Spring 4-24-2019

Predicting Parameters for Bertini Using Neural Networks

Makenzie Clower

Predicting parameters for bertini using neural networks

Makenzie L. Clower

submitted in partial fulfillment of the requirements for Honors in Mathematics at the University of Mary Washington

Fredericksburg, Virginia

April 2019

This thesis by **Makenzie L. Clower** is accepted in its present form as satisfying the thesis requirement for Honors in Mathematics.

Date	Approved
_____	_____
	James Collins, Ph.D. thesis advisor
_____	_____
	Julius Esunge, Ph.D. committee member
_____	_____
	Suzanne Sumner, Ph.D. committee member

Contents

Abstract

The purpose of this research is to use machine learning algorithms to predict the fastest settings for a program called Bertini. Bertini is a computer program that approximates solutions to systems of polynomial equations. Settings for this program can be changed by the user, but the user may not necessarily know the best settings to use to optimize the run time for a particular system of polynomial equations. The settings that were focused on were the differential equation predictor methods when tracking the homotopy to the solution of the system. A neural network was used on a training set of data to create a model and then a test set was run through to obtain a percent accuracy for this model. Increased accuracy for the model was obtained by changing hyperparameters of the neural network. Neural networks with training sets of 3,000 and 8,000 polynomials were used and results were found for different parameter settings.

1 Introduction

A polynomial system of equations is a set of polynomials, $f_1, ..., f_n$, where each f_i can depend on one or several variables, each in varying degrees. To solve the system, each function f_i is set equal to zero and solved for each variable that the system contains. One program that can approximate solutions to these polynomial systems of equations is Bertini. This research focused on using and improving the Bertini program.

1.1 Bertini Classic and 2.0

Bertini is a computer program that approximates solutions to systems of polynomial equations. Bertini Classic was written by Daniel J. Bates, Jonathan D. Hauenstein, Andrew J. Sommese, and Charles W. Wampler.

Bertini 2.0, which was used in this research, was written by Danielle Brake. This program was written to be more efficient and modularizable than Bertini Classic. Bertini 2.0 can evaluate multivariate systems and efficiently find an approximate solution depending on the size of the system. How the solution approximation is calculated changes with the settings of the program. These settings can be changed by the user but it may not be apparent to the user how these settings should be changed to best optimize the run time of the program. This research focused on using machine learning to predict the best settings for different polynomial systems that users may input into Bertini 2.0.

1.2 Real World Applications

Systems of polynomial equations can be very useful for solving problems outside of the mathematics field. These applications include problems in Global Positioning Systems (GPS), robotics, and chemical reaction networks. Examples of these applications will be explained further throughout the next sections.

1.2.1 GPS

GPS location services use solutions to systems of polynomial equations to find locations of target objects. Usually four satellites near the target object are used, with a coordinate point (A_i, B_i, C_i) known for each satellite. These coordinates are then substituted into the following system.

$$\begin{aligned}
(x - A_1)^2 + (y - B_1)^2 + (z - C_1)^2 - (c(t_1 - d))^2 &= 0 \\
(x - A_2)^2 + (y - B_2)^2 + (z - C_2)^2 - (c(t_2 - d))^2 &= 0 \\
(x - A_3)^2 + (y - B_3)^2 + (z - C_3)^2 - (c(t_3 - d))^2 &= 0 \\
(x - A_4)^2 + (y - B_4)^2 + (z - C_4)^2 - (c(t_4 - d))^2 &= 0
\end{aligned}$$

The constant d is used as a difference between the clock of the satellite and the object's time. Each variable t_i is the time variable for how long the signal takes to travel from the satellite to the object. The speed of light is used for the constant c. Solving this system for the coordinate variables (x, y, z) then gives the approximate placement of the object of interest. [1]

1.2.2 Robotics

In the area of robotics, the range of motion of robots can be found by solving systems of equations. Each point on the robot, such as the joints on an arm, have a range of motion that can be represented by a system of polynomial equations. The system comprised of all polynomial equations can then be solved to find a range of motion.

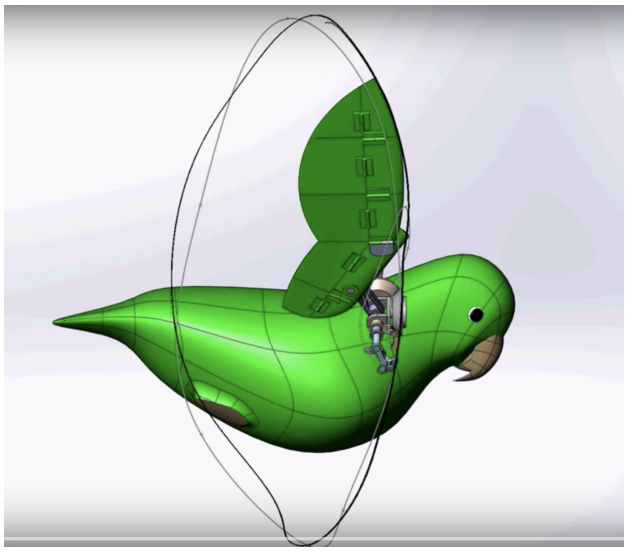

Figure 1: The figure above shows an animation of a robotic bird during flight. The range of motion outlined by the black curve is found by solving the system of polynomial equations that incorporates the range of motion at each joint of the wing. [4]

1.2.3 Chemistry

Solving systems of equations can also be useful in chemistry. When finding the equilibrium values for certain chemical reaction networks, systems of polynomial equations are helpful in solving for the concentrations of elements and compounds. For example, consider the simple chemical reaction network below.

$$\mathrm{H_2O} \rightleftharpoons 2\,\mathrm{H} + \mathrm{O}$$

From this equation, we can then set up the chemical reaction equation

$$kX_{H_2O} = X_H^2 X_O,$$

where X is the concentration of the element or compound and k is the equilibrium constant. It is also important to follow conservation equations so that the concentration of each individual element on both sides of the chemical reaction do not exceed the total concentration possible for each element. These equations are

$$\begin{aligned} 2X_{H_2O} + X_H &= T_H \\ X_{H_2O} + X_O &= T_O. \end{aligned}$$

Here, T is the total concentration of the element throughout the whole system. The total polynomial system comprised of each function above can then be rewritten as,

$$\begin{aligned} f_1 &= kX_{H_2O} - X_H^2 X_O \\ f_2 &= 2X_{H_2O} + X_H - T_H \\ f_3 &= X_{H_2O} + X_O - T_O. \end{aligned}$$

Solving this system for each of the X and T variables would then give us the equilibrium concentrations of each element and compound in this chemical reaction network. [5]

2 Mathematics Behind Bertini

Homotopy continuation is a method of approximating solutions to systems of polynomial equations. This method finds the solutions to $f(z) = 0$, by relating the $f(z)$ system to $g(z)$, which has known solutions for $g(z) = 0$. This z variable in each equation may represent more than one variable. For example, f and g may be a functions of x and y, such that $f(z) = f(x, y)$ and $g(z) = g(x, y)$. For a thorough description of the mathematics behind Bertini, reference *Numerically Solving Polynomial Systems with Bertini*, by Bates, Hauenstein, Sommese, and Wampler. [2]

2.1 Introducing Variables and Functions

The homotopy function used for approximating a solution is

$$H(z(t), t) = (1 - t)f(z) + t(g(z)).$$

The variable $t \in [0, 1]$ is used during tracking. Our "known" function is $g(z)$ and our function or system that we are solving is $f(z)$. The system that is chosen as the "known" system needs to have a number of solutions equal to the maximum number of possible solutions for the target system so that when tracking, no solutions to the target system are left unsolved. Thus, the degree of $f(z)$ is equal to the degree of $g(z)$. When solving a system with only one variable and one polynomial, the simplest known system is $g(z) = z^d - 1$, so that $g(z)$ has d solutions, which we can find very easily. This d is the maximum number of solutions that the system $f(z)$ can have.

2.2 Tracking

Tracking is the process through which the Bertini program approximates the solution to the system of polynomial equations.

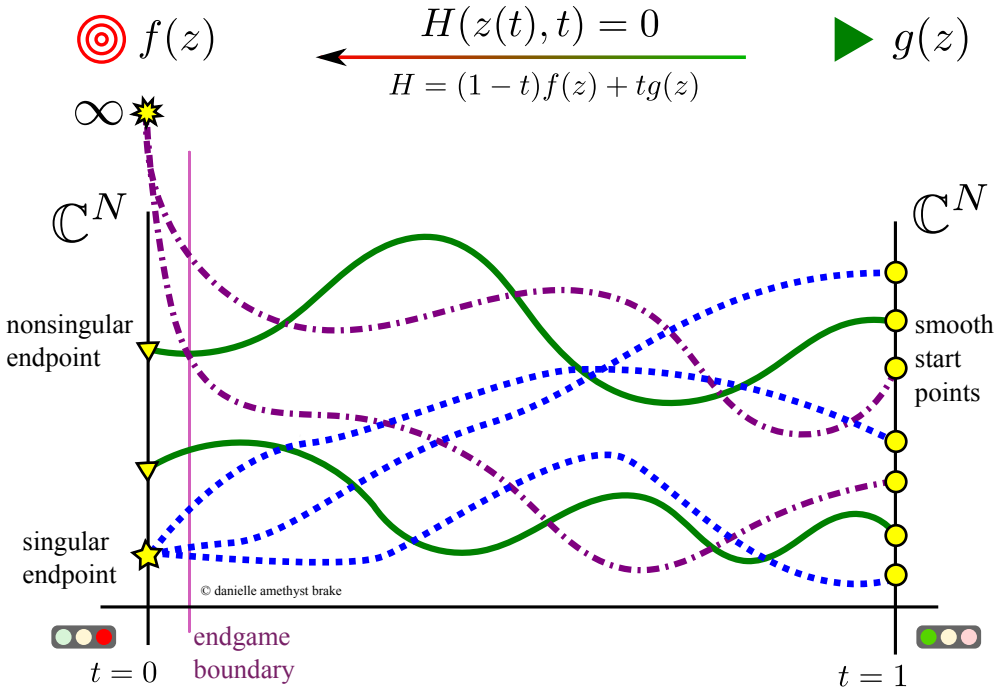

Figure 2: This picture, made by Danielle Brake, shows the homotopy continuation visually.[3]

The tracking variable t begins at 1 and ends at 0. When $t = 1$, the circle points along the right side of Figure 2 represent the solutions to the previously mentioned system $g(z)$. At this point, the homotopy function is equal to the known system as seen below.

$$\begin{aligned} H(z(1), 1) &= (1-1)f(z) + (1)g(z) \\ &= g(z) \end{aligned}$$

We will then follow along these paths seen above in Figure 2. These paths are $z(t)$ solved at the various values of t that are used in tracking. At the end of tracking, $t = 0$, we have

arrived at the triangle and star endpoints on the left side of Figure 2. These endpoints are the solutions to the system that is being solved. The homotopy function is now equal to the system that is being solved as seen below.

$$\begin{aligned} H(z(0),0) &= (1-0)f(z) + (0)g(z) \\ &= f(z) \end{aligned}$$

Tracking t from 1 to 0 requires many small steps to be taken. At each of these steps, it is necessary to solve $H(z,t) = 0$ for z, using each step's t value. We know that at $t = 1$, $H(z,1) = g(z)$, which is a system with known solutions. Also, when $t = 0$ at the end of tracking, $H(z,0) = f(z)$, which will give us the approximate solutions to our "unknown" system. Each of these steps consists of two stages. These stages are the predictor and corrector steps, each of which are methods of approximation to move along the paths correctly. How the predictor and corrector steps approximate the next value can be set by the user. This research focused on various predictor approximation settings available in Bertini 2.0.

As seen above in the homotopy continuation image, these tracking paths seem to cross, but in three dimensional space, they will never intersect. By using the complex plane to track these paths across, we can say that these paths will cross with probability zero. It is important that the paths never cross, because if they did, the program would fail. Paths crossing could result in one path incorrectly following a second path along its tracking. If this occurs and the second path does not converge, switching paths could result in no outputted solutions. Another outcome could be that the first path would converge to incorrect solutions, thus omitting the first path's solutions.

2.3 Predictor Step

The predictor step, as mentioned above, is the first step in tracking the homotopy variable t along the path from $t = 1$ to $t = 0$. This predictor step begins by finding the line tangent to the curve at $t = 1$ when using Euler's predictor method. For other predictor methods used, a similar equation to the tangent line is used for the approximation. To find this equation, we will need to solve $H(z,t) = 0$ at the various step values of t that are taken across the tracking path. We will do this process by taking the homotopy function, differentiating it with respect to t, and solving the differential equation as explained below. A visualization of this prediction step process can be seen below.

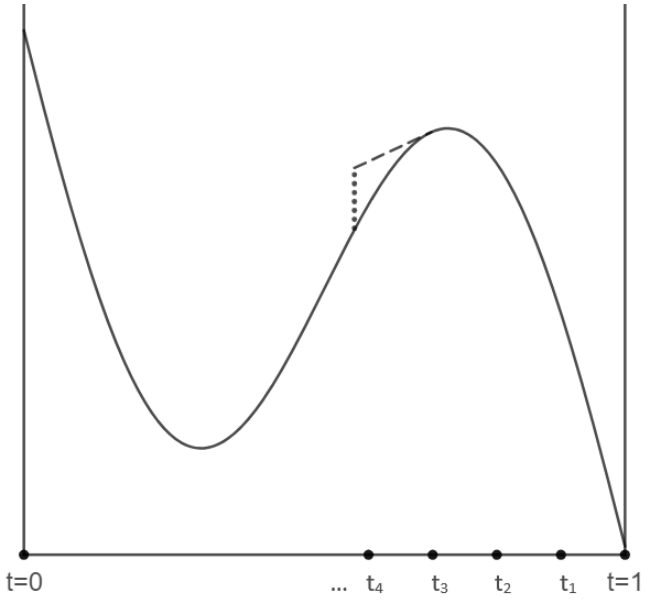

Figure 3: In this graph, the straight broken line is the predictor step and dotted line is the corrector step in tracking from t_3 to t_4. Many of these small steps that include a predictor and corrector method are used when tracking the path along from $t = 1$ to $t = 0$.

The equation below can now be used to solve for $z(t)$ at the t_i step value, where t_i is the next value of t that is being "stepped" towards.

$$\frac{d}{dt}(H(z(t),t)) = \frac{\partial H}{\partial z}\frac{dz(t)}{dt} + \frac{\partial H}{\partial t} = 0$$

We then separate to obtain $\frac{dz(t)}{dt}$ on one side of the equation so that an ordinary differential equation will be left to solve.

$$\frac{dz(t)}{dt} = -(\frac{\partial H}{\partial z})^{-1}\frac{\partial H}{\partial t}, \text{ where } \frac{\partial H}{\partial z} \neq 0$$

Now, all that is left is to solve the above ordinary differential equation, which is called the Davidenko equation. An initial condition of $z(1) = y_0$ can then be used to solve for the final solution to the prediction step. Here, y_0 is a solution of $g(z)$. Once this is calculated, the corrector step needs to be taken to continue moving along the path to obtain accurate solutions.

2.3.1 Predictor Example

For the following example, we will use an example differential equation $\frac{dz(t)}{dt} = z^2t$. Suppose we begin at $t = 1$ and use a step size of $\Delta t = 0.05$. Let our starting point be $z(1) = 0.4$.

Euler's method is a method that can be use to approximate the solution to this ordinary differential equation. Using this method, we will obtain the following equation

$$z_1 = z_0 + (\Delta t) z_0^2 t$$

where z_1 is the approximate solution at the point $t = 0.95$. Then we can solve for the z_1 equation above to complete the prediction step.

$$\begin{aligned} z_1 &= 1 + (-0.05)(-0.16) \\ &= 1.008 \end{aligned}$$

Thus, $z_1 = 1.008$ approximates $z(0.95)$. The next step in the process is to correct this approximation of z_1 using Newton's method which we consider in Section 2.4.

2.3.2 Predictor Parameters

The possible predictor approximators that can be used in Bertini are Constant, Euler, Heun, HeunEuler, RK4, RKCashKarp, RKCashKarp45, RKDormandPrince56, RKF45, RKNorsett34, and RKVerner67. Each of these predictor approximators are well known numerical methods for approximating solutions to differential equations. The default setting for the predictor step is RKF45, which is a fourth order predictor method. Not all predictor parameters were evaluated in this research. Only Euler, RKCashKarp, RK4, and RKF45 were considered. First order predictors, such as Euler, may be faster at approximating through each step in tracking, but these predictors are less accurate, and hence many steps need to be taken to approximate with high accuracy. Higher order predictor methods such as RKF45 and RKCK are more accurate at each step and can take larger steps, but each step speed may be slower. Trying to balance between speed per step and number of steps taken is necessary in optimizing the time taken to run the program for each polynomial. Some polynomials may run faster with lower order predictors while others may run faster using higher orders.

The time it takes a single polynomial system to run through the program may be minuscule, but when running multiple systems of polynomials through the program, the time accumulates and thus picking the best predictor method for each system is important.

2.4 Corrector Step

The next component in the stepping process is the corrector step. During this step, Newton's method, which is an iterative method, is used to make a better approximation of the function $H(z, t_i) = 0$, using the approximation obtained from the predictor as the initial iterate. A good initial iterate is necessary for Newton's method to work efficiently.

Below is the formula for Newton's method during the corrector steps.

$$z_{n+1} = z_n - \frac{H(z_n, t^*)}{\frac{\partial H}{\partial z}\big|_{z=z_n}}$$

We also will make the assumption that $\frac{\partial H}{\partial z}|_{z=z_n} \neq 0$ because the curves in the tracking paths are assumed not to cross. Newton's method as explained above only uses about 2-3 iterations in each step of tracking. It is a very accurate method of approximation, but the method does require a good initial iterate.

2.4.1 Corrector Example

For this example, let us start with the system $f(z) = 5 - z^2 - 2z^4$. If we begin with an initial iterate of $z_0 = 1$, we follow the steps below for Newton's method.
The general equation used for Newton's method is

$$z_{n+1} = z_n - \frac{f(z_n)}{f'(z_n)}$$

First, we see that $f(1) = 2$ and $f'(1) = -10$. Then the first iteration will be

$$\begin{aligned} z_1 &= 1 - \frac{2}{-10} \\ &= 1.2 \end{aligned}$$

Using $z_1 = 1.2$, we can calculate that $f(1.2) = -0.5872$ and $f'(1.2) = -16.2240$.
Then the second iteration is

$$\begin{aligned} z_2 &= 1.2 - \frac{-0.5872}{-16.2240} \\ &= 1.1638 \end{aligned}$$

Next, we see that $f(1.1638) = -0.0235$ and $f'(1.1638) = -14.9382$. The third iteration is then

$$\begin{aligned} z_3 &= 1.1638 - \frac{-0.0235}{-14.9382} \\ &= 1.1622 \end{aligned}$$

For the fourth iteration, $f(1.1622) = 0.0004624.$ and $f'(1.1622) = -14.8828$.
This result gives us

$$\begin{aligned} z_4 &= 1.1622 - \frac{0.0004624}{-14.8828} \\ &= 1.1622 \end{aligned}$$

We have now reached an iterate that is fairly close to the previous iterate. Once this matching occurs, we can assume that the z values have converged and we move on to the next step.

2.4.2 Corrector Parameters

The corrector steps are used after the predictor step to correct the previously approximated predictor step and continue along the path. Visually we can see this process in Figure 3.

Settings that affect the corrector step are explained further in the next section. These settings were not studied during this research, but it is a very important step for the Bertini program to run.

2.5 Predictor-Corrector Failure

In the process of tracking, "checkpoints" are in place to make sure that we are following the path at a high accuracy of approximation. One way that the Bertini program checks this accuracy is through the tracking tolerance settings. Tolerance overall is how far each corrector iteration of predictor-corrector approximations are from the previous corrector iteration. This tolerance will tell us when the Newton iterations have converged. So that we can move through with accurate values, the tracking tolerance can be changed to allow the user to determine how accurate is "accurate enough" to move along the path. If each step approximation does not fall within the tolerance after the maximum number of iterations, then the step size (Δt) is cut in half. If several consecutive steps are successful and fall within the tolerance, then the step size will double.

Step size is another parameter that can be changed in Bertini to allow the user to decide the maximum and minimum step size. Setting a small maximum size would make the program run slowly, because it has to take many small steps. On the other hand, if the minimum size is too large, then the tolerance for approximations may not be met quickly, if at all, and thus the program will continuously need to re-approximate each step value.

Another important setting that can be changed in the Bertini program is the maximum number of Newton iterations that can be taken. This number allows the user to set the maximum iterations that the corrector step can take in an attempt to get the approximations to converge. These settings in Bertini are very important in running the program, but they also were not evaluated in this research.

2.6 Endgame Boundary

As we approach the end of tracking each path, a new process must occur, especially when the system of polynomials has solutions of mutliplicity greater than one. This new process happens in the endgame boundary during the final steps of tracking as seen in Figure 2. The default time when this endgame boundary begins in tracking is when $t = 0.1$. As mentioned previously, it is necessary that the multiple paths being tracked do not cross or the program will fail. If a singularity occurs in the final solution of $f(z) = 0$, multiple paths may converge to a single point at $t = 0$. One way that we can approximate the solutions to these paths is through Cauchy's integral formula. This formula,

$$f(0) = \oint f(s)ds,$$

is used to to approximate the solution at $t = 0$, using a neighborhood in the complex plane around t with radius ϵ. We need $\epsilon > 0$, but small enough that the approximation at this point is very close to the actual value at $t = 0$. By doing this method we can complete the tracking paths with an accurate approximation to the system of polynomial equations.

Multiple settings can be changed by the user for the endgame. These settings include FinalTol, EndgameNum, EndgameBdry, and NbhdRadius. The FinalTol is the tolerance used by the endgame method of approximation to determine whether the approximations are accurate. The EndgameNum setting determines which Endgame method is used. For this setting there are three options, one being a fractional power-series endgame, and two using Cauchy's integral formula. The EndgameBdry sets the t value for which the program switches from predictor-corrector steps to this new endgame method of approximation. The default value for this setting is $t = 0.1$. Lastly, the NbhdRadius setting determines the radius value (ϵ) of the neighborhood used in Cauchy's integral formula.

3 Machine Learning

Machine learning uses algorithms to find patterns in data sets. These patterns can then be used to predict outputs on new sets of data. The data set that is used to learn is called a training set. The machine learning algorithm can then test how well it is able to predict outputs by using a validation set and a test set. These sets of data contain input data and labels. For a simple example, consider students' tests scores as results of hours of sleep and hours of studying. The number of hours of sleep and studying would be the input data that are used to predict the label, which in this case would be the test scores.

Validation sets are used before test sets are implemented to make sure the learning algorithm does not over fit the original training data. Overfitting is a term used when the machine learning algorithm fits its prediction method too closely around the training data, such that when given new data, it cannot predict well. What predictions the machine learning algorithm uses are determined by the data. For example, in this research, classification prediction is used. This method means that the predictions of the neural network must fit into a certain number of discrete classes. The main machine learning algorithm used in this research is a neural network.

3.1 Neural Networks

A neural network is the specific machine learning algorithm that was used in this research. Neural networks learn from data patterns to predict outputs on new data. The general structure of a neural network is seen in Figure 4.

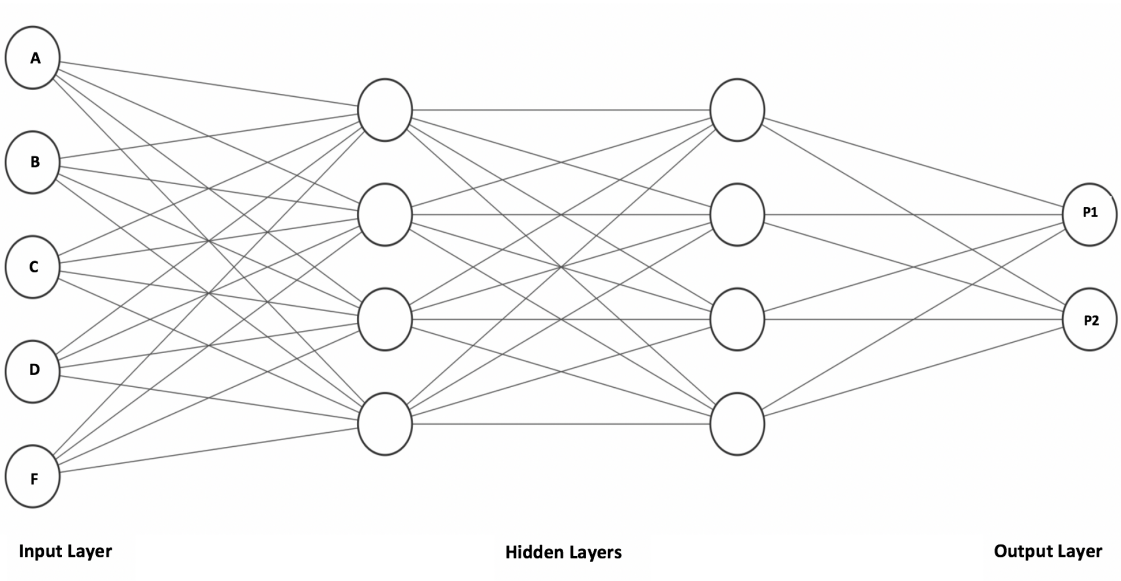

Figure 4: This diagram is the general structure of a neural network with five inputs and two outputs.

The inputs into this network structure are the coefficients of the system of polynomial equations. These numbers then propagate through the network, and at each node the number is inputted into the equation of the node, which will then return an output for that node. Each node output then becomes an input into the following layer of nodes. The number of layers in the structure and the number of nodes in each layer can be changed to help the accuracy of the model. In the neural network, hyperparameters can be changed other than the network structure to help maximize the accuracy of the model. These parameters will be explained further in section 3.2.

For the initial neural networks that were used in this research, Google's Tensor Flow was used to create the structure of the network. Once the network was built, we then switched to a more user friendly software called keras. The keras code structure gives the same neural network as before, but allows for changes in the structure to be made much faster.

3.2 Minimizing Error

The learning rate and number of training steps for the model are hyperparameters that can be changed in the neural network to effectively minimize the error in the outputs of the neural network. The weight values in a neural network are also important in minimizing error, but these are not controlled by the user. We are trying to maximize the number of correctly classified polynomial systems by changing these settings.

Weights in the neural network are values associated with each node that change the output value of the node. Weight values differ across all nodes and they determine how important that input value is to the prediction output at the end of the process. Once these weights are determined by the neural network and a prediction is given, the network weights can change using gradient descent by taking the gradient of the error function with respect to the weights to minimize the error of prediction in accordance with the weight

values throughout the network.

To begin with a simple example of minimizing error, we can relate this concept to linear regression. In linear regression, points (x, y) on a plane are given and the goal is to find the equation of the line, $y = mx + b$, so that the distances from each point to the line are the smallest.

In a neural network, the objective is similar but distance calculation will not be used to find the error. Instead, the mean squared error formula will be used and is shown below.

$$Error = \frac{1}{m}\sum_{i=1}^{m}(h(x^{(i)}) - y^{(i)})^2$$

Here, $h(x^{(i)})$ is the predicted output given by the neural network and $y^{(i)}$ is the actual label for the given polynomial. We want to minimize this function so that the accuracy of the neural network model is maximized.

To minimize the error function in the neural network, the gradient descent method is used. Gradient descent is a process that finds the minimum of a function through an iterative optimization algorithm. The steps/iterations that are taken are proportional to the negative gradient of the function at the current point. In the neural network, hyperparameters such as the learning rate and training steps affect how this gradient descent method minimizes the function. The learning rate decides the size of the steps. If the step size is too large, the gradient descent function cannot minimize because the large steps fluctuate around the function and may never reach the minimum of the error. On the other hand, if the learning rate is too small, then the steps taken in an attempt to minimize will result in going towards the minimum but never actually reaching the minimum or taking too long to do so.

Another critical setting in the neural network for the gradient-descent function is the number of training steps. This value is the number of iterations/steps taken in the minimization process. Once again, if the number of training steps is too large, it may take too long to run the algorithm, and if the number of training steps is too small, then we may begin to minimize without actually reaching the minimum error.

3.3 Generating Data

Once we have constructed a neural network, the next step is to generate data sets that the network can use to learn from and test its accuracy on. The data sets previously mentioned were comprised of single polynomial systems. These polynomials were generated with the following structure:

$$(x + a)^b(x + c)^d,$$

where $a, b, c, d \in \mathbb{Z}$ and $-100 < a, b, c, d < 100$. At first,we started with only integer roots to make sure that the neural network could predict solutions with a generally high accuracy.

The coefficients of the randomly generated polynomial systems were then inputted into a data file, which is a matrix of the coefficients.

	A	B	C	D	F	Label
System 1	1	54	-5987	-181332	11276164	0
System 2	1	24	-752	-10752	200704	1
System 3	0	1	157	6720	87300	0
System 4	0	1	-12	-7155	264046	0
System 5	1	-24	198	-648	729	0
System 6	0	1	-251	20935	-580413	1
System 7	1	100	3748	62400	389376	1
System 8	0	1	-165	8307	-132327	0
System 9	1	280	29398	1371720	24000201	1
System 10	0	1	251	20935	580413	0

Figure 5: The data chart for a single variable system would look similar to this matrix. Each row is the coefficients for a single variable polynomial that was generated. The column on the right contains the labels that each polynomial is given to correspond to the fastest setting that system will run under in Bertini.

3.3.1 Training Data

The training data of the single variable polynomials was made by running each polynomial through the program twice. Each polynomial ran once under the RKF45 (default) setting and once under the Euler setting. The run time under each of these settings was recorded. After each polynomial runs through Bertini twice, the timing of each run is compared to label the polynomial as to which setting is the optimal run time setting for the polynomial. If the system ran through Bertini fastest under the RKF45 setting, it was labeled with a 1, and if the system ran through fastest under the Euler setting, it was labeled with a 0.

3.3.2 Test Data

Test data was constructed in the same form as the training data. These data sets are polynomial coefficients with labels that give the best predictor parameter. The test set is used to test the accuracy of predictions that the machine learning algorithm gives. The neural network is given the polynomial coefficients without the labels and is then expected to predict the fastest setting for this polynomial system to run through Bertini. Once the neural network predicts the best setting for each polynomial system in the data set, the predicted parameter outcomes are compared with the actual best output for each polynomial system. The number of correct predictions is turned into a percentage out of the total number of polynomial systems to give an accuracy for this model.

3.4 Probability Outputs

The outputs of the neural network structure as seen in Figure 4 are probabilities that each predictor parameter will be the fastest setting for the given system. In the single polynomial system data the following are the outputs.

```
P1=Probability that the system will run fastest under the RKF45 setting
```

```
P2=Probability that the system will run fastest under the Euler setting
```

The outputs P1 and P2 (shown in Figure 4) will be probabilities between 0 and 1. The neural network can then decide the highest probability of these three options and choose that setting as the optimal parameter for Bertini to run this polynomial system.

4 Results

4.1 Neural Network Parameter Settings

Below are the parameter settings used for the neural network. Each setting was changed to obtain higher accuracy on the neural network model. The most effective hidden layer structure that was used was two hidden layers with 100 nodes in each layer, so this setting was used in each of the following neural network structures.

	Training Steps	Learning Rate
Parameters_1	100,000	0.08
Parameters_2	50,000	0.05
Parameters_3	50,000	0.08
Parameters_4	100,000	0.05

4.2 Single Variable Systems

For the single variable systems, the Google Tensor flow package for neural networks was used to construct the code. Below are the accuracies of the neural network models when trained on a set of 3,000 polynomials and tested on a set of 2,300 polynomials.

	Parameters_1	Parameters_2	Parameters_3	Parameters_4
RKF45	83.37%	81.27%	43.86%	88.10%
Euler	62.06%	55.92%	80.11%	50.42%
Overall Accuracy	73.45%	69.28%	61.01%	69.95%

Below are the accuracies of a network model that trained with a set of 8,000 polynomials and tested with a set of 3,000 polynomials.

	Parameters_1	Parameters_2	Parameters_3	Parameters_4
RKF45	89.65%	78.25%	83.60%	81.23%
Euler	42.14%	62.81%	57.43%	59.93%
Overall Accuracy	65.71%	69.97%	70.41%	70.50%

The overall accuracy of the models did not seem to change with the varying structures of the neural network or the size of the training and test sets given. Prediction accuracy's above over 50%, give us a neural network that helps Bertini run at faster than default settings.

4.3 Future Work

New labels for systems of polynomial equations can be created using probabilities instead of the original classification of 0 or 1. This idea originated because the timing of the Bertini program can be somewhat random, and thus a probability that each system will run fastest under each setting should be the label instead of a single best setting for each system.

The new data for polynomial systems of equations can be obtained in the following way. Each system can be run through the Bertini program thirty times. A loop will be used to split up the runs into ten loops. Each one of the ten loops consists of the polynomial system running through the program under three predictor settings. In each of the ten loops run, the fastest setting is recorded. After the 10 loops are finished, the best settings can then be recorded as probabilities. The label below is an example of a label for one system of polynomial equations.

Probability of RKF45	Probability of RKCK	Probability of RK4
0.5	0.4	0.1

For example, with the above labels, the RKF45 setting is the fastest setting for this system in 5 out of the 10 times that it was used. Similarly, the RKCK setting is the fastest for this system in 4 of the 10 times that it was used. Lastly, the RK4 setting is the fastest setting for this system in only 1 of the 10 times that it was used.

These new labels may be able to increase the accuracies of the neural networks predictions for the predictor parameter settings for each system of polynomial equations. We believe this increase in accuracy may occur because the outputs of the neural network are probabilities, similar to these new labels.

5 Conclusion

The Bertini program is a very useful way to approximate solutions to systems of polynomial equations. Understanding the settings of this program can be essential to its efficiency. This research was concentrated around using machine learning algorithms to improve the run-time of this program.

The specific setting that was focused on was the predictor setting. This predictor step is used in solving the homotopy continuation function along the tracking paths, which are essential to approximating the solution to the overall system of polynomial equations. By using a neural network, we can predict the optimal setting for each individual system that Bertini is given so that the fastest setting is used for each system.

In conclusion, we have made a neural network that can predict settings for single variable polynomial systems so that the program will run faster than it would under default settings. We have also considered future work that contains different labels in an attempt to obtain higher accuracies on original data as well as accuracies for higher variable systems.

References

[1] H. Arneja, A. Bender, S. Jugus, and T. Reid. Solving the GPS equations. URL `http://mason.gmu.edu/~treid5/Math447/GPSEquations/`.

[2] D. Bates, J. Hauenstein, A. Sommese, and C. Wampler. *Numerically Solving Polynomial Systems with Bertini*. Society for Industrial and Applied Mathematics, 2013.

[3] Danielle Brake. Homotopy continuation. Image.

[4] Ferdi Kahraman. Flapping wing mechanism. Youtube.com, November 2014. URL `https://www.youtube.com/watch?v=7aXmze9Ynis`.

[5] A. Sommese and C. Wampler. *The Numerical Solution of Systems of Polynomials Arising in Engineering and Science*. World Scientific Publishing, 2005.