Kriging Convolutional Networks

Gabriel Appleby,[*] **Linfeng Liu,**[*] **Li-Ping Liu**
Department of Computer Science, Tufts University
{Gabriel.Appleby, Linfeng.Liu, Liping.Liu}@tufts.edu

Abstract

Spatial interpolation is a class of estimation problems where locations with known values are used to estimate values at other locations, with an emphasis on harnessing spatial locality and trends. Traditional kriging methods have strong Gaussian assumptions, and as a result, often fail to capture complexities within the data. Inspired by the recent progress of graph neural networks, we introduce Kriging Convolutional Networks (KCN), a method of combining advantages of Graph Neural Networks (GNN) and kriging. Compared to standard GNNs, KCNs make direct use of neighboring observations when generating predictions. KCNs also contain the kriging method as a specific configuration. Empirically, we show that this model outperforms GNNs and kriging in several applications.

Spatial data is ubiquitous in a wide variety of fields such as ecology (Fink et al. 2010), economics (Gao and Liu 2014), and meteorology (Xingjian et al. 2015). A common task within these fields is to estimate values at target locations from nearby known values. Improving these estimations should provide clear benefits for these applications. Estimation techniques tailored to spatial data must leverage the fact that every data point is associated with a location. Most importantly, these techniques should be able to capture the spatial correlation among these locations.

In many fields, the most prevalent method for spatial data modeling is kriging (Cressie 1991). The fundamental assumption of kriging is that observations at locations are from an underlying Gaussian process. After estimating the *variogram*, which is essentially the strength of spatial correlations between data points, kriging uses a *linear interpolation of observed values* to predict the value at a new location. The kriging prediction is the best linear unbiased estimator for spatial points given its Gaussian assumption. However, this assumption is quite constrictive, as data in many applications are not from a Gaussian distribution. For example, we will show in our experiments that this assumption leads to poor performance when estimating integer counts that contain a significant fraction of zeros.

Researchers also use flexible machine learning algorithms (Hengl et al. 2018) for spatial data modeling. Given the huge success of GNNs and the similarity between spatial data and graph data, researchers have started to apply Graph Neural Networks (GNN) (Wu et al. 2019) to spatial data (Li et al. 2017; Yu, Yin, and Zhu 2017; Zhu and Liu 2018; Yan et al. 2019). GNNs were first developed for explicit graph data, but can model any data that can be transformed into a graph either by their spatial vicinity or their physical connections (e.g. routes). The main idea is to propagate information along graph edges, so graph nodes can share information during the learning process. GNNs are relatively generic, and can find nonlinear relationships between the inputs, hidden layers, and neighborhood information of each node. By design, GNNs are more flexible than kriging.

However, kriging has an advantage over GNNs: kriging directly uses observed training labels to predict the label of a new data point. In comparison, there is no straightforward way to feed training labels as input to a GNN. It is not feasible to directly feed training labels as part of the input because the GNN will directly output the given label of a training data point and learn nothing. Furthermore, spatial data modeling requires inductive learning – the model needs to be able to make predictions for new locations that are not in the graph formed by training data. While kriging is intrinsically inductive, only a few GNNs such as GraphSAGE (Hamilton, Ying, and Leskovec 2017) can work inductively.

Inspired by these two observations, we develop a new model, the Kriging Convolutional Network (KCN), as an improvement to GNNs. The KCN is still a type of GNN. However, it does not form a single large graph over all data points. Every time a KCN fits the label of a data point (call it the *center*), it forms a small graph over the center and its neighboring *training* data points. These neighbors are the K nearest neighbors according to a distance metric. In the input to the KCN, we hide the label of the center node. The input consists of feature vectors for all nodes in the graph ($K+1$ nodes), as well as the labels of the *neighbors*. The KCN also needs the adjacency matrix of the graph, which is defined to be the spatial kernel matrix or a normalized version of that. The target value of the KCN is the label of the center node.

[*]Equal contribution. The first two authors are arranged by the alphabet order

We iterate over all of the training data, treating each node as the center to train the KCN model. The KCN uses the same structure to predict the label of a new data point.

The KCN combines the best parts of both models. In comparison to the GNN, it is able to directly leverage training labels in prediction, and no re-training is necessary when new data points are introduced. In contrast to kriging, the KCN is more versatile. On a large dataset where overfiting is not an issue, the KCN has a clear advantage over kriging. Even though the KCN's underlying mechanisms are very different from kriging, our theoretical analysis reveals a deep connection between the two models. In fact, with a special configuration, the KCN can emulate kriging.

In summary, this work has three contributions:

- the development of the KCN, which is a GNN that directly uses training labels for prediction;
- the theoretical result showing that the KCN approximately recovers local universal kriging; and
- empirical studies indicating the KCN's advantage over baseline models.

Related Work

Kriging (Cressie 1991) has been widely used in spatial data modeling. Using kriging to model non-Gaussian data is often accomplished through careful transformation of labels (Saito and Goovaerts 2000). However, it is not always feasible to transform a variable to be Gaussian (Dance 2018). One direction of exploration is to weaken the Gaussian assumption of kriging models (Wallin and Bolin 2015), but these methods are often specially designed for their respective applications.

GNNs are neural networks that work on graph data (Gori, Monfardini, and Scarselli 2005). Wu et al. (2019) and Zhou et al. (2018) have done extensive surveys of this topic. A GNN typically consists of a few layers, each of which has a non-linear transformation of the hidden vectors and a step of information propagation between nodes. GNN architectures differ by how they propagate information among graph nodes (Kipf and Welling 2016; Atwood and Towsley 2016; Hamilton, Ying, and Leskovec 2017; Veličković et al. 2018). When a GNN is applied to spatial data (Wu et al. 2019; Yan et al. 2019), one first builds a graph over data points in the spatial area and then runs the GNN on the graph. To the best of our knowledge, all of these methods feed features as the input and fit labels by the output of the network. In this work, we develop our KCN model based on the Graph Convolutional Network (GCN) (Kipf and Welling 2016), Graph Attention Network (GAT) (Veličković et al. 2018), and GraphSAGE (Hamilton, Ying, and Leskovec 2017).

Background

Suppose there are N spatial data points, $(\mathbf{s}, \mathbf{X}, \mathbf{y}) = (s_i, \mathbf{x}_i, y_i)_{i=1}^{N}$, where s_i, $\mathbf{x}_i$, and y_i are respectively the location, the feature vector, and the label of data point i. Usually a location s_i is a GPS coordinate, $s_i \in \mathbb{R}^2$. There are d features in a feature vector $\mathbf{x}_i \in \mathbb{R}^d$. The domain of the target value y_i is application-dependent. For example, $y_i \in \mathbb{N}$ when y_i is a count, and $y_i \in \mathbb{R}^+$ when y_i represents the precipitation level. One important task of spatial data modeling is to predict or estimate the value y_* for a new location s_* with a feature vector $\mathbf{x}_*$. Let $\hat{y}_*$ denote the prediction.

Kriging

There are many variants of kriging, of which *universal kriging* is the most appropriate for the setting above. Universal kriging has the following model assumption (Eq. 3.4.2 in (Cressie 1991)).

$$y_i = \boldsymbol{\beta}^\top \mathbf{x}_i + \epsilon(s_i), i = 1, \dots, n, * \tag{1}$$

Here $\boldsymbol{\beta}$ is the coefficient vector. $\epsilon(\cdot)$ is a zero-mean random process with *variogram* $2\gamma(\cdot)$. The variogram $2\gamma(\cdot)$, which specifies the spatial correlation between data points, is a function of spatial distance: $2\gamma(\|s_i - s_j\|) = \mathbb{E}\left[(\epsilon(s_i) - \epsilon(s_j))^2\right]$. The variogram often takes a special function form with its parameters estimated from the data. With this model assumption, kriging minimizes the expected squared error, $\mathbb{E}_{y_*}\left[(\hat{y}_* - y_*)^2\right]$, in closed form. Then the prediction y_* of universal kriging is $\hat{y}_*^{kriging} = \boldsymbol{\lambda}^\top \mathbf{y}$ with

$$\boldsymbol{\lambda} = \boldsymbol{\Gamma}^{-1}\left(\boldsymbol{\gamma} - \mathbf{B}\mathbf{X}^\top \boldsymbol{\Gamma}^{-1}\boldsymbol{\gamma} + \mathbf{B}\mathbf{x}_*\right), \tag{2}$$

with $\mathbf{B} = \mathbf{X}(\mathbf{X}^\top \boldsymbol{\Gamma}^{-1}\mathbf{X})^{-1}$, $\boldsymbol{\Gamma} = [\gamma(\|s_i - s_j\|)]_{i,j=1}^{n}$, and $\boldsymbol{\gamma} = [\gamma(\|s_i - s_*\|)]_{i=1}^{n}$.

Note that kriging uses known training labels as well as all features as the input to make the prediction. Despite its complex form, kriging has a subtle relation with the KCN model proposed later.

Graph Convolutional Networks

Suppose we have a graph $G = (V, E)$, where $V = \{1, \dots, M\}$ is set of data points, and E is the edge set. Each data point $i \in V$ has a feature vector $\mathbf{x}_i$ and a label y_i. All feature vectors and labels are collectively denoted by $\tilde{\mathbf{X}}$ and $\tilde{\mathbf{y}}$). Let $\mathbf{A}$ denote the adjacency matrix of the graph, and $\bar{\mathbf{A}}$ denote the normalized adjacency matrix,

$$\bar{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}}(\mathbf{A} + \mathbf{I})\mathbf{D}^{-\frac{1}{2}}, \tag{3}$$

with $\mathbf{D} = \mathrm{diag}(\mathbf{A}\mathbf{1} + \mathbf{1})$ being the degree matrix plus one. Then a GCN (Kipf and Welling 2016) takes $\bar{\mathbf{A}}$ and $\tilde{\mathbf{X}}$ as the input and fits known labels in $\tilde{\mathbf{y}}$ as the target. The GCN consists of L GCN layers. Each GCN layer ℓ takes an input $\mathbf{H}_{\ell-1} \in \mathbb{R}^{n \times d_{\ell-1}}$ and outputs a matrix $\mathbf{H}_\ell \in \mathbb{R}^{n \times d_\ell}$. The layer is parameterized by a matrix $\mathbf{W}^\ell$ with size $d_{\ell-1} \times d_\ell$. Formally, the GCN is defined by

$$\mathbf{H}^0 = \tilde{\mathbf{X}}, \tag{4}$$

$$\mathbf{H}^\ell = \sigma\left(\bar{\mathbf{A}}\mathbf{H}^{\ell-1}\mathbf{W}^\ell\right), \quad \ell = 1, \dots, L \tag{5}$$

$$\hat{\mathbf{y}} = \mathbf{H}^L. \tag{6}$$

Here $\sigma(\cdot)$ is a non-linear activation function.

Usually only part of the node labels in $\tilde{\mathbf{y}}$ are observed, and the task is to predict unknown labels of other nodes. A GCN defines its training loss based on known labels and predicts unknown labels as corresponding entries in $\hat{\mathbf{y}}$. To apply a GCN to the previous task, we form a graph over data points based on their locations $(\mathbf{s}, s_*)$, e.g. nearest-neighbor graph. Then each feature vector and its label are attached to the corresponding graph node. The prediction for the new location * is just $\hat{y}_*$ taken from $\hat{\mathbf{y}}$.

Kriging Convolution Network

In this section, we develop a new learning model that directly use training labels as the input for predictions. We call this model a Kriging Convolution Network (KCN).

We will first demonstrate how a KCN will be used for prediction. Let's treat a KCN model as a function $KCN(\cdot;\theta)$ parameterized by θ. When predicting the label of a new data point (s_*, x_*), the model ideally should use all information available to make the prediction, that is, $\hat{y}_* = KCN(\mathbf{s}, \mathbf{X}, \mathbf{y}, s_*, \mathbf{x}_*)$. However, it is not feasible to consider all of the training points for just one prediction. It is not necessary either, because data points far from s_* often have little influence over y_* in many spatial problems. Therefore, we use the K nearest neighbors of the new data point as the input. Denote the index set of these neighbors as $\alpha_* \subset \{1, \ldots, N\}$, then the predictive function becomes

$$\hat{y}_* = KCN(\mathbf{s}_{\alpha_*}, \mathbf{X}_{\alpha_*}, \mathbf{y}_{\alpha_*}, s_*, \mathbf{x}_*). \tag{7}$$

To train the model, we treat every training point i as a test point and fit its training label y_i. The model's output, $\hat{y}_i$, is compared against the true label. The difference of the two is measured by some loss function $\mathrm{loss}(y_i, \hat{y}_i)$. The learning objective of the model is to minimize the summation of all training losses

$$\min_\theta \sum_{i=1}^{N} \mathrm{loss}(y_i, \hat{y}_i),$$
$$\hat{y}_i = KCN(\mathbf{s}_{\alpha_i}, \mathbf{X}_{\alpha_i}, \mathbf{y}_{\alpha_i}, s_i, \mathbf{x}_i). \tag{8}$$

Here α_i is the set of neighbors of i in the training set.

Now we construct the network architecture of the KCN, i.e. the function $KCN(\mathbf{s}_{\alpha_i}, \mathbf{X}_{\alpha_i}, \mathbf{y}_{\alpha_i}, s_i, \mathbf{x}_i)$. Instead of using locations, s_i and $\mathbf{s}_{\alpha_i}$, as features, we define a complete graph over the data points i and its neighbors and then use a GCN to construct the predictive model. Denote $\beta_i = \{i\} \cup \alpha_i$ as the set containing the data point i and its neighbors. We first define a graph over β_i by constructing its adjacency matrix $\mathbf{A}$ from a Gaussian kernel,

$$A_{jk} = \exp\left(-\frac{1}{2\phi^2}\|s_j - s_k\|_2^2\right), \ \forall j, k \in \beta_i. \tag{9}$$

Here ϕ is the kernel length, which is a hyperparameter. In this graph, the edge (j, k) has a large weight when j and k are near each other and vice versa.

Next we define the feature input to the GCN. The input should include features, $\mathbf{x}_i$ and $\mathbf{X}_{\alpha_i}$, and neighboring labels $\mathbf{y}_{\alpha_i}$. Incorporating this information into a matrix will require a bit of care. We place $\mathbf{y}_{\alpha_i}$ and a zero in place of y_i into a vector with length $(K+1)$, so the model has no access to y_i. We also use an indicator vector $\mathbf{e}$ to indicate that the instance i is the one to be predicted. Then the GCN input is expressed

```
Input: $(\mathbf{s}, \mathbf{X}, \mathbf{y})$, $K$
Output: $\theta = (\mathbf{W}^1, \ldots, \mathbf{W}^L, \mathbf{w}_{den})$
for $i \leftarrow 0$ to $N$ do
    $\beta_i$ = the $K$ nearest neighbors of $s_i$ and $i$ ;
    Compute $\mathbf{A}$ from $s_{\alpha_i}$ by (9) ;
    Prepare $\mathbf{H}^0$ from $(x_i, \mathbf{X}_{\alpha_i}, y_{\alpha_i})$ by (10) ;
end
for $iter \leftarrow 0$ to num training iter do
    $i = iter\%N$ ;
    $\mathbf{H}^L = GCN(\mathbf{A}, \mathbf{H}^0; \mathbf{W}^1, \ldots, \mathbf{W}^L)$ ;
    $\hat{y}_i = \sigma(\mathbf{e}^\top \mathbf{H}^L)\mathbf{w}_{den}$ ;
    Compute $loss(y_i, \hat{y}_i)$ and its derivative ;
    Update weights $\theta = \mathbf{W}^1, \ldots, \mathbf{W}^L, \mathbf{w}_{den}$
end
```

Algorithm 1: The training algorithm of KCN

by a matrix $\mathbf{H}^0$ with size $(K+1) \times (2+d)$.

$$\mathbf{H}^0 = \begin{bmatrix} 0 & 1 & \mathbf{x}_i^\top \\ \mathbf{y}_{\alpha_i} & \mathbf{0} & \mathbf{X}_{\alpha_i} \end{bmatrix}. \tag{10}$$

The locations $\mathbf{s}_{\beta_i}$ can be included in the feature matrix $\mathbf{X}$ as features if there is reason to suspect spatial trends.

Then the KCN model is defined to be a GCN followed by a dense layer. The KCN is formally defined as

$$\mathbf{H}^L = GCN(\mathbf{A}, \mathbf{H}^0), \tag{11}$$
$$\hat{y}_i = \sigma\left(\mathbf{e}^\top \mathbf{H}^L \mathbf{w}_{den}\right). \tag{12}$$

Here $\mathbf{A}$ and $\mathbf{H}^0$ are the adjacency matrix and the input feature matrix constructed from the neighborhood of i. Note that every data point i gets its own $\mathbf{A}$ and $\mathbf{H}^0$, whose index i is omitted for notational simplicity. The vector $\mathbf{e}$ is the indicator vector for i: it takes the first vector of $\mathbf{H}^L$, corresponding to i, as the input to the dense layer. The dense layer allows for a final transformation of the data without interference from neighbors.

The KCN parameters are all weight matrices, $\theta = \{\mathbf{W}^1, \ldots, \mathbf{W}^L, \mathbf{w}_{den}\}$. We train the KCN model by minimizing the loss in (8). Then we can predict the label of a new data point using its features and neighbors in the training set. Algorithm 1 summarizes the training procedure of the KCN.

Compared to local kriging, which only uses nearest neighbors for kriging, the KCN uses the same input. However, the KCN is much more flexible. When the training set is large enough such that the overfitting issue is less of a concern, the KCN model has clear advantages.

Compared to the direct application of a GCN on spatial data, a KCN is able to use labels from neighbors directly. Furthermore, a KCN does not need to use the test data points to form the graph. Therefore, it does not need to re-train the model when there is a new batch of test data points.

The KCN is also similar to the KNN classifier but is much more powerful: while the KNN simply averages the labels of neighbors, the KCN uses a neural network as the predictive function.

KCN with Graph Attention

The recent success of attention mechanism on GNNs inspires us to try the Graph Attention network (GAT) (Veličković et al. 2018) as the predicting model. The original GAT model computes attention weights with a neural network; it also requres that the attention weights of a node's neighbors sum up to 1. Here we use the dot-product self-attention (Vaswani et al. 2017) so that the model has a choice to fall back on the GCN model.

Suppose the input feature at the ℓ-th layer of the GCN is $\mathbf{H}^{\ell-1}$, then we compute an attention matrix $\mathbf{U}$ by

$$\mathbf{P} = \mathbf{H}^{\ell-1}\mathbf{W}_{att}, \quad \mathbf{M} = \sigma(\mathbf{P}\mathbf{P}^\top),$$
$$\mathbf{\Lambda} = \text{diag}(\mathbf{M}), \quad \mathbf{U} = \mathbf{\Lambda}^{-\frac{1}{2}}\mathbf{M}\mathbf{\Lambda}^{-\frac{1}{2}}. \tag{13}$$

Here $\mathbf{W}_{att}$ is the weight matrix for the attention mechanism. It projects input features into a new space. Then the attention weights are decided by inner products between features in this new space. We normalize the attention matrix so that the diagonal elements of $\mathbf{U}$ are always one.

In each layer ℓ, we get an attention matrix $\mathbf{U}_\ell$ as above. Then we use $\mathbf{A}_\ell^{att} = \mathbf{A} \odot \mathbf{U}_\ell$ as the new adjacency matrix used in layer ℓ. The actual computation is

$$\mathbf{H}^\ell = \sigma\left(\mathbf{A}_\ell^{att}\mathbf{H}^{\ell-1}\mathbf{W}^\ell\right), \quad \ell = 1, \ldots, L \tag{14}$$

We call this new model the KCN-att. When the matrix $\mathbf{W}_{att}$ has small weights, then $\mathbf{U}$ approaches a matrix with all entries being one. In this case, the KCN-att becomes similar to the KCN. When the matrix $\mathbf{W}_{att}$ has large weights, then $\mathbf{U}$ tends to approach the identity matrix, and then the KCN-att tends to reduce neighbors' influence.

KCN based GraphSAGE

We also use GraphSAGE (Hamilton, Ying, and Leskovec 2017) as the predictive model of the KCN given that GraphSAGE performs well on several node classification tasks. GraphSAGE cannot use a weighted graph, so we treat the graph over the neighborhood of i as a complete graph. Let $\mathbf{H}^{\ell-1} = \{\mathbf{h}_k^{\ell-1} : k \in \beta_i\}$ be the input to the GraphSAGE layer ℓ, then the layer computes its output $\mathbf{H}^\ell$ as follows.

$$\mathbf{g}_j^\ell = \text{AGG}\left(\{\mathbf{h}_k^{\ell-1}, k \in \beta_i, k \neq j\}\right), \quad \forall j \in \beta_i \tag{15}$$
$$\mathbf{h}_j^\ell = \sigma\left(\mathbf{W}_1^\ell\mathbf{h}_j^{\ell-1} + \mathbf{W}_2^\ell\mathbf{g}_j^\ell\right), \quad \forall j \in \beta_i \tag{16}$$
$$\mathbf{H}^\ell = \left\{\mathbf{h}_j^\ell / \|\mathbf{h}_j^\ell\|_2 : \forall j \in \beta_i\right\} \tag{17}$$

The function $\text{AGG}(\cdot)$ aggregates a list of vectors into one. We use the max-pooling aggregator, one the three aggregators proposed in the original work (Hamilton, Ying, and Leskovec 2017).

$$\text{AGG}(\mathbf{H}_{\beta_i \setminus j}^{\ell-1}) = \max(\sigma(\mathbf{W}_{pool}\mathbf{h}_k^{\ell-1} + \mathbf{b}), k \in \beta_i, k \neq j)$$

Here $\max$ takes the element-wise max values over a list of vectors. We refer to this model as the KCN-sage.

Analysis

Computation Complexity

The time complexity of the KCN and the two variants includes nearest-neighbor search and network training. In order to find the K nearest neighbors we utilize a KD tree, which takes $O(N\log(N))$ time to build. Here we treat the dimensionality of spatial coordinates as a constant because it is usually a small number (2 or 3). Querying a single data point in the tree takes time $O(K\log(N))$, and searching neighbors for all data points takes a total of $O(NK\log(N))$ time.

When we train the model on a single instance, the computation of the adjacency matrix takes time $O(K^2)$. The computation within each layer takes time $O(K^2 d_{\max})$, with $d_{\max}$ being the largest dimensionality of hidden layers. The forward computation and backpropagation for one instance takes time $O(K^2 L d_{\max})$, and one training epoch takes time $O(NK^2 L d_{\max})$.

Relation to Kriging

The KCN is a flexible model and approximately includes local kriging as a special case. This fact is shown by the following theorem.

Theorem 1: Assume the variogram of a kriging model satisfies $2\gamma(0) > 0$ [1]. Also assume $\tilde{\mathbf{X}} = [\mathbf{x}_*, \mathbf{X}_{\alpha_*}^\top]^\top$ has full column rank. Then there exists a set of special parameters and activations with which a KCN makes the same prediction as the kriging prediction, i.e. $\hat{y}_*^{KCN} = \hat{y}_*^{kriging}$.

Proof sketch: Let $\tilde{\mathbf{\Gamma}}$ be the covaraince matrix corresponding to the new data point and training data point.

$$\tilde{\mathbf{\Gamma}} = \begin{bmatrix} 0 & \boldsymbol{\gamma}^\top \\ \boldsymbol{\gamma} & \mathbf{\Gamma} \end{bmatrix}. \tag{18}$$

Here $\boldsymbol{\gamma}$ and $\mathbf{\Gamma}$ are semivariograms defined in the same way as kriging. To approximate kriging, we set the KCN to have one convolutional layer and a dense layer. We set

$$\bar{\mathbf{A}} = \tilde{\mathbf{\Gamma}}^{-1} - \tilde{\mathbf{\Gamma}}^{-1}\tilde{\mathbf{X}}(\tilde{\mathbf{X}}^\top\tilde{\mathbf{\Gamma}}^{-1}\tilde{\mathbf{X}})^{-1}\tilde{\mathbf{X}}^\top\tilde{\mathbf{\Gamma}}^{-1} \tag{19}$$

as the "normalized adjacency matrix" and directly use it to multiply the hidden input. We consider a 1-layer GCN with a special activation function $\sigma_{div}(\cdot)$. The first row of the GCN output is $\mathbf{e}^\top\mathbf{H}^L = \sigma_{div}\left(\mathbf{e}\bar{\mathbf{A}}\mathbf{H}^0\mathbf{W}^1\right)$. In the Appendix we show $\mathbf{e}\bar{\mathbf{A}} = \left[z^{-1}, -z^{-1}\boldsymbol{\lambda}^\top\right]$, then

$$\mathbf{e}\bar{\mathbf{A}}\mathbf{H}^0 = \left[-z^{-1}\boldsymbol{\lambda}^\top\mathbf{y}_{\alpha_*},\ z^{-1},\ z^{-1}(\mathbf{x}_* - \boldsymbol{\lambda}^\top\mathbf{X}_{\alpha_*})\right].$$

Here z is a scalar, and $\boldsymbol{\lambda}$ is the kriging coefficient defined in (2). Take the first two elements of $\mathbf{e}\bar{\mathbf{A}}\mathbf{H}^0$ and denote it as $\mathbf{u} = [-z^{-1}\boldsymbol{\lambda}^\top\mathbf{y}_{\alpha_*}, z^{-1}]$. We can see that $-u_1/u_2 = \boldsymbol{\lambda}^\top\mathbf{y}_{\alpha_*} = \hat{y}_*^{kring}$. If the rest of the network is able to emulate $f(\mathbf{u}) = -u_1/u_2$, then $\hat{y}_*^{KCN}$ is the same as $\hat{y}_*^{kring}$.

Now consider the network computation on $\mathbf{u}$. It is a combination of the first two rows of $\mathbf{W}^1$, the GCN activation, and then the dense layer, so it can be viewed as a two-layer feedforward network applied to $\mathbf{u}$. If special activations are allowed, such as step functions and the logarithm, then the two-layer neural network can realize the function $-u_1/u_2$. With normal network settings, the network can also approximate the function due to the universal approximation theorem. □

[1]The value $2\gamma(0)$ is called as the nugget of the variogram, which is usually greater than zero.

This theorem and its proof have strong implications for our model development. First, if the KCN uses $\bar{\mathbf{A}}$ defined above as the normalized adjacency matrix, then the KCN has a straightforward way to discover kriging solutions. Since $\bar{\mathbf{A}}$ has a small size, $(K+1) \times (K+1)$, the computation of $\bar{\mathbf{A}}$ is affordable. Second, the matrix $\bar{\mathbf{A}}$ indicates that we should introduce the feature matrix into the computation of the normalized adjacency matrix. Otherwise, the KCN may need complicated computations to recover kriging results. This is one main motivation behind our usage of graph attention in KCN-att.

Experiment

We evaluate our methods on three tasks: bird count modeling, restaurant rating regression, and precipitation regression. We use kriging, Random Forest, GCN, and GraphSAGE as baselines.

Experiment setup

Kriging: we use the implementation of kriging within Automap (Hiemstra et al. 2008). Automap essentially automates the process of Kriging by automatically fitting variograms and testing a number of different models. In all of our experiments, Automap tests spherical, exponential, gaussian, matern, and stein variograms and picks the best one based on the smallest residual sum of squares. Since all of the datasets have a large number of data points, we use local kriging and only consider the closest 100 points.

Random Forest: Hengl et al. (2018) use Random Forest to make predictions for spatial data. For each data point, the algorithm calculates the distances between that point and all training points. These distances are then used as the feature vector of that data point. This algorithm does not scale to very large datasets, so we downsample the training set to a size of 1000. We use the implementation of Random Forest (Wright and Ziegler 2017), and method of tuning (Probst, Wright, and Boulesteix 2018) used by the authors of Hengl et al. (2018). The implementation tunes four hyperparameters of Random Forest: the number of trees to use, the number of variables to consider at a node split, the minimal node size, and the sample fraction when training each tree.

GCN: we modify Kipf's implementation of (Kipf and Welling 2016) for regression problems. Before we run the GCN on spatial data, we first build an undirected graph over data points: we connect two data points if one is among the other's K nearest neighbors. We only consider a GCN with two hidden layers. We tune the hyper-parameters of the GCN in the same way as we tune the KCN and the KCN-att below.

GraphSAGE: we implement GraphSAGE with the *Spektral* graph deep learning library. For each experiment, we build an undirected graph in the same way as GCN. Then we train a two hidden layer GraphSAGE whose hyperparameters are tuned as below.

KCN & KCN-att & KCN-sage: the three models use two hidden layers respectively. We tune the following hyperparameters: hidden sizes $\in ((20, 10), (10, 5), (5, 3))$, dropout rate $\in (0, 0.25, 0.5)$, and kernel length $\in (1, .5, .1, .05)$.

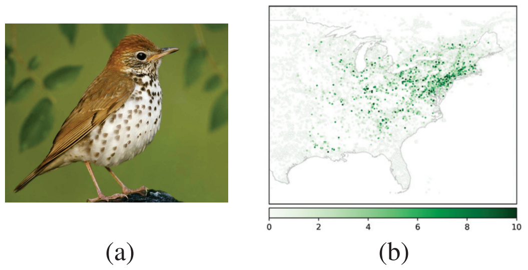

Figure 1: Wood thrush (a) and observed counts over eastern US, June 2014 (b).

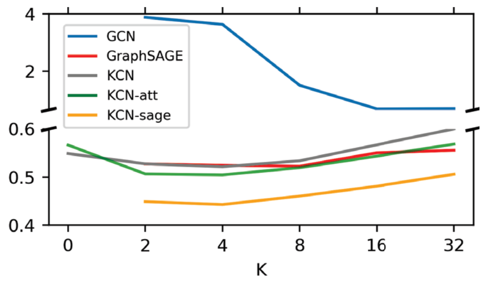

Figure 2: Mean Squared Error of GCN, GraphSAGE, KCN, KCN-att, and KCN-sage using different numbers of neighbors.

Note that GraphSAGE and KCN-sage do not consider edge weights of the adjacency matrix, so we do not tune kernel length for them. We also employed early stopping to decide the number epochs.

Bird count modeling

One application of KCNs is modeling bird count data from the eBird project (Fink et al. 2010), which contains over one billion records of bird observation events. Modeling bird data from the eBird project provides an opportunity to deepen our understanding of birds as part of the ecosystem. In this experiment, we model the distribution of *wood thrush* in June, which is of great interest to ornithologists (Johnston et al. 2019). Figure 1 shows a picture of a wood thrush and the distribution of observed counts over the eastern US.

We restrict our data to a subset of records of wood thrush in June 2014. Each record has a GPS location, a count of wood thrushes observed, and a list of features such as observation time, count type (stationary count, traveling count, etc.), effort hours, and effort area. After removing 583 records with uncertain counts or counts over 10, we get 107,246 records to form our dataset. Bird counts in this dataset are highly sparse: only 11,468 records (fraction of 0.11) have positive counts. We split the dataset into a training set and a test set by 1:1.

When we test our models and baselines, we consider two evaluation metrics. The first one is mean squared error (MSE), so we have a fair comparison with Kriging, the minimization objective of which is the mean squared error. The second one is negative log-likelihood. We use a zero-inflated Poisson distribution (Lambert 1992) as the predictive distri-

methods	Kriging	RF	GCN	GraphSAGE	KCN	KCN-att	KCN-sage
MSE	$1.56 \pm .85$	$0.68 \pm .03$	$0.70 \pm .02$	$0.53 \pm .01$	$0.50 \pm .01$	$0.49 \pm .01$	$\mathbf{0.44 \pm .01}$
NLL	n.a.	n.a.	$1.82 \pm .00$	$1.73 \pm .00$	$1.60 \pm .00$	$1.58 \pm .00$	$\mathbf{1.51 \pm .00}$

Table 1: Experiment results on the bird count dataset. Performances are measured by the mean squared error and the negative log likelihood of preditions. Smaller values are better.

bution for each count. The model needs to output a logit u for the Bernoulli probability and the mean λ of the Poisson component. The probability of a count y given u and λ is

$$p(y) = \begin{cases} (1 - \text{expit}(u)) + p_{poisson}(y = 0) & \text{if } y = 0, \\ p_{poisson}(y = 0) \text{ if} & y > 0. \end{cases} \tag{20}$$

Table 1 shows the performance of the KCN, KCN-att, KCN-sage, and baseline models. From this table, we can see that the three KCN models significantly outperform baseline methods. We also observed that GraphSAGE based methods have better performance than the GCN based methods, we speculate it is because of the concatenation operation (plays a role similar to a skip link) used in the GraphSAGE. Given that bird counts are highly non-Gaussian, we do not expect kriging to perform very well. Random Forest gets much better performance than kriging, but it overly smooths the training data given the small number of training points it can use. The KCN and KCN-att achieve similar performances.

We also study the performances of the GCN, GraphSAGE, KCN, KCN-att, and KCN-sage when different numbers of neighbors are used to form the graph. Figure 2 shows performance values of the five models using different numbers. The GCN perform poorly when the number of neighbors is small in the construction of the graph. In this case, a test point might only connect to another test data point, then the message propagation between two test points is not helpful. GraphSAGE is robust to the number of neighbors. In KCN models, a data point has K training points as direct neighbors, so KCN models can make better use of the training data in this sense. When a KCN model uses zero neighbors, it is equivalent to a fully connected neural network. In this case, its performance deteriorates significantly, which indicates that spatial correlation exists in the data. KCN, KCN-att, and KCN-sage only need a small number of neighbors to perform well. We speculate that a bird or its nest can be observed only in a small spatial range, so the correlation between near sites is strong but diminishes quickly as the distance increases. The KCN-att performs slightly better than the KCN because it is able to use observatory features to decide whether a neighboring count is from a similar situation or not.

Restaurant rating regression

Yelp is a popular website, which allows users to rate and provide information about businesses. They have hosted an extensive collection of these business ratings and attributes for download. In this experiment, we only consider the restaurants within that dataset. Each restaurant has a GPS location and an average rating rounded to the nearest .5, from 0 to 5. Additionally, we choose 13 related attributes from the dataset, all but one of which is categorical. We turn these categorical covariates into 30 indicator variables. These indicators give information about restaurant attributes such as whether it serves alcohol, and whether it takes credit card. After we drop any rows where the ratings, coordinates, or number of reviews is NA, we obtain 188,586 restaurants. We then split the data 1:1 to form a training and test set.

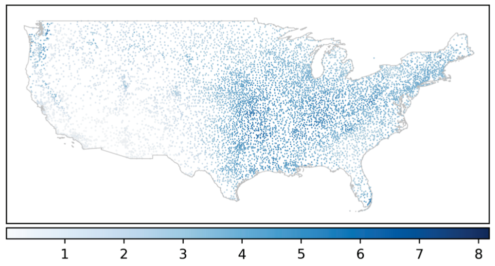

Figure 3: Distribution of Precipitation values.

Table 2 shows the experimental results on this dataset. The KCN, KCN-att, and KCN-sage improve the performance of their corresponding baseline models. This task is hard: the features do not seem to be very useful, and the spatial correlations are weak. It is actually hard to overfit the labels with a standard feedforward neural network. Ratings are correlated in a strange way since it is normal that good and bad restaurants are often mixed in the same area. The bad performance of the kriging method can be explained by the fact that the correlations between ratings are highly non-Gaussian.

Precipitation regression

The National Oceanic and Atmospheric Administration keeps detailed records of precipitation levels across the United States. One such dataset provides monthly average precipitation in inches from 1981 to 2010 across the US. We average the precipitation level in May for 8,832 stations. We then take the log of these average precipitation values as target values for the regression task. Essentially, we assumes a log-normal distribution of precipitation levels. Finally we have a target value, coordinates of each station, and one feature (the elevation) of each station. Data are split 1:1 as a training and testing. Figure 3 shows the data distribution over the US.

Table 3 summarizes the experimental results using the mean squared error. The target values in the log-scale are more likely to be from a Gaussian distribution than the previous two datasets, so the kriging method performs relatively well compared to other methods. The Random Forest method only uses 1000 data points as training data, so it

Method	Kriging	Random Forest	GCN	GraphSAGE	KCN	KCN-att	KCN-sage
MSE	1.49 ± .008	1.04 ± .005	1.37 ± .006	**0.969 ± .004**	0.990 ± .004	0.977 ± .004	**0.959 ± .004**

Table 2: The results on the dataset of restaurant ratings. Performances are measured by MSE. Smaller values are better.

Method	Kriging	Random Forest	GCN	GraphSAGE	KCN	KCN-att	KCN-sage
MSE	.155 ± .013	.046 ± .003	.640 ± .023	.056 ± .003	**.029 ± .002**	**.029 ± .002**	**.030 ± .002**

Table 3: Experiment results on the precipitation dataset. Performances are measured by MSE. Smaller values are better.

omits a lot of detailed variations. The GCN models perform poorly on this dataset. One reason is that there are not many features for the GCN to learn. The GCN model becomes more like a "generative" model that "generates" correlated labels from hidden values. Compared to GNN baselines, the KCN models benefit in particular from using neighbroring labels because the KCN models work more like a discriminative model. Note that discriminative models often outperform generative models in supervised learning tasks.

Conclusion

In this work, we have introduced the Kriging Convolutional Network, a novel approach to modeling spatial data. Like kriging, the KCN model directly uses training labels in the prediction. However, by employing GNNs as backbone predictive models, the KCN is far more flexible than kriging. We have tried the GCN, GAT, and GraphSAGE as the predictive model of the KCN. The empirical study shows that KCN has significant performance improvement over kriging. Compared with baseline GNNs, the KCN directly uses known labels, which has a clear benefit as indicated by the experimental results. Our analysis also reveals that the KCN has the ability to emulate kriging models. This connection further indicates the flexibility of the proposed KCN model.

Appendix: Detailed Proof of Theorem 1

We need to derive $\mathbf{e}\bar{\mathbf{A}}$, where $\mathbf{e} = [1, \mathbf{0}^\top]^\top$, $\bar{\mathbf{A}}$ is "normalized adjacency matrix" defined in (19), and $\tilde{\mathbf{X}} = [\mathbf{x}_*, \mathbf{X}^\top]^\top$ are feature vectors. We create the following shorthand notations to facilitate our derivation.

$$t = -\boldsymbol{\gamma}^\top \boldsymbol{\Gamma}^{-1}\boldsymbol{\gamma},\ \ \mathbf{a} = -\boldsymbol{\Gamma}^{-1}\boldsymbol{\gamma},\ \ \mathbf{c} = \mathbf{x}_* + \mathbf{X}^\top \mathbf{a},$$
$$\mathbf{T} = \mathbf{X}^\top \boldsymbol{\Gamma}^{-1}\mathbf{X},\ \ \mathbf{B} = \mathbf{X}\mathbf{T}^{-1},\ \ r = \mathbf{c}^\top \mathbf{T}^{-1}\mathbf{c}$$

We will show that the first row of $\bar{\mathbf{A}}$ is

$$\mathbf{e}^\top \bar{\mathbf{A}} = \left[z^{-1}, -z^{-1}\boldsymbol{\lambda}^\top\right]$$

with z being a scalar.

By checking (19), we first compute the inverse $\tilde{\boldsymbol{\Gamma}}^{-1}$ is

$$\tilde{\boldsymbol{\Gamma}}^{-1} = \begin{bmatrix} t^{-1} & t^{-1}\mathbf{a}^\top \\ t^{-1}\mathbf{a} & \boldsymbol{\Gamma}^{-1} + t^{-1}\mathbf{a}\mathbf{a}^\top \end{bmatrix}$$
$$= t^{-1}\begin{bmatrix} 1 \\ \mathbf{a} \end{bmatrix}[1, \mathbf{a}^\top] + \begin{bmatrix} 0 & \mathbf{0}^\top \\ \mathbf{0} & \boldsymbol{\Gamma}^{-1} \end{bmatrix}.$$

Denote $\mathbf{v}_1 = \mathbf{e}\tilde{\boldsymbol{\Gamma}}^{-1} = t^{-1}[1, \mathbf{a}^\top]$.

We then consider the second term in (19). We have

$$\tilde{\mathbf{X}}^\top \tilde{\boldsymbol{\Gamma}}^{-1} = t^{-1}\mathbf{c}[1, \mathbf{a}^\top] + [\mathbf{0}, \mathbf{X}^\top \boldsymbol{\Gamma}^{-1}]$$
$$= [t^{-1}\mathbf{c}, t^{-1}\mathbf{c}\mathbf{a}^\top + \mathbf{X}^\top \boldsymbol{\Gamma}^{-1}].$$

Denote $\mathbf{S} = (\tilde{\mathbf{X}}^\top \tilde{\boldsymbol{\Gamma}}^{-1}\tilde{\mathbf{X}})^{-1}$,

$$\mathbf{S} = (t^{-1}\mathbf{c}\mathbf{c}^\top + \mathbf{T})^{-1}$$
$$= \mathbf{T}^{-1} - \mathbf{T}^{-1}\mathbf{c}(t + \mathbf{c}^\top \mathbf{T}^{-1}\mathbf{c})^{-1}\mathbf{c}^\top \mathbf{T}^{-1}.$$

The first line is from the equation $[1, \mathbf{a}^\top]\tilde{\mathbf{X}} = \mathbf{c}^\top$.

Denote $\mathbf{v}_2 = \mathbf{e}^\top \tilde{\boldsymbol{\Gamma}}^{-1}\tilde{\mathbf{X}}\mathbf{S}\tilde{\mathbf{X}}^\top \tilde{\boldsymbol{\Gamma}}^{-1}$. Insert the expansion of $\tilde{\mathbf{X}}^\top \tilde{\boldsymbol{\Gamma}}^{-1}$, we have

$$\mathbf{v}_2 = t^{-1}\mathbf{c}^\top \mathbf{S}[t^{-1}\mathbf{c},\ \ t^{-1}\mathbf{c}\mathbf{a}^\top + \mathbf{X}^\top \boldsymbol{\Gamma}^{-1}].$$

By $\mathbf{c}^\top \mathbf{S} = t(t{+}r)^{-1}\mathbf{c}\mathbf{T}^{-1}$ and $w = \mathbf{c}^\top \mathbf{S}\mathbf{c} = rt(t{+}r)^{-1}$, we have

$$\mathbf{v}_2 = (t + r)^{-1}[rt^{-1},\ rt^{-1}\mathbf{a}^\top + \mathbf{c}^\top \mathbf{T}^{-1}\mathbf{X}^\top \boldsymbol{\Gamma}^{-1}].$$

Since $t^{-1} - (t + r)^{-1}rt^{-1} = (t + r)^{-1}$, we have

$$\mathbf{e}^\top \bar{\mathbf{A}} = \mathbf{v}_1 - \mathbf{v}_2$$
$$= (t + r)^{-1}[1,\ \ (\mathbf{a}^\top - \mathbf{c}^\top \mathbf{T}^{-1}\mathbf{X}^\top \boldsymbol{\Gamma}^{-1})].$$

Then we expand $\mathbf{a}$ and $\mathbf{c}$ to get

$$\mathbf{e}^\top \bar{\mathbf{A}} = (t + r)^{-1}[1,\ \ -(\boldsymbol{\gamma} + \mathbf{B}\mathbf{x}_* - \mathbf{B}\mathbf{X}\boldsymbol{\Gamma}^{-1}\boldsymbol{\gamma})^\top \boldsymbol{\Gamma}^{-1})]$$
$$= z^{-1}[1,\ \ -\boldsymbol{\lambda}^\top]$$

Here $z = t + r$.

Since $\bar{\mathbf{A}}$ are computed from normal matrix operations, its entries are bounded. Therefore, $z \neq 0$.

Acknowledgments

This work has been supported in part by Gordon & Betty Moore Foundation, NSF CISE-1908617, and and NSF CRII-1850358. We thank all reviewers of this work for their insightful comments.

References

Atwood, J., and Towsley, D. 2016. Diffusion-convolutional neural networks. In *Advances in Neural Information Processing Systems*, 1993–2001.

Cressie, N. A. C. 1991. *Statistics for spatial data.* Wiley series in probability and mathematical statistics. Applied probability and statistics. New York: J. Wiley.

Dance, T. 2018. *A comparison of linear and non-linear kriging techniques for predicting the probability of exceeding a threshold value*. Ph.D. Dissertation.

Fink, D.; Hochachka, W. M.; Zuckerberg, B.; Winkler, D. W.; Shaby, B.; Munson, M. A.; Hooker, G.; Riedewald, M.; Sheldon, D.; and Kelling, S. 2010. Spatiotemporal exploratory models for broad-scale survey data. *Ecological Applications* 20(8):2131–2147.

Gao, H., and Liu, H. 2014. Data analysis on location-based social networks. In *Mobile social networking*. Springer. 165–194.

Gori, M.; Monfardini, G.; and Scarselli, F. 2005. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, 729–734. IEEE.

Hamilton, W.; Ying, Z.; and Leskovec, J. 2017. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, 1024–1034.

Hengl, T.; Nussbaum, M.; Wright, M. N.; Heuvelink, G. B.; and Gräler, B. 2018. Random forest as a generic framework for predictive modeling of spatial and spatio-temporal variables. *PeerJ* 6:e5518.

Hiemstra, P.; Pebesma, E.; Twenh"ofel, C.; and Heuvelink, G. 2008. Real-time automatic interpolation of ambient gamma dose rates from the dutch radioactivity monitoring network. *Computers & Geosciences*. DOI: http://dx.doi.org/10.1016/j.cageo.2008.10.011.

Johnston, A.; Hochachka, W.; Strimas-Mackey, M.; Gutierrez, V. R.; Robinson, O.; Miller, E.; Auer, T.; Kelling, S.; and Fink, D. 2019. Best practices for making reliable inferences from citizen science data: case study using ebird to estimate species distributions. *bioRxiv* 574392.

Kipf, T. N., and Welling, M. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Lambert, D. 1992. Zero-inflated poisson regression, with an application to defects in manufacturing. *Technometrics* 34(1):1–14.

Li, Y.; Yu, R.; Shahabi, C.; and Liu, Y. 2017. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. *arXiv preprint arXiv:1707.01926*.

Probst, P.; Wright, M.; and Boulesteix, A.-L. 2018. Hyperparameters and tuning strategies for random forest. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*.

Saito, H., and Goovaerts, P. 2000. Geostatistical interpolation of positively skewed and censored data in a dioxin-contaminated site. *Environmental Science & Technology* 34(19):4228–4235.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L. u.; and Polosukhin, I. 2017. Attention is all you need. In Guyon, I.; Luxburg, U. V.; Bengio, S.; Wallach, H.; Fergus, R.; Vishwanathan, S.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 30*. 5998–6008.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Liò, P.; and Bengio, Y. 2018. Graph attention networks. In *International Conference on Learning Representations*.

Wallin, J., and Bolin, D. 2015. Geostatistical modelling using non-gaussian matérn fields. *Scandinavian Journal of Statistics* 42(3):872–890.

Wright, M. N., and Ziegler, A. 2017. ranger: A fast implementation of random forests for high dimensional data in C++ and R. *Journal of Statistical Software* 77(1):1–17.

Wu, Z.; Pan, S.; Chen, F.; Long, G.; Zhang, C.; and Yu, P. S. 2019. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*.

Xingjian, S.; Chen, Z.; Wang, H.; Yeung, D.-Y.; Wong, W.-K.; and Woo, W.-c. 2015. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *Advances in neural information processing systems*, 802–810.

Yan, X.; Ai, T.; Yang, M.; and Yin, H. 2019. A graph convolutional neural network for classification of building patterns using spatial vector data. *ISPRS journal of photogrammetry and remote sensing* 150:259–273.

Yu, B.; Yin, H.; and Zhu, Z. 2017. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. *arXiv preprint arXiv:1709.04875*.

Zhou, J.; Cui, G.; Zhang, Z.; Yang, C.; Liu, Z.; and Sun, M. 2018. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*.

Zhu, D., and Liu, Y. 2018. Modelling spatial patterns using graph convolutional networks (short paper). In *10th International Conference on Geographic Information Science (GIScience 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.