ABSTRACT

Cellular automata are discrete models that can be used to simulate many physical systems. Cellular automata have been used to model gas diffusion, different types of chemical reactions, population growth, and land use change over time. Recent research into cellular automata networks has shown that if sparse long range connections are added to a cellular automata, then it will exhibit properties of complex networks. Furthermore, research into modeling climate systems has shown that modeling the global climate as a complex network can be used to predict individual climate variables. In this work we attempt to connect these ideas by simulating global climate variables, from the National Center for Environmental Prediction / National Center for Atmospheric Research Reanalysis 1 Dataset, as a cellular automata model and as a cellular automata network model.

In our experiments we use neural networks as the cellular automata transition functions, using both single and multi-variable data. The results of our work suggest that cellular automata networks are better at modeling climate variables than standard cellular automata and that cellular automata based modeling is a viable approach to modeling climate data.
# TABLE OF CONTENTS

ABSTRACT ....................................................... ii

LIST OF FIGURES .............................................. iv

INTRODUCTION .................................................. 1

LITERATURE SURVEY ........................................... 4

EXPERIMENT DESIGN .......................................... 14

EXPERIMENTAL WORK .......................................... 24

CONCLUSION .................................................... 35

BIBLIOGRAPHY .................................................. 37

APPENDICES .................................................... 41
LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Wolfram’s Rule 30, A one dimensional cellular automaton.</td>
<td>6</td>
</tr>
<tr>
<td>2.2</td>
<td>1-Dimensional Cellular Automata with a neighborhood radius of 1.</td>
<td>7</td>
</tr>
<tr>
<td>2.3</td>
<td>2-Dimensional Cellular Automata with an R-radial neighborhood radius of 1 (von Neumann Neighborhood).</td>
<td>7</td>
</tr>
<tr>
<td>2.4</td>
<td>2-Dimensional Cellular Automata with an R-axial neighborhood radius of 1 (Moore Neighborhood).</td>
<td>8</td>
</tr>
<tr>
<td>3.1</td>
<td>More Neighborhood with a long range connection.</td>
<td>22</td>
</tr>
<tr>
<td>4.1</td>
<td>$r^2$ values for linear regression model predicting Air Temperature.</td>
<td>25</td>
</tr>
<tr>
<td>4.2</td>
<td>Linear regression model for Air Temperature on cell [23,108].</td>
<td>26</td>
</tr>
<tr>
<td>4.3</td>
<td>Linear regression model MAE for Air Temperature.</td>
<td>27</td>
</tr>
<tr>
<td>4.4</td>
<td>Results from the single CA experiment over a 2-year period.</td>
<td>28</td>
</tr>
<tr>
<td>4.5</td>
<td>An example of the neural net training function overfitting the data.</td>
<td>29</td>
</tr>
<tr>
<td>4.6</td>
<td>Average Mean Absolute Error values for Air Temperature over the Earth.</td>
<td>30</td>
</tr>
<tr>
<td>4.7</td>
<td>Average Mean Absolute Error values for Air Temperature over North America.</td>
<td>31</td>
</tr>
<tr>
<td>4.8</td>
<td>Multi variable cellular automata simulation results for Air Temperature.</td>
<td>32</td>
</tr>
<tr>
<td>4.9</td>
<td>All pair Pearson Correlation Coefficients $r$ and associated $p$ values for the cell [23,108] for the Air Temperature variable.</td>
<td>32</td>
</tr>
<tr>
<td>4.10</td>
<td>Plots of the most often strongly correlated cells for different variables.</td>
<td>33</td>
</tr>
<tr>
<td>4.11</td>
<td>Global MAE scores of the Cellular Automata Network and Standard Cellular Automata simulations for air temperature.</td>
<td>34</td>
</tr>
<tr>
<td>4.12</td>
<td>North America MAE scores of the Cellular Automata Network and Standard Cellular Automata simulations for air temperature.</td>
<td>34</td>
</tr>
<tr>
<td>B.1</td>
<td>Sequoia parallelized experiment toolchain.</td>
<td>50</td>
</tr>
</tbody>
</table>
CHAPTER 1

INTRODUCTION

Cellular automata (CA) have been used since the early 1950’s as a tool to model many different types of physical systems in a discrete manner and as a framework to perform grid based computations. A cellular automaton is a grid of cells where each cell changes through time as a function of the values of its neighbors. An example of a cellular automaton is the Game of Life, a famous cellular automaton invented by John Conway in 1970. In the Game of Life, a cell that is ”alive“ (or has a value of 1) stays alive in the next iteration if two or three of its eight neighbors are also alive. A cell that is ”dead“ becomes alive in the next iteration if exactly three of its neighbors are alive. It has been shown with just these two rules it is possible to create systems that function as Turing machines and are therefore capable of universal computation.

The simplicity of cellular automata make them a popular choice for modeling natural phenomena. The rules of a CA usually are applied at a local level where each cell in the automaton function is independently from everything except for its immediate neighbors. The interesting property of CA is that although they are governed by these local rules, as a system, they can exhibit complex macroscopic behavior. This is useful for modeling purposes because most physical systems can be described as systems of local partial differential equations.
If the systems are able to be reduced to rules that function locally on some dimension then they can be modeled by CA. Some examples of cellular automata that are used for modeling purposes include: lattice-gas cellular automata [18] for modeling fluid flows and stochastic cellular automata for modeling processes such as urban growth over time [20].

In this work we compare the predictive power of standard cellular automata against a specific class of CA called cellular automata networks (CAN) using a large amount of climate data from the National Center for Environmental Prediction (NCEP) / National Center for Atmospheric Research (NCAR) Reanalysis 1 Dataset [5]. Previous work on cellular automata networks and modeling climate data as complex networks, discussed in Chapter 2, suggest that CAN can be used to model climate data. We have designed a series of experiments, described in Chapter 3, that will allow us to see, first, if cellular automata coupled with neural networks are a feasible method for predicting climate variables, and second, if cellular automata networks are better than standard cellular automata at this task. The results from our experiments are discussed in Chapter 4 and conclusions in Chapter 5. In Appendix A we benchmark several popular Python neural network libraries in order to find an efficient library to use in the implementation of our cellular automaton experiments. Finally, in Appendix B we describe the process of designing a toolchain to run our experiments on the Mississippi Center for Supercomputer Research (MCSR) supercomputer.

The experiments in this work all deal with the NCEP/NCAR Reanalysis 1 Dataset; however, the main objective of the work is not to build a state of the art
climate regression model or make inferences about climate models in general. We model the climate system as a cellular automaton where the transition functions are neural networks trained on the existing data from the NCEP/NCAR dataset; however, the climate system is a physical system, and is represented by systems of differential equations called the primitive equations [10]. Solving these equations directly with numerical methods in order to make predictions would surpass our method of approximation with machine learning methods. For our purposes, approximating the transition function for cellular automata is sufficient because we don’t have to incorporate numerical solvers into our experiments and because we are only interested in discovering whether cellular automata based methods are feasible for modeling complex systems.
CHAPTER 2

LITERATURE SURVEY

In this chapter we will give a brief history of cellular automata in Section 2.1, provide a definition of cellular automata in Section 2.2, examine a study on cellular automata networks in Section 2.3, and examine several studies on complex networks and climate science in Section 2.4. The combined ideas from all of these sections support our goal of showing how cellular automata networks can be used to model climate data as a complex network.

2.1 Cellular Automata Background

At a very basic level a cellular automaton is made up of a grid of cells, and a transition function. A cell in a cellular automaton can take on discrete or continuous values and models the state of a system at a point in time. The transition function describes how the value of an individual cell changes at the next discrete time step as a function of that cell’s spatial neighborhood. The entire grid of cells is updated in parallel to “advance” time in the simulation.

Cellular Automata were first described by John von Neumann in the late 1940’s in his work on creating an abstract universal constructor. Von Neumann was trying to create a self replicating model and came up with a very large two-dimensional cellular automaton with 29 states and a complicated ruleset to
emulate the function of digital computers. It has been shown that this model is capable of universal computation [16]. However, this model was not actually implemented due to its complexity and size. The study of CA was continued by Stephen Wolfram who studied one dimensional cellular automata in the 1980’s and published the well known work “A New Kind of Science” in 2002 which shows the implication of cellular automata in many different scientific fields [1]. Wolfram shows some surprising results about the 255 different one dimensional “elementary” cellular automata (all the different combinations of 8 rules on 2 states) including: rule number 30 which can act as a pseudo random number generator and rule 110 which is capable of universal computation. Several iterations of rule 30 can be seen in Figure 2.1. As mentioned in Chapter 1, John Conway invented the Game of Life, which is a two dimensional cellular automaton. Although the Game of Life only has two simple rules, it is also capable of universal computation. The simplicity of these automata have allowed them to be enthusiastically studied by hobbyists and researchers alike since their creation.

2.2 Cellular Automata

There are many different types of cellular automata. This makes formulating a general formal definition difficult. For a comprehensive description of the different definitions of CA see the Cellular Systems chapter from [4]. Briefly, a cellular automata is made up of a grid (or lattice), a neighborhood, and a transition function. This can be stated formally (in 2 dimensions) as, \( C = (G^t_{n \times m}, F) \), where \( C \) is a cellular automata, \( G \) is an \( n \) by \( m \) grid of cells at time \( t \), and \( F \) is a
transition function where $G_{n+m}^{t+1} = F(G_{n+m}^t)$.

The grid in which the automata functions can consist of any number of dimensions depending on the application. As mentioned in 2.1, Wolfram studied 1 dimensional cellular automata, where each cell is only connected to 2 neighbors as shown in Figure 2.2. Two-dimensional grids are used in the Game of Life and in most automata that model physical phenomena. An example of a two dimensional grid is shown in Figure 2.3. The borders of the grid can be: null boundaries, where each cell “outside” of the grid is considered to be in some unchanging null state, cyclical boundaries, where the grid wraps to the other side on the edges, or limitless, where there are not any boundary conditions. In the limitless case there has to be some “inactive” state where there is not any evaluation of the transition function. The grid that cellular automata are executed on can also be modeled as a network where each cell is connected to its
adjacent cells. This allows for more complex cellular automata networks to be create and simulated.

Figure 2.2: 1-Dimensional Cellular Automata with a neighborhood radius of 1.

Figure 2.3: 2-Dimensional Cellular Automata with an R-radial neighborhood radius of 1 (von Neumann Neighborhood).

The neighborhoods used by most cellular automata fall into the categories of R-radial or R-axial neighborhoods. An R-radial neighborhood includes all cells that are a distance \( r \) away from the center cell. An R-radial neighborhood with \( r = 1 \) is known as a von Neumann Neighborhood, as shown in Figure 2.3. An R-axial neighborhood contains all cells in the area \( (x + r, y + r) \) where \( r \in \{-R, R\} \). A R-axial neighborhood with \( r = 1 \) is known as a Moore Neighborhood, as shown in Figure 2.4. Both the R-radial and R-axial neighborhoods are discussed more formally in [2].

The transition function is the set of rules or function that, given a cell,
maps the configuration of the neighborhood to a new value for the cell. With a finite number of states per cell the transition function becomes a simple mapping, but this function can have a continuous range/domain as well. The transition functions for Wolfram’s one dimensional cellular automata are a mapping from all the different configurations of three cells to the two different configuration that the middle cell can take on. In some cases genetic algorithms or other optimization techniques have been used to learn transition functions to fit to data as discussed in [6]. Transition functions have also been hand crafted to approximate solutions to fluid dynamic problems, model the spread of forest fires, model billiard ball collisions, and study self organized criticality as discussed in [19].

2.3 Cellular Automata Networks

Standard cellular automata can be seen as networks where each cell in the automaton is a vertex. Edges in the network are the spatial connections between cells and their neighbors. The network formed by a standard cellular
automata on a two dimensional grid will have an average degree of 4 if the Von Neumann neighborhood is used, because each cell will be connected to 4 other cells. The Manhattan distance between cells \(a\) and \(b\) will be equivalent to the number of iterations (or timesteps) that must take place before the information that started at \(a\) can propagate to \(b\). When a cellular automata network is used, as described by [21], the resulting network behaves as a complex network (or small world network). Small world networks are a class of networks in which the average number of steps that it takes to get from two randomly chosen nodes grows in proportion with the logarithm of the number of nodes. These networks have a high degree of local clustering, meaning that most nodes are not neighbors of one another. Examples of small world networks include the Internet, social networks, financial networks, and many naturally occurring systems such as the climate network and neural networks as described by Watts and Strogatz in [17].

Cellular automata networks are an extension of cellular automata that allow for non local (or long range) connections to other cells. Cellular automata networks have been studied by X. Yang and Y. Yang [21] who have shown that if a small fraction of cells in a cellular automaton have long range connections then the entire network exhibits properties of small-world or complex networks.

In the introduction of his paper, Yang states that partial differential equations that include long distance shortcuts cannot be solved analytically, which is why modeling with some sort of complex network is important. He further states that cellular automata models are more stable than equation-based models, due to their finite states and local interacting rules. Also, if the local rules (transi-
tion function) of a cellular automaton can be derived from the partial differential
equations model they will both simulate the same process. In Section 2.4 we
will review literature that suggests that the global climate can be modeled as a
complex network, which supports the idea that a cellular automaton network will
also be able to model it.

2.4 Complex Networks and Climate Research

The main motivation for our work comes from the idea that cellular au-
tomata networks can take on the properties of complex networks, which in turn
can be used to model climate variables. The research that shows how complex
networks can be used to model climate data is described below:

In [12], Steinhæuser describes the process of making a complex network
out of the data from the NCEP/NCAR Reanalysis Project. The paper outlines a
de-seasonalization and de-trending method to use on the reanalysis data in order
to normalize the data on a per cell basis. To construct the actual network, the
cells are used as vertices and all pairs of edges between cells are weighted with the
Pearson correlation coefficient between the two series. The Pearson correlation
coefficient, \( r \), between two series \( A \) and \( B \) of length \( t \) is given as:

\[
r(A, B) = \frac{\sum_{i=1}^{t} (a_i - \overline{a})(b_i - \overline{b})}{\sqrt{\sum_{i=1}^{t} (a_i - \overline{a})^2 \sum_{i=1}^{t} (b_i - \overline{b})^2}}
\]

(2.1)

In the described technique all of the edges whose correlation \( r \) had a cor-
responding \( p \) value over a threshold \( \tau \) were pruned away. This was done so that
the remaining edges represented strong connections that had significant deviation
from the mean. This paper looked at the topology of the resulting graph at different levels. The number of nodes, density of edges, clustering coefficient, and characteristic path length were all examined. It was found that the clustering coefficient and characteristic path length for all 7 examined variables (air temperature, sea level pressure, geopotential height, etc...) from the data were greater than the expected values for a random graph of the same size. The area weighted connectivity for each node was plotted and it was observed that highly connected areas correspond with the location of global climate indicators, such as: El Nino, and the Pacific Index.

Steinheuser’s later work, [14], describes further the technique he uses to prune graph edges, and touches on predictive modeling with the resulting complex network. A clustering technique, WalkTrap (described in [13]), is used on the graph made from the cells and Pearson correlation coefficients, then each of the clusters are used to predict target variables with linear regression. The dataset in this case is split into a 50 year training set, 10 year test set, and the root mean squared error (RMSE) is used to evaluate the regression experiments. Also described is a technique called “Lift” for determining the predictive power of the regression model, which compares the experimental RMSE to a random based RMSE. Steinhaeuser admits that his approach is mainly constrained to a single variable, but he attempts a multivariate analysis of the climate data using vectors of correlation coefficients between the different combinations of variables at two nodes as edge weights. In order to choose the variables included in the multivariate approach Steinhauser consulted a domain expert, who recommended
air temperature, pressure, relative humidity, and precipitable water.

A different author, Steinbach, also uses clustering as a tool for analyzing the NCEP/NCAR dataset. In his paper “Discovery of Climate Indices using Clustering” [11], Steinbach describes the well-known climate indices, and performs clustering on the climate data with both Singular Value Decomposition (SVD) and Shared Nearest Neighbor (SNN) techniques. He then successfully correlates the found clusters with the known climate indices which further supports the idea that the “complex network” approach to analysis of climate data is a valid approach.

Lastly, Tsonis’ paper, [15], also describes how to build a complex network from the NCAR/NCEP dataset. The differences between Tsonis’ and Steinhaeuser’s work is that Tsonis does not use a deseasonalization step, examines the data at a coarser resolution, and focuses more on showing that the resulting network is a complex network. The paper shows the small world properties of the network and how the same properties do not show up in randomly generated networks.

All three of these authors’ works suggest that the climate data can be successfully modeled as a complex network. Steinhaeuser and Tsonis both build a complex network from the cell locations where the nodes in the network represent a physical location from the dataset, and vertices represent a high correlation between two locations. From this work we hypothesize that introducing long range connections between highly correlated cells in a cellular automaton will allow it to make more accurate predictions as it will be able to model the underlying
physical processes better.
The literature suggests that cellular automata function as small world networks when a small fraction of cells are allowed to have a long range connection to non-adjacent cells. Small world networks are a type of complex networks that exhibit many of the same properties found in networks in the physical world. Furthermore, the global climate system has been modeled successfully as a complex network. This work will take these ideas and try to model the global climate with cellular automata networks. We use the same climate dataset that is used elsewhere in the climate modeling literature, described in Section 3.1. Our experiments first establish a baseline score with linear regression to predict a single variable on individual cells as described in Section 3.2. The next experiment uses a cellular automata with a neural network transition function without long range connections to predict different climate variables, see Sections 3.3 and 3.4. Finally, a cellular automata with a small number of long range connections is used to predict climate variables as described in Section 3.5.

3.1 NCEP/NCAR Reanalysis 1 Dataset

The experiments done in this work have all used the The National Center for Environmental Prediction / National Center for Atmospheric Research
Reanalysis 1 Dataset [5]. The NCEP/NCAR Reanalysis 1 Dataset has measurements for a large number of climate variables at several pressure levels 4 times daily since 1948. The measurements are done at a 2.5 degree resolution over the entire surface of the Earth. This data is taken from a large variety of sources and reanalyzed to produce the actual measurements reported in the dataset. In order to reduce computational effort we will use the monthly means of the 12 surface level variables which are included in the “NCEP/NCAR Reanalysis Monthly Means and Other Derived Variables” subset. Unlike other papers in the literature [13], we do not perform de-seaonalization on the data because our main focus is not on analyzing trends in the data, so it is not necessary to remove the seasonal component. The dataset is distributed in the NetCDF4 file format with a single file for each of the 12 variables. All interaction with the dataset is done using the netcdf4-python library\(^1\).

The following variables are included in the dataset, and are described further in Table 3.1:

**Air Temperature**: Basic air temperature.

**Surface Lifted Index**: Indicates the stability of the air, negative values indicate unstable air, positive values are stable air (stable air resists vertical motion).

**Best 4-layer Lifted Index**: The most unstable lifted index from 4 different...

---

\(^1\)https://github.com/Unidata/netcdf4-python
layers.

**Omega**: Vertical motion in atmosphere.

**Potential Temperature**: The temperature an unsaturated air would have if lowered (or raised) to a level of 1000 mb.

**Precipitable Water**: Depth of water in a column of atmosphere.

**Pressure**: The force exerted by the atmosphere on a surface.

**Relative Humidity**: The ratio of the partial pressure of water vapor in the mixture to the saturated vapor pressure of water at a given temperature.

**Sea Level Pressure**: The pressure of the atmosphere at sea level.

**U Wind**: East-West wind speed (Zonal).

**V Wind**: North-South wind speed (Meridional).

**Wind Speed**: Average absolute local wind speeds.

<table>
<thead>
<tr>
<th>Name</th>
<th>Units</th>
<th>Precision</th>
<th>Range Start</th>
<th>Range End</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monthly Mean Relative Humidity at sigma level 0.995</td>
<td>%</td>
<td>2</td>
<td>0.00</td>
<td>100.01</td>
</tr>
<tr>
<td>Monthly Mean Meridional Wind at sigma level 0.995</td>
<td>m/s</td>
<td>2</td>
<td>-15.44</td>
<td>19.79</td>
</tr>
<tr>
<td>Sea Level Pressure</td>
<td>millibars</td>
<td>1</td>
<td>958.98</td>
<td>1482.56</td>
</tr>
<tr>
<td>Monthly Mean Wind Speed at sigma level 0.995</td>
<td>m/s</td>
<td>2</td>
<td>0.51</td>
<td>21.33</td>
</tr>
<tr>
<td>Monthly Mean Air Temperature at sigma level 0.995</td>
<td>degC</td>
<td>1</td>
<td>-73.78</td>
<td>41.75</td>
</tr>
<tr>
<td>Monthly Mean of Surface Pressure</td>
<td>millibars</td>
<td>2</td>
<td>495.93</td>
<td>1690.44</td>
</tr>
<tr>
<td>Monthly mean potential temperature at sigma level 995</td>
<td>degK</td>
<td>1</td>
<td>214.60</td>
<td>334.70</td>
</tr>
<tr>
<td>Monthly Mean Zonal Wind at sigma level 0.995</td>
<td>m/s</td>
<td>2</td>
<td>-18.84</td>
<td>18.84</td>
</tr>
<tr>
<td>Monthly mean omega at sigma level 0.995</td>
<td>Pascal/s</td>
<td>3</td>
<td>-1.11</td>
<td>2.30</td>
</tr>
<tr>
<td>Monthly Mean Surface Lifted Index</td>
<td>degK</td>
<td>2</td>
<td>-13.39</td>
<td>30.22</td>
</tr>
<tr>
<td>Monthly Mean of Precipitable Water Content</td>
<td>kg/m²</td>
<td>2</td>
<td>-3.97</td>
<td>68.97</td>
</tr>
<tr>
<td>Monthly Mean Surface Lifted Index</td>
<td>degK</td>
<td>2</td>
<td>-8.91</td>
<td>48.23</td>
</tr>
</tbody>
</table>

Table 3.1: Description of the different climate variables in the Reanalysis 1 dataset.
With data points every 2.5 degrees over the surface of the Earth, there are 73 latitude points and 144 longitude points. This forms a 72x143 grid over the surface of the earth for a total of 10,296 individual cells. At each cell location there are 802 measurements (once monthly from January 1948 to October 2014) for each of the 12 variables at the surface level in the dataset, for a total of 99,088,704 measurements. It is convenient to think of the dataset as a volume, where the base of the volume is a map of the Earth, with 12 columns at each cell representing the time series information for the different variables. In this description of the dataset, a horizontal cross section at some point along the vertical-axis would be a description of the entire map at that point in the time series. If the monthly means were not taken there would be 97,520 measurements per variable for a total of 12,048,791,040 measurements. The monthly mean dataset is sufficient, however, because the main objective of this research is not to make precise predictions, and using the larger dataset would require a much greater amount of computing power.

3.2 Linear Regression Experiment

The objective of the first experiment is to determine a baseline score that we hope to be able to beat with a cellular automata implementation. For each cell in the map, a LinearRegression model from the python sklearn library[8] is “trained” with the entire time series data from that cell. This is equivalent to doing a least squares line fitting. To estimate how well this model does the coefficient of determination and mean squared error is calculated per cell between
the actual time series and the predicted values from the model. This experiment provides the bare minimum baseline for which we will compare the neural network models against.

3.3 Cellular Automata with Neural Network Experiment

In this experiment we train a neural network to act as the transition functions for a single cell in a cellular automata and evaluate the prediction results. In this experiment we used the cell from the dataset at 32.5 North, 90 East (index [23,108]) which roughly corresponds to Mississippi.

The Python bindings\(^2\) of the FANN library\(^7\) are used to train all of the neural networks in our experiments. See Appendix A for the benchmark results of different neural network libraries for Python.

The model was tested by splitting the input data into training and testing partitions, training the model on the training set, and determining the Mean Squared Error (MSE) on both the training and testing sets for the air temperature variable. This process was done 40 times for various different training/testing split sizes and the average and standard deviation mean squared error (MSE) was recorded for each split size. Results from this experiment are shown in Section 4.2.

The data in this, and subsequent experiments, is scaled to the range [0,1] prior to training using the following formula, \(X_{\text{scale}} = (X - X_{\text{min}})/(X_{\text{max}} - X_{\text{min}})\). After the network is trained, the inverse formula, \(X = X_{\text{scale}} \times (X_{\text{max}} - X_{\text{min}}) + X_{\text{min}}\), is used to scale all further output from network back to equivalent

---

\(^2\)PyFANN https://github.com/orso82/python-fann
The neural networks are trained using the Cascade-Correlation training method [3] implemented by the FANN library. This training method automatically adjusts the size of the network during the training. The following parameters are used in the FANN library for training:

```c
ann.set_training_algorithm(libfann.TRAIN_RPROP);
ann.set_activation_function_hidden(libfann.SIGMOID_SYMMETRIC);
ann.set_activation_function_output(libfann.LINEAR PIECE);
ann.set_activation steepness_hidden(0.5);
ann.set_activation steepness_output(0.5);
ann.set_train_error_function(libfann.ERRORFUNC_LINEAR);
ann.set_rprop_increase_factor(1.2);
ann.set_rprop_decrease_factor(0.5);
ann.set_rprop_delta_min(0.0);
ann.set_rprop_delta_max(50.0);
ann.set_cascade_output_change_fraction(0.01);
ann.set_cascade_output_stagnation_epochs(12);
ann.set_cascade_candidate_change_fraction(0.01);
ann.set_cascade_candidate_stagnation_epochs(12);
ann.set_cascade_weight_multiplier(0.4);
ann.set_cascade_candidate_limit(1000.0);
ann.set_cascade_max_out_epochs(150);
ann.set_cascade_max_cand_epochs(150);
```
This experiment does not actually test using a cellular automata simulation to predict climate variables. However, it is necessary to show that the neural network transition function is a viable method for approximating the physical processes that cause the changes in the climate variables.

3.4 Full Cellular Automata Experiments

These experiments are scaled up versions of the previous experiment described in Section 3.3. In this case each cell in the grid will have an individual neural network trained on the first half of the time series data from the variables in its Von Neumann neighborhood. The neural network will have inputs for each variable being trained on from the current cell as well as the neighboring cells (e.g. if 4 different variables are included in an experiment, each neural network will have a size 20 input layer).

Similar to the previous experiment, the neural networks are tested on the second half of the data and the mean absolute error (MAE) and mean squared error (MSE) metrics are recorded for each variable. The cellular automata functionality of the model is tested by simulating climate variables starting from baseline data. The cellular automata grid is populated from ground truth data then simulated for a number of iterations. The average MAE is calculated per variable per cell.

The experiments in this case are all run on the Mississippi Center for Su-
percomputing Research Sequoia supercomputer to take advantage of the parallel nature of this model of computation. See Appendix B for more information about how the experiment was set up and run in a parallel fashion on Sequoia.

We have run the full cellular automata experiment with single and multi-variable configurations. The air temperature variable was used again, as in the previous experiment for the single variable experiment. For the multivariable experiment we used the 4 variables that were used in [14], namely: air temperature, pressure, relative humidity, and precipitable water.

For the full grid single variable experiment it is assumed that the results will be similar to the single cell single variable experiment as the same parameters are used. Results from this experiment are shown in Section 4.3.

3.5 Cellular Automata Network Experiment

In the Cellular Automata Network Experiment we modify the cellular automata structure from 3.4 to include a small number of long range connections not found in traditional cellular automata models. See Figure 3.1 for an example long range connection on a cellular automata grid.

The long range connections in this experiment occur with probability \( p = 0.05 \) to a long range location where there is high Pearson cross correlation. As Steinhauer and Tsonis do in [15], [12], [13], and [14], we have calculated all pairs Pearson Cross Correlation for each variable in the dataset. The long range connection is made for each variable included in the simulation to a random choice from the top 200 highly correlated cells. This is done to allow the information
Figure 3.1: More Neighborhood with a long range connection.

from an area that is not directly spatially linked to travel and affect nodes that it otherwise would not have been able to reach. In addition, the resulting complex network will be similar to the climate networks created in [12] and [14], where connection between two nodes is not dependent upon spatial location and there exists long range connections between nodes that are strongly correlated. The idea is that certain “climate indicator” regions, as discussed in [11] are strong predictors of weather in other parts of the world. If the information can easily be transported to the necessary parts in the model then the model should have a greater predictive power.

The presence or absence of a long range connection is determined at training time for our experiment in order to train a neural network to accept an additional input to the normal Von Neumann neighborhood. Once a neural net-
work has been trained for cell (i,j) that includes a long range connection to cell (k,l) it can be exchanged seamlessly with a neural network that does not have that same long range connection.

This experiment is trained and tested on the same four variables from the multivariable test from Section 3.4. The same variables were used because they were recommended by the literature and so that the results from the two experiments can be directly compared. The results can be found in Section 4.4.
CHAPTER 4

EXPERIMENTAL WORK

4.1 Linear Regression Experiment Results

As expected, the predicted values from this model are not representative of the data, as the linear regression method is not able to model the seasonal component of the data. The linear regression model ends up predicting an average temperature for the location it is trained on, as seen in Figure 4.2, which forces the $r^2$ values to be very low. We see from Figure 4.1 that all of the $r^2$ scores are practically zero. Finally, from Figure 4.3, we see that the MAE for the air temperature variable fluctuates from 0, in regions of generally static temperature, to 17.5.

4.2 Single Cell Cellular Automata Experiment Results

This experiment tested how well the neural network for a single cell performed on one variable (air temperature). The results from a two year prediction period are shown in Figure 4.4. The “input” line on the graph shows the previous value at each timestep for the cell being examined. From this graph you can see that the predictions generally match the shape of the input data, however do not model any anomalies in the data.
Figure 4.1: $r^2$ values for linear regression model predicting Air Temperature.

Figure 4.5 shows the results from a neural network trained on the first half of the input data and tested on the second half of the data. The strong fit to the first half of the data compared to the erratic fit to the second half suggests that the Cascade Correlation training method is overfitting the neural networks to seen data. The data from Table 4.1 suggests that the model is able to generalize at some level though because the training sizes of 50% and 60% produce the lowest MAE on the testing sets, whereas if the model was being completely overfit to the data the lowest MAE should be with the largest training set size. Table 4.2
Figure 4.2: Linear regression model for Air Temperature on cell [23,108].

shows the summary statistics for the cell that these results were obtained from.

<table>
<thead>
<tr>
<th>Testing Size(%)</th>
<th>Mean Training MSE</th>
<th>STDEV Training MSE</th>
<th>Mean Testing MSE</th>
<th>STDEV Testing MSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.00%</td>
<td>0.10837395</td>
<td>0.0052210762</td>
<td>14.97219503</td>
<td>2.826443425</td>
</tr>
<tr>
<td>20.00%</td>
<td>0.108498825</td>
<td>0.0128114703</td>
<td>15.86206858</td>
<td>1.818641207</td>
</tr>
<tr>
<td>30.00%</td>
<td>0.105996075</td>
<td>0.0074800226</td>
<td>15.74647818</td>
<td>1.832946476</td>
</tr>
<tr>
<td>40.00%</td>
<td><strong>0.10503765</strong></td>
<td><strong>0.0091229185</strong></td>
<td><strong>15.51759805</strong></td>
<td><strong>1.418296801</strong></td>
</tr>
<tr>
<td>50.00%</td>
<td>0.10512295</td>
<td>0.0083995017</td>
<td>15.9019352</td>
<td>1.418296801</td>
</tr>
<tr>
<td>60.00%</td>
<td>0.101702279</td>
<td>0.0143694174</td>
<td>16.3940707</td>
<td>1.168416567</td>
</tr>
<tr>
<td>70.00%</td>
<td>0.09245585</td>
<td>0.0118975515</td>
<td>17.2831438</td>
<td>1.541466034</td>
</tr>
<tr>
<td>80.00%</td>
<td>0.09883375</td>
<td>0.0146033128</td>
<td>18.06803153</td>
<td>3.194061864</td>
</tr>
<tr>
<td>90.00%</td>
<td>0.083968075</td>
<td>0.017279106</td>
<td>20.05882088</td>
<td>4.36917064</td>
</tr>
</tbody>
</table>

Table 4.1: Results from different training/testing sizes on a single cell.

4.3 Full Cellular Automata Experiment Results

This experiment tested the predictive ability of a full standard cellular automata model. The first part of the experiment tests the neural networks on the single variable air temperature. The results from this part of the experiment
can be seen in Figures 4.6 and 4.7. These results are equivalent to running the previous experiment on every cell and graphing the resulting MAE on a heatmap.

In the second part of the experiment the cellular automata portion of the model was used. The results from a timestep $t$ are used to predict the next timestep at $t + 1$. The cellular automata was initially populated with the ground truth values from the first timestep from unseen testing data. The model was then run for 50 iterations and the difference from the ground truth values for each variable was calculated at each iteration. This process cause any errors
introduced by the cellular automata to be greatly amplified as time passes. The results for the air temperature variable from the cell [23,108] are shown in Figure 4.8 where the red line (diamond markers) is the ground truth predictions and the blue line (square markers) is the predicted values. The results from the entire grid and the grid over North America are shown in Figures 4.11 and 4.12.

The results of this experiment show that the air temperature at the longitudes close to the poles are very difficult for the algorithm to predict accurately. Air temperature in the northern U.S., Canada, and northern Asia seem to be difficult to predict compared to the rest of the globe. Generally, as latitude increases the MAE values seem to trend higher, with the equator having the most accurate predictions. The prediction values over water seem to be more accurate
Figure 4.5: An example of the neural net training function overfitting the data.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>17.352114</td>
</tr>
<tr>
<td>STDEV</td>
<td>7.440082</td>
</tr>
<tr>
<td>Min</td>
<td>-2.378379</td>
</tr>
<tr>
<td>Max</td>
<td>29.833548</td>
</tr>
</tbody>
</table>

Table 4.2: Summary Statistics corresponding to the Air Temperature series at cell [28,108].

than predictions over land, this could be an artifact of the reanalysis aspect of the data.

4.4 Cellular Automata Network Experiment Results

As mentioned in the experiment design in Section 3.5, the locations for the long range connections in the cellular automata network were chosen randomly from a list of the top 200 highly correlated cells. The Pearson Correlation Coef-
ficient, $r$, and corresponding $p$ value was calculated for each pair of cells on the grid. An example plot for the cell where Mississippi is located is shown in Figure 4.9. As described in [13], to form the complex network, connections between cells that have a $p$ value higher than some handpicked threshold are kept and the rest of the connections are discarded. We follow a similar procedure. For each cell we look at the top 10 highly correlated cells which have $p$ values over 0.5. For each of these strongly correlated cells we increase a counter at their location. After this process is done for each cell on the grid, the cells whose counters are highest are the ones that have the most influence for that particular variable. It is these top 200 highly correlated cells (per variable) that long range connections are made to. An visualization of this data can be seen for several variables in Figure 4.10.

The results from this experiment are obtained in the same way as the results from the full cellular automata experiment. These results for the air tem-

![Figure 4.6: Average Mean Absolute Error values for Air Temperature over the Earth.](image)
Figure 4.7: Average Mean Absolute Error values for Air Temperature over North America.

The temperature variable can be seen in Figures 4.11 and 4.12. The Cellular Automata Network simulation results are slightly better than the Standard Cellular Automata Results. For the Cellular Automata Network simulation the minimum MAE score from the Global results is 0.44 and the maximum MAE is 23.84. This range is better than the Standard Cellular Automata simulation which has a minimum value of 0.61 and a maximum value of 29.41. The heatmaps also show that the Cellular Automata Network simulation has noticeably better results than the Standard Cellular Automata simulation over various location such as Canada and parts of Northern Europe.
Figure 4.8: Multi variable cellular automata simulation results for Air Temperature.

Figure 4.9: All pair Pearson Correlation Coefficients $r$ and associated $p$ values for the cell [23,108] for the Air Temperature variable.
Figure 4.10: Plots of the most often strongly correlated cells for different variables.
Figure 4.11: Global MAE scores of the Cellular Automata Network and Standard Cellular Automata simulations for air temperature.

Figure 4.12: North America MAE scores of the Cellular Automata Network and Standard Cellular Automata simulations for air temperature.
CHAPTER 5

CONCLUSION

The results from the Single Cell Cellular Automata experiment and the first part of the Full Cellular Automata experiment suggest that the neural network approach to modeling the transition function is acceptable. From observing prediction from the cell [23,108] it seems neural networks are able to capture at least the shape of the climate variables. The MAE results from the neural network methods are predictably better than the LinearRegression results from the Linear Regression Experiment. These experiments lead to the Cellular Automata Simulation experiment and Cellular Automata Network Simulation experiment. The results from these experiments suggest that modeling climate variables with cellular automata networks is more descriptive than modeling with a standard cellular automata. This conclusion is weakened however as we do not know how much the improvements in average accuracy are due to the deviations in training between the sets of neural networks used for each experiment. Even though the training method for the neural networks was held the same between the experiments, any randomness introduced by the training technique will have an affect on the final predicted values of the network. Regardless, the results from these four experiments and the results obtained in other works from the literature seem
to suggest that cellular automata networks are a suitable tool for modeling complex network based data and that cellular automata networks are more effective than cellular automata at this task.

There are many opportunities to carry on and extend this work. Immediately, more experiments could be done with the same, and different, datasets to support our primary conclusion that cellular automata networks are better suited than cellular automata to model complex network based problems. More work could also be done in setting up experiments that compare cellular automata networks with cellular automata where the effectiveness of the transition function is known beforehand. This would allow for stronger conclusions about the effectiveness of the two different techniques to be drawn. Studies on the effectiveness of different techniques for learning transition functions could be done in the context of simulating timeseries data modeled as complex networks, as it is unknown to us of any research that has been done in this area. Finally, as far as we know, cellular automata networks have only been studied in [21]. More experiments that use cellular automata networks to model complex networks need to be done to investigate the scope of effectiveness of cellular automata in these applications. This work only compared cellular automata networks with cellular automata, future work could compare the effectiveness of cellular automata networks against other simulation techniques.


APPENDICES
APPENDIX A

PYTHON NEURAL NETWORK LIBRARY BENCHMARKING

To set up the cellular automata network tests we needed a library to quickly and easily train neural networks from Python. We considered four different Python libraries that accomplished this task: FANN, Neurolab, and PyBrain [9]. In order to test the speed of setting up and training networks with these libraries we used a 2 input, 4 hidden, 1 output layer network that was trained using ResilientBackPropagation as a benchmark. The training input was random floating point numbers in the range [-0.5,0.5] and the output was the sum of the inputs.

In the first experiment the networks were trained for 1000 iterations bootstrapped 40 times and the average training time and MSE were recorded. In the second experiment the networks were trained until a goal MSE of .0001 or 1000 iterations was reached and the average MSE, training time and number of iterations were recorded. Both experiments were done on training sizes of 10, 50 and 100.
The results from this experiment shown in Table A.1 and A.2 show that the FANN library is several magnitudes faster than the other networks. This is to be expected as the FANN library is a native C library with Python bindings, while PyBrain is a native Python library, and Neurolab is mainly a Python library that uses optimization routines from SciPy. PyBrain, although it is the slowest library in our tests, converges with a smaller number of iterations than the other two libraries. The relatively large differences in standard deviation in the time taken to train from experiment 1 to experiment 2 suggests that in some cases the networks converge very quickly to the desired error rate based on initial random weights. This result could be further investigated to support that fact that PyBrain could be used to the same effect as the other two libraries despite having a vastly slower training time.

Considering that the FANN library is still an active project, its extremely fast training time compared to the other two libraries, and good results from training on the benchmark data, we will use the FANN library for the cellular automata experiments.
### Table A.1: Benchmarking results for different size datasets trained on for 1000 iterations.

<table>
<thead>
<tr>
<th>N=10</th>
<th>M=40</th>
<th>Name</th>
<th>Mean error</th>
<th>STD error</th>
<th>Mean time</th>
<th>STD time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Neurolab</td>
<td>0.000296</td>
<td>0.00047</td>
<td>0.589155</td>
<td>0.007946</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FANN</td>
<td><strong>0.000123</strong></td>
<td>0.000471</td>
<td><strong>0.004967</strong></td>
<td>0.000189</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PyBrain</td>
<td>0.000162</td>
<td>0.000095</td>
<td>3.810644</td>
<td>0.173006</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>N=50</th>
<th>M=40</th>
<th>Name</th>
<th>Mean error</th>
<th>STD error</th>
<th>Mean time</th>
<th>STD time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Neurolab</td>
<td>0.000365</td>
<td>0.000348</td>
<td>2.922027</td>
<td>0.446531</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FANN</td>
<td>0.00041</td>
<td>0.000279</td>
<td><strong>0.021111</strong></td>
<td>0.000605</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PyBrain</td>
<td><strong>0.000081</strong></td>
<td>0.000179</td>
<td>18.241002</td>
<td>1.320556</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>N=100</th>
<th>M=40</th>
<th>Name</th>
<th>Mean error</th>
<th>STD error</th>
<th>Mean time</th>
<th>STD time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Neurolab</td>
<td>0.000414</td>
<td>0.000424</td>
<td>6.0927</td>
<td>1.06329</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FANN</td>
<td>0.000348</td>
<td>0.000211</td>
<td><strong>0.044655</strong></td>
<td>0.005376</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PyBrain</td>
<td><strong>0.000031</strong></td>
<td>0.000032</td>
<td>36.5323</td>
<td>0.563232</td>
</tr>
</tbody>
</table>

### Table A.2: Benchmarking results for different size datasets trained on until convergence.

<table>
<thead>
<tr>
<th>N=10</th>
<th>M=40</th>
<th>Name</th>
<th>Mean error</th>
<th>STD error</th>
<th>Mean time</th>
<th>STD time</th>
<th>Mean iterations</th>
<th>STD iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Neurolab</td>
<td>0.000109</td>
<td>0.000064</td>
<td>0.149833</td>
<td>0.17914</td>
<td>244.95</td>
<td>295.012453</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FANN</td>
<td><strong>0.000093</strong></td>
<td>0.000007</td>
<td>1.49833</td>
<td>0.17914</td>
<td><strong>0.000469</strong></td>
<td><strong>0.000073</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PyBrain</td>
<td>0.000123</td>
<td>0.000072</td>
<td>2.944967</td>
<td>1.254019</td>
<td>649</td>
<td>277.678141</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>N=50</th>
<th>M=40</th>
<th>Name</th>
<th>Mean error</th>
<th>STD error</th>
<th>Mean time</th>
<th>STD time</th>
<th>Mean iterations</th>
<th>STD iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Neurolab</td>
<td>0.0000414</td>
<td>0.000424</td>
<td>6.0927</td>
<td>1.06329</td>
<td>636.45</td>
<td>387.656933</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FANN</td>
<td><strong>0.000049</strong></td>
<td>0.000424</td>
<td>1.79989</td>
<td>1.09346</td>
<td><strong>0.000469</strong></td>
<td><strong>39.075</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PyBrain</td>
<td>0.000123</td>
<td>0.000072</td>
<td>2.944967</td>
<td>1.254019</td>
<td>649</td>
<td>277.678141</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>N=100</th>
<th>M=40</th>
<th>Name</th>
<th>Mean error</th>
<th>STD error</th>
<th>Mean time</th>
<th>STD time</th>
<th>Mean iterations</th>
<th>STD iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Neurolab</td>
<td>0.000289</td>
<td>0.000243</td>
<td>5.678297</td>
<td>2.67872</td>
<td>813.375</td>
<td>311.881683</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FANN</td>
<td><strong>0.000288</strong></td>
<td>0.000183</td>
<td>5.678297</td>
<td>2.67872</td>
<td><strong>0.033989</strong></td>
<td><strong>841.525</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PyBrain</td>
<td><strong>0.000097</strong></td>
<td>0.000005</td>
<td>7.307003</td>
<td>5.15694</td>
<td><strong>167.125</strong></td>
<td><strong>118.239627</strong></td>
</tr>
</tbody>
</table>

Table A.1: Benchmarking results for different size datasets trained on for 1000 iterations.

Table A.2: Benchmarking results for different size datasets trained on until convergence.
The resources provided by the Mississippi Center for Supercomputer Research (MCSR) were used to run the larger full cellular automata experiments for this study. The structure of cellular automata lend themselves to being highly parallelizable. In most work on parallel computing with cellular automata, the execution of the actual model is the important computational task, as certain cellular automata allow for image and string processing algorithms to be approached in a massively parallel form. In this work we parallelize the training of the neural network transition functions “offline” on the supercomputer, because it is the training that is the most computationally expensive part of the process. After the neural network model is trained it can be run very quickly on a single machine by simply feeding forward the data for the necessary variables.

The Sequoia supercomputer cluster was used for all of our work. The supercomputer works through a PBS (portable batch system) job scheduling program. You can log onto the main server node and configure your home directory
as needed, then submit jobs to be run using the \texttt{qsub} command. This appendix describes how to build all of the necessary libraries used in our experiments and how the toolchain for running experiments works.

B.1 Building Libraries

Sequoia has python version 2.6 running on it which is sufficient to run the libraries required for the transition function training program. All of the supporting libraries and their dependencies that were used had to be built from source in the home directory. The following packages were all built from source and installed into a local directory\footnote{./configure \texttt{–prefix=/home/ums/r1968/usr} was used to setup most of the packages to be built/installed into the /usr/ directory}:

- setuptools-12.0.1
- FANN-2.2.0
- hdf5-1.9.210
- Netcdf-4.3.2
- zlib-1.2.8
- OpenBlas
- swig-3.0.4

Several complications came up with the build process of the NetCDF4 library: We found, fixed and reported a bug to the NetCDF development team.
concerning how NetCDF was built on the earlier versions of gcc used on Sequoia. The second argument of the method “nc4_rec_find_nc_type” was not consistently typed throughout the codebase and caused gcc to throw the error: previous declaration of “nc4_rec_find_nc_type”. This was corrected by simply editing the function declarations to use consistent argument types. Secondly, the configure command required the CPPFLAGS and LDFLAGS environment variables to show where the hdf5 libraries were installed. Finally, if the hdf5 installation that the NetCDF4 installation links to is not built with zlib support the netCDF4 python library will fail to install, however the main source library will install without error or warning.

After the above libraries are successfully built from source, the Setup-tools (easy_install.py) python installer was used to install the remaining support libraries:

- fann2
- swig
- upgrade scipy (originally had to be built from source so that compiler flags could be set pointing to Lapack and BLAS)
- numpy
- netCDF4
- redis

Footnote: 
2“CPPFLAGS=-I/home/ums/r1963/usr/include LDFLAGS=-L/home/ums/r1963/usr/lib ./configure –prefix=/home/ums/r1968/usr”
scikit-learn

B.2 Building the Experiment Runner Toolchain

The PBS system is configured to let a normal user run 35 separate jobs concurrently with a total of 40 jobs in the queue. Each job is submitted as a bash or python script with PBS directives at the top of it that specify the number of processors and memory that it will need to execute. The number of processors and amount of memory that are available to a job depends on what queue it is placed in; the available queues to run in are described in Table B.1. The NCEP/NCAR dataset has cells on a 72(latitude)*143(longitude) grid (2.5 degree resolution over the globe), which requires 10656 individual experimental runs (*10 bootstrapped trainings for each experiment). To handle this parallelization we split the problem on a job level along the dataset rows. Each latitude row receives its own job (74 jobs) and further parallelizes its execution across the row with the number of processors available to it (which depends upon which queue it is placed in).

<table>
<thead>
<tr>
<th>Queue</th>
<th>MCSR-1N</th>
<th>MCSR-2N</th>
<th>MCSR-4N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max Memory</td>
<td>35gb</td>
<td>30gb</td>
<td>140gb</td>
</tr>
<tr>
<td>Max Number of CPUS</td>
<td>12</td>
<td>16</td>
<td>128</td>
</tr>
<tr>
<td>Max Nodes</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Max CPU Time</td>
<td>4032:00:00</td>
<td>4032:00:00</td>
<td>44000:00:00</td>
</tr>
</tbody>
</table>

Table B.1: Job Queues on Sequoia.

The toolchain for running experiments in a parallelized manner is illustrated in Figure B.1. A description of each script in the process is given below.

**queueAll.sh** Submits a range of jobs to the PBS system through qsub for given
rows in the dataset

**watchdog.py** Monitors the number of currently running jobs on the cluster and
starts up the next batch automatically when the current run is finished.

**runner.py** Is started by the PBS system. Runs on a single node in the cluster.
Runs the client.py script for every longitude cell in the current row, parallelizes this process using all the processors available to the current node\(^3\).

**client.py** Script that actually runs the experiments, is parameterized as described in Section B.2.1.

**submitResults.py** Is called at the end of the watchdog loop to submit all of
the experiment data on file to a Redis database and make backups of the
experiment logs.

The toolchain was designed around the lack of network access in each
node, the inability to queue more than 40 jobs at a time in the PBS queue, the
inability to pass command line parameters to a job through qsub, and the desire
to maximize computing resources.

Because the nodes that the experiments were actually being run on do
not have access to any outside networks and the queue could only hold 40 jobs
at a given time it was necessary to setup an automated system that could check
on the progress of jobs that were currently running and execute code (such as
reporting results) when a set of jobs finished. The inability to pass command line

\(^3\)runner.py with latitude \(LA\) will spawn \(X\) number of processes that each run \(144/X\) experiments for the given \(LA\), where \(X\) is the number of processors available on the given node
parameters to jobs in a queue made it necessary to have the queueAll.sh script to handle setting/unsetting environment parameters for each submitted job to break the work up into rows. To keep the experiment code as separated from the process as possible the we created the runner.py script to handle the in node parallelization.

To run a batch experiment first the queueAll.sh script was run from rows 0 to 39 ("./queueAll.sh 0 39") and then “touch runNext” is run to let the watchdog.py script to run the next set of commands after the current set is done. The
watchdog.py script is run every 30 minutes as a cron job and monitors the output from the qstat command. When the job queue is empty and the runNext file is present it will execute the next batch of jobs and delete the runNext file. This process can be chained repeatedly to continue running multiple experiments.

The nodes that the client.py script are run on do not have a connection to any external networks, which makes it impossible for them to directly report their results to a database. Instead of automatically reporting its results, each experiment writes its results to file and at the end of a macro-experiment an updateDB.py script is called which parses all of the output files and updates a Redis Database that is run on our workstation in the Computer Science Department.

B.2.1 client.py

The client.py script handles the training of a neural network for a given cell. An example call to client.py for given latitude and longitude cell indices and the usage for the script are as follows:

```
python client.py -lat LAT -lon LON -vars airTemp -t 0.5 -m 10 -o output/lat.LAT.lon.LON.output
```

```
usage: client.py [-h] [-v] -lat LATITUDE -lon LONGITUDE [-t TRAININGSIZE
[-m ITERATIONS] [-o OUTPUTFN] -vars VARIABLES [VARIABLES ...]
[-analysis ANALYSIS]
```

optional arguments:
-h, —help    show this help message and exit
-v, —verbose Increase output verbosity
-lat LATITUDE, —latitude LATITUDE
    Latitude index of the cell to process
-lon LONGITUDE, —longitude LONGITUDE
    Longitude index of the cell to process
-t TRAININGSIZE, —trainingSize TRAININGSIZE
    Percent of series to use as the training set
-m ITERATIONS, —iterations ITERATIONS
    Number of times to bootstrap for MSE calculation
-o OUTPUTFN, —outputFn OUTPUTFN
    Name of the output file
-vars VARIABLES [VARIABLES ...], —variables VARIABLES [VARIABLES ...]
    Variables to include in the simulation
-analysis ANALYSIS File name for analysis output