Spring 4-26-2019

Improving Bertini 2.0: Classifying Singular Polynomials with Machine Learning

Riley Anderson

Improving Bertini 2.0: Classifying Singular Polynomials with Machine Learning

Riley J. Anderson

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

Fredericksburg, Virginia

April 2019

This thesis by **Riley J. Anderson** 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
_____	_____
	Jangwoon Lee, Ph.D. committee member

Contents

Abstract

The purpose of this research is to decrease the run time of Bertini, a program that approximates solutions of polynomial systems. Bertini can be run more efficiently if it is known whether a polynomial is singular or non-singular. In this research, we focus on polynomials in one variable. We use a machine learning algorithm to classify polynomials into these two categories. To do so, we create and use a set of polynomials to train a neural network and create a model. Then, we create and use a test set to assess the accuracy of the model. By changing the hyper-parameters of the system and by changing the functions used in the system, the accuracy of the model is able to be increased.

1 Introduction

Solving polynomial systems has applications in many areas including chemistry, robotics and GPS. System of polynomial equations can be used in chemistry to find points of chemical equilibrium in chemical reactions. These reactions occur when two or more substances, known as reactants, combine to form a new substance. Equilibrium is reached when the concentration of reactants and products are constant. For example, in a sealed carbonated drink, carbon dioxide is both in the liquid and a gas between the cap and liquid. It is constantly changing between liquid and a gas and vise versa, however it does not appear to be changing as the system is at equilibrium. A case study in *The Numerical Solution of Systems of Polynomials Arising in Engineering and Science* asserts"While the transient behavior of the reaction is governed by differential equations, the final equilibrium conditions are well-modeled by a system of polynomial equations" [9]. These systems often have a real solution in which the concentration of reactants and products are positive, meaning they are applicable to real-word scenarios.

Numerical solutions to a system of polynomial equations can also represent the range of motion of a machine. This fact is especially true with robotic machines referred to as linkage systems. Imagine there are two bars: one bar is connected to a table with a movable joint, and on the other end, a second bar is connected again with a movable joint. The end of the second bar that is not attached to anything is often referred to as the hand. A three bar linkage system can be seen in Figure 1.

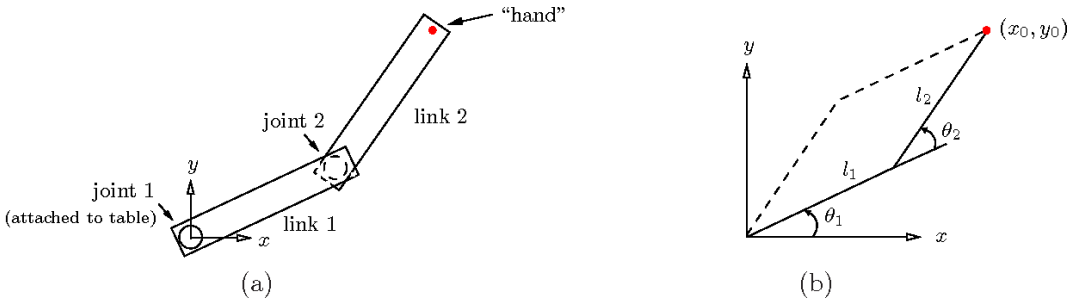


Figure 1: Three bar linkage system [8]

If the hand needs to be in a specific location, a polynomial system can be used to find all possible angles of the two joints that puts the hand at the specific point. The opposite problem can also be solved. More specifically, if the angles of the two joints are known, a system can be

solved to find the location of the hand. This process can also be applied for more complex system of bars and linkages.

GPS uses a system of polynomial equations to find locations of receivers, such as those used in cars and smartphones. Currently, the United States Global Positioning System has 29 satellites but only three or four are used to locate a receiver at any one time. Using the coordinates of the satellites and the difference in time of the receiver and satellites clocks, a system of equations can be formed [1]. If four satellites are used, four equations can be formed and solved for the x, y and z coordinates of the receiver on earth and d, the difference in time of the receiver and satellites clocks."The difference in time is important because the location is approximated using the travel time of a signal from the satellite to the receiver, and this happens in much less than a second. The difference in time can cause errors because the satellites' clocks are accurate to 10^{-8} of a second while the receiver's clock is much less accurate." If three satellites are used, everything but z, the altitude, can be solved using a system of three equations.

One way of calculating these numerical solutions of polynomial systems is with a program called Bertini. It is our goal to reduce the run time of this program. We do this by utilizing and adapting machine learning algorithms to classify polynomials as singular and non-singular. To accomplish this, coefficients of a polynomial will be the inputs of a neural network. Then, to increase the accuracy of the system we will adjust the parameters associated the network as well as adjust the network itself to in an attempt to approximate the discriminant. Based on the classification of the polynomial, we are able to bypass sections of the program and run it in less time.

2 Polynomials

2.1 Bertini

Bertini Classic is an open-source program that approximates the the numerical solution of systems of polynomial equations. It was originally authored by Daniel J. Bates, Jonathan D. Hauenstein, Andrew J. Sommese, and Charles W. Wampler. Bertini 2.0, a redevelopment of Bertini Classic, was created by Danielle Brake and is what was used for this project. The program works in two stages.

First, homotopy continuation is used. Bertini uses the homotopy function,

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

where $f(z)$ is the polynomial we would like to find the solutions of, $g(z)$ is an equation that has known solutions and the number of solutions is the same as the maximum number of solutions of $f(z)$, and $t \in [0,1]$. In our case, $g(z) = z^d - 1$, where d is the degree of $f(z)$. Bertini begins with $t = 1$, which corresponds to $g(z)$. By decreasing t and solving $h(z(t),t)$ the solutions of $g(z)$ can be tracked to solutions to $f(z)$, as seen in the image below. The solutions of $z(t)$ are known as paths. In order to solve $h(z(t),t)$ as the value of t decreases, both Newton's method and a predictor step are used. Since Newton's methods requires a Jacobian matrix it is important that these paths do not cross as it would make this matrix is singular and Newton's method unusable. To avoid this, these calculations take place in $\mathbb{C}^N$, where the crossing of paths will take place with probability 0 [2].

A problem arises when $f(z)$ has a repeated root. When this occurs the Jacobian matrix, used in Newton's method is no longer invertible . Thus a second stage, the endgame, is used as the end of the program as t approaches zero. During this stage other methods, such as the Cauchy integral formula, are used to find solutions.

Since we do not know before we run the program whether $f(z)$ will be singular or non-singular, the endgame is always used. This section of the program can be expensive to run. Therefore, if we can identify $f(x)$ as non-singular, then we can remove the endgame boundary and run the program more efficiently.

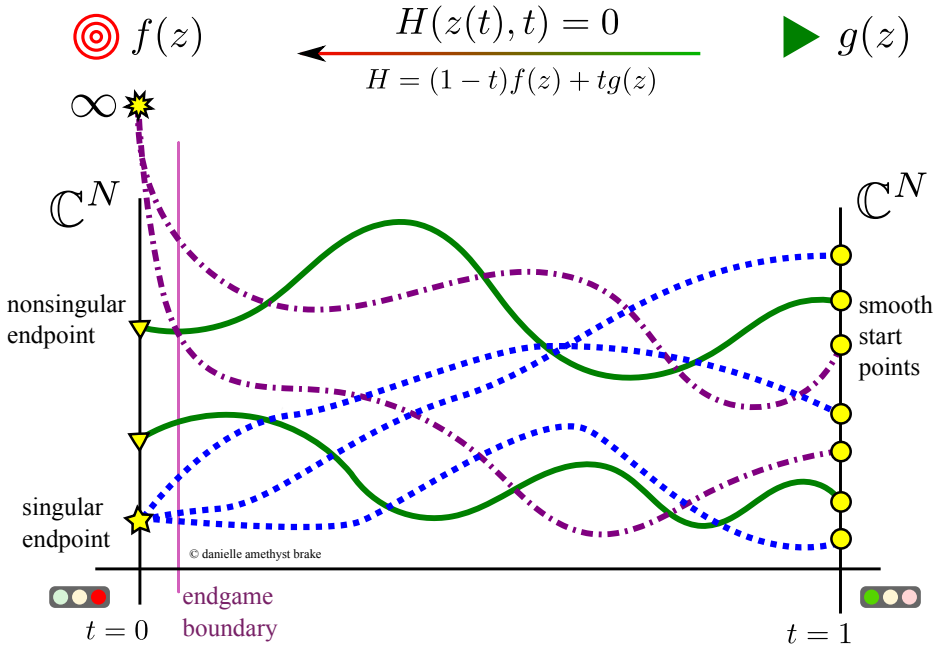


Figure 2: Homotopy Continuation Courtesy of of Danielle Brake [3]

2.2 Singular and Non-Singular Polynomials

Definition 2.1 (Multiplicity)**.** Let $p(x)$ and $s(x)$ be polynomials such that $s(a) \neq 0$ and $p(x) = (x-a)^k s(x)$. Then we say that a is a root of $p(x)$ with **multiplicity** k.

Definition 2.2 (Singular Polynomials)**. Singular polynomials** have at least one root that appears with multiplicity two or greater.

Example 2.3. The polynomial $x^3 - 2x^2 - 4x + 8 = (x+2)(x-2)(x-2)$ is singular.

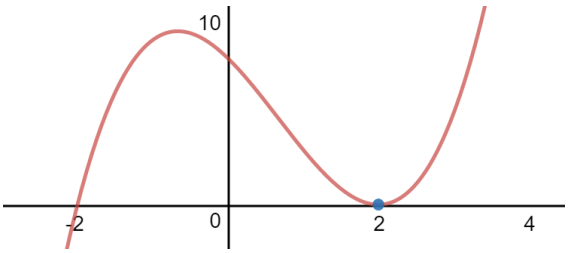


Figure 3: Singular Polynomial

Definition 2.4 (Non-Singular Polynomials)**. Non-Singular polynomials** have all roots that appear with multiplicity one.

Example 2.5. The polynomial $x^3 + 3x^2 - 4x - 12 = (x+2)(x-2)(x+3)$ is non-singular.

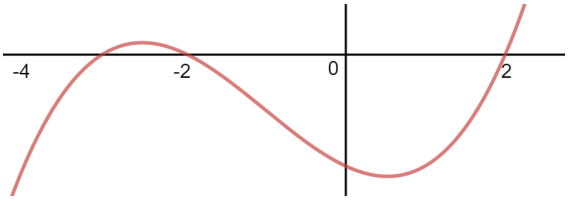


Figure 4: Non-SingularPolynomial

2.3 Discriminant

The **discriminant** of a polynomial reveals information about the polynomial's roots. It can be expressed as a polynomial function of its coefficients. We are most interested in the case where the discriminant is equal to zero, as it is known that this occurs if and only if the polynomial is singular.

Definition 2.6. For polynomials in the form

$$p(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_2 x^2 + a_1 x + a_0, a_n = 1, n \geq 0,$$

the **discriminant** is defined to be

$$D(p) = \prod_{\substack{i,j=0 \\ i \neq j}}^{n} (r_j - r_i)^2,$$

where $i < j$ and r_i is the ith root of $p(x)$.

Example 2.7. Let $p(x) = x^2 - 2x - 15$, which has the roots $x = -3$ and $x = 5$. Hence,

$$D(p) = (-5 - 3)^2 = 1(-8)^2 = 64.$$

Remark. The discriminant of a polynomial is 0 if and only if the polynomial has at least two roots are equal.

Proof. First, let a p be a polynomial with degree n and let p have two roots r_k and r_l such that $r_k = r_l$. Therefore,

$$\prod_{i \neq j}^{n} (r_j - r_i)^2 = 0$$

and thus $D(p) = 0$.

Now, let $D(p) = 0$. Then,

$$\prod_{i \neq j}^{n} (r_j - r_i)^2 = 0.$$

Therefore $r_k = r_l$ for some $k, l \in \mathbb{N}$ such that $k \neq l$. □

Therefore if the determinant of a polynomial is 0, the polynomial is singular. However, this definition is not helpful to us as it requires the roots of the polynomial to be known. We will not know this information before running the program. There is a second way to define the determinant that uses the coefficients of the polynomial.

Definition 2.8 (Resultant)**.** Given two polynomials,

$$f(x) = a_n x^n + a_{n-1}x^{n-1} \cdots + a_2 x^2 + a_1 x + a_0, a_n \neq 0, n \geq 0$$
$$g(x) = b_m x^m + b_{m-1}x^{m-1} + \cdots + b_2 x^2 + b_1 x + b_0, b_m \neq 0, m \geq 0.$$

the **resultant** [4] of $f(x)$ and $g(x)$ is defined as,

$$Res(f(x), g(x)) = \det \begin{bmatrix} a_n & & & & b_m & & & \\ a_{n-1} & a_n & & & b_{m-1} & b_m & & \\ a_{n-2} & a_{n-1} & \ddots & & b_{m-2} & b_{m-1} & \ddots & \\ & a_{n-2} & \ddots & a_n & & b_{m-2} & \ddots & b_m \\ \vdots & & \ddots & a_{n-1} & \vdots & & \ddots & b_{m-1} \\ a_0 & \vdots & \ddots & a_{n-2} & b_0 & \vdots & \ddots & b_{m-2} \\ & a_0 & & & & b_0 & & \\ & & \ddots & \vdots & & & \ddots & \vdots \\ & & & a_0 & & & & b_0 \end{bmatrix}$$

where are unspecified entries in the matrix above are zeros.

Definition 2.9 (Discriminant)**.** For polynomials in the form

$$p(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_2 x^2 + a_1 x + a_0, a_n \neq 0, n \geq 0$$

the **discriminant** [4] is defined to be,

$$\frac{(-1)^{\frac{n(n-1)}{2}}}{a_n} Res(p(x), p'(x)).$$

However, using this definition the discriminant gets considerably more complicated as the degree of the polynomial increases.

Example 2.10. Consider a quadratic polynomial in the form

$$p(x) = ax^2 + bx + c.$$

Then, $p'(x) = 2ax + b$. Hence,

$$\frac{(-1)^{\frac{2(2-1)}{2}}}{a} \cdot \det \begin{bmatrix} a & 2a & 0 \\ b & b & 2a \\ c & 0 & b \end{bmatrix} = \frac{-1}{a}(-ab^2 + 4a^2 c)$$
$$= b^2 - 4ac$$

Therefore, $D(p) = b^2 - 4ac$.

The following examples are calculated in the same manner.

Example 2.11. For a cubic polynomial in the form

$$ax^3 + bx^2 + cx + d,$$

the discriminant is defined to be

$$D(p) = b^2c^2 - 4ac^3 - 4b^3 d - 27a^2 d^2 + 18abcd.$$

Example 2.12. For a quartic polynomial of the form

$$p(x) = ax^4 + bx^3 + cx^2 + dx + e,$$

the discriminant is defined to be

$$\begin{aligned} D(p) =& 256a^3e^3 - 192a^2bde^2 - 128a^2c^2e^2 + 144a^2cd^2e \\ & - 27a^2d^4 + 144ab^2ce^2 - 6ab^2d^2e - 80abc^2de \\ & + 18abcd^3 + 16ac^4e - 4ac^3d^2 - 27b^4e^2 + 18b^3cde \\ & - 4b^3d^3 - 4b^2c^3e + b^2c^2d^2. \end{aligned}$$

As it is not efficient for higher power polynomials, we would like to find a more efficient way to determine whether a polynomial is singular or non-singular that does not rely on solving the discriminant.

3 Machine Learning

Our goal is to use a machine learning algorithm to predict whether a polynomial will be singular or non-singular. Machine learning, defined by *Deep Learning* is the "science of programming computers so they can learn from data." [6] It is our goal that our program will learn to classify polynomials as singular and non-singular from patterns of the polynomials coefficients. We will be using supervised learning, where we will be giving a specific data set to the algorithm in order to train it. This set includes lists of coefficients of polynomials as well as labels attached to these lists identifying them as belonging to a singular or non-singular polynomial. We use a 1 to label a polynomial as singular and a 0 to label a polynomial as non-singular. When given a new polynomial, our program will be able to apply the knowledge gained from the set of labeled data to predict whether the polynomial is singular or non-singular.

Definition 3.1 (Training Set)**.** A **training set** is a set of "examples that the system uses to learn" [7].

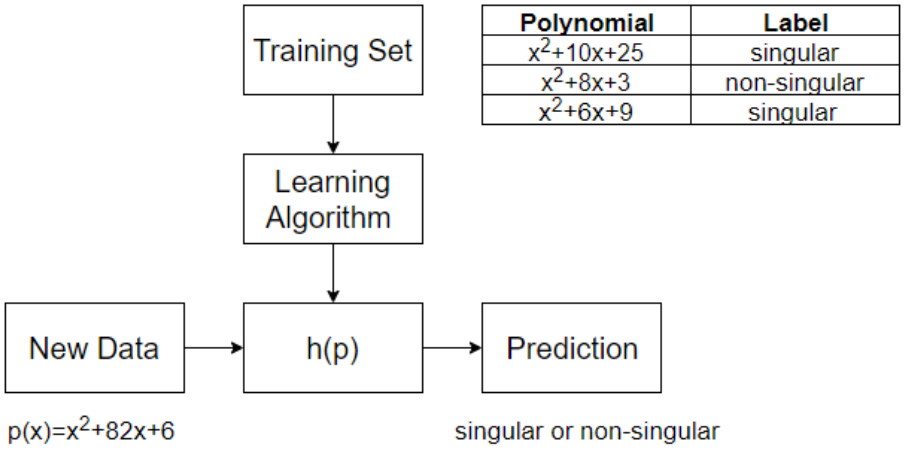
Example 3.2. For the polynomial $x^2 + 4x + 4$, since the discriminant $b^2 - 4ac = 4^2 - 4(1)(4) = 0$, the polynomial is singular. Hence the associated element of the training set is $(1, 4, 4, 1)$, where the first three elements are the coefficients of the polynomial and the last element is the label. In this case, the last element,1, indicates the polynomial is singular.

Definition 3.3 (Test Set)**.** A **test set** is a group "of examples that were collected separately from the training set." [6] This set is used to assess the model after it has been trained.

It is important to test the program on examples that it did not train on as this is what the program will be doing when in use. A high accuracy on the training set does not guarantee that the model will generalize well. This is referred to as overfitting.

Definition 3.4 (Overfitting)**. Overfitting** occurs when "the model performs well on the training data, but does not generalize well." [7]

The algorithm functions as follows. The training set is fed into the machine learning algorithm, which then creates a predictor function. Once this function is created, a new polynomial can be fed through the predictor function which will classify it as either singular or non-singular. The test set is then used to assess to accuracy of the model. Below is a visualization of this process, where $h(p)$ is the predictor function.

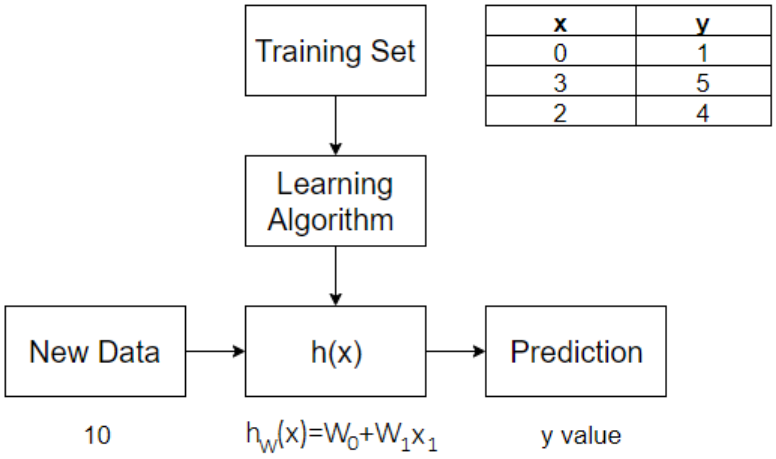


Polynomial	Label
$x^2+10x+25$	singular
x^2+8x+3	non-singular
x^2+6x+9	singular

Figure 5: Machine Learning

3.1 Linear Regression

To demonstrate this process we will use the example of linear regression which is shown in Figure 6.



x	y
0	1
3	5
2	4

Figure 6: Linear Regression

The training set in this case is a collection of ordered pairs. The machine learning algorithm will create the line of best fit for the training set, choosing the best values of $_0$ and W_1. This differs slightly from the original problem, as this predictor function will output any real number, while our predictor function will only have two outputs, 1 or 0 to denote singular or non-singular. However, the majority of the process is still the same. The line of best fit created by the algorithm will be the predictor function,

$$h_W(x) = W_0 + W_1 x,$$

Use the convention of letting $x_0 = 1$. This function contains parameters W_0 and W_1, which the algorithm will determine the values of. To do so, the algorithm must find the line that minimizes the distance of every point in the training set to the line. This is achieved using a loss function such as the Mean Squared Error which is shown below. In this function, m is the number of examples in the training set, $h_W(x_i)$ is the predicted y value the ith element of the training set and y_i is the actual y value from the ith element of the training set. We want to minimize this loss function.

$$J(W) = \frac{1}{m}\sum_{i=1}^{m}(h_W(x_i) - y_i)^2$$

In order to minimize this function, we use an iterative process called gradient descent shown below,

$$W_j := W_j - \alpha\frac{\partial}{\partial W_j}J(W).$$

This process finds the value of the jth weight in the predictor function. The first value of W_j is to be randomly chosen by the program. Then, in each iteration of the process, its value changes to be the left hand side of the equation. The number of iterations this process is followed is referred to as the training steps. This must be chosen appropriately. If the number of training steps is too low, the function will not be minimized. This process also requires a learning rate, α. The learning rate is a parameter that can be changed in the program. It, too, needs to be chosen appropriately to effectively minimized the function. If it is too large, the function will not be minimized and if it is too small it will take a long time for function to be minimized. Finally gradient decent also uses $\frac{\partial}{\partial W_j}J(W)$. Solving for $\frac{\partial}{\partial W_0}J(W)$ we find,

$$\begin{aligned}
\frac{\partial}{\partial W_0}J(W) &= \frac{\partial}{\partial W_0}(\frac{1}{m}\sum_{i=1}^{m}(h_W(x_i) - y_i)^2) \\
&= (\frac{1}{m})\sum_{i=1}^{m}\frac{\partial}{\partial W_0}(h_W(x_i) - y_i)^2 \\
&= (\frac{1}{m})\sum_{i=1}^{m}2(h_W(x_i) - y_i)\frac{\partial}{\partial W_0}(h_W(x_i) - y_i) \\
&= (\frac{2}{m})\sum_{i=1}^{m}(h_W(x_i) - y_i).
\end{aligned}$$

Note that

$$\frac{\partial}{\partial W_0}(h_W(x_i) - y_i) = \frac{\partial}{\partial W_0}(W_0 + W_1 x_i - y_i) = 1.$$

Therefore, we can rewrite the process of gradient decent for a single example in the training set as,

$$W_0 := W_0 + \alpha(\frac{2}{m})\sum_{i=1}^{m}(h_W(x_i) - y_i).$$

Now, solving for $\frac{\partial}{\partial W_1}J(W)$ we find,

$$\begin{aligned}\frac{\partial}{\partial W_1}J(W) &= \frac{\partial}{\partial W_1}(\frac{1}{m}\sum_{i=1}^{m}(h_W(x_i)-y_i)^2)\\ &= (\frac{1}{m})\sum_{i=1}^{m}\frac{\partial}{\partial W_1}(h_W(x_i)-y_i)^2\\ &= (\frac{1}{m})\sum_{i=1}^{m}2(h_W(x_i)-y_i)\frac{\partial}{\partial W_1}(h_W(x_i)-y_i)\\ &= (\frac{2}{m})\sum_{i=1}^{m}(h_W(x_i)-y_i)x_i.\end{aligned}$$

This is because

$$\frac{\partial}{\partial W_0}(h_W(x_i)-y_i) = \frac{\partial}{\partial W_0}(W_0+W_1x_i-y_i) = x_i.$$

Hence,

$$W_1 := W_1 + \alpha(\frac{2}{m})\sum_{i=1}^{m}(h_W(x_i)-y_i)x_i.$$

3.2 Neural Networks

We will use a neural network as our learning algorithm. This algorithm was originally based on neurons in the brain but currently has little resemblance to these first networks. The purpose of the network is to create the predictor function. It does this by representing the predictor function, which has the potential to be a very complex function, as the composition of simpler functions.

The network is comprised of layers of nodes, which are multi-variable functions. The output values of a node are then inputs in the following layer. These layers are known as the input layer, the output layer and the hidden layers.

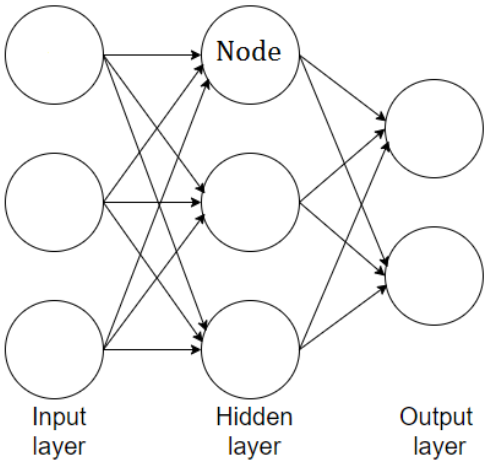


Figure 7: Neural Network

Definition 3.5 (Input Layer)**.** The **input layer** is the left-most layer in Figure 7. This is where the coefficients are input into the system.

Definition 3.6 (Hidden Layer(s))**. Hidden layers** are the layers in between the input and output layers.

Definition 3.7 (Output layer)**.** The **output layer** is the right-most layer in Figure 7. This layer will output the classification of the polynomial.

The multi-variable functions that are used in these nodes. traditionally have a linear component. The ReLU function is them applied to this linear component.

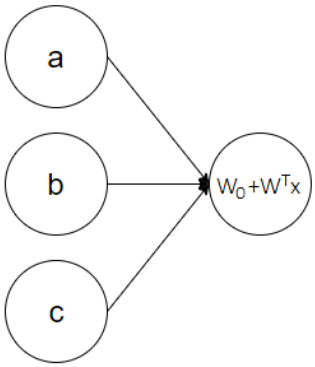


Figure 8: Linear Node

For a quadratic functions in the form $p(x) = ax^2 + bx + c$ the output of a node is

$$L_W(p) = max(W_0 + aW_1 + bW_2 + cW_3, 0).$$

Through training, the values of the weights and bias are decided. The result of this node is then an input of the nodes in the following layer.

Since we have a binary classification problem, identifying polynomials as singular or non-singular, we will be using one node in the output layer with the sigmoid function,

$$\sigma(z) = \frac{1}{1 + e^{-z}},$$

where z is the output of the last layer of hidden nodes effect by the weights and bias of the output layer.

The range of this function is $[0, 1]$. This represents the probability the polynomial is singular. If the output of the function is greater than or equal to .5, the polynomial will be classified as singular. If the output is less than .5, the polynomial will be classified as non- singular.

A visualization of a neural network with three hidden layer is shown in Figure 9. The inputs of this network are coefficients of quadratic polynomials in the form of $ax^2 + bx + c$.

Training the system works almost identically to the process of linear regression, using the process of gradient decent. However, $\frac{\partial}{\partial W_j} J(W)$ can not be found directly in an efficient manner. Instead, a process called backpropagation is used. This is based on the chain rule of calculus. The usefulness of this process was fully recognized in a paper by David Rumehart, Goeffrey Hinton and Ronald Williams [5].

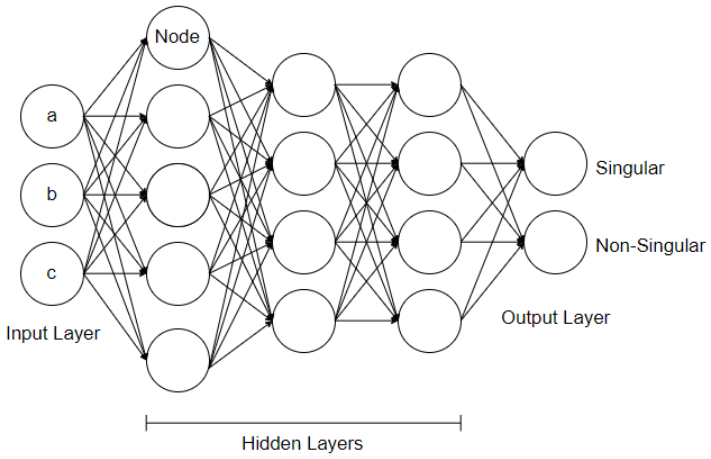


Figure 9: Neural Network 2

3.3 Linear Node Results

We used a variety of neural networks to classify quadratic, cubic, and quartic polynomials. We chose these polynomials to begin with as they are the simplest to classify. Our training set was composed of 640,000 polynomials and the test set was composed of 160,000 polynomials. Both the training and the test sets were fifty percent singular and fifty percent non-singular.

To do so, we first generated factors from the normal distribution with a mean of 0 and a variance of 1. For singular polynomials, we repeated one of these roots. Then, we multiplied these factors to expand the polynomial and find its coefficients.

Example 3.8. To create a singular quadratic polynomial, we randomly generate the number 2.678. Then we calculate

$$(x + 2.678)(x + 2.678) = x^2 + 5.356x + 7.171684.$$

Hence we record $(1, 5.356, 7.171684, 1)$, where the first three entries are the coefficients of the polynomial and the last entry, 1, denotes the polynomial is singular.

To judge the effectiveness of our model, we considered three accuracies: singular accuracy, which is the percentage of correctly classifies singular polynomials, non-singular accuracy, the percentage of correctly classified non-singular polynomials, and total accuracy, the percentage of correctly classified singular and non-singular polynomials. Singular accuracy is the most important accuracy out of these three, as incorrectly classifying a singular polynomial will cause the program to fail. In contrast, incorrectly classifying a non-singular polynomial will cause the program to run on its default settings.

Below are the highest accuracies we were able to achieve on the test sets. Results for the quadratics and cubics where achieved using a network with 2 hidden layers with 100 nodes in each layer, while results for the quartics used a network with 2 hidden layers with 500 nodes in each layer.

Polynomial	Learning Rate	Training Steps	Non-Singular Accuracy	Singular Accuracy
Quadratic	.05	1,000,000	91.379	100
Cubic	.5	1,000,000	80.937	99.526
Quartic	.05	2,000,000	74.116	94.065

Polynomial	Total Accuracy
Quadratic	95.670
Cubic	90.233
Quartic	84.091

We were able to achieve the highest total accuracy and singular accuracy with quadratic functions. This was expected as quadratics have the simplest discriminant of these degrees of polynomial using Definition 2.8. This indicates that quadratic polynomials have the simplest relationship between their coefficients and being singular or non-singular. We also noticed complexity of the network increased substantially from cubic to quartic polynomials and yet our accuries were not as high. This also expected, since we saw in Example 2.11 that the relationship between the coefficients and multiplicity of the roots is complicated.

3.4 Quadratic Node

We know from Definition 2.9 the discriminant of a single variable polynomial. However, there is no known definition of the discriminant for systems of polynomials of more than two variables. A machine learning algorithm such this could be used to classify multivariate systems in future. Since the discriminant indicates whether a polynomial is singular or non-singular, if we can create a node that will output the discriminant, we may be able to achieve higher accuracies. Therefore, we decided to create a quadratic node, as it should produce a better approximation of the discriminant than the linear function normally used. The discriminant is a quadratic function for second degree polynomials. We decide to change the output of the node to be,

$$Q_W(x) = x^T Ax,$$

where x is the row matrix of coefficients of a polynomial and A is a square matrix of weights with the same number of rows as x^T has columns.

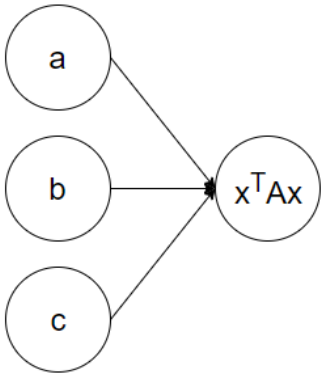


Figure 10: Quadratic Node

A network with a single quadratic node will output all possible second degree polynomials based on the inputs. Since the discriminant is a second-degree polynomial of the coefficients for quadratic

functions, the discriminant will be a possible output of the network. This should allow the network to more easily approximate the discriminate hopefully letting us achieve higher accuracies than we did previously with the traditional node.

For the quadratic node shown in Figure 10,

$$\begin{aligned} Q_W(x) &= \begin{bmatrix} a & b & c \end{bmatrix} \begin{bmatrix} W_1 & W_2 & W_3 \\ W_4 & W_5 & W_6 \\ W_7 & W_8 & W_9 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} \\ &= \begin{bmatrix} W_1(a) + W_4(b) + W_7(c) & W_2(a) + W_5(b) + W_8(c) & W_3(a) + W_6(b) + W_9(c) \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} \\ &= W_1(a^2) + (W_2 + W_4)(ab) + (W_3 + W_7)(ac) + W_5(b^2) + (W_6 + W_8)(bc) + W_9(c^2). \end{aligned}$$

4 Results from Quadratic node

We tested the new model using the quadratic node on quadratic, cubic and quartic polynomials as we had done with the tradition network with linear nodes. The training and test set was again composed of fifty percent singular and non-singular second degree polynomials. To create the set, factors were generated from the normal distribution with a mean of 0 and a variance of 1. Then these factors were expanded. To prevent incorrectly classifying singular polynomials and to increase our accuracy, we considered only polynomials with a discriminant of less than a difference of 10^{-12} from 0 to be singular. There were 640,000 elements in the training set and 160,000 elements in the test set. All of the trials below were run with 640,000 training steps. Below are tables of accuracies in percents comparing neural networks composed of traditional and quadratic nodes in the same network structure. We considered only total accuracy to compare the two networks. The accuracies seen below are those from the test sets.

Table 1: Quadratic Polynomials

Learning Rate	Traditional Node	Quadratic Node
.005	64.4494	83.1922
.05	74.6216	90.28062
.5	64.3888	94.3241

Network structure of one hidden layer with one node

Table 2: Quadratic Polynomials

Learning Rate	Traditional Node	Quadratic Node
.005	67.4806	83.9794
.05	82.5444	90.7363
.5	84.0028	94.7522

Network structure of one hidden layer with 5 nodes

Table 3: Cubic Polynomials

Learning Rate	Traditional Node	quadratic Node
.005	62.9022	66.5178
.05	56.1587	66.5991
.5	63.1631	66.6662

Network structure of one hidden layer with one node

Table 4: Cubic Polynomials

Learning Rate	Traditional Node	Quadratic Node
.005	66.2491	66.7469
.05	75.1515	66.0141
.5	73.155	66.1922

Network structure of one hidden layer with 5 nodes

Table 5: Quartic Polynomials

Learning Rate	Traditional Node	Quadratic Node
.005	55.625	59.4644
.05	56.6160	59.6984
.5	57.9203	59.0156

Network structure of one hidden layer with one node

Table 6: Quartic Polynomials

Learning Rate	Traditional Node	Quadratic Node
.005	55.4412	59.9347
.05	59.1866	59.8650
.5	57.5862	59.8550

Network structure of one hidden layer with 5 nodes

When used on quadratic data, the network with the quadratic node achieves significantly higher accuracies than the network with the linear nodes. The largest difference occurs with the network structure of one node with the highest learning rate, where the quadratic node performed over 29.9% better than the traditional node. This was the second highest accuracy reached at 94.3241%, over 2.9% higher than the accuracy previously reached using linear nodes. It also had a substantially simpler network structure, using one layer with one hidden node versus the previous structure with 2 hidden layers and 100 nodes in each layer. We predict this outcome is because the predictor function from the quadratic node network is a better approximation of the discriminant. The new network also achieved a higher accuracy with a higher learning rate. Due to these two factors, the network structure with the quadratic node was able to train substantially faster than that of the larger network with the linear node. However, the difference between the two accuracies decreased when the number of nodes in the network increased.

With cubic polynomials, our network with quadratic nodes does not have the discriminant of a cubic polynomial as a possible predictor function. This is because the discriminant of a cubic polynomial is a fourth-degree polynomial, yet the output of our node is only a second-degree

polynomial. Predictably, our accuracies for cubic polynomials decreased substantially from those of quadratic polynomials. With structure of one node, the quadratic node still outperforms the linear node, but this changes when the number of nodes in the network increases to five. When this occurs the network achieves the highest accuracies for cubic polynomials at 75.1515%, over 19% less than the highest accuracy reached for quadratic polynomials.

The accuracies decrease again when we test quartic polynomials. We were able to reach an accuracy of 59.9347 %, which is over 15% less accurate then results reached with cubic polynomials and over 34% less accurate than results reached with quadratic polynomials. Again, the discriminant of a quartic polynomial is not a possible predictor function in this system. However, our quadratic node does outperform the linear node even when the number of nodes increases.

5 Future work

In the future we would like to answer a variety of other questions including:

- How close to the discriminant is our predictor function?

  To assess how close the predictor function created by our neural network is to the discriminant, we may be able to evaluate the Euclidean norm.

- Can we create cubic and quartic nodes?

- Can we create a network that has the discriminant as a possible predictor function for any degree polynomial?

  This outcome maybe possible by using a new activation function or using a quadratic node in a specific network structure.

- Can we effectively use neural networks to identify if systems of polynomial equations of more than one variable will be singular or non-singular?

Answering these questions will continue to help decrease the run time of Bertini 2.0.

6 Conclusion

Classifying singular polynomials has the potential to decrease the runtime of Bertini 2.0. Using neural networks with both traditional linear nodes and our quadratic node, we were able to classify polynomials as singular or non- singular. However, our accuracies did decrease as the degree of the polynomial increased. We also found our quadratic node to be substantially more efficient at classifying quadratic polynomials. We believe this is because the discriminant is a possible predictor function of the network. Answering the above questions will get us closer to implementing with with Bertini.

References

[1] Harnam Arneja, Andrew Bender, Sam Jugus, and Tim Reid, *Solving the gps equations*, George Mason University, http://mason.gmu.edu/ treid5/Math447/GPSEquations/.

[2] Daniel J. Bates, Jonathan D. Haunstein, Andrew J. Sommese, and Charles W. Wampler, *Numerically solving polynomial systems with bertini*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2013.

[3] Danielle Brake, *Homotopy continuation*.

[4] David A. Cox, John Little, and Donal OShea, *Using algebraic geometry*, Springer, 2005.

[5] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams, *Learning representations by back propagating errors*, Nature **323** (1986), 533–536.

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville, *Deep learning*, The MIT Press, 2017.

[7] Aurlien Gron, *Hands-on machine learning with scikit-learn and tensorflow: concepts, tools, and techniques to build intelligent systems*, OReilly, 2018.

[8] Yan-Bin Jia, *Roots of a polynomial system*, Roots of a Polynomial System (2018), http://web.cs.iastate.edu/ cs577/handouts/polysystem.pdf.

[9] Andrew John. Sommese and Charles W. Wampler, *The numerical solution of systems of polynomials: arising in engineering and science*, World Scientific, 2005.