving image classification with location context. *In*: *ICCV 2015.* 1008–1016.

Te Stroet, C.B. and Snepvangers, J.J., 2005. Mapping curvilinear structures with local anisotropy kriging. *Mathematical geology*, 37 (6), 635–649.

Tobler, W.R., 1970. A computer movie simulating urban growth in the detroit region. *Economic geography*, 46 (sup1), 234–240.

Trisedya, B.D., Qi, J., and Zhang, R., 2019. Entity alignment between knowledge graphs using attribute embeddings. *In*: *AAAI 2019.* vol. 33, 297–304.

Trouillon, T., *et al.*, 2017. Knowledge graph completion via complex tensor factorization. *The Journal of Machine Learning Research*, 18 (1), 4735–4772.

Valsesia, D., Fracastoro, G., and Magli, E., 2019. Learning localized generative models for 3d point clouds via graph convolution. *In*: *ICLR 2019.*

Vapnik, V., 2013. *The nature of statistical learning theory.* Springer science & business media.

Vaswani, A., *et al.*, 2017. Attention is all you need. *In*: *Advances in neural information processing systems.* 5998–6008.

Veličković, P., *et al.*, 2018. Graph attention networks. *In*: *ICLR 2018.*

Wang, J., Hu, Y., and Joseph, K., 2020. Neurotpr: A neuro-net toponym recognition model for extracting locations from social media messages. *Transactions in GIS.*

Wang, Y., *et al.*, 2019. Dynamic Graph CNN for learning on point clouds. *Acm Transactions On Graphics (tog)*, 38 (5), 1–12.

Weyand, T., Kostrikov, I., and Philbin, J., 2016. Planet-photo geolocation with convolutional neural networks. *In*: *European Conference on Computer Vision.* Springer, 37–55.

Wu, Y., *et al.*, 2020a. Inductive graph neural networks for spatiotemporal kriging. *arXiv preprint arXiv:2006.07527.*

Wu, Z., *et al.*, 2020b. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems.*

Xie, J., *et al.*, 2021. Generative pointnet: Deep energy-based learning on unordered point sets for 3d generation, reconstruction and classification. *In*: *CVPR 2021.* 14976–14985.

Xu, K., *et al.*, 2019. How powerful are graph neural networks? *In*: *ICLR 2019.*

Xu, Y., Piao, Z., and Gao, S., 2018. Encoding crowd interaction with deep neural network for pedestrian trajectory prediction. *In*: *CVPR 2018.* 5275–5284.

Yan, B., *et al.*, 2019. A spatially explicit reinforcement learning model for geographic knowledge graph summarization. *Transactions in GIS*, 23 (3), 620–640.

Yan, X., *et al.*, 2021. Graph convolutional autoencoder model for the shape coding and cognition of buildings in maps. *International Journal of Geographical Information Science*, 35 (3), 490–512.

Yang, G., *et al.*, 2019. Pointflow: 3d point cloud generation with continuous normalizing flows. *In*: *ICCV 2019.* 4541–4550.

Yin, Y., *et al.*, 2019. GPS2Vec: Towards generating worldwide gps embeddings. *In*: *ACM SIGSPATIAL 2019.* 416–419.

Zaheer, M., *et al.*, 2017. Deep sets. *In*: *Advances in neural information processing systems.* 3391–3401.

Zhong, E.D., *et al.*, 2020. Reconstructing continuous distributions of 3d protein structure from cryo-em images. *In*: *ICLR 2020.*

Zhong, L., Hu, L., and Zhou, H., 2019. Deep learning based multi-temporal crop classification. *Remote sensing of environment*, 221, 430–443.

Zhou, Y. and Tuzel, O., 2018. Voxelnet: End-to-end learning for point cloud based 3d object detection. *In*: *CVPR 2018*. 4490–4499.

Zhu, D., *et al.*, 2020. Understanding place characteristics in geographic contexts through graph convolutional neural networks. *Annals of the American Association of Geographers*, 110 (2), 408–420.

Zhu, M., *et al.*, 2019a. Location2Vec: a situation-aware representation for visual exploration of urban locations. *IEEE Transactions on Intelligent Transportation Systems*, 20 (10), 3981–3990.

Zhu, R., Janowicz, K., and Mai, G., 2019b. Making direction a first-class citizen of tobler's first law of geography. *Transactions in GIS*, 23 (3), 398–416.