A Low Rank Weighted Graph Convolutional Approach to Weather Prediction

Tyler Wilson*, Pang-Ning Tan*, Lifeng Luo†
*Department of Computer Science and Engineering, Michigan State University, East Lansing, MI, 48823
†Department of Geography, Michigan State University, East Lansing, MI, 48823
Email: {wils1270,ptan,lluo}@msu.edu

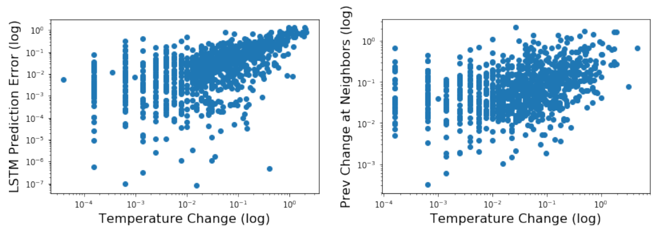
***Abstract*—Weather forecasting is an important but challenging problem as one must contend with the inherent non-linearities and spatiotemporal autocorrelation present in the data. This paper presents a novel deep learning approach based on a coupled weighted graph convolutional LSTM (WGC-LSTM) to address these challenges. Specifically, our proposed approach uses an LSTM to capture the inherent temporal autocorrelation of the data and a graph convolution to model its spatial relationships. As the weather condition can be influenced by various spatial factors besides the distance between locations, e.g., topography, prevailing winds and jet streams, imposing a fixed graph structure based on the proximity between locations is insufficient to train a robust deep learning model. Instead, our proposed approach treats the adjacency matrix of the graph as a model parameter that can be learned from the training data. However, this introduces an additional $O(|V|^2)$ parameters to be estimated, where V is the number of locations. With large graphs this may also lead to slower performance as well as susceptibility to overfitting. We propose a modified version of our approach that can address this difficulty by assuming that the adjacency matrix is either sparse or low rank. Experimental results using two real-world weather datasets show that WGC-LSTM outperforms all other baseline methods for the majority of the evaluated locations.**

***Index Terms*—Graph convolution, long short-term memory, deep learning, weather prediction**

I. Introduction

Weather prediction is an important modeling task due to its significant impact on agriculture, water resources, transportation, and many other aspects of our daily lives. Among the most popular approaches to weather prediction in use today is numerical weather prediction, which employs physics-based models to generate forecasts of future weather conditions [1]. In addition, there has also been substantial research on using data driven approaches to weather prediction [2] [3]. However, because of the nonlinearity inherent within the task, simple linear models are often insufficient to produce reliable forecasts. This has led to considerable interest in applying non-linear techniques, such as artificial neural networks [4][5] [6], to weather prediction problems. These approaches are mostly implemented using fully connected neural networks [7] [8], recurrent neural networks [9], and long short-term memory (LSTM) networks [10]. However, a major limitation of using these approaches is that they do not adequately consider the spatial autocorrelation within the dataset.

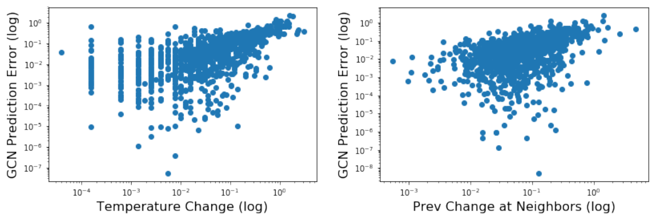
To illustrate this problem, we examine the prediction errors of an LSTM on a dataset of 12-hourly weather measurements taken at weather stations across the continental United States. LSTMs are a powerful deep learning model for time series prediction that have found success in various domains such as speech recognition [11] and machine translation [12]. However, an LSTM that makes predictions based solely on its historical weather measurements completely ignores the important spatial relationships that exist within the data. For example, Figure 1a compares the temperature prediction error of an LSTM at a given time step t against changes in temperature from its previous time step (i.e., $\text{Temp}_t - \text{Temp}_{t-1}$). Since the LSTM is trained for each station using its historical observations only, it tends to predict temperature values that are similar to the temperature of its previous time step, and thus, unable to anticipate rapid changes in the temperature time series. This is illustrated in Figure 1a, which shows the high correlation between LSTM prediction error and the temperature changes at a location. Furthermore, Figure 1b shows that these sudden changes tend to be preceded by large temperature changes at nearby locations in the previous time step. This observation suggests the importance of incorporating spatial information into weather prediction models. In fact, the correlation between LSTM prediction error and average temperature changes in the previous time step at nearby locations is indeed high (0.44). Thus, if the information about temperature changes in neighboring locations can be incorporated into LSTM, this may help improve its prediction accuracy.



(a) Correlation = 0.59 (b) Correlation = 0.34.

Fig. 1: Plot of changes in temperature at a weather station (x-axis) against (a) LSTM prediction error and (b) average temperature change at nearby locations in the previous time step.

In contrast, a convolutional neural network (CNN) is a deep learning approach for modeling spatial relationships in data.



(a) Correlation = 0.52 (b) Correlation = 0.56.

Fig. 2: Plot of changes in temperature at a weather station (x-axis) against (a) GCN prediction error and (b) temperature change at previous time step

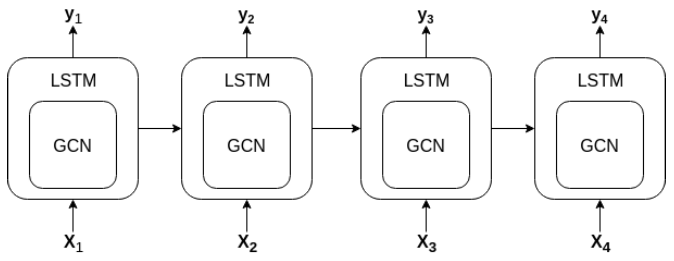


Fig. 3: Proposed model

CNNs have been successfully applied to images [13], videos [14], and other spatially gridded datasets [10]. For weather prediction, as the locations of weather stations are irregularly spaced, an ordinary convolutional neural network cannot be used for modeling the data. However, recent work has explored methods of extending convolutional neural networks so that they can be applied to arbitrary graphs [15] [16]. Just as each layer of an ordinary convolutional neural network represents each pixel as a linear combination of its neighboring pixels, a graph convolutional neural network represents each vertex in the graph as a linear combination of nearby vertices. A more detailed discussion of graph convolutional neural networks is reserved for Section III. We can apply graph convolutional neural networks to the weather prediction problem by treating each weather observation location as a vertex in a graph and form edges between vertices based upon the distance between the corresponding observation locations. However, Figures 2a and 2b show that a graph convolutional neural network encounters similar type of problems as an LSTM. Whereas an LSTM is limited by its inability to consider spatial relationships, the graph convolution is unable to consider temporal relationships.

The goal of this paper is to improve spatiotemporal prediction with deep learning by combining graph convolution with an LSTM. This can be achieved by replacing the fully connected layers within each LSTM cell with graph convolutional layers. However, care must be taken when constructing the graph to be used as input into the graph convolutional LSTM. For example, a typical approach is to consider the geographic distance between locations as edge weights of the graph. For weather prediction, the strength of the relationship between locations may also be affected by prevailing winds, topography, and other factors not directly related to distance. Consequently, distance alone may not be an optimal way to determine edge weights. To address this problem, we propose an approach to automatically learn the graph structure from data by treating the edge weights as model parameters. The learned adjacency matrix not only improves predictive performance, it also provides useful insights into how the current weather condition at a location influences the weather pattern at other locations. However, learning the full adjacency matrix introduces an additional $O(|V|^2)$ parameters, which can be computationally expensive for large graphs and may potentially lead to overfitting. To overcome these problems, we develop a technique called weighted graph convolutional LSTM (WGC-LSTM) for learning an adjacency matrix for the weather prediction problem. We propose two approaches to control overfitting in WGC-LSTM, a variant with a sparse adjacency matrix and another based on the low rank assumption. Experimental results performed on two real-world datasets demonstrate the effectiveness of the approach in terms of its predictive performance, computational efficiency, and interpretability of the learned graph. We also show that the high prediction accuracy can still be maintained but with faster runtime when using a low rank adjacency matrix for our graph convolution network.

II. Preliminaries

Consider a sequence of T weather observations at $|V|$ locations, i.e., $\mathbf{X}_1, \mathbf{X}_2, ..., \mathbf{X}_T$, where each $\mathbf{X}_t \in \mathbb{R}^{|V| \times d}$. The d features correspond to values of weather variables such as temperature, wind speed, and humidity from previous time steps. A full discussion of the specific weather variables used is reserved for Section IV-A. Given this sequence of historical observations, the goal is to predict the target sequence $\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_T$, where $\mathbf{y}_t \in \mathbb{R}^{|V|}$ is a vector of the targeted weather variable to be predicted at time t for all $|V|$ locations. Note that $\mathbf{y}_t$ is also a column in $\mathbf{X}_{t+1}$, i.e., the values of the target variable $\mathbf{y}_t$ at the current time step become one of the predictor variables at the next time step.

In this study, we will employ a deep learning approach for the weather prediction problem. A deep learning architecture can be thought of as layers of simple functions. Each layer takes an input, $\mathbf{h}^{(l-1)}$, from the layer below and maps it to an output, $\mathbf{h}^{(l)}$, using a function $f^{(l)}$. This can be expressed mathematically as: $\mathbf{h}^{(l)} = f^{(l)}(\mathbf{h}^{(l-1)})$. We denote the dimension of the input vector to the l^{th} layer as $d^{(l-1)}$ and the dimension of the output vector of the l^{th} layer as $d^{(l)}$. The input to the first layer ($\mathbf{h}^{(0)}$) corresponds to the $\mathbf{X}_t$ whereas the output of the final layer ($\mathbf{h}^{(L)}$) corresponds to $\mathbf{y}_t$.

Our proposed WGC-LSTM framework is a novel extension of two deep learning architectures—long short-term memory (LSTM) networks and convolutional neural networks. Below we provide some background information on these architectures to motivate discussion of our proposed framework in the next section. A graphical representation of our model is provided in Figure 3.

A. Long Short-Term Memory Networks

One of the most common ways of capturing temporal or sequential relations with neural networks is with a variation of recurrent neural networks called a long short-term memory network (LSTM) [17]. In contrast to an ordinary recurrent neural network, LSTMs are capable of learning temporal relationships extending over long periods of time while also alleviating the vanishing gradient problem present in ordinary recurrent neural networks.

The LSTM has an internal memory in each layer, $\mathbf{c}_t^{(l)}$, that it updates at each time step. The update at time t, $\tilde{\mathbf{c}}_t^{(l)}$ depends upon the output of the layer below at that time step, $\mathbf{h}_t^{(l-1)}$ as well as the output from the same layer at the previous time step, $\mathbf{h}_{t-1}^{(l)}$. So the change of the memory is given by the equation:

$$\tilde{\mathbf{c}}_t^{(l)} = \tanh(\mathbf{W}_c^{(l)}\mathbf{h}_t^{l-1} + \mathbf{U}_c^{(l)}\mathbf{h}_{t-1}^{(l)} + b_c^{(l)}) \tag{1}$$

In this equation and all equations that follow, any $\mathbf{W}$, $\mathbf{U}$, or b with a subscript is a trainable model parameter.

The amount that $\tilde{\mathbf{c}}_t^{(l)}$ is allowed to modify the memory is regulated by an input gate, $\mathbf{i}_t^{(l)}$ and the amount that the memory is allowed to leak from one time step to the next is regulated by the forget gate $\mathbf{f}_t^{(l)}$. The input and forget gate are determined by the following two equations:

$$\mathbf{f}_t^{(l)} = \sigma(\mathbf{W}_f^{(l)}\mathbf{h}_t^{(l-1)} + \mathbf{U}_f^{(l)}\mathbf{h}_{t-1}^{(l)} + b_f^{(l)}) \tag{2}$$

$$\mathbf{i}_t^{(l)} = \sigma(\mathbf{W}_i^{(l)}\mathbf{h}_t^{(l-1)} + \mathbf{U}_i^{(l)}\mathbf{h}_{t-1}^{(l)} + b_i^{(l)}) \tag{3}$$

Here σ represents the sigmoid function applied component wise to ensure that each component of the input and forget gates ranges between 0 and 1. The updated value of the memory is given by:

$$\mathbf{c}_t^{(l)} = \mathbf{i}_t^{(l)} \circ \tilde{\mathbf{c}}_t^{(l)} + \mathbf{f}_t^{(l)} \circ \mathbf{c}_{t-1}^{(l)} \tag{4}$$

where $\circ$ represents the Hadamard product. A portion of the memory "leaks" to form the LSTM output $\mathbf{h}_t$. The amount of memory allowed to leak is regulated by an output gate o_t, which is calculated as follows:

$$\mathbf{o}_t^{(l)} = \sigma(\mathbf{W}_o^{(l)}\mathbf{h}_t^{(l-1)} + \mathbf{U}_o^{(l)}\mathbf{h}_{t-1}^{(l)} + b_o^{(l)}) \tag{5}$$

$$\mathbf{h}_t^{(l)} = \mathbf{o}_t^{(l)} \circ \tanh(\mathbf{c}_t^{(l)}) \tag{6}$$

The final output of the model, $\hat{\mathbf{y}}_t$ is:

$$\hat{\mathbf{y}}_t = \mathbf{h}_t^{(L)} = \mathbf{w}_{prediction}^T \mathbf{h}_t^{(L-1)} + b_{prediction}, \tag{7}$$

where $\mathbf{w}_{prediction}$ and $b_{prediction}$ are parameters used to convert the output of the last layer of the LSTM into the final prediction.

B. Convolutional Neural Networks

Convolutional neural networks are designed to learn representations of data by taking a linear combination of each data point with its neighbors with an operation called convolution. For example, we may learn the representation of each point in a time series by considering the point itself along with the time points immediately before and after it. As an illustration, let $\mathbf{x}$ be a univariate time series of length T, $\mathbf{w}$ be a weight vector, and b is a bias. Then the t^{th} element of the convolution of $\mathbf{x}$ with $\mathbf{w}$ can be expressed as:

$$(\mathbf{x} * \mathbf{w})_t = \sum_{j=-k}^{k} \mathbf{x}_{t+j}\mathbf{w}_{j+k} + b$$

where $*$ represents the convolution operation. Note that t^{th} element of the convolution depends upon points of $\mathbf{x}$ within k steps away from the t^{th} element of $\mathbf{x}$.

In contrast, the representation learned by a fully connected neural network depends upon the entire time series rather than localized segments of it. Thus, fully connected neural networks would require $O(T^2 \times d^{(l-1)} \times d^{(l)})$ parameters to be learned for a time series of length T, whereas convolutional neural networks with filter size k would only require $O(k \times d^{(l)} \times d^{l-1})$ parameters with $k << T$.

III. Proposed Approach

Convolution is useful to handle datasets with autocorrelated observations. For example, convolutional neural networks are often applied to images as nearby pixels are correlated with each other. Similarly, convolution has been applied to time series, where the measurements at consecutive points in time are autocorrelated. However, convolutional neural networks are typically formulated in such a way that it can be applied only to datasets where observations are taken at regular intervals. In the next section we discuss how to address this problem using graph convolution and how to incorporate the graph convolution into an LSTM.

A. Graph Convolution

Convolution can be applied to spatial data by imposing a graph structure on the data. Several different approaches to graph convolution have been discussed in the literature [16] [18] [15]. In this section we describe the approach used in this paper. For a graph convolutional neural network with L layers, the output of each layer, $H^{(l)}$, is a $|V| \times d^{(l)}$ matrix with each row representing one of the vertices (i.e. locations) of the graph. The number of vertices in each layer will stay the same, only the dimension of the vertex representation $d^{(l)}$ changes. The graph convolution of a set of vertices, $\mathbf{H}^{(l-1)}$ with a matrix of weights, $\mathbf{W}^{(l)}$ is defined as:

$$\mathbf{H}^{(l-1)} * W^{(l)} = \mathbf{A}\mathbf{H}^{(l-1)}\mathbf{W}^{(l)} \tag{8}$$

where $\mathbf{A}$ is the adjacency matrix associated with the graph and σ is a non-linear activation function. The product $\mathbf{A}\mathbf{H}_t^{(l-1)}$ will sum up the features in the neighborhood around each vertex, while right multiplying this product by $\mathbf{W}^{(l)}$ allows us to consider linear combinations of features.

We can then utilize graph convolution in the layers of a neural network. The operations performed in each graph convolutional layer can be expressed as:

$$\begin{aligned}\mathbf{H}^{(l)} &= f^{(l)}(\mathbf{H}^{(l-1)}) \\ &= \sigma(\mathbf{H}^{(l-1)} * W^{(l)}) \\ &= \sigma(\mathbf{A}\mathbf{H}^{(l-1)}\mathbf{W}^{(l)})\end{aligned} \tag{9}$$

In our proposed approach, the adjacency matrix is shared between all layers of the network to reduce the number of model parameters.

B. Graph Convolutional LSTM

Graph convolutions allow the model to utilize spatial relationships and by incorporating them into a long short-term memory network it is possible for the model to consider spatial and temporal relationships simultaneously. We call the combination of graph convolution inside an LSTM a graph convolutional LSTM (GC-LSTM). The modified LSTM equations are:

$$\begin{aligned}
\mathbf{C}_t^{(l)} &= \mathbf{I}_t \circ \tilde{\mathbf{C}}_t^{(l)} + \mathbf{F}_t^{(l)} \circ \mathbf{C}_{t-1}^{(l)} \\
\mathbf{F}_t^{(l)} &= \sigma(\mathbf{H}_t^{(l-1)} * \mathbf{W}_f^{(l)} + \mathbf{H}_{t-1}^{(l)} * \mathbf{U}_f^{(l)} + \mathbf{b}_f^{(l)}) \\
\mathbf{I}_t^{(l)} &= \sigma(\mathbf{H}_t^{(l-1)} * \mathbf{W}_i^{(l)} + \mathbf{H}_{t-1}^{(l)} * \mathbf{U}_i^{(l)} + \mathbf{b}_i^{(l)}) \\
\tilde{\mathbf{C}}_t^{(l)} &= tanh(\mathbf{H}_t^{(l-1)} * \mathbf{W}_c^{(l)} + \mathbf{H}_{t-1}^{(l)} * \mathbf{U}_c^{(l)} + \mathbf{b}_c^{(l)}) \\
\mathbf{H}_t^{(l)} &= \mathbf{O}_t^{(l)} \circ tanh(\mathbf{C}_t^{(l)}) \\
\mathbf{O}_t^{(l)} &= \sigma(\mathbf{H}_t^{(l-1)} * \mathbf{W}_o^{(l)} + \mathbf{H}_{t-1}^{(l)} * \mathbf{U}_o^{(l)} \\
&\quad + \mathbf{C}_t^{(l)} * \mathbf{V}_o^{(l)} + \mathbf{b}_o^{(l)})
\end{aligned}$$

where $*$ represents graph convolution. There are two primary differences between these equations and the original LSTM formulation given in Equations 1-6. First, many of the variables are now matrices with $|V|$ rows rather than vectors. Second, all the matrix multiplications in the ordinary LSTM equations are replaced by graph convolutions.

C. Weighted Graph Convolutional LSTM

Previous work has often applied graph convolutions to datasets with known graph structure or used simple heuristics to form the graph. However, in the case of weather prediction, there are many factors that can potentially affect the spatial relationships between different locations, which makes it difficult to manually create the adjacency matrix. To overcome this problem, we propose to treat the entries of the adjacency matrix as a model parameter. This gives the model complete freedom to determine, in a data driven way, how to organize the relationships between vertices. Once this adjacency matrix is learned it can be examined for insight into weather phenomena as well as future research into climate networks.

One published work [19] has also explored the possibility of using graph convolutional LSTMs for language processing and traffic prediction applications based on diffusion convolution [15]. Specifically, the $j^{(th)}$ column of the convolution between the $|V| \times d^{(l)}$ matrix of vertex features $\mathbf{H}$ with an order 3 tensor of parameters with dimension $d^{(l-1)} \times d^{(l)} \times K$ is defined as:

$$(\mathbf{H} * \mathbf{W})_{:,j} = \sum_{i=1}^{d_{in}} g_{\mathbf{W}_{i,j}}(\mathbf{L})\mathbf{H}_{:,i},$$

where $\mathbf{L}$ is the graph Laplacian. If, $\mathbf{1}$ is the identity matrix and $T_k(\mathbf{L})$ is the order k Chebyshev polynomial of $\mathbf{L}$ and λ_{max} is the largest eigenvalue of $\mathbf{L}$ then:

$$g_{\mathbf{w}}(\mathbf{L})\mathbf{x} = \sum_{k=0}^{K-1} \mathbf{w}_k T_k(\tilde{\mathbf{L}})$$
$$\tilde{\mathbf{L}} = 2\mathbf{L}/\lambda_{max} - \mathbf{1}$$

However, learning the weighted graph convolution using the Chebyshev polynomial approach is a very expensive operation. This is because if we modify the graph Laplacian at each training step, then the value of λ_{max} would have to be recomputed. As a consequence, these previous approaches assume that the adjacency matrix of the graph is known and fixed unlike our proposed WGC-LSTM approach, which learns the adjacency matrix from the training data. We avoid the costly calculation of eigenvalues by using the following graph convolution operation as described in Section III-A:

$$\mathbf{H} * \mathbf{W} = \mathbf{AHW}$$

where $\mathbf{W}$ is a $d^{(l-1)} \times d^{(l)}$ parameter matrix.

In our framework, the value of $\mathbf{A}$ can be trained using gradient descent based methods such as Adam [20] as long as the loss function is a differentiable function. However, the edge weights introduce an additional $|V|^2$ parameters into our model. The addition of such a large number of parameters can potentially lead to overfitting. It is worth noting, however, that as bad as the addition of $|V|^2$ model parameters may initially seem, even with a fully learned adjacency matrix this convolution-like operation still has fewer parameters than a similar fully connected network. If we are given a graph of $|V|$ vertices each with dimension $d^{(l)}$, mapping to a $d^{(l+1)}$ dimensional representation for each vertex, a fully connected neural network layer would require $|V|^2 d^{(l)} d^{(l+1)}$ parameters since every feature of each of the output vertices depends on all the features of all the vertices in the input. In contrast, using graph convolution with learned edge weights requires only $|V|^2 + d^{(l)} d^{(l+1)}$ parameters.

For the weather prediction task we expect that the vast majority of potential edges in the dataset are unnecessary as any given station will be related to just a small percentage of the other stations. One way to create sparser models is by using L1 regularization, $\lambda\|\mathbf{A}\|_1$. The choice of the value for the hyper-parameter λ determines the desired sparsity of the adjacency matrix. Another way to reduce the number of parameters is to discard some of the edges in the graph before learning. This makes sense for weather prediction, which is driven primarily by the current weather at nearby locations. Consequently, we can remove edges between distant weather stations. When we pre-sparsify the adjacency matrix by removing edges between distant stations, we call the model a **sparse WGC-LSTM**. An added bonus of using a sparse adjacency matrix is that it enables us to compute the convolution operation using sparse matrix multiplication rather than dense matrix multiplication thus reducing the amount of computation required. However, in practice deep learning models are often deployed on GPUs which cannot evaluate sparse matrix operations very quickly. To better take advantage of the unique capabilities of GPUs we propose to factorize the adjacency matrix[1] $A = U^T V$, where U and V are $k \times |V|$ dimensional matrices. Factorizing the adjacency matrix in this way can significantly reduce the number of model parameters while also reducing the required computation in a GPU friendly way. Computing this low rank graph convolution has time complexity $O(k \times |V| \times d + d^2)$. The low rank assumption is justified because we are dealing with spatial data and we expect the intrinsic dimensionality of the data to be much lower than the number of stations in the dataset. We call this approach **low-rank WGC-LSTM**.

IV. Experimental Evaluation

We have performed extensive experiments using two real-world datasets to evaluate the performance of our proposed WGC-LSTM framework. In this section, we describe the datasets and baseline algorithms used along with experimental setup and the results obtained.

A. Weather Datasets

There are two real-world weather datasets are used in this study: (1) Radiosonde (weather balloon) data from the Integrated Global Radiosonde Archive (IGRA)[2] and (2) NOAA Global Surface Summary of the Day (GSOD) dataset[3]. In both datasets, we consider only measurements taken from locations in the continental United States. Furthermore, a location is included in the dataset only if less than 10% of their data is missing. Any missing values for the selected location are linearly interpolated. A summary of the datasets is given in Table I.

For the IGRA dataset, measurements of dew point, wind speed, wind direction, temperature, and geopotential height were obtained from weather balloons released at particular heights twice a day. Our goal is to predict one of the weather variables (temperature or wind speed) twelve hours in the future based on the weather measurements from its previous time steps. The resulting dataset contains weather measurements from 67 locations, spanning the years between 1996 and 2015. Measurements from the first 10,000 time steps are used for training, the next 2,300 time steps are used as validation set, and the remaining 2,300 time steps are used as a dedicated test set. All the variables are standardized so that their mean is zero and standard deviation is one.

For the GSOD dataset, daily measurements of minimum, maximum, and average temperature are recorded along with the maximum and average wind speed as well as average surface pressure from various weather stations. Our objective is to predict one of the weather variables (temperature or wind speed) at every selected weather station for the next day. Similar to the IGRA dataset, all variables are standardized to mean zero and unit variance. The dataset contains measurements from 332 weather stations, spanning the years from 1990 to 2016. Measurements from the first 7,800 days are used as the training set and the next 1,000 days are used as validation set. Finally, we reserve the data from another 1,000 days as test set.

[1]If the adjacency matrix is expected to be symmetric then this can be further simplified by factorizing A as $A = U^T U$

[2]https://www.ncdc.noaa.gov/data-access/weather-balloon/integrated-global-radiosonde-archive

[3]https://data.noaa.gov/dataset/global-surface-summary-of-the-day-gsod

TABLE I: Summary of weather datasets.

Dataset	# Locations	Time Period	Training Size
IGRA	67	1996-2015	10,000
GSOD	332	1990-2016	7,800

B. Baseline Algorithms

We compared the performance of WGC-LSTM against the baseline methods listed below.

1) **Previous Time Step:** This baseline simply predicts the weather observations at the next time step to be the same as those at the current time step.
2) **Gaussian Process:** This baseline uses spatio-temporal Gaussian process regression. Gaussian process regression makes predictions at the next time step by taking a linear combination of the observations at previously observed time steps. We use a kernel that is the product of a spatial kernel and a temporal kernel. The spatial kernel is an RBF kernel based on the geographic distance between two stations. The temporal kernel is an RBF kernel that takes the time differences between observations as input. Only the 10 most recent observations are used to predict the next time step.
3) **Graph Convolution:** This baseline uses only $\mathbf{X}_t$ to predict $\mathbf{Y}_t$ with a graph convolutional neural network. This approach primarily relies on spatial relationships to make predictions and has very little temporal information.
4) **Local LSTM:** This baseline trains an ordinary LSTM to predict the future weather at a given location based on weather information from its previous time steps. This is equivalent to using a graph convolutional LSTM with an identity matrix as its spatial adjacency matrix.
5) **Global LSTM** At each time step we combine the data from all weather stations into a single vector to form a multivariate time series. This multivariate time series is then used as the input to an LSTM. The LSTM will be tasked with predicting a multivariate time series of length V, where each element of the prediction vector corresponds to the forecasted weather at a particular station. In this approach, the LSTM's prediction depends not only on the weather condition at a given location, but also on the weather conditions at other locations as well. Due to the large number of parameters in the model, an L2 regularizer is placed on the LSTM parameters to avoid overfitting.
6) **GC-LSTM (Fixed Adjacency)** This corresponds to a graph convolutional LSTM approach with a fixed adjacency matrix, where the edge weights are computed

based on a Gaussian kernel on the geographic distance between each pair of locations, i.e., $A_{ij} = \exp[-\frac{d_{ij}^2}{2\sigma^2}]$.

C. Experimental Setup

All LSTMs were trained with a variant of the gradient descent algorithm called Adam [20]. Each LSTM based model has the following hyper parameters: weight of an L_2 norm regularization term, number of LSTM layers, number of hidden units. The low-rank WGC-LSTM has a hyper-parameter controlling the rank of the adjacency matrix. The rank is varied between 2 and one half of the number of stations in the dataset. Hyper-parameters were chosen through a random search [21]. The initial learning rate was set to 0.01 and periodically reduced.

D. Experimental Results

The following experiments are performed to evaluate the proposed approach. First we compare the R^2 of WGC-LSTM against the baseline methods. Second, we investigate the effect of reducing the rank of the adjacency matrix. Finally, we examine the impact of varying the number of LSTM layers on the predictive performance.

1) Comparison against Baseline Methods: Table II shows the results of our experiments comparing graph convolutional LSTMs (WGC-LSTM and GC-LSTM) against other baseline methods. For WGC-LSTM, we use the sparse implementation of the framework and reported the results on both IGRA and GSOD datasets, with temperature and wind speed as the response variables. The results in Table II demonstrate the superiority of both GC-LSTM and WGC-LSTM on all four prediction tasks. For wind speed prediction the improvement in R^2 is greater than 3% when compared against the strongest baseline. For temperature prediction, the gain is more modest (aproximately 1%), which is not surprising as the simple baseline of using temperature from previous time step is sufficient to obtain high accuracy. The global LSTM is also capable of making accurate predictions as it considers the weather observations at multiple locations. However, the weighted graph convolutional LSTMs still outperform the global LSTM by 2% (for temperature prediction) and 4% (for wind speed prediction), which suggests that the graph convolutional LSTMs are more effective at utilizing observations from multiple locations to improve their predictions while using fewer parameters. We also observe significant performance gain using WGC-LSTM compared to GC-LSTM on all tasks, which suggests the advantages of treating edge weights as learned parameters in graph convolutional LSTM. The performance gain is less significant for temperature prediction as the R^2 is already very close to 1.

A more detailed analysis on the performance comparison is shown in Tables III and IV. We compare the predictive performance of the competing methods on a location by location basis. Each number represents the percentage of locations in which the method specified in the given row outperformed the method specified in the given column. We see that the proposed WGC-LSTM method is superior than all

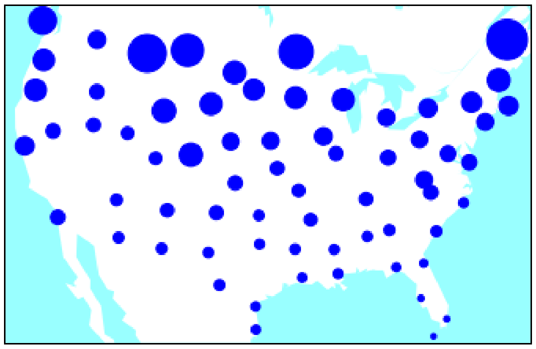


Fig. 4: Plot of IGRA stations with station size scaled to be proportional to the cubic value of MSE for temperature prediction at that station. We see that there are larger errors in the north. Correlation between a station's average error and station temperature standard deviation is 0.82.

other baseline methods for the majority of the locations on the GSOD dataset. In particular, WGC-LSTM outperforms other competing methods between 90% to 99% of the locations for temperature prediction and by more than 93% of the locations for wind speed prediction.

In Figure 4 we plot a map of the MSE error for temperature prediction at each station. Each circle represents a station and the size of the circle is scaled according to the cubic value of MSE for temperature prediction at each station. Observe that the errors tend to be smaller near the coastal and southern stations and larger for those located to the north. Further analysis shows that the correlation between a station's average squared error and standard deviation of its temperature is 0.82. This suggests the lower error for stations located to the south and along the coast could be due to the less variability in their temperature compared to the northern stations.

E. Low-Rank versus Sparse WGC-LSTM

This section examines the effect of reducing the rank of the adjacency matrix on predictive performance and runtime. A low rank adjacency matrix can be created by factorizing the adjacency matrix into the product of two $|V| \times k$ matrices where k is the matrix rank. By factorizing the adjacency matrix in this way we help to reduce run time while also reducing the number of model parameters. In Figure 5a we see that it is possible to obtain strong performance on the GSOD wind speed prediction task with a rank 40 matrix (the original matrix is 332×332).

In addition, we compare the runtime of a model with a low rank adjacency matrix against a model with a full rank adjacency matrix sparsified by removing edges between very distant locations. We see in Figure 5b that a low rank adjacency matrix requires much less time to compute than a sparse adjacency matrix with the same number of model parameters. This large discrepency is due to the fact that the GPUs used in many deep learning tasks are not optimized

	IGRA		GSOD	
Approach	Temperature	Wind Speed	Temperature	Wind Speed
Previous Time Step	0.8766	0.3554	0.9103	0.1739
Gaussian Process	0.8940	0.4467	0.9016	0.1888
Graph Convolution (without LSTM)	0.9094	0.5567	0.9299	0.3863
Local LSTM	0.9187	0.5400	0.9476	0.4796
Global LSTM	0.9423	0.6147	0.9523	0.5600
GC-LSTM (fixed adjacency)	0.9353	0.6117	0.9459	0.4930
WGC-LSTM (learned adjacency)	**0.9523**	**0.6412**	**0.9705**	**0.5982**

TABLE II: Comparison of R^2 value for graph convolutional LSTM against other baseline methods.

	Gaussian Process	Graph Convolution	Local LSTM	Global LSTM	GC-LSTM	WGC-LSTM
Gaussian Process	0.0%	0.0%	0.0%	0.3%	0.0%	0.3%
Graph Convolution	100.0%	0.0%	0.6%	26.8%	6.6%	2.4%
Local LSTM	100.0%	99.4%	0.0%	44.6%	60.2%	9.6%
Global LSTM	99.7%	73.2%	55.4%	0.0%	60.2%	0.6%
GC-LSTM	100.0%	93.4%	39.8%	39.8%	0.0%	6.3%
WGC-LSTM	99.7%	97.6%	90.4%	99.4%	93.7%	0.0%

TABLE III: Percent of stations for which the model on each row outperforms the model in the column on the temperature prediction task for the GSOD dataset.

	Gaussian Process	Graph Convolution	Local LSTM	Global LSTM	GC-LSTM	WGC-LSTM
Gaussian Process	0.0%	2.4%	0.0%	0.3%	0.3%	0.0%
Graph Convolution	97.6%	0.0%	2.1%	4.8%	13.6%	2.7%
Local LSTM	100.0%	97.9%	0.0%	13.9%	47.0%	6.9%
Global LSTM	99.7%	95.2%	86.1%	0.0%	89.2%	3.6%
GC-LSTM	99.7%	86.4%	53.0%	10.8%	0.0%	1.2%
WGC-LSTM	100.0%	97.3%	93.1%	96.4%	98.8%	0.0%

TABLE IV: Percent of stations for which the model on each row outperforms the model in the column on the wind speed prediction task for the GSOD dataset.

for sparse-dense matrix multiplications but are optimized for dense-dense matrix multiplications.

F. Number of Layers

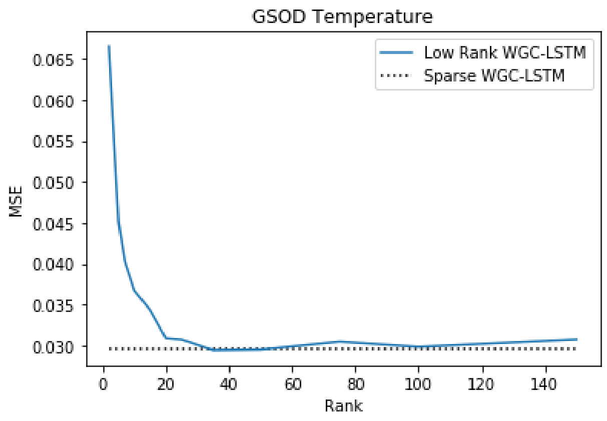
Next we consider the impact of the number of layers on model performance. Figure 6 shows predictive performance as a function of the number of LSTM layers in the network. The y-axis shows the maximal R^2 reached across all runs for graph convolutional LSTMs with a learned adjacency matrix and a given number of layers. Only 2 layers are necessary to achieve near optimal performance on the IGRA dataset while the GSOD dataset benefits from increasing the number of layers further. Since each added layer represents an additional multiplication by the adjacency matrix, adding more layers to the WGC-LSTM allows it to incorporate weather information from larger areas.

G. Learned Adjacency Matrix Visualization

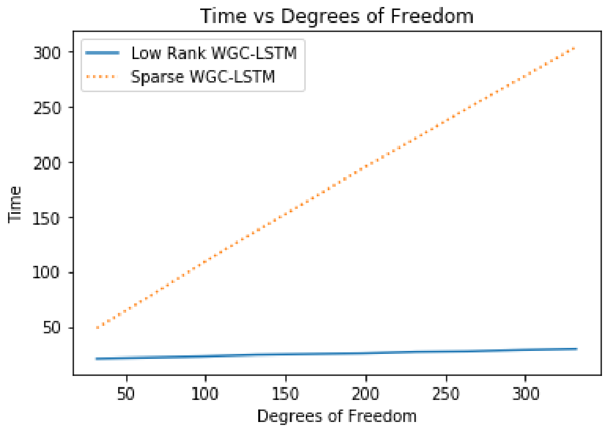
In this section we visualize the learned adjacency matrix in two ways. First, we directly visualize the adjacency matrix by plotting the most important weights. Second, we use the learned adjacency matrix to cluster stations. Throughout this section we compare results for models with the low rank assumption against models with a sparsity assumption.

In Figure 8 we visualize the edges learned by a WGC-LSTM for the IGRA temperature prediction task. To do this, we take the absolute value of the learned adjacency matrix and find the largest edge weight in each row of the adjacency matrix. The edge corresponding to each row's largest weight is then plotted along with its direction. In cases where both A_{ij} and A_{ji} would be plotted we display the edge as a purple line segment. We visualize a sparse full rank adjacency matrix that includes edges between stations less than 1400 km long and a low rank adjacency matrix with edges longer than 1400 km removed before plotting. What we observe is that the most significant edge for each station closely matches what we would expect from climate research. In particular, we see that stations are almost always most heavily influenced by stations to the west. This phenomenon is clearer for the sparse adjacency matrix than the low rank adjacency matrix.

After learning the adjacency matrix it becomes possible to assign different weather stations to different regions using a clustering algorithm. Our choice of clustering algorithm depends on the structure of the adjacency matrix. For a sparse adjacency matrix, we apply spectral clustering. When using a low rank adjacency matrix, we concatenate the learned latent factors and use them as input to K-Means clustering. We compare the clusters obtained from the adjacency matrix learned in the IGRA temperature prediction task to the clusters found using a fixed adjacency matrix based on the RBF kernel computed from the geographic distance. These clusters are formed using spectral clustering. The clusters resulting from the fixed adjacency matrix are generally quite "spherical" and are spatially contiguous. The clusters derived from the sparse adjacency matrix tend to be geographically contiguous but not quite as spherical as those arising from the fixed adjacency matrix. This is most apparent when observing stations located



(a) MSE for temperature prediction as a function of adjacency matrix rank.



(b) Average time per training iteration (for the GSOD dataset) as a function of degrees of freedom. The low rank WGC-LSTM results in a much faster model than the sparse WGC-LSTM.

Fig. 5: Comparison between low rank WGC-LSTM and sparse WGC-LSTM.

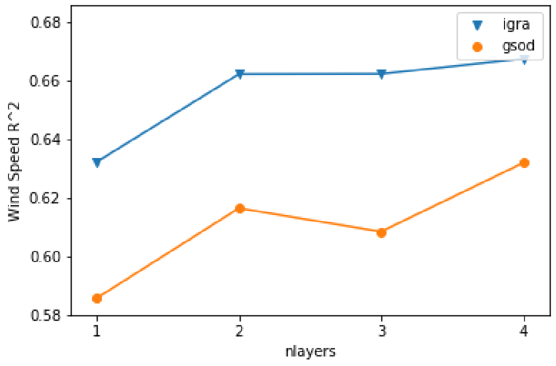


Fig. 6: Prediction error for wind speed as a function of the number of layers in WGC-LSTM.

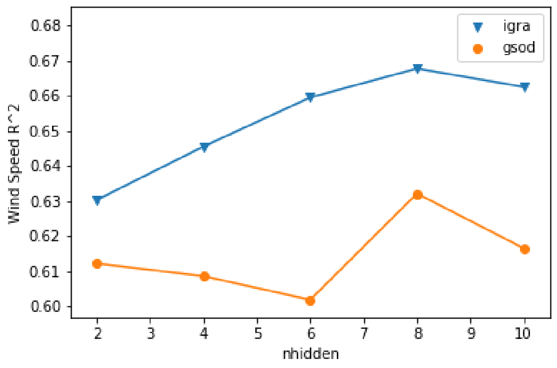
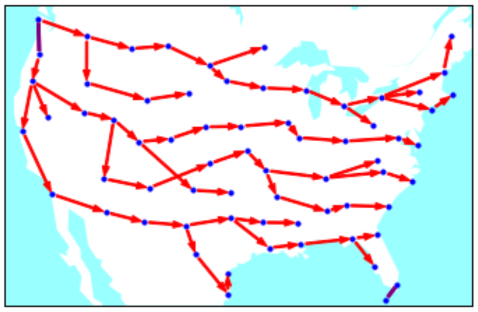
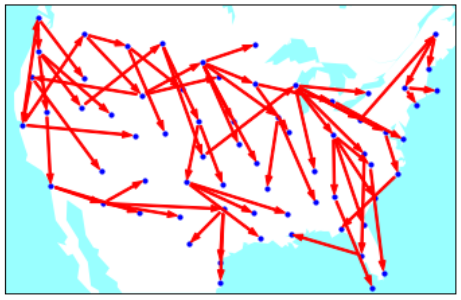


Fig. 7: Prediction error for wind speed as a function of the number of layers in WGC-LSTM.



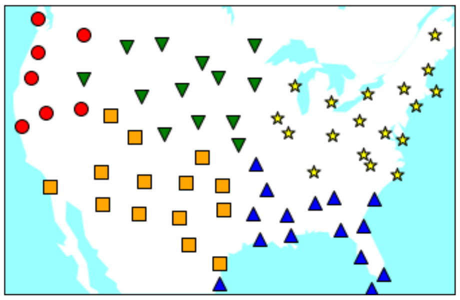
(a) Sparse adjacency matrix

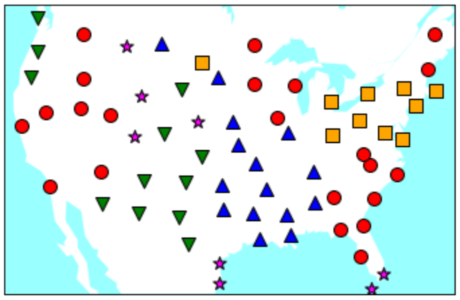


(b) Low rank adjacency matrix

Fig. 8: Comparison between the sparse and low rank adjacency matrices learned for temperature prediction on the IGRA dataset.

around the Gulf of Mexico. Nevertheless, the clusters obtained from the sparse adjacency matrix are still quite similar to those found using the fixed adjacency matrix. In contrast, when the low-rank adjacency matrix is used for clustering, the stations within a cluster may be geographically distant from each other. Its possible that the clusters formed based on the low rank model are less geographically contiguous because the adjacency matrix formed by the low rank factors is a dense matrix with edges spanning the entire United States whereas the sparse adjacency matrix only has edges between nearby


(a) Sparse adjacency matrix clustering


(b) Low rank adjacency clustering

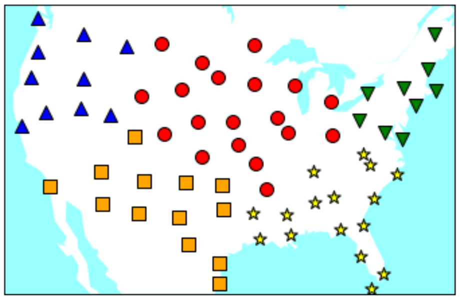

(c) Clustering based on geographic distance

Fig. 9: Comparing the results of clustering performed on the sparse and low rank adjacency matrices learned for temperature prediction on the IGRA dataset are used for clustering. An adjacency matrix formed from geographic distance is also clustered.

stations.

In both of these tasks, the sparsity assumption yielded more interpretable results. This suggests that, though low rank adjacency matrices may be much more computationally efficient, sparse adjacency matrices may still be useful in situations where the interpretability of the learned adjacency matrix is highly important.

V. Related Work

A. Deep Learning

Over the past decade, deep learning has exploded in popularity. Thanks to the availability of large labeled datasets and access to cheap computing power (especially GPUs), deep neural networks have been able to achieve strong results on challenging tasks. They have successfully been applied to object recognition [13], speech recognition [11], and machine translation [12].

To date, most work applying deep learning to spatiotemporal data has focused on sequences of gridded observations like videos. Because each sample in the sequence is gridded convolutional layers can easily be incorporated into the deep learning. For example, [10] develops that convolutional LSTM model that incorporates convolution into an LSTM and applies their model to the problem of precipitation now-casting based on sequences of gridded radar observations.

B. Graph Convolution

Many of the most successful applications of deep learning involve convolutional neural networks. Consequently, there has been a wide range of research exploring the possibility of generalizing convolutional neural networks so that they are applicable to a broader class of problems.

[18] devises a neural network architecture based on graph convolutions and apply it to graph classifications problems on synthetic datasets. Building on this, [15] use a polynomial approximation of graph convolution that is computationally efficient and localized. The effectiveness of the method is demonstrated for the classification of vertices in a graph. [16] further approximates graph convolution and also proposes a technique for applying graph convolutions to semi-supervised learning problems. Our approach to graph convolution is most similar to [16] but our approach to graph convolution is distinct from theirs in several important ways. First, [16] used a dataset with known graph structure and weights. In contrast, we learn edge weights over the course of training. Second, their approach is based on multiplication by the normalized graph Laplacian of a self-loop augmented graph and is formulated in the spectral domain. Our approach however is formulated in the vertex domain and involves multiplication by a learned adjacency matrix.

One previously published work has also used graph convolutional LSTMs for prediction. In [19] the authors use a graph convolutional LSTM for traffic prediction. We distinguish our work in the following ways. First, in [19], the authors assume that the graph structure is known. However, we show that incorporating edge weights as a part of the learning process can improve predictive performance. In addition, [19] use a formulation of graph convolution based on random walks while the graph convolution operation employed in this paper is formulated in the vertex domain. Since we are learning the edge weights graphs, the approach to graph convolution utilized in [19] would require computing the powers of the adjacency matrix at each training step which takes $O(|V|^3)$ time. We avoid this costly computation in our formulation.

C. Weather Prediction

The most common approach to weather prediction is based on numerical simulations [1]. In addition, data-driven approaches such as [2] [3] have also been developed. The framework in [2] contains three components: the first component consists of a regression model at each location for which they would like to produce predictions that predicts weather at that location in the future based on the historical weather measurements at that location, the second component enforces smoothness constraints using a gaussian process with a custom kernel, and the third component uses a restricted Boltzman machine to enforce physical constraints on the weather variables at each location. [10] uses a convolutional LSTM to perform precipitation now casting using ground based radar observations. In [3], the authors propose an extension of decision trees to spatiotemporal data that they evaluate on weather prediction tasks. Finally, in [22], the authors use an LSTM for soil moisture prediction.

VI. Conclusions and Future Work

In conclusion, an approach to weather prediction using graph convolutional LSTMs was proposed. We evaluated our WGC-LSTM method against several baselines and found that it outperformed the baselines on all prediction tasks. In particular, we found that learning the edge weights gave noticeable performance improvements. We also evaluated predictive performance as a function of adjacency matrix rank and found that a relatively small rank was necessary in order to reach near optimal performance. This makes it possible to reduce the computational requirements of training and reduce the chance of overfitting without compromising prediction quality.

In this paper we focused on making predictions using a fixed graph. Future work might explore ways to extend the framework proposed here to allow weather stations to be added or removed over time. Another possible research direction is to explore ways of leveraging datasets where different vertices have different predictor variables. For example, some vertices may contain observations of temperature and wind speed while others contain only precipitation measurements. In addition, we will also explore the possibility of designing a hybrid approach that leverages the computational benefits of the low rank structure and the interpretability yielded by sparse models simultaneously.

Acknowledgement

This research is partially supported by the National Science Foundation under grant #IIS-1615612. Any use of trade, firm, or product names is for descriptive purposes only and does not imply endorsement by the U.S. Government. This work was supported in part by Michigan State University through computational resources provided by the Institute for Cyber-Enabled Research.

References

[1] P. Bauer, A. Thorpe, and G. Brunet, "The quiet revolution of numerical weather prediction," *Nature*, vol. 525, no. 7567, pp. 47–55, Sep. 2015.

[2] A. Grover, A. Kapoor, and E. Horvitz, "A Deep Hybrid Model for Weather Forecasting." ACM Press, 2015, pp. 379–386.

[3] A. McGovern, N. C. Hiers, M. Collier, D. J. Gagne II, and R. A. Brown, "Spatiotemporal Relational Probability Trees: An Introduction," in *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, ser. ICDM '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 935–940.

[4] M. Hossain, B. Rekabdar, S. J. Louis, and S. Dascalu, "Forecasting the weather of Nevada: A deep learning approach," in *2015 International Joint Conference on Neural Networks (IJCNN)*, Jul. 2015, pp. 1–6.

[5] N. Nasrollahi, K. Hsu, and S. Sorooshian, "An Artificial Neural Network Model to Reduce False Alarms in Satellite Precipitation Products Using MODIS and CloudSat Observations," *Journal of Hydrometeorology*, vol. 14, no. 6, pp. 1872–1883, Jul. 2013.

[6] J. N. Liu, Y. Hu, J. J. You, and P. W. Chan, "Deep neural network based feature representation for weather forecasting," in *Proceedings on the International Conference on Artificial Intelligence (ICAI)*. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2014, p. 1.

[7] R. J. Kuligowski and A. P. Barros, "Localized Precipitation Forecasts from a Numerical Weather Prediction Model Using Artificial Neural Networks," *Weather and Forecasting*, vol. 13, no. 4, pp. 1194–1204, Dec. 1998.

[8] Y. Tao, X. Gao, K. Hsu, S. Sorooshian, and A. Ihler, "A Deep Neural Network Modeling Framework to Reduce Bias in Satellite Precipitation Products," *Journal of Hydrometeorology*, vol. 17, no. 3, pp. 931–945, Jan. 2016.

[9] M. A. Zaytar and C. E. Amrani, "Sequence to Sequence Weather Forecasting with Long Short-Term Memory Recurrent Neural Networks," *International Journal of Computer Applications*, vol. 143, no. 11, pp. 7–11, Jun. 2016.

[10] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-k. Wong, and W.-c. WOO, "Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 802–810.

[11] A. Graves, N. Jaitly, and A. r. Mohamed, "Hybrid speech recognition with Deep Bidirectional LSTM," in *2013 IEEE Workshop on Automatic Speech Recognition and Understanding*, Dec. 2013, pp. 273–278.

[12] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105.

[14] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2014, pp. 1725–1732.

[15] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering," in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Curran Associates, Inc., 2016, pp. 3844–3852.

[16] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations (ICLR)*, 2017.

[17] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[18] J. Bruna, W. Zaremba, A. Szlam, and Y. Lecun, "Spectral networks and locally connected networks on graphs," in *International Conference on Learning Representations (ICLR)*, 2014.

[19] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," in *International Conference on Learning Representations (ICLR '18)*, 2018.

[20] D. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *International Conference on Learning Representations (ICLR)*, 2015.

[21] J. Bergstra and Y. Bengio, "Random Search for Hyper-Parameter Optimization," *Journal of Machine Learning Research*, vol. 13, no. Feb, pp. 281–305, 2012.

[22] F. Kuai, S. Chaopeng, K. Daniel, and Y. Xiao, "Prolongation of smap to spatiotemporally seamless coverage of continental u.s. using a deep learning neural network," *Geophysical Research Letters*, vol. 44, no. 21, pp. 11,030–11,039.