ESTIMATING RAINFALL WITH NEURAL NETWORKS AND CONDITIONAL RANDOM FIELDS

A Thesis
Presented to
The Academic Faculty

by

Evan Ott

In Partial Fulfillment of
the Requirements for the Dean's Scholars Honors Program and
the Degree of Bachelor's of Science in Physics – Honors
and Computer Science – Honors
in the
Department of Physics and Department of Computer Science

University of Texas at Austin

ESTIMATING RAINFALL WITH NEURAL NETWORKS AND CONDITIONAL RANDOM FIELDS

Approved by:

Professor Michael Marder, Advisor
Department of Physics
University of Texas at Austin

Professor Kristen Grauman
Department of Computer Science
University of Texas at Austin

Professor Pradeep Ravikumar, Advisor
Department of Computer Science
University of Texas at Austin

Professor Brent Waters
Department of Computer Science
University of Texas at Austin

Professor Greg Sitz
Department of Physics
University of Texas at Austin

Date Approved: May 13, 2015

SIGNATURE PAGE

In partial fulfillment of the requirements for graduation with the Dean's Scholars Honors Degree in the Department of Physics and Department of Computer Science.

Physics Supervising Professor, Dr. Michael Marder — Date

Physics Honors Advisor, Dr. Greg Sitz — Date

CS Supervising Professor, Dr. Pradeep Ravikumar — Date

CS Second Reader, Dr. Kristen Grauman — Date

CS Honors Thesis Committee, Dr. Brent Waters — Date

To all those who have pushed me, pulled me, taught me, and challenged me.

"Though the rain comes in torrents and the floodwaters rise and the winds beat against that house, it won't collapse because it is built on bedrock."

– Jesus Christ, Matthew 7:25 (New Living Translation)

ACKNOWLEDGEMENTS

First, I want to note that I worked with Dr. Michael Marder and Dr. Pradeep Ravikumar at UT-Austin, along with Jon Zeitler and Dr. Larry Hopper from National Weather Service office in New Braunfels, Texas on my weather estimation project. In particular, the National Weather Service helped me to refine my project in scope and focus to specifically estimating rainfall, rather than cloud patterns, as an effort to create a more objective system for flash-flood predictions here in the "flash-flood capital of the world" in Central Texas. Further, I want to acknowledge Dr. Tu-Thach Quach and Dr. Jonathan Woodbridge at Sandia National Laboratories, who were my mentors on a project this summer dealing with machine learning and conditional random fields.

Thanks also to my parents who have supported me in my wild and crazy academic pursuits (including helping to edit this document). Thank you for all you have done for me. You have always worked to set me up for success.

TABLE OF CONTENTS

SUMMARY

Over the past 30 years, an average of 85 people died each year in the US due to flash-floods, making them the most fatal severe weather condition [1]. Particularly in Central Texas, the "most flash-flood prone area in the United States," [2] we need to accurately predict rainfall. However, meteorologists continue to manually adjust state-of-the-art physical models based on experience [3].

The National Weather Service creates flash-flood warnings based on Doppler radar station estimates of rainfall occurring over the past hour. As such, estimating rainfall directly impacts public safety – overestimates cause extraneous warnings that are easily ignored while underestimates fail to warn those in danger. Unfortunately, Doppler radar can miss the mark, yielding misleading results. If a method existed to update these estimates to make them more accurate, meteorologists could make better flash-flood predictions, saving lives. Furthermore, if this method were robust and based on data, rather than heuristics, it could be trusted as a step in post-processing of radar scans, seamlessly integrating with existing systems.

This project uses neural networks and conditional random fields – equipment from the toolbox of machine learning – to create a data-based model for updating Doppler radar rainfall estimations. To do this, the neural network is "trained" (uses actual observations to learn patterns) using the Lower Colorado River Authority's network of rain sensors. These rain sensors provide a ground-truth value for the rainfall in the Central Texas area. The neural network compares the Doppler radar estimates and the rain sensor ground-truth to learn how to better predict rainfall from radar scans. The neural network can also employ a rough estimate of the true rainfall from a

subset of the rain sensors to make even better overall rainfall estimates. Furthermore, conditional random fields provide a method of smoothing these predictions, leveraging the fact that drastic changes in rainfall are not physically reasonable (at least, in general case).

Based on the machine learning techniques referenced above, I sought to create a system that could make rainfall estimates more accurately (based on the ground-truth rain sensors) than the naïve estimate provided directly by Doppler radar. To do this, I implemented neural network and conditional random field algorithms, using many different experimental configurations. Each of these was used to create a potential system which was then compared against the Doppler radar estimates.

Although testing more configurations would provide additional statistical certainty, the system "in the middle" of those tested produced the best results. To be more explicit, the neural network model with a reasonably large number of neighboring cells used in the calculation, but no conditional random field algorithms applied, gave the smallest error under both utilized error metrics, meaning that it produced the best rainfall estimates. Furthermore, each tested configuration is consistent with or out-performs the standard Doppler radar estimate. As such, any of the tested network configurations seem to be a valid post-processing tool to create better rainfall estimates, providing a simple avenue to make more accurate flash-flood predictions in Central Texas.

CHAPTER I

BACKGROUND

Theoretical weather models date to the 1940s and '50s, as access to digital computers became possible[4]. Since then, meteorology shifted from an experience-based field to a scientific one. However, because even the simplest propagation models are chaotic, state-of-the-art predictions often still combine physical simulations with first-hand experience.

One method for producing forecasts has been largely unexplored: machine learning. Machine learning algorithms create approximation functions based on "training" data – examples where expected output is known. Neural networks are one such method, which are particularly good at approximating nonlinear functions.

Neural networks (and conditional random fields, to a smaller extent) have been used previously [5, 6, 7, 8]. Hung provides an excellent survey of previous work in [5]. Many of these predict rainfall at particular weather stations or even forecast weather conditions for an entire city. While many employ advanced neural network strategies (including past measurements in the estimation for current values with recurrent networks, adding in additional channels of data such as humidity, etc.), few attempt to predict rainfall on a point-by-point basis. French does this with simulated data over a region of approximately the same size as in my project (roughly, 100km by 100km), but uses roughly a quarter of the number of input and output nodes as mine [6]. Thus, my project exceeds French's by increasing the resolution of each cell over a similarly sized region, and applying real-world data sets to the neural network to determine how well an applied system can perform.

CHAPTER II

DATA DESCRIPTION

2.1 Doppler Radar

For my project, I used the region $-99°\ 14'\ 42"$ E to $-98°\ 25'\ 30"$ E and $30°\ 17'\ 57"$ N to $30°\ 59'\ 8"$ N. This region was split into a grid of 82×98 cells, each representing a region of $(0.0084°)^2$ or approximately $.74\text{mi}^2$ (a total area of $2300\text{mi}^2 = 6000\text{km}^2 = 1.5 \times 10^6\text{acres}$). Based on a Voronoi diagram (Thiessen polygons, see Section 2.4) of the rain gauges in the Lower Colorado River Authority's network of weather stations, the region is covered by 71 rain sensors (see Figure 2.1). Additionally, the National Weather Service has two radar stations that cover this region: EWX (Austin / San Antonio region, based in New Braunfels, Texas) and GRK (Central Texas region, based in Granger, TX). This allows for many experimental configurations of input to the neural network.

Using NOAA's Storm Events Database at www.ncdc.noaa.gov/stormevents/, I created a listing of all known major storms in Travis county between 2010 and 2014 that were categorized as tornadoes, hail, thunderstorm wind, flash-flood, flood, lightning, winter storm, strong wind, heavy rain, winter weather, or high wind. This request resulted in 230,294 NetCDF files from Doppler radar, split between the two radar locations. This large number is due to the construction of the requests: I requested data for the date of the recorded storm and the two days on either side. The Doppler radar stations sweep 360° every 3 to 15 minutes, running at the higher rate during storm activity.

National Weather Service radar data was requested using a custom Python script that makes requests through the National Oceanic and Atmospheric Administration's

(NOAA) National Climatic Data Center (NCDC), through the NEXRAD Data Inventory system ncdc.noaa.gov/nexradinv/. In particular, I requested the N1P channel of data, which is the one-hour precipitation. This is a Doppler-only-based estimate of the total precipitation over the last hour, which is "used to assess rainfall intensities for flash-flood warnings" [9]. This is based on a simplified model where the reflectivity measured by the difference in reflected signal in decibels (Z) is estimated (from prior data) to be related to the rain fall rate (in inches per hour) rate by a simple power law: $r = 1.54 \times 10^{-11} Z^{6.57}$ (based on data from the base reflectivity at srh.noaa.gov/jetstream/doppler/baserefl.htm). This measure is integrated over the last hour of estimates to estimate the one-hour precipitation. While this is a reasonable estimate, it is not the "whole story" as the radar is limited in terms of its ability to make this estimate. The NEXRAD Data Inventory system provides the data in NetCDF format, which can be read using the **ucar.nc2** package in Java.

2.2 LCRA Rain Gauges

The Lower Colorado River Authority data is available for individual rain gauges at hydromet.lcra.org/chronhist.aspx. In personal correspondence, I obtained the historical data needed for the region listed above in bulk. This entailed the rainfall recorded at each of the rain gauges between two measurements, with the times of those measurements given. To have the data sets represent the same physical quantity (estimated and observed precipitation over the past hour), I wrote a small script to add together the past hour's rainfall for every available data point at each sensor, and used this augmented value in all experiments. The Doppler and LCRA data are all given in inches, so the neural network's role is simply to update estimates from Doppler radar on the same scale as originally provided (see Section 3).

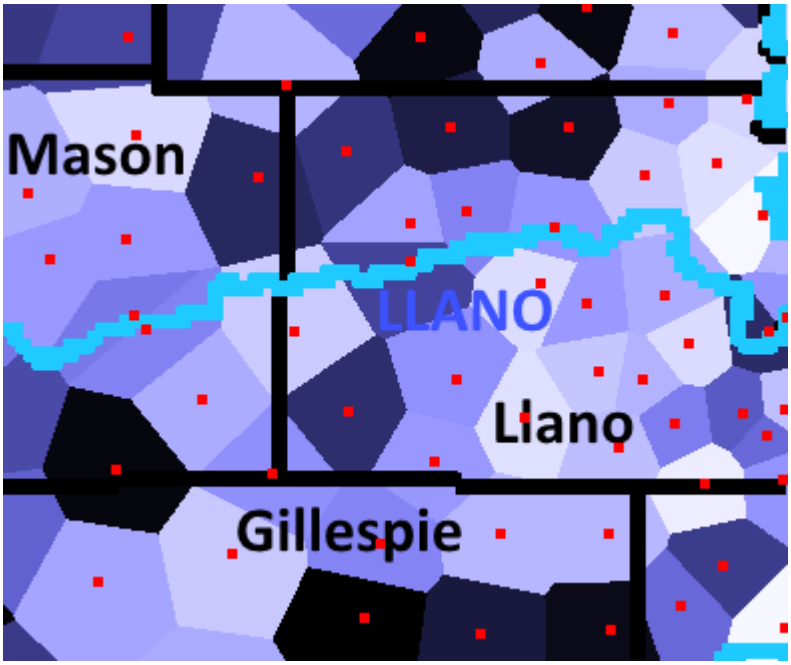


Figure 2.1: Map indicating the counties represented in the data analyzed (black), rivers (light blue), location of rain gauges (red), and a Voronoi diagram corresponding to the regions closest to each of the sensors (various shades of blue).

2.3 Cleaning Data

In order to create an effective training and test data set, I pruned the 230,000 NetCDF files to those that have corresponding data from at least 65% of the LCRA sensors in the region within 15 minutes of each radar scan. Furthermore, I only included those files from one radar station that are within 15 minutes of a scan from the other station (so that data from both stations can be used simultaneously). Finally, I removed all files where there was not a single pixel where both radar and LCRA sensors had a non-zero value. This left 2,189 files for each radar station. These correspond roughly to approximately 36 storms. Further, the rain values were normalized based on the maximum value from either data set (5.16 inches in a single hour). That way, the sigmoid used in training could theoretically produce any rain value.

The NetCDF files are given in polar coordinates as the distance d in miles from the radar station to the point observed and θ in degrees, measured clockwise from

North. To convert these points to a latitude–longitude pair (λ, ϕ), the following transformation was applied:

$$\begin{aligned}
\lambda_0 &= \text{radar station latitude} \\
\phi_0 &= \text{radar station longitude} \\
d &= \text{distance in miles} \\
\theta &= \text{bearing angle, measured clockwise from North} \\
R_E &= \text{radius of Earth, in miles} \\
\delta &= \frac{d}{R_E} && (2.1) \\
\lambda &= \arcsin\left[\sin(\lambda_0) \cdot \cos(\delta) + \cos(\lambda_0) \cdot \sin(\delta) \cdot \cos(\theta)\right] && (2.2) \\
\phi &= \phi_0 + \arctan\left[\frac{\sin(\theta) \cdot \sin(\delta) \cdot \cos(\lambda_0)}{\cos(\delta) - \sin(\lambda_0) \cdot \sin(\lambda)}\right] && (2.3)
\end{aligned}$$

2.4 Voronoi diagrams

Because the Doppler radar data gives a rainfall estimate over each cell in the grid (see Section 2.1), the rain gauge measurements should also be interpolated over the grid space. One particularly apt, straightforward way to interpolate values over a grid with discrete measurements is to use Voronoi diagrams.

Voronoi diagrams (otherwise known as Thiessen polygons) are a fairly simple construct. The following description generalizes to N dimensions, but for this project, a 2D discussion and visualization will suffice. Given a set S of points $p_i \in S$ with $p_i = (x_i, y_i)$, and a distance metric $d : \mathbb{R}^2 \times \mathbb{R}^2 \to [0, \infty)$, identify the corresponding set of points $R_i \in \mathbb{R}^2$ such that $\forall p \in R_i : \forall q \in S : d(p, p_i) \leq d(p, q)$. In other words, the Voronoi diagram partitions the plane into the sections that are each closest to a particular point under a given distance metric. So, by using the location of each of the rain gauges, I generated a Voronoi diagram of each region that is closest to a particular rain gauge. The measurement from each rain gauge is then applied over the corresponding region.

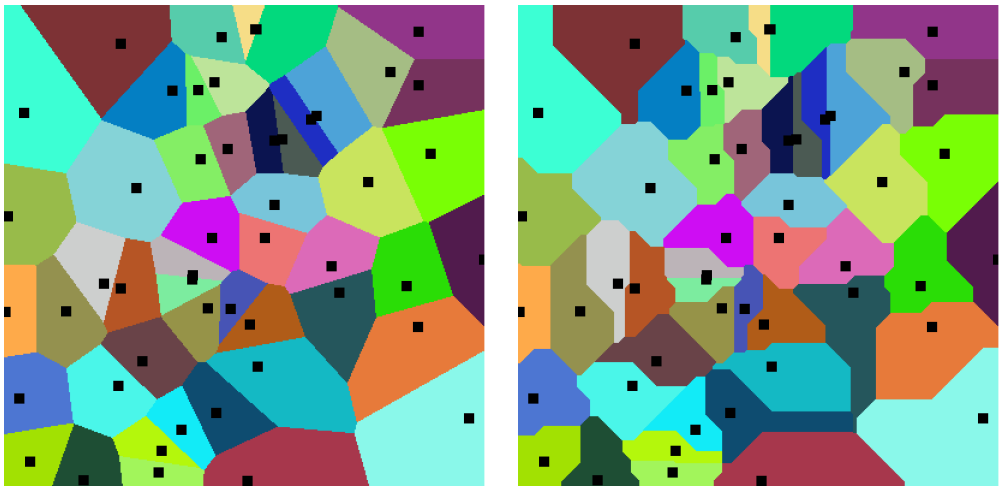


Figure 2.2: Figures demonstrating Voronoi diagrams for a set of points (black) under the Euclidean distance metric $d_E(p_1, p_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ (left), and the Manhattan distance metric $d_M(p_1, p_2) = |x_1 - x_2| + |y_1 - y_2|$ (right).

CHAPTER III

ARTIFICIAL NEURAL NETWORK STRUCTURE

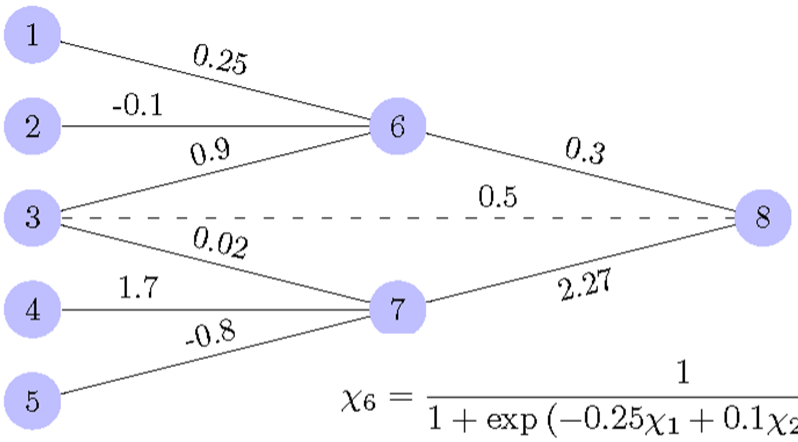
3.1 General Structure

An artificial neural network (ANN), for my purposes, is simply an approximation function that takes an input vector in $\mathbb{R}^n$ and produces a vector in $\mathbb{R}^m$, where n may not equal m. Often (as in the case for my project), the domain is reduced for simplicity to $[0,\ 1]^n$ and range to $[0,\ 1]^m$, creating a "normalized" ANN. To create the output vector, the ANN creates a fixed number of intermediate values for aiding the approximation. To do so, the ANN takes repeated transformations of weighted sums.

To make this more concrete, let x_i represent a value (input, output, or intermediate) in the network (also referred to as a "node," given the graphical nature of the structure of the ANN). For $\forall i \leq n$, x_i is simply the ith component of the input vector. Let N be the total number of nodes in the ANN ($N = n + m + k$, where k is the number of intermediate nodes). Then for $\forall i$ s.t. $N - m + 1 \leq i \leq N$, x_i is the $(i - n - k)$th component of the output vector. To compute the values of x_i for $i > n$, we have the following:

$$\begin{aligned} z_i &= \sum_{j<i} W_{i,j} \cdot x_j + \theta_i \\ x_i &= f(z_i) \end{aligned} \tag{3.1}$$

where θ_i is a fixed offset for each node, $W_{i,j}$ is the weight matrix, which indicates the relative weights in the sum (the entries of which will be discussed below), z_i is the weighted sum of all previous nodes, and $f(z)$ is called the "activation function." For a



$$\chi_6 = \frac{1}{1+\exp\left(-0.25\chi_1 + 0.1\chi_2 - 0.9\chi_3\right)}$$

$$\chi_7 = \frac{1}{1+\exp\left(-0.02\chi_3 - 1.7\chi_4 + 0.8\chi_5\right)}$$

$$\texttt{output} = \hat{Y}_1 = \chi_8 = \frac{1}{1+\exp\left(-0.3\chi_6 - 2.27\chi_7\right)}$$

Figure 3.1: Example neural network. Circles represent "nodes" in the network, where 1-5 are input nodes, 8 is the output node, and 6-7 are hidden. Values along the edge represent the weight in the sum seen in Equation 3.1. The dashed line between 3 and 8 shows that a node in the network can, in general, depend on any previous node (though this is not represented in the functions above).

normalized ANN, with no constraint on the weight matrix's values, f must be defined on $f : \mathbb{R} \to [0, 1]$. Typical examples used are: the step function $f(z) = \begin{cases} 0 & z \leq 0 \\ 1 & z > 0 \end{cases}$, the hyperbolic tangent function $f(z) = \frac{1}{2}(1 + \tanh(z))$, and the sigmoid function $f(z) = (1 + \exp(-z))^{-1}$.

In context, x_i (where $i < n$) refers to the estimated rainfall from one of the Doppler radar stations for a particular cell in the grid. For the last m values of x_i, it represents the ANN's best estimate of rainfall at a particular cell in the grid. The values z_i are all intermediate weighted sums of the rain values from the previous values in the network. Physically, this is essentially taking a weighted average of cells in the grid to produce an estimate. Once spatial structure and layers are added to the network, this is exactly the quantity z_i represents (with different layers representing a sort of

iterative calculation thereof). $f(z)$, the activation function, represents the constraints of the ANN's values for x_i (the estimated rainfall). The data are normalized (scaled to a range [0,1]) so each node can only take on such a value. So, $f(z)$ ensures our estimates are physically realizable.

3.2 Incorporating Spatial Structure

The weights in $W_{i,j}$ will be determined using the "backpropagation" algorithm (see Appendix A). But by applying constraints based on the problem at hand, many weights can be configured to be 0. For example, as shown above, $\forall j \geq i, W_{i,j} = 0$. One logical constraint when considering spatial data is a spatial dependence: when two nodes refer to locations that are "too far" away from each other, their values do not depend on each other (at least, not directly). Mathematically, we can associate each node x_i with a physical location (e.g., (x, y), or (latitude, longitude), etc.), called L_i. We assign a maximum distance $d_{\max}$, such that $\forall i, j$,

$$\|L_i - L_j\| > d_{\max} \Rightarrow W_{i,j} = 0$$

3.3 Layers

As an additional optimization, many ANNs have "layers" of nodes. To simplify the scenario, assume that the input and output vectors have the same number of dimensions ($n = m$). Now, let $N \mod n = 0$, meaning that the number of intermediate nodes in the network is a multiple of the number of input/output nodes. We can impose the following constraint on the weight matrix (in addition to those outlined above): $W_{i,j}$ can only be non-zero if $\exists p \in (\mathbb{Z} \bigcap [1, N/n])$ s.t. $\exists i', j' < n$ s.t. $i = pn + i'$ and $j = (p-1)n + j'$. In other words, the network is partitioned into sets $s_1 = \{x_1, x_2, ..., x_n\}$, $s_2 = \{x_{n+1}, x_{n+2}, ..., x_{2n}\}$ and so forth, where all the nodes in set s_p can only have non-zero weights for nodes in set s_{p-1}. Each of these sets is then called a "layer," because if the weights were viewed in terms of a weighted adjacency

matrix, there would be discrete groupings of nodes based on the connections in and out of each group. Thus, we have an "input layer" containing the values for the input vector, an "output layer" containing the values for the output vector, and $N/n - 2$ "hidden layers" which hold the intermediate values.

The layered approach enables greater computational efficiency when attempting to calculate the output vector. Calculations have to propagate from x_{n+1} through to x_N, but the layered approach only requires all previous sets of values to be computed, rather than all values. Said another way, each node in a layer is completely independent of the other nodes in the layer. So, multithreading can help speed up calculations by attempting to compute all values for a layer simultaneously (though obviously, unless $n \sim$ the number of CPUs available, not all values can be computed at the same time – just that they can be computed as an atomic unit).

My implementation relaxes the layering constraint slightly. Nodes in any set s_p with $p > 1$ can be connected to the input vector (nodes in s_1). The intuition here is that if the best approximation involves fewer iterations of the activation function f than the number of layers in the ANN, this implementation should be able to converge on that approximation instead.

CHAPTER IV

TESTING ANN IMPLEMENTATION

One test I performed on my neural network implementation to determine if it was indeed generating a reasonable approximation function was to use the simple case of a sigmoid. I randomly generated 100 points $x_i \in [0, 1)$, then applied the function $f(x) = 1/(1+\exp(-x))$, creating a test set of points $(x_i, f(x_i))$. I then applied various ANN configurations to this data set, with varying number of iterations (training the network on all 100 values each time):$\{10, 20, 50\}$, and varying number of hidden layers (as defined in Section 3.3): $\{0, 1, 2, 4, 8\}$. 100 ANNs with randomly initialized starting weights were trained for each configuration (number of training iterations and number of hidden layers). Since the ANN is actually an iterated sum of sigmoids (as given in Equation 3.1 with f defined as the sigmoid above), the entire network can be reduced to a function of the input variable x_i through expanding the weighted sums.

4.1 Series Approximation

For example, the following function was produced as the result of 50 training iterations with a single hidden layer (so two total layers):

$$\hat{f}(x_i) = \frac{1}{1 + e^{-(0.865 \cdot x_i + 0.683 \cdot (1/\ (1+\exp(-(0.987 x_i + 0.482))))+\ -0.426)}} \tag{4.1}$$

Because this is no longer a simple sigmoid, we have to construct a means by which we can determine if it indeed approximates the function $f(x_i) = 1/(1 + \exp(-x_i))$. One method is the Maclaurin (Taylor, centered at $x_i = 0$) series approximation of both functions. To eighth order, the sigmoid function can be approximated as:

$$f(x_i) \approx 0.5 - 0.25x_i + 0.02083x_i^3 - 0.00208x_i^5 + 0.00021x_i^7$$

The ANN represented in Equation 4.1 has coefficients that are extremely close to those of the ideal sigmoid:

$$\begin{aligned}\hat{f}(x_i) \approx & 1.08 \times 10^{-4} x_i^8 - 4.27 \times 10^{-4} x_i^7 - 0.000495 x_i^6 \\ & + 0.00322 x_i^5 + 0.00186 x_i^4 - 0.0251 x_i^3 - 0.00438 x_i^2 + 0.256 x_i + 0.499 \end{aligned} \quad (4.2)$$

To the doubting statistician, this seems rife with potential for coincidence. However, the tests described above show conclusively that this alignment is no accident.

For each configuration of the ANN (number of training iterations and number of hidden layers), each of the 100 ANNs were expanded mathematically as in Equation 4.1, which simply expands all the levels of the network to their dependence on the input variable x_i. The resultant functions (which were iterated sigmoid functions) were then approximated to 8th order (choice of 8th order was arbitrary, but meant to reflect a broad range of coefficients and help model the nonlinearity of the function being approximated). For each configuration, the mean and standard deviation of each coefficient was computed. The average coefficient for each configuration is used to create Figure 4.1. For example, for 10 iterations and 0 hidden layers, the coefficients for the approximation $\hat{f}(x_i) = \sum_{n=0}^{8} c_0 x_i^n$ are:

Coefficient	Exact	Measured
c_0	0.5	0.508 ± 0.048
c_1	0.25	0.411 ± 0.023
c_2	0	-0.001 ± 0.001
c_3	-0.020833...	-0.012 ± 0.002
c_4	0	0.0001 ± 0.0001
c_5	0.0020833...	0.0018 ± 0.0003
c_6	0	-0.000015 ± 0.000010
c_7	-0.000210813...	-0.00018 ± 0.00003
c_8	0	$1.7 \times 10^{-6} \pm 1.2 \times 10^{-6}$

Table 4.1: Table of Maclaurin coefficients taken from the mean and standard deviation of 100 ANNs with 10 training iterations and 0 hidden layers, compared with the exact result of the Maclaurin series for $1/(1 + \exp(-x))$.

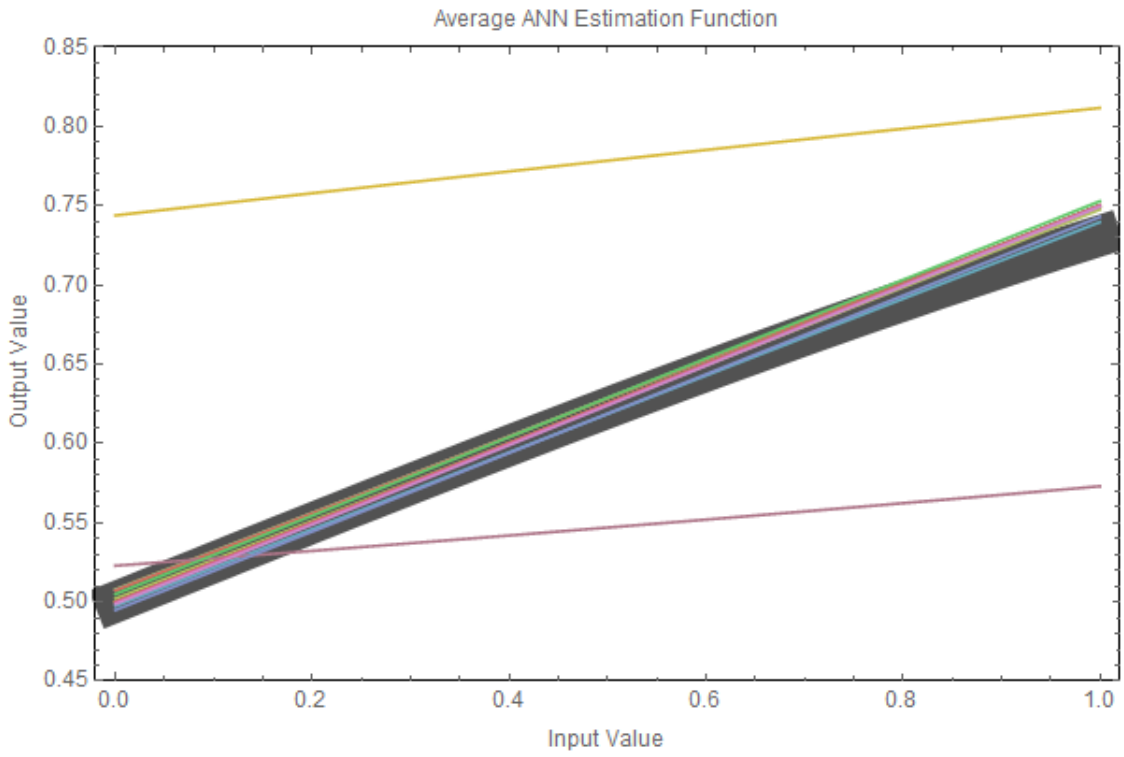


Figure 4.1: Average function predicted by each of the 15 configurations. In black is the true sigmoid function. The two functions that are visually far from the true sigmoid are those with 20 and 50 iterations and 0 hidden layers. Their non-correspondence with the rest of the results would then appear to be a simple case of overfitting an insufficiently complex model.

By similar construction for all 9 coefficients for all 15 configurations, I computed the z-score for each coefficient, taking the model under the null hypothesis to be Gaussian with mean value (μ_i) of each coefficient to be the Maclaurin approximation of a sigmoid, and the standard deviation taken to be the standard error from each sample ($\hat{\sigma_i} = \mathrm{SE}_i$). With this, I performed a two-tailed z-test of the distributions (since each coefficient was the average from 100 ANNs, the Normal/Gaussian approximation to the t-distribution should be valid) for each coefficient at the $\alpha = 0.05$ level. In other words, for each coefficient, I determined how many standard deviations away from the exact result it was ($z_i = \frac{x_i - \mu_i}{\hat{\sigma_i}}$), and determined how likely a result that extreme was:

$$P(|z| > |z_i|) = \int_{-|z_i|}^{|z_i|} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{z^2}{2}\right) dz = \text{erf}\left(\frac{|z_i|}{\sqrt{2}}\right) \tag{4.3}$$

This is known, perhaps more simply as the p-value for each coefficient in each configuration. With $N = 9 \times 15 = 135$ total coefficients, I determined how many had a p-value less than $\alpha = 0.05$: 7. That means 5.19% of the time, the result was so strange that it should have appeared 5% of the time. Said another way, 5.19% of the time, the coefficient has a z-score with $|z_i| > 1.96$, which corresponds to $\alpha = 0.05$.

Under the assumption that this distribution draws from the Binomial distribution with size $N = 135$ and "success" probability $p = \alpha = 0.05$ (where the mean number of times of seeing such a strange result is $\mu = Np = 6.75$ and the standard deviation is $\sigma = \sqrt{Np(1-p)} = 2.53$), we could expect to observe 7 or more coefficients to have $|z| > 1.96$ very often (51.5% of the time, to be more precise). This is one way to look at multiple hypothesis testing, and at least provides reasonable evidence that we shouldn't be surprised by the result.

Furthermore, I performed the same analysis after using the Benjamini-Hochberg procedure [10], which controls the false discovery rate (number of rejections of true hypotheses over total number of hypothesis rejections). See Appendix B for details. This resulted in 3 coefficients deemed significant out of 135. This is a more significant result (in a very strong sense, this procedure controls for false discoveries). If a second-order approximation is used instead, 0 coefficients are significant (with third-order and above creating significant coefficient differences). To me, this indicates that using the series approximation to a high order (especially if the high-order terms are close to zero), the slight variation in estimated coefficients may be large enough to falsely indicate a significant difference. However, this is indeed a potential source of doubt of reliability in the ANN implementation. Because it was only a small number of coefficients (that go to zero with a smaller-order approximation), and because other metrics indicate reliability in the ANN, I am inclined to believe they are

only significant through what amounts to insufficient training to fine-tune parameters beyond a reasonable amount.

Because of this, we can fully expect to see such a distribution of z-scores for each of the coefficients, so we cannot reject the overall null hypothesis. In other words, we have failed to provide sufficient evidence to show that the results obtained from the ANN code are inconsistent with an approximation of the function used for training. As such, we can then proceed to assume that the ANN code produces functions that are reasonable approximations of the underlying training function.

4.2 Best Fit

One less extensive way to test whether the ANN code produces reasonable approximation functions is leveraging nonlinear model fitting functions, such as *Mathematica*'s `FindFit` function. In this case, the ANN's approximation function is used to predict pairs $\left(x_i, \hat{f}(x_i)\right)$, for input values $x_i \in \{0, 0.001, 0.002, 0.003, ..., 0.998, 0.999, 1\}$. `FindFit` is then used to fit the function $f_{a,b}(x) = (1 + \exp[-(a \cdot x + b)])^{-1}$ for coefficients a, b. Once again grouping the ANNs by configuration, I can find the mean and standard deviation of the estimations of a and b, and compare them to the true values $a = 1, b = 0$. Again assuming that the standard deviation for each parameter for each configuration is the standard error from the sample, I can determine if the average estimate is within 1.96 standard deviations of the true value (again, where $z = 1.96$ corresponds to $\alpha = 0.05$). And indeed, this is the case for all 15 configurations for both parameters. The likelihood of having all of the 30 values be within 1.96 standard deviations (again assuming a Binomial distribution, this time with $N = 30$ and $p = \alpha = 0.05$, and calculating the probability that the number of "successes" is 0) is 21.5%. Thus, once again, our result is well within reason for the null hypothesis that the ANN code produces functions that are reasonable approximations of the underlying training function. So, we fail to reject the null hypothesis and can assume

(with two different tests in hand) that the code works as intended.

CHAPTER V

CONDITIONAL RANDOM FIELDS

The idea behind conditional random fields (or CRFs) stems from the classification problem in a graph structure, $G = \langle V, E \rangle$. In relation to the discussion above, the vertices would represent the individual cells in the grid, and the edges would represent the connections to neighboring points on the grid. While the ANN produces a single value for the rainfall estimate, we can easily modify the result to be categorical in nature, rather than strictly numerical. In particular, we can instead create an estimated probability distribution at each point (vertex) of the rainfall being one of several different levels of rain (no rain, light rain, heavy rain, etc.). Leaving the particulars of this change unspecified for this discussion for now, we can think about the implications of having a probability distribution among labels at each vertex.

Assume we have output from the ANN that represents the probability distribution between $|\mathcal{L}|$ "labels" at each vertex $v \in V$ (so the output size obtained from the neural network would then be $|V| \cdot |\mathcal{L}|$). When attempting to report the rain level at each vertex, we can report the label with the highest probability (to introduce some notation, let $P_v(l)$ be the probability of observing label $l \in \mathcal{L}$ at vertex $v \in V$ – the level reported is then $\text{argmax}_{l \in \mathcal{L}} P_v(l)$). But can we do better? For example, if we know $\exists (u, v) \in E$ and for some $l \in \mathcal{L}$, we have $P_v(l) \approx 1$, then we might expect $P_u(l)$ to similarly be close to 1. Said another way, if we are confident about the label for one vertex, and it is connected to a neighbor, should we not expect the neighbor have the same label? If we apply this to weather, we could say that if we have high confidence in the probability of severe rain at some point (a latitude and longitude), we should be surprised if we are predicting the weather to be dry 1000 ft away.

CRFs look to smooth out these discrepancies to provide a better regionalized solution, that (theoretically) creates a better global solution. We could try all possible combinations of labels, but that takes too much computation time. In an acyclic graph, there is always an exact solution, but in general, there is not always an exact global solution.

One additional aspect we need for the construction of CRFs is a "pairwise probability distribution" that describes how likely it is to have two vertices to have a particular set of labels. That is, we need some function

$$P_{v,u}(l,k) \in [0,1] \text{ for } v,u \in V \text{ and } l,k \in \mathcal{L} \tag{5.1}$$

Often, this is defined in terms of some energy function:

$$P_{v,u}(l,k) = \exp\left[-E_{v,u}(l,k)\right]/Z \tag{5.2}$$

This energy function can be seen as a cost of having this set of labels for these two vertices. One common example is the Potts function (for some $A > 0$):

$$E_{v,u}(l,k) = \begin{cases} A \cdot (1 - \delta_{l,k}) & (v,u) \in E \\ 0 & \text{otherwise} \end{cases} \tag{5.3}$$

In other words, if these points are not connected directly in the graph, they have no consequence on each other. If they are connected, and have the same label, there is also no cost – the only cost comes from changing labels between neighboring nodes. This is a somewhat naïve function. In the weather example, "dry" and "moist" should theoretically have a smaller cost of switching than "dry" and "severe." But in practice, the Potts model ends up being a reasonable guess (that can then be improved). However, it is worth noting that any energy function we define adds subjectivity back to our model, which provides a potential source of bias and thus a source of systematic error.

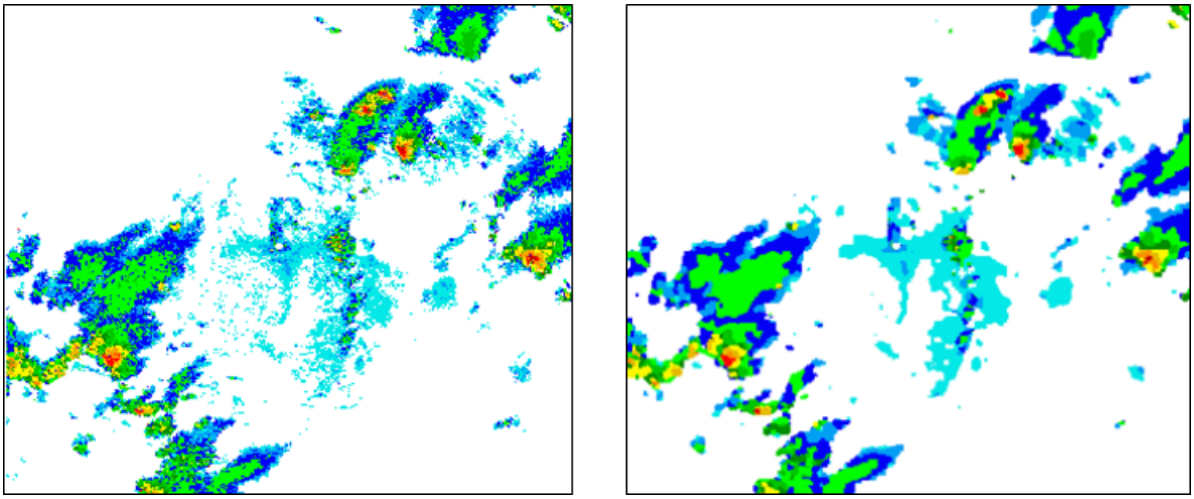


Figure 5.1: (left) Example of neural network's cloud reflectivity forecast for Albuquerque, New Mexico. (right) Same prediction, with conditional random field algorithm applied, showing the "smoothing" effect. Training data courtesy of the National Weather Service.

5.1 Loopy Belief Propagation

One algorithm to use this CRF paradigm is "Loopy" Belief Propagation (so-called, because it ignores cycles [or "loops"] in the graph, allowing the following iterative calculation to be "loopy", violating some of the assumptions made in the construction, see [11]). In this case, we will have neighboring nodes send "messages" to one another, giving each other the best impression they have of what labels their neighbors should take on.

So here, we'll define $\Gamma_v = \{u \in V \mid (u, v) \in E\}$ as the "neighbors" of $v \in V$. Then, for each $s \in V$, for each $t \in \Gamma_s$, and label $l \in \mathcal{L}$, we have messages defined as:

$$m_{s \to t}^{(i)}(l) = \alpha(s, t, i) \cdot \sum_{k \in \mathcal{L}} \left(P_s(k) \cdot P_{s,t}(k, l) \cdot \prod_{u \in \Gamma_s - \{t\}} m_{u \to s}^{(i-1)}(k) \right) \tag{5.4}$$

where the $^{(i)}$ part of the message is simply to label the messages sent at each iteration of the procedure. The $\alpha(s, t, i)$ term is to normalize the messages:

$$\sum_{l \in \mathcal{L}} m_{s \to t}^{(i)}(l) = 1 \tag{5.5}$$

After a sufficient number of iterations I, we can compute the updated "belief" at each vertex:

$$B_v(l) = \alpha(v) \cdot P_v(l) \cdot \prod_{u \in \Gamma_v} m^{(I)}_{u \to v}(l) \tag{5.6}$$

where again $\alpha(v)$ is simply for normalization:

$$\sum_{l \in \mathcal{L}} B_v(l) = 1 \tag{5.7}$$

The final estimate of $B_v(l)$ replaces our original estimate $P_v(l)$, by taking into account the values of neighbors in the graph.

5.2 Converting ANN Output to CRF Input

Unless the ANN is trained to predict a probability distribution from the start, the rainfall prediction it makes at each point must be modified in such a way as to be used in a CRF model. To create the "labels" in the CRF, one can simply segregate rainfall estimates into discrete bins (not unlike the process used to create a histogram). In this project, I employed the following transformation. Given a scaled (normalized w.r.t. the maximum rainfall observed: 5.16" in an hour) rainfall estimate in the unit interval, segregate it into one of 9 bins. These bins correspond to the values 0, 0.125, 0.25, ... 0.875, 1.0 (again, normalized rain per hour). If the label l_i corresponds to the rainfall estimate $i/8$, then the corresponding probability distribution for each label given a rainfall estimate r is:

$$P(l_i|r) = \begin{cases} 1 - 8 * (r - i/8) & i/8 < r < (i+1)/8 \\ 8 * [r - (i-1)/8] & (i-1)/8 < r < i/8 \\ 0 & \text{otherwise} \end{cases} \tag{5.8}$$

So, if $r = .3$, it is .05 above 0.25, which, out of a bin size of 0.125, means it is 40% of the way along the number line between 0.25 and 0.375. In that case, we assign a likelihood of 60% of the rainfall being 0.25 (label l_2) and 40% to being label l_3. A

value halfway between these values has an equal probability of taking on either label. But one very close to one of the label's rainfall values will most likely be found with that label.

In this project, neighboring nodes in the grid were defined by a maximum distance in units of grid cells. A maximum distance of 1 indicates that each cell is only related to adjacent cells in the 4 cardinal directions. So, a distance of 2 would be all adjacent cells including diagonals plus the cells a distance of 2 away in the cardinal directions.

5.3 Cost Models

The cost models employed in this project were not dependent on geography; the distance between two points did not impact the pairwise probability distribution directly – only by whether the points were close enough to be neighbors (the distance for this condition is an experimental parameter). Four pairwise cost models were applied to each ANN prediction as a means of testing multiple models. Two were Potts models with varying strengths, and two were motivated by a physical cost. The Potts models were exactly those represented by Equation 5.3, with $A = .5, 2$. The physical models added some intuition. Whereas the Potts model only looks for differences, a more realistic model may incorporate the physical differences between the quantities being estimated. For example, while we may expect slight variation in rainfall estimates between neighboring locations, we generally don't expect severe weather to occur within visible distance of calm weather. As such, two energy models were used:

$$E^{(1)}_{v,u}(l,k) = \ln\left[\sqrt{1+|l-k|}\right] \tag{5.9}$$

$$E^{(2)}_{v,u}(l,k) = \ln\left[(1+|l-k|)^2\right] \tag{5.10}$$

These correspond to probabilities (before normalization) of $\sqrt{\frac{1}{|l-k|+1}}$ and $\left(\frac{1}{|l-k|+1}\right)^2$. These provide varying levels of penalty for drastic differences between neighboring

labels. For non-neighbors, there is no cost, because knowing the value of one has no impact on the other. So outside any maximum length scale, the energy cost is 0.

CHAPTER VI

RESULTS

6.1 Testing Methodology

The goal of my project was to reliably create a system that produces better rainfall estimates than the Doppler radar alone. To do this, I investigated many configurations of the ANN and multiple CRF cost functions in an effort to hone in on prediction systems that could improve flash-flood warning systems. Each trial consisted of training the ANN on 80% of the 2,189 Doppler radar scans, and computing the error (defined below) for the training and test sets (only 25% of the training set is considered for this metric, and is chosen randomly, ensuring that number of scans is not a significant contributor to any difference in error). Once the ANN prediction has been made, the prediction is binned and transformed into a discrete probability distribution (see Section 5.2). Four CRF algorithms are applied in turn, with the same error metrics calculated for these. The goal then, is to determine which scenario produces the best results for error across varying number of ANN hidden layers, number of training iterations, maximum neighbor distance, and percentage of the LCRA sensors used as input. For example, if one of the Potts models systematically performs better than the other configurations, it is the best bet for being able to produce a better estimate than the Doppler radar alone. The best performing model's error will then be compared against the same metrics run on the 2,189 Doppler radar scans, to determine if the best system described by the methods outlined above can improve predictions using post-processing alone.

6.2 Error Metrics

In determining the "best" system to process the Doppler radar scans into more accurate rainfall predictions, an appropriate error measure must be chosen. The most straightforward metric of error is the mean of the square of the difference between observed and expected rainfall at each point in the grid. This is represented by

$$\epsilon_{\text{mse}} = \frac{1}{N} \sum_i (y_{\text{obs}} - y_{\text{exp}})^2 \tag{6.1}$$

Considering that 99% of the full data set of 203,000 files are entirely dry, being able to predict identically the value 0 at all points would provide a great "best guess." By taking only the selection of the ∼1% of the full data set where both radars were active, and the radars and rain gauges all registered some rain helps ensure the approximation function $f(\vec{x}) \approx \vec{0}$ is not among the best models for rainfall. Since only situations where rain was present were chosen, this is less likely the case, but there is still something potentially useful to be said for weighting the error by the strength of the storm. So small differences in estimation when the weather is relatively dry are less important than small differences when the weather is severe. In this case, the difference between observed and expected value is multiplied by the square of the average of the observed and expected value. This is represented by:

$$\epsilon_{\text{str}} = \frac{1}{N} \sum_i (y_{\text{obs}} - y_{\text{exp}})^2 \cdot \left(\frac{y_{\text{obs}} + y_{\text{exp}}}{2} \right)^2 \tag{6.2}$$

Because even in the subset of the data used, many input and output grid points predict zero rainfall, `strength` is likely to have a smaller value on average, compared to `mse`. For example, running this metric on the ∼2,000 files, taking LCRA data as the expected value and the average estimate from the two Doppler radar stations as the observed value, and finding the mean and standard deviation of each, I arrived at `mse`= 0.0026 ± 0.0057 and `strength`= 0.000054 ± 0.0020. These show how we

might expect the two error metrics to behave for each configuration of the system, and provide a baseline for determining whether any configuration can out-perform the standard Doppler estimate.

6.3 Configurations Tested

For the scope of my thesis, I was able to test 6 main neural network configurations (with 6 "baseline" neural networks, see below), with five of those augmented by application of conditional random fields. These were:

Configuration Number	Number of Hidden Layers	Maximum Distance for Neighbors	Iterations of Training
1	1	1.0	1
2	1	1.0	5
3	5	1.0	5
4	5	3.0	5
5	2	3.0	10
6	5	3.0	10

Table 6.1: Table of configurations used to generate models for estimating rainfall.

Configurations 1–5 were post-processed with the Loopy Belief Propagation algorithm with the following pairwise probability functions (using notation from Chapter 5), which are examples of the Potts energy (Equation 5.3) and custom "strength" energy (Equation 5.9):

$$P_1 \rightarrow P_{s,t}(k,l) = \begin{cases} \frac{1}{1+\exp(-.5)} & k = l \\ \\ \frac{.5}{1+\exp(-.5)} & k \neq l \end{cases} \tag{6.3}$$

$$P_2 \rightarrow P_{s,t}(k,l) = \begin{cases} \frac{1}{1+\exp(-2)} & k = l \\ \\ \frac{2}{1+\exp(-2)} & k \neq l \end{cases} \tag{6.4}$$

$$P_3 \rightarrow P_{s,t}(k,l) \propto \sqrt{\frac{1}{|l-k|+1}} \tag{6.5}$$

$$P_4 \rightarrow P_{s,t}(k,l) \propto \left(\frac{1}{|l-k|+1}\right)^2 \tag{6.6}$$

As a "baseline" to determine whether the approach outlined above (using the entire region for input and output of the ANN), I also created a "simple" version with the same number of hidden layers and training iterations, but only a single cell for input and output. It is trained on each cell of the grid for each radar scan the "regional" ANN was trained on. This simple ANN provides a metric to determine whether adding neighbors is truly the source of any improvement, rather than just creating a generally better-fitting point-wise nonlinear model. As such, results between the two types of ANNs for the "same" configuration (number of hidden layers and number of training iterations) are compared.

6.4 Configuration Results

Figures 6.1–6.8 display the results of the tests. Plotted in each is the mean of the error metric referenced for the "test" data set (the 20% of the full data set that was not used

for training), with the standard error of the mean used for error bars (SEM = $\frac{\sigma}{\sqrt{n}}$). The same error metrics calculated for the average Doppler-radar estimate as the "observed" value in the calculation can be seen in gray in each figure (constant value across configurations).

6.4.1 Results for $\epsilon_{\mathbf{mse}}$

This error metric is calculated with Equation 6.1.

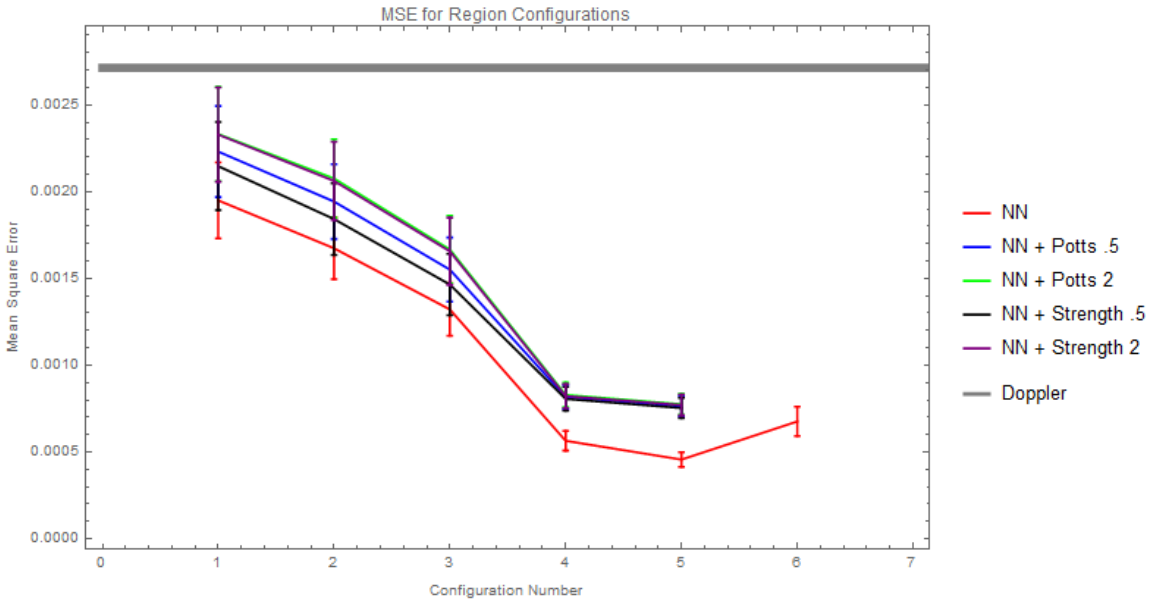


Figure 6.1: ϵ_{mse} for the region-based ANNs. The same metric calculated between the average Doppler-radar estimate and the rain gauge grid is in gray. Other colors as noted in the legend.

Figures 6.1–6.3 indicate several conclusions (insofar as can be noted from these configurations under this error metric):

1. CRFs do not decrease the overall error (perhaps the resolution was not refined enough for neighbors to sufficiently contribute to one another).

2. For small numbers of training iterations, the "simple" ANN performs better than the "regional" counterpart.

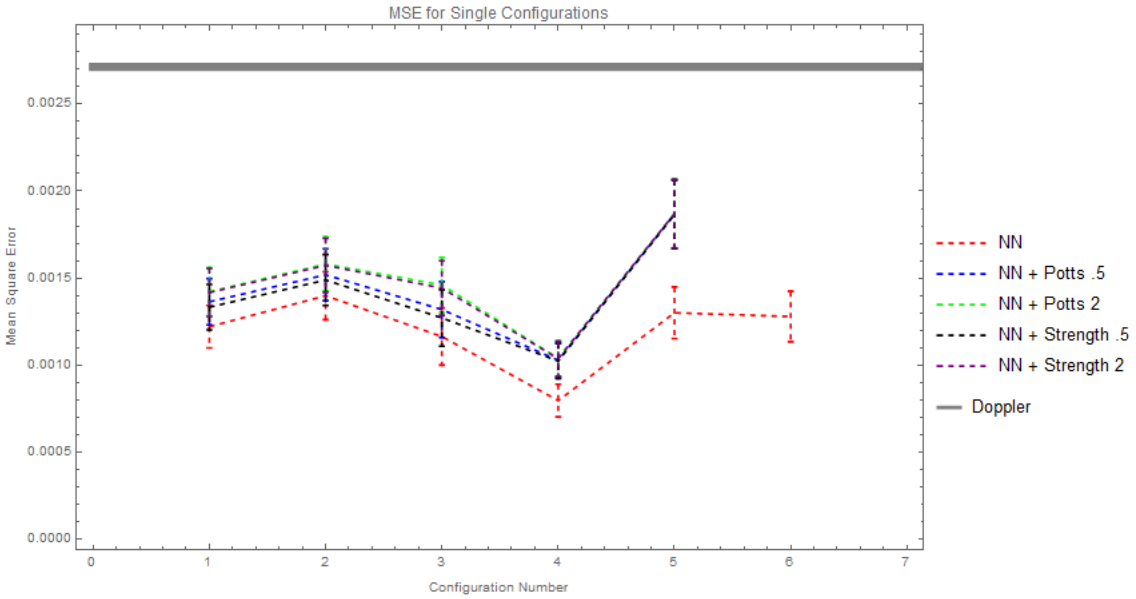


Figure 6.2: ϵ_{mse} for the single-cell ANNs. Colors as above in Figure 6.1.

3. When the regional ANN is allowed to view a multitude of neighboring cells, results improve dramatically.

The first conclusion is evident from each figure. The post-processed results tend to have a higher mean error than the un-processed results. Even though in some cases, the values were within 1-2 standard errors of the mean, there is no indication that CRFs decrease error. This was unexpected. However, some potential reasons are that:

a. the CRF pairwise probability functions are of arbitrary forms – perhaps a better physical model would improve results,

b. the constants used were selected arbitrarily – perhaps they were each either systematically too low or to high to produce valuable results, and

c. the Loopy BP algorithm was applied for 4 iterations each time – perhaps additional iterations would improve results.

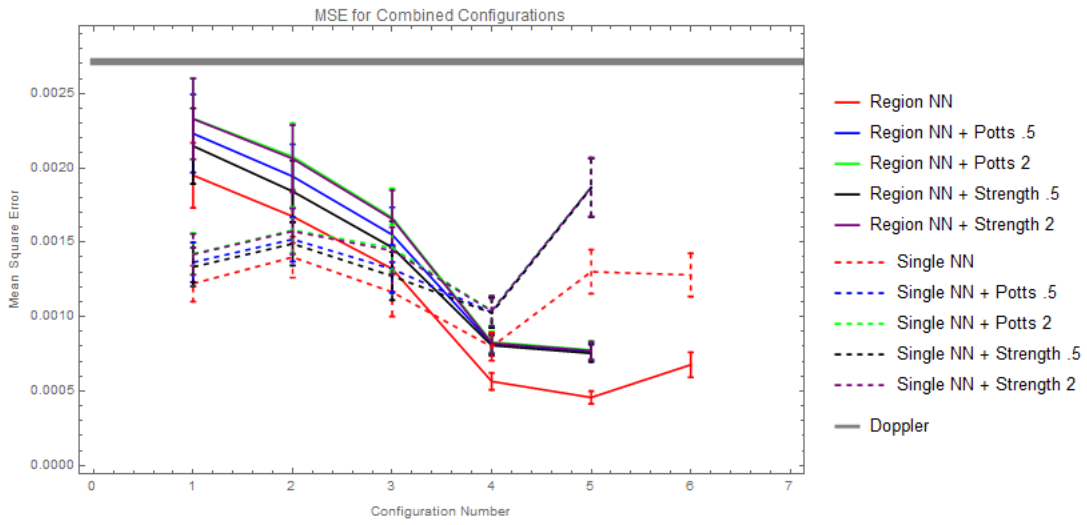


Figure 6.3: Combined results for ϵ_{mse}, combining Figure 6.1 and 6.2.

The second conclusion – that small amounts of training led the "simple" ANN to perform better – may have arisen because the simple network essentially has more training data. By altering the problem to produce cell-by-cell results (rather than the entire grid all at once), the simple network trains on 82×98 times more data (in terms of the problem it is solving), due to the grid being 82×98 cells.

The fact that the regional network saw drastic improvements in ϵ_{mse} when additional neighboring cells were considered is extremely encouraging. That indicates that increasing spatial structure improves rainfall estimates – which was the hope for this project from the start.

6.4.2 Results for ϵ_{str}

This error metric is calculated with Equation 6.2.

In this instance, it appears that the addition of the post-processing step with conditional random field algorithms had no significant impact either way on the error (Figures 6.4–6.6). Once again, the regional strategy appears to produce a smaller error than the "single" counterpart, for situations of sufficient complexity for the regional ANN. However, the SEM for the distributions is large enough to suggest

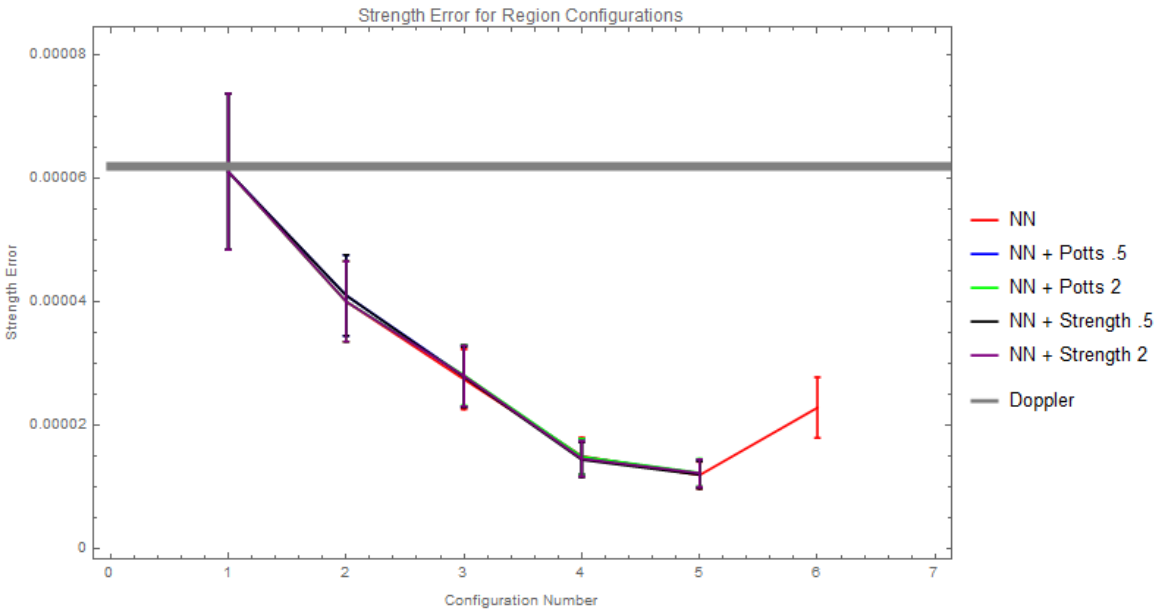


Figure 6.4: ϵ_{str} for the region-based ANNs. The same metric calculated between the average Doppler-radar estimate and the rain gauge grid is in gray. Other colors as noted in the legend.

potentially little or no difference between regional and single strategies.

6.4.3 Overall Results

Figures 6.7–6.8 strip out the ANN-only results given in the previous two sections. This is because in each case the ANN-only results were consistent with or better than the CRF-augmented results. These two figures indicate that given sufficient complexity (number of neighbors, number of training iterations, number of hidden layers), the regional strategy seems to out-perform the single strategy. What is more is that for both error metrics, both ANN solutions provided a smaller error than the naïve estimate from averaging the Doppler radar scans. Intuitively, this should be the case, so long as the networks don't become over-fitted to the training data (and then perform poorly on the test data set), but comforting to see it realized in the experiment.

Thus, based on this selection of ANN configurations, I determined that the regional

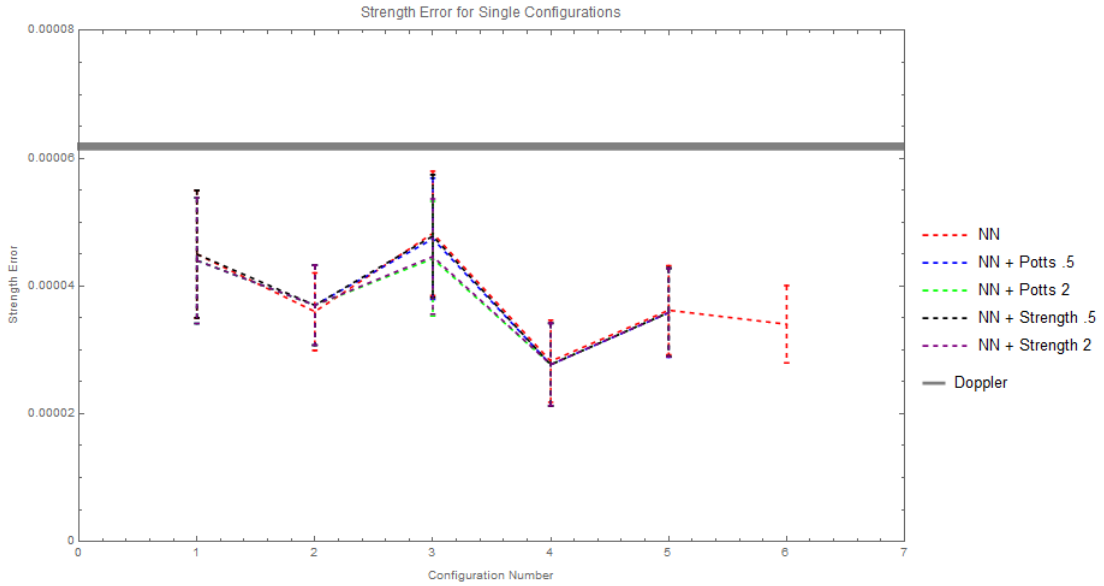


Figure 6.5: ϵ_{str} for the single-cell ANNs. Colors as above in Figure 6.4.

strategy – leveraging spatial structure in the grid by allowing the ANN to consider physical neighbors at each layer in the ANN – produces better rainfall estimates. As such, a system like this could be applied to real-world data processing streams for precipitation estimation. In doing so, the error would be decreased in estimates, which in turn provides a better baseline for flash-flood predictions. With better flash-flood predictions, warning systems can improve, keeping people safer during extreme weather in Central Texas.

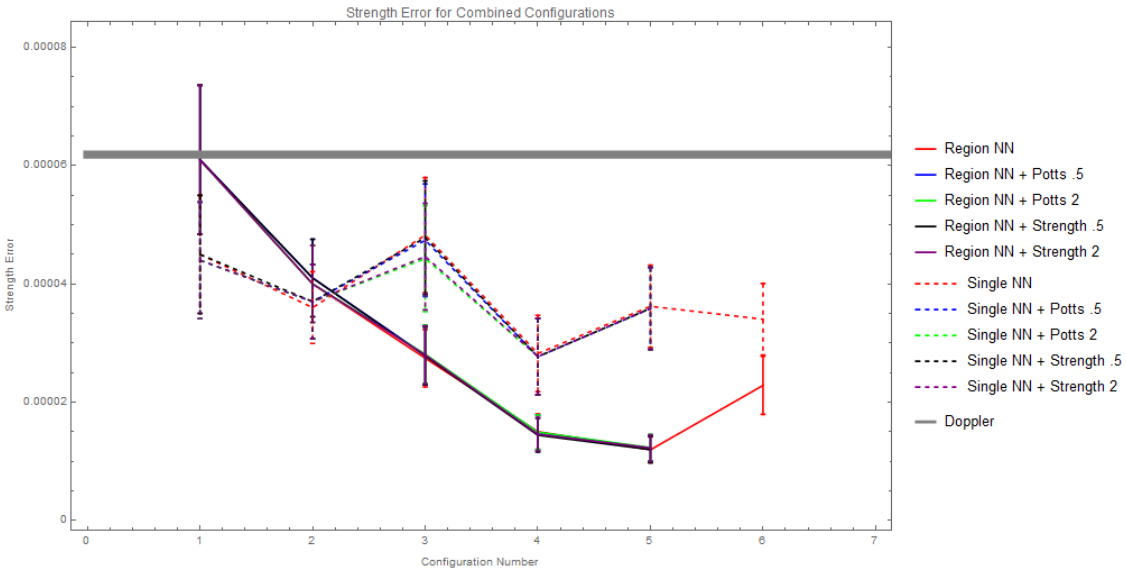


Figure 6.6: Combined results for ϵ_{str}, combining Figure 6.4 and 6.5.

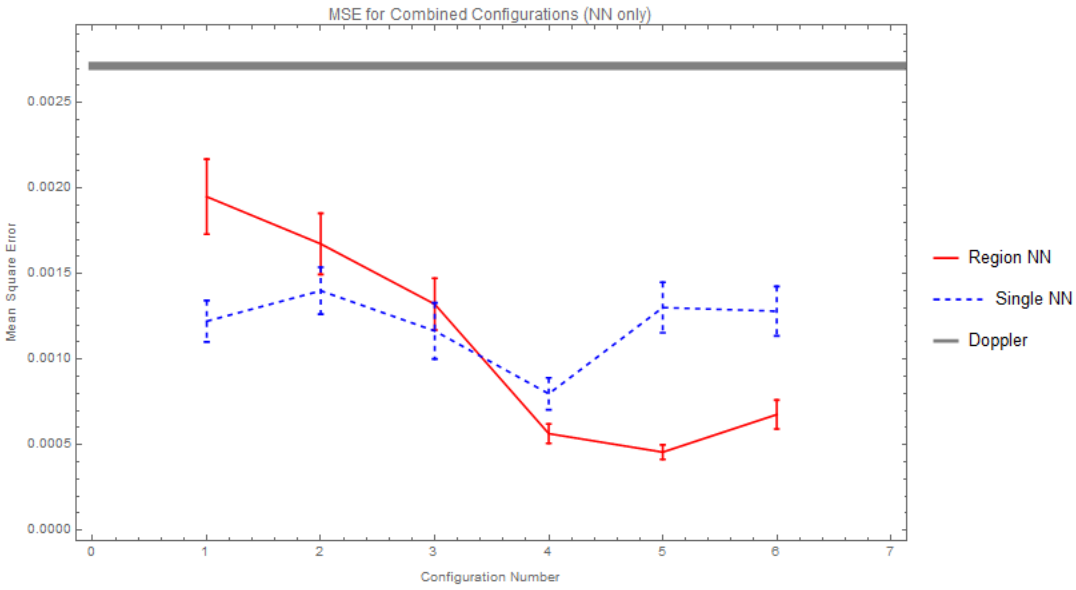


Figure 6.7: Results for ϵ_{mse}, showing just the "plain" ANNs (no CRFs).

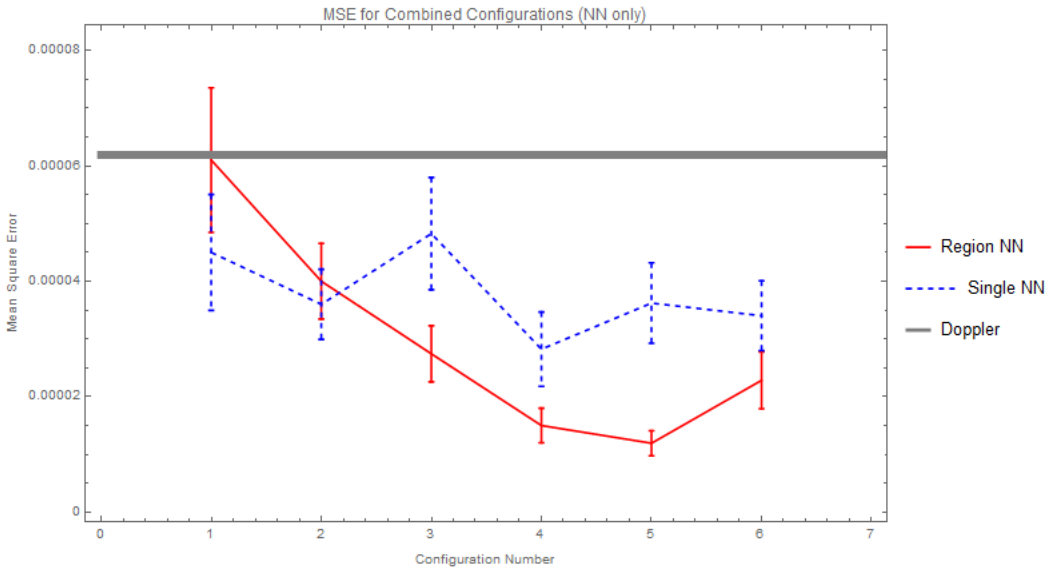


Figure 6.8: Results for ϵ_{str}, showing just the "plain" ANNs (no CRFs).

CHAPTER VII

POTENTIAL EXTENSIONS OF WORK

Some ways that this project could be extended to further investigate the claims made and to produce even better models for rainfall estimation are:

- Increasing the number of hidden layers, training iterations and maximum distance for neighbors for the regional network to determine where improvements to error become stagnant.
- Identify and test additional pairwise probability models for the CRF algorithm.
- Apply the CRF algorithm for more than 4 iterations to see if increased inferencing produces smaller errors.
- Compare results against the National Weather Service's Quantitative Precipitation Estimation system, which incorporates Doppler, satellite, rain gauge, and other data for its estimates.
- Use a network of a size between the simple and regional models – for example, a network just large enough to cover neighboring nodes – in order to see if the ANN can be used as a sort of regional smoothing algorithm that could be applied more readily at a national scale.

APPENDIX A

BACKPROPAGATION

One algorithm for learning the weights $W_{i,j}$ in the neural network is backpropagation. The main method of updating values is just gradient descent, but because of the complicated dependence on each node, the algorithm is provided. I'll follow conventions from [12]. Assume we have input $\chi \in \mathbb{R}^n$ and expected (training) output $Y \in \mathbb{R}^m$, with total number of nodes N. So, as before, for $i \leq n$, $\chi_i = x_i$ and for $i \geq N - m$, $\hat{Y}_{i+m-N} = x_i$ (the observed output vector $\hat{Y}$ is represented by the last nodes in the network).

Given a set of training data, the algorithm then iteratively calculates the coefficients in the network, minimizing the square of the error $E = \sum_{i=1}^{m}(Y_i - \hat{Y}_i)^2$, where $\hat{Y}$ is the current output from the neural network for input χ. We'll also use the notion of an "ordered derivative" with z_i defined as in Equation 3.1:

$$D(E, z_i) = \frac{\partial E}{\partial z_i} + \sum_{j>i} \frac{\partial E}{\partial z_j} \cdot \frac{\partial z_j}{\partial z_i} \tag{A.1}$$

which encapsulates the notion of both the explicit and implicit dependence of the error on a particular variable.

To summarize Werbos' analysis, we obtain the following:

$$D(E, \hat{Y}_i) = \begin{cases} \hat{Y}_i - Y_i & 0 < i \leq n \\ 0 & \text{otherwise} \end{cases} \tag{A.2}$$

$$D(E, W_{i,j}) = D(E, z_i) \cdot D(E, X_j) \tag{A.3}$$

$$D(E, z_i) = \left.\frac{ds(z)}{dz}\right|_{z_i} \cdot D(E, X_i),\ i = N, N-1, ..., n+1 \tag{A.4}$$

$$D(E, X_i) = D(E, \hat{Y}_{i-(N-n)}) + \sum_{j=i+1}^{N} W_{j,i} \cdot D(E, z_j),\ i = N, N-1, ...n+1 \tag{A.5}$$

From this, we are able to calculate these many quantities in order to update the coefficients:

$$W'_{i,j} = W_{i,j} - r \cdot D(E, W_{i,j}) \tag{A.6}$$

where r is "some small constant" for iterating this procedure (for example, $r = 0.3$). Generally, if $0 < r < 1$, this is sufficient for the algorithm to converge to a locally optimal solution.

We then redefine $W_{i,j} = W'_{i,j}$, and continue this process for another pair of input and expected output vectors. This is done until some notion of convergence is achieved. In my implementation, this is a fixed number of iterations through the entire training set.

APPENDIX B

BENJAMINI-HOCHBERG PROCEDURE

The Benjamini-Hochberg Procedure [10] is a method to control the false discovery rate in multiple hypothesis testing. If data are taken from a simulation assuming the null hypothesis, we should expect the p-values of many such hypotheses to be distributed uniformly on $U(0,1)$. For example, if all of the hypotheses involve a parameter estimated from a normal distribution $N(\mu,\sigma)$, we should expect roughly 5% of the z-scores to be greater in absolute value than 2. Rather than assuming all such z-scores are significant, we can instead consider a method to identify which are truly significant, rather just due to random variation. This minimizes the false discovery rate (number of true hypotheses marked as rejected over total number of hypotheses rejected).

The Benjamini-Hochberg procedure does so by taking the hypotheses H_i and sorting them by p-value, creating a new list $H_i^{(s)}$ with associated p-values $p_i^{(s)}$ where for $i < j$, $p_i^{(s)} \leq p_j^{(s)}$. The idea here is that these p-values should follow a uniform distribution, so their CDF should follow the line $\hat{p}_i^{(s)} = \frac{i}{N}$. Now, instead of comparing the p-values naively to α as $p_i \overset{?}{<} \alpha$, we can instead use the estimated p-values as a basis for comparison.

In particular, for a given value q^* (which can be set to α), let k be the maximum value for which $p_k^{(s)} \leq \frac{k \cdot q^*}{N}$. In other words, find the largest sorted p-value that is significantly less than the estimated value (by a factor of q^*). This indicates the p-values that are abnormally small. For all $i \leq k$, we reject $H_i^{(s)}$. So, the hypotheses rejected under this procedure obey a much harsher constraint for rejection, which helps control the FDR.

APPENDIX C

CODE DESCRIPTION

All code for this project is available on GitHub at github.com/eaott/weather-prediction. It is largely written in Java, with a few Python and *Mathematica* scripts used for data processing and analysis. The relevant package structure is as follows:

weather.data Contains some files for processing intermediate data, and constants associated with the data set (for example, the boundaries of the region under investigation).

weather.network Code to produce an artificial neural network (ANN) as described in Chapter 3, specifically **SimpleNetwork.java**. **NetworkTest.java** contains the test code described in Chapter 4.

weather.process Major utility functions for relevant processing when running experiments. In particular, this includes code for producing Voronoi diagrams (see Section 2.4), and for performing Loopy Belief Propagation on neural network results (see Section 5.1).

weather.scripts Scripts contains the runnable sections of code to create experiments. This includes processing of the rain gauge data and the main experiments used to create the results for this project.

weather.util Utility functions for serialization, error calculation, and more, along with interfaces for some customizable aspects of the ANN system.

REFERENCES

[1] W. National Weather Service Office of Climate and W. Services, *74-year list of severe weather fatalities*, 2014.

[2] C. of New Braunfels, *Floodplain management*.

[3] N. W. S. Weather Prediction Center, *Quantitative precipitation forecasts (QPFs)*.

[4] *Weather by the Numbers* (MIT Press, 2012).

[5] N. Q. Hung, M. S. Babel, S. Weesakul, and N. Tripathi, *An artificial neural network model for rainfall forecasting in Bangkok, Thailand*, Hydrology and Earth System Sciences **13**, 1413 (2009).

[6] M. N. French, W. F. Krajewski, and R. R. Cuykendall, *Rainfall forecasting in space and time using a neural network*, Journal of Hydrology **137**, 1 (1992).

[7] E. Toth, A. Brath, and A. Montanari, *Comparison of short-term rainfall prediction models for real-time flood forecasting*, Journal of Hydrology **239**, 132 (2000).

[8] K. Koizumi, *An objective method to modify numerical model forecasts with newly given weather data using an artificial neural network*, Weather and forecasting **14**, 109 (1999).

[9] N. National Weather Service, *JetStream - online school for weather*, http://forecast.weather.gov/jetstream/doppler/ridgeimages.htm.

[10] Y. Benjamini and Y. Hochberg, *Controlling the false discovery rate: a practical and powerful approach to multiple testing*, Journal of the Royal Statistical Society. Series B (Methodological) **57**, 289 (1995).

[11] K. P. Murphy, Y. Weiss, and M. I. Jordan, *Loopy belief propagation for approximate inference: An empirical study*, in *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pp. 467–475, Morgan Kaufmann Publishers Inc., 1999.

[12] P. J. Werbos, *Backpropagation through time: what it does and how to do it*, Proceedings of the IEEE **78**, 1550 (1990).