UNIVERSIDADE D
COIMBRA

Ana Maria Marques Cruz

PREDICTING THE PERFORMANCE OF BUCHBERGER`S ALGORITHM

Dissertação no âmbito do Mestrado em Matemática, Ramo Análise Aplicada e Computação, orientada pelos Professores Doutores João Eduardo da Silveira Gouveia e Eduardo Manuel Dias e apresentada ao Departamento de Matemática da Faculdade de Ciências e Tecnologia.

Setembro de 2022

Predicting the performance of Buchberger's algorithm

Ana Maria Marques Cruz

UNIVERSIDADE D
COIMBRA

Master in Mathematics

Mestrado em Matemática

MSc Dissertation | Dissertação de Mestrado

Setembro 2022

Acknowledgements

Quero agradecer ao Eduardo Dias e ao Professor João Gouveia pela orientação, disponibilidade e ensinamentos ao longo da elaboração desta dissertação. Agradeço também ao Departamento de Matemática pelos recursos disponibilizados e a todos os Professores com quem me cruzei ao longo do meu percurso académico.

À empresa Inductiva Research Labs, e à incrível equipa que a constitui, obrigado por me terem dado a oportunidade de fazer parte deste projeto durante o último ano, por tudo o que aprendi ao longo do estágio e por me terem aberto as portas a novas áreas de conhecimento.

Agradeço à minha família, principalmente aos meus pais por me terem dado esta oportunidade e, sobretudo, por todo o apoio, paciência e carinho presentes ao longo do meu percurso académico. À minha irmã, pelos conselhos e bons momentos partilhados, enquanto estudantes e irmãs, e à Luna por ter estado sempre ao meu lado.

Por último, quero agradecer a todos os meus amigos que sempre me apoiaram e estiveram presentes, pelos abundantes momentos de descontração e diversão que desempenharam um importante papel na minha vida e no processo de escrita desta dissertação.

Abstract

Gröbner bases are a fundamental concept in computational algebra. Since the creation of the theory behind them in 1949, by Wolfgang Gröbner, they became an important tool in any area where polynomial computations play a part, both in theory and in practice. Although they have proved to be very useful, their calculation is very expensive in certain cases. The first algorithm ever developed to compute these bases is the so-called Buchberger's Algorithm, and is still one of the most commonly used algorithms for this purpose.

As a preliminary step in improving the efficiency of the algorithm, one would like to be able to predict, given an ideal, how complicated it is to compute its Gröbner basis using Buchberger's Algorithm. In this dissertation, we cover precisely this question, following recent work of Mojsilović, Peifer and Petrović. More precisely we introduce and apply linear regression and Machine Learning tools to attempt to predict the number of iterations needed, and show that it is, in certain cases, possible for us to achieve this goal.

Resumo

As bases de Gröbner são um conceito fundamental em álgebra computacional. Desde a criação desta teoria, em 1949, por Wolfgang Gröbner, elas tornaram-se numa ferramenta importante em qualquer área onde exista computação polinomial, tanto na teoria como na prática. Embora se tenham demonstrado extremamente úteis, o seu cálculo é muito pesado em certos casos. O primeiro algoritmo desenvolvido para calcular estas bases é chamado Algoritmo de Buchberger, que ainda é um dos algoritmos mais utilizados para esse fim.

Como passo preliminar para melhorar a eficiência do algoritmo, gostaríamos de poder prever, dado um ideal, quão complicado é calcular a respetiva base de Gröbner usando o Algoritmo de Buchberger. Nesta dissertação, abordamos precisamente esta questão, seguindo o trabalho recente de Mojsilović, Peifer e Petrović. Mais precisamente, introduzimos e aplicamos ferramentas de regressão linear e Machine Learning para tentar prever o número de iterações necessárias, e mostramos que, em certos casos, é possível atingir esse objetivo.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

The concept of Gröbner basis was first introduced in 1949, by Wolfgang Gröbner, an Austrian mathematician. Later, in 1965, Bruno Buchberger, with Gröbner by his side, developed the Gröbner basis theory for ideals in commutative polynomial rings [4]. This development of the Gröbner basis began with the initial task to find the complete solution of a system of algebraic equations:

$$f_1(x_1,...,x_n) = 0, f_2(x_1,...,x_n) = 0, \ldots, f_m(x_1,...,x_n) = 0. \tag{1.1}$$

As is commonly known, the fewer the number of variables, compared to the number of equations, the easier it is to solve the system. But when the number of variables is big, it is necessary to approach the problem in a different way. One approach is to find a new system of equations, 1.2, equivalent to the initial system 1.1, with less variables to work with, such that the roots of 1.1 are also the roots of 1.2. What was described is the core idea of the elimination theory, it starts by eliminating the variable x_n, by finding equivalent equations, and so on until there is an equivalent system with less variables.

$$g_1(x_1,...,x_t) = 0, g_2(x_1,...,x_t) = 0, \ldots, g_s(x_1,...,x_t) = 0. \tag{1.2}$$

A few classical methods for cases with only one variable already existed. Even though the methods work, they were not efficient enough for the multivariate case, as it is flawed. Therefore, Professor Wolfgang Gröbner figured that another more efficient method must exist, this method is now known as the Gröbner Basis. In order to avoid these flaws, ideal theory was introduced [17]: *"All of the polynomials of the ideal $A = (f_1, \ldots, f_m)$ which do not contain the variable x_n form an ideal $(\ldots)$ in the polynomial ring $K[x_1,...,x_n - 1]$, whose basis polynomials set equal to zero produce the system of equations 1.2."*

It was a few years later that, with the help of Buchberger, this theory was defined and denominated Gröbner basis, and it also developed the algorithmic theory for the computing of a basis that could take any set of polynomials and always construct an unique Gröbner basis for any set of polynomials. In the following years, work to improve the algorithms performance in problems related to Gröbner basis was done. Having the Buchberger algorithm as a starting point, other algorithms such as Faugère's F4[13] and F5[14] algorithms were designed to fix some performance problems in the initial algorithm. The F4 algorithm shows more efficiency for large input, by implementing simultaneous reductions of several S-polynomials, computing the reduced row echelon form of coefficient matrices. And the

F5 algorithm was created to avoid redundant computations from F4, the idea behind it is to detect unnecessary reductions before they appear. By using machine learning training methods such as Supervised Learning and Reinforcement Learning, useful information was discovered. For example, discovering new selection strategies for steps with freedom of choice, data domain information and that Buchberger's algorithm can be *machine learnable*.

The applications of Gröbner basis are important in many fields of research in mathematics such as: commutative algebra, geometric theorem proving and elimination theory among others. Also, in science and engineering, for example: in coding theory, robotics, software engineering and many others. The Gröbner basis theory can be introduced in almost anywhere polynomials are used. In computer algebra, some problems with polynomials are quite complex to solve, between them there are the problem of solving polynomial equations and the ideal membership problem. Solving these problems provide an answer to many practical challenges and Gröbner bases are a powerful tool that can be used in the process of solving those problems.

In this dissertation, Chapter 2 introduces some needed base notions for understanding what are and how Gröbner basis work. First, the Ideal Membership Problem is stated as motivation for the need of Gröbner basis. Then, following the motivation problem, and in order to try and solve it, Monomial Ordering, Division algorithm and Gröbner basis are defined. In the end of the subchapter 2.4 the problem initially introduced as a motivation, is solved using a Gröbner basis. After that, the needed algorithmic theory for the Gröbner basis computation, the Buchberger's algorithm, completes the second chapter.

Some machine learning tools were used in the study of the Buchberger's Algorithm, such as in the article *Learning a performance metric of Buchberger's algorithm*[26]. Where the complexity of Buchberger's Algorithm is studied, by trying to predict the number of polynomial additions during one run of the algorithm. This article is the backbone of this dissertation. To better understand the methods used in the article, some background related to Machine Learning, that will be mentioned and used in the following chapters, is presented. Since, for this dissertation, only some Supervised learning algorithms were needed, this chapter focuses on Linear Regression models, Simple Neural Networks and Recursive Neural Networks.

For a proper review, some concepts of abstract algebra referred in the article and in the follow up chapters, Binomial Ideals and Toric Ideals, are firstly described in Chapter 4. Then, the articles methodology and results are described, to be used ahead for comparison purposes. Following the review, the research goals, succeeding the article's for this dissertation, are described.

Chapter 5 contains information regarding the data used for the research and how it was generated. It also compares the performance of multiple selection strategies in different datasets, and from it, the main selection performance used for the calculation of Gröbner bases is defined.

In Chapter 6, the linear regression models used are described, as well as respective performance results, using linear and multiple regressions. Those results are compared with the linear regression models from the article in Chapter 4. Next, the architecture of the neural networks models are detailed, both simple neural network and recursive neural network. The models have multiple parameters to decide, the selection process is explained for the purpose of understanding how different datasets have different model architecture. The final results from the training of the models are depicted in the last section of the chapter.

Lastly, in the last Chapter 7, the results are discussed and compared with the article results in Chapter 4. Throughout this research, some remarks and suggestions that surfaced for possible future work and improvements that were not possible to execute for this dissertation, are also mentioned.

Chapter 2

Gröbner Basis

2.1 Ideal Membership Problem

The Ideal Membership problem is a fundamental algorithmic problem with multiple applications in solving polynomial systems. It essentially consists of finding when some polynomial identity is implied by a set of other polynomial relations.

In order to properly define the Ideal Membership problem, the notion of a polynomial ideal needs to be introduced first:

Definition 2.1.1. *Let $R = K[x_1,...,x_s]$ be a polynomial ring. Its elements are all polynomials with $x_1,...,x_s$ variables and coefficients from a field K. A polynomial ideal, $I = \langle f_1,...,f_k \rangle \subseteq R$ is the set of all polynomials generated by $f_1,...,f_k$, i.e.*

$$I = \{g_1 f_1 + \cdots + g_k f_k, | g_i \in R\}.$$

Now, the problem can be stated:

Ideal Membership Problem (IMP): Given $f_1,...,f_s$, a set of polynomials on multiple variables. Is there a way to know if a polynomial f belongs to the ideal I generated by $f_1,...,f_s$? To put it more simply, are there $q_1,...,q_s$ polynomials such that f can be written as

$$f = q_1 f_1 + ... + q_m f_s?$$

As stated before, this is a fundamental problem in commutative algebra, and has numerous applications. One illustrative example is geometry theorem proving. Consider the classic Circle Theorem of Apollonius whose illustration is given in Figure 3.2. The theorem states:

Example 2.1.1. *Let ABC be a right triangle in the plane, with a right angle at A. The midpoints of the three sides and the foot of the altitude drawn from A to the line BC lies in a circle.*

The result can be rewritten into an IMP. For this, it is necessary to express the premises as multivariate polynomials. First, the coordinates of each point need to be defined. Without loss of generality, one can assume A as the origin, $A = (0,0)$, B in the y axis, $B = (y,0)$, C in the z axis

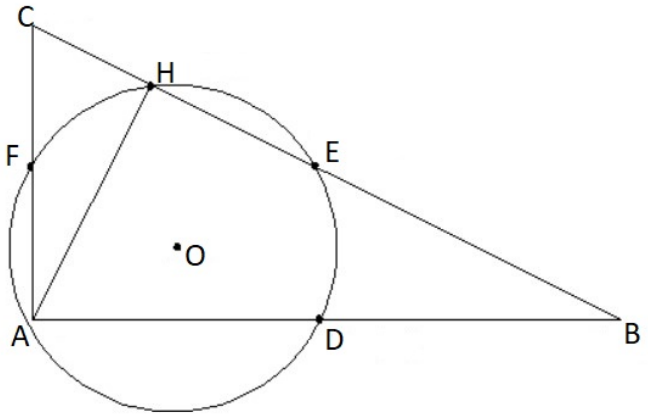


Fig. 2.1 A representation of the *Circle Theorem of Apollonius*

$C = (0,z)$, the center of the circle $O = (x_7,x_8)$ $H = (x_5,x_6)$, $D = (x_1,0)$, $F = (0,x_2)$ and $E = (x_3,x_4)$, the last three are the midpoints of each side. Having the coordinates defined, one can translate the geometric hypothesis into eight polynomial identities, as shown in Table 2.1.

Polynomial Equations	Description
$p_1 := 2x_1 - y = 0$	D is the midpoint of AB
$p_2 := 2x_2 - z = 0$	F is the midpoint of AC
$p_3 := 2x_3 - y = 0$ $p_4 := 2x_4 - z = 0$	E is the midpoint of BC
$p_5 := x_5y - x_6z = 0$	AH $\perp$ BC
$p_6 := x_5z + x_6y - yz = 0$	H lies in BC
$p_7 := (x_1 - x_7)^2 + x_8^2 - ((x_3 - x_7)^2 + (x_4 - x_8)^2) = 0$ $p_8 := (x_1 - x_7)^2 + x_8^2 - x_7^2 - (x_8 - x_2)^2) = 0$	$\|OD\| = \|OE\| = \|OF\|$
$c =: (x_5 - x_7)^2 + (x_6 - x_8)^2 - ((x_1 - x_7)^2 - (0 - x_8)^2) = 0$	$\|OD\| = \|OH\|$

Table 2.1 Polynomial equations for the Circle Theorem of Apollonius.

Now it is just necessary to show that c lies in the ideal generated by the polynomials $p_1,...,p_8$.

How can one solve this IMP? If it was the univariate case, finding if a polynomial can be expressed as a polynomial combination of elements of $F = \{f_1,...,f_k\}$ is quite easy, since there is a single polynomial that can be used to verify the membership. In fact, the ring of polynomials in one variable over any field is a *Principal Ideal Domain* i.e., every ideal is generated by a single polynomial.

This polynomial is the *greatest common divisor* (GCD) of F, and can be obtained, for example, by a simple use of the Euclidean Algorithm. One way of thinking of the Euclidean Algorithm to find the greatest common divisor of F is the following:

- Take a pair of elements from F and divide the one with the largest degree by the other.
- Replace the one with the largest degree by the remainder of the division. If the remainder is zero, remove it from F.
- Repeat until only one polynomial is left, that is the GCD of F.

Example 2.1.2. *Consider the two polynomials, $p(x) = x^5 - 4x^3 + 4x + 3$ and $g(x) = x^3 + 3x + 1$, with $x \in \mathbb{Z}$. To calculate the greatest common divisor of the pair $(f(x), g(x))$, one need to perform repeated divisions with the remainder, like so, $gcd(f,g) = gcd(g,r_1) = gcd = (r_1,r_2) = ... = gcd(r_n,0)$, where r_n represent the remainder from the division of the previous two polynomials. First, it starts by having $gcd(f,g)$:*

$$x^5 - 4x^3 + 4x + 3 = (x^3 + 3x + 1)(x^2 + 1) + (4x^2 + x + 2)$$

obtaining $r_1 = 4x^2 + x + 2$. Then, calculate $gcd(g,r_1)$:

$$x^3 + 3x + 1 = (4x^2 + x + 2)(4x + 4) + (x + 3),$$

obtaining $r_2 = x + 3$, and finally see that r_2 divides r_1,

$$4x^2 + x + 2 = (x + 3)(4x + 4)$$

so $gcd(r_2,r_1) = 0$, and therefore $gcd(f,g) = r_2 = x + 3$.

Once the GCD(F) is found, simply divide f by GCD(F), to verify if it is or is not generated by F, which happens if and only if the rest of the division is zero.

The problem appears when there is more than one variable to consider as it stops being a Principal Ideal Domain. One can try to see what fails in the division algorithm, to see what needs to be fixed. The whole idea of the Euclidean Algorithm is that at each step the degree of one of the elements of the set can be reduced by dividing it by another. This fails in a set of multivariate polynomials, and it is not even clear what highest degree should be replaced with. The first step towards a multivariate division algorithm is therefore to clarify this notion.

2.2 Monomial Ordering

To be able to generalize the long division algorithm to multivariate polynomials, the task of finding the leading term of a polynomial requires to sort the terms of the polynomial in a descending or ascending order, without ambiguity. To do that, it is necessary to define what is needed from this order.

Definition 2.2.1. *A monomial order ($>$) is a relation on the set of monomials of $K[x_1,...,x_s]$ that satisfies the following properties:*

- *is a total order, meaning every two monomials are comparable;*

- *is a well order, i.e., in a non-empty set of monomials there is always a smallest element;*
- *respects multiplication, i.e., if $a \geq b$ and c is another monomial in $k[x_1,...,x_n]$, then $ac \geq bc$.*

These three properties play an important role in the division algorithm. The first property allows to identify the leading term from a polynomial with multiple terms, the second property implies that the algorithm will eventually end and finally, the last property ensure that the leading terms will not change even with the multiplications involved in divisions.

There are many monomial orderings, and picking which one to use has an important numerical impact. For theoretical purposes and simplicity, only the lexicographic order is defined.

Definition 2.2.2. *The lexicographic order is defined as in the lexicographical order on words. When having multiple variables $x_1,x_2,...,x_n$ (where the order of the variables is fixed) it works by comparing the exponents of x_1 in the monomial first, and if the exponents are equal, it will break the tie by comparing the exponents of x_2 and so on.*

Example 2.2.1. *Consider the following two examples for monomial representation and comparing two monomials in lexicographic order:*

- $xy^2 > y^3z^4$ *in lex order,*
- $x^3y^2z^4 > x^3y^2z$ *in lex order.*

Note that it will be used x^q to symbolize an arbitrary monomial with q representing a vector of exponents, $q = (q_1,...,q_s)$ and x a vector of its variables with $x = (x_1,...,x_s)$, e.g., for the monomial $x_1x_2^4x_3^5$ then the correspondent vector of exponents and vector of variables are: $q = (1,4,5)$ and $x = (x_1,x_2,x_3)$. With this convention it is possible to identify monomials with their exponent vectors and the inequalities above can be interpreted as

- $(1,2,0) > (0,3,4)$,
- $(3,2,4) > (3,2,1)$.

It is a simple exercise to verify that the lexicographic order verifies all the conditions to be a monomial order. Knowing what an order implies, the concept of leading term for a multivariate polynomial can be defined rigorously, a concept that will depend on the order chosen.

Definition 2.2.3. *Given a fixed monomial ordering in $R = K[x_1,...,x_s]$, and $f \in R$ then the leading term of f, $LT(f)$, is the term associated to the maximum monomial of f.*

For instance, if $f = 3xy^3 + 2x^2y - 3y^5 + 7$, then the leading term is $LT(f) = 2x^2y$ in the lexicographic order induced by $x > y$. If the chosen order is induced by $y > x$, instead the lexicographic order, then the leading term would be $LT(f) = y^5$.

2.3 Division Algorithm in multivariate polynomial rings

The idea behind the division algorithm in the multivariate case is similar to the case of a single variable. While for one variable, the algorithm divides two polynomials by comparing their highest degree terms, in the multivariate case it does the same but by comparing the leading terms with respect to some ordering. The following pseudo-code and definition describes properly the algorithm.

Algorithm 1 Multivariate Division Algorithm with remainder

Require: a polynomial f and a set of non-zero polynomials $F = f_1, ..., f_s$ and a monomial order
Ensure: $r = remainder(f, F)$
  $q_1 = q_2 = ... = q_n = 0$, $r = 0$
  $p \leftarrow f$
  while p $\neq 0$ **do**
    if $LT(f_i)|LT(p)$ for some i **then**
      choose i such that $LT(f_i)|LT(p)$
      $q_i \leftarrow q_i + \frac{LT(p)}{LT(f_i)}$
      $p \leftarrow p - \frac{LT(p)}{LT(f_i)} f_i$
    else
      $r \leftarrow r + LT(p)$
      $p \leftarrow p - LT(p)$
    end if
  end while

The basic idea is that at each step it will decrease the leading term of f by subtracting from f $\frac{LT(f)}{LT(f_i)} f_i$ while there is some f_i such that divides the leading term of f, and if there is not such f_i, collect the leading term in the remainder and proceed with what is left.

Example 2.3.1. *Consider the polynomial $p := f = x^3y + y^2 + yz$ divided by the polynomials $\{xy+1, z\}$ with the lex order induced by $x > y > z$*

- *$LT(p) = x^3y$ which is divisible by $xy = LT(xy+1)$ so p becomes $p - x^2(xy+1) = -x^2 + y^2 + yz$.*
- *$LT(p)$ is now $-x^2$ which is not divisible by any of the leading terms of the divisors, so that term is moved to the remainder that becomes $r(x) = -x^2$, leaving with $p = y^2 + yz$*
- *Now $LT(p) = y^2$, which once again is not divisible by any leading term, so $r = -x^2 + y^2$ and $p = yz$*
- *Finally $LT(p) = yz$ is divisible by z and p becomes $p - yz = 0$, so it stops.*

In the end, it may be observed that the remainder is $-x^2 + y^2$ and that in fact $f = (xy+1)x^2 + (z)y + (-x^2 + y^2)$.

Below it is possible to see that this algorithm always stops, and contributes with the following result.

Proposition 2.3.1 (Division Algorithm in $k[x_1, ..., x_n]$)**.** *Let $>$ be a fixed monomial order (e.g. lex) and $(f_1, ..., f_n)$ polynomials of multivariate variables, $x_1, ..., x_n$. Then every polynomial f can be written*

as $f = q_1 f_1 + ... + q_n f_n + r$, with $q_i, r \in k[x_1, ..., x_n]$ and no term of r (where r is the remainder of the division from f by $(f_1, ..., f_n)$) being divisible by $LT(f_1), ..., LT(f_n)$.

Proof. First, it is necessary to show that f can be written as

$$f = q_1 f_1 + ... + q_n f_n + p + r$$

in every iteration of the above algorithm. The algorithm begins with $q_1 = q_2 = ... = q_n = 0$ and $r = 0$, so in the initial stage it holds. On the IF-ELSE step, if $LT(p)$ is divided by a $LT(f_i)$ for some i=1,...,n, this way:

$$q_i := q_i + \frac{LT(p)}{LT(f_i)}$$

$$p := p - \frac{LT(p)}{LT(f_i)} f_i$$

By checking the equality:

$$q_1 f_1 + ... + q_n f_n + p = (p - \frac{LT(p)}{LT(f_i)} f_i) + q_1 f_1 + ... + (q_i + \frac{LT(p)}{LT(f_i)}) f_i + ... + q_n f_n$$

It is apparent that the equation holds, and will hold for the next loop. Else, is the remainder step, in this stage p and r will change, but $p + r$ will remain the same:

$$p + r = p - LT(p) + r + LT(p).$$

In the remainder step, terms are added to the remainder and subtracted to p if $LT(p)$ is not divisible by any $LT(f_i)$. Meaning, that the leading term of p strictly decreases, so the number of loops in the while stage are limited and when $p = 0$ the final equation computed is $f = q_1 f_1 + ... + q_n f_n + r$. □

However, even if the above algorithm works up to a point, it is still not enough on its own to solve the IMP. This is because there can be more than one output to the algorithm, depending on the choices made.

Example 2.3.2. *Given a polynomial set $F = \{f_1, f_2\}$, with $f_1 = xy - 1$ and $f_2 = x^2 - 1 \in k[x, y]$. To figure out if $f = xy^2 - x$ belongs to the ideal generated by F, the f needs to be divided by F. If f is firstly divided by f_1 first, and then f_2, the result is:*

$$xy^2 - x = y(xy - 1) + (-x + y),$$

so the remainder is $-x + y$. However, if a different order is attempted, picking f_2 as the first divisor, the answer will be different from the first division order, $xy^2 - x = x(y^2 - 1)$, so the remainder is 0 and therefore, f is in the ideal generated by F.

As stated before, the previous example shows where the division algorithm fails. Even with a fixed monomial order, the remainder is not always the same, given that there is an ambiguous step in the algorithm.

To fix this ambiguity one would need to find a set of polynomials, a basis, where the remainder would always be the same and unique, once a monomial order was fixed. This basis is the Gröbner

basis, a "magical" basis where irrespective of the order by which the divisors were chosen, the outcome does not vary.

2.4 Gröbner Basis

Now there is enough information to define a Gröbner basis.

Definition 2.4.1. *Fix a monomial order on the polynomial ring $k[x_1,...,x_n]$. Given an ideal $I \subseteq k[x_1,...,x_n]$ and consider $LT(I)$, the leading term ideal of I, as the ideal generated by the leading terms of all polynomials in I. A finite subset $G = \{g_1,...,g_m\}$ is called a Gröbner basis if $\langle LT(g_1),...,LT(g_m)\rangle = LT(I)$. [8]*

Putting the previous definition in simpler terms: a set $G = \{g_1,...,g_m\}$ from an ideal I, is a Gröbner basis of I if the leading term of any element of I is divisible by one of the leading terms of G.
This basis has important properties that help solving the Ideal membership Problem.

Proposition 2.4.1. *Consider $I \subseteq k[x_1,...,x_n]$ an ideal and $G = \{g_1,...,g_n\}$ a correspondent Gröbner basis for some fixed monomial ordering. If $f \in k[x_1,...,x_n]$ then there is a remainder, $r \in k[x_1,...,x_n]$, from the division of f by the Gröbner basis. This remainder is unique and follows two properties:*

1. *No term of r is divisible by any of the leading terms from G, $(LT(g_1),...,LT(g_t))$.*
2. *There is a polynomial $g \in I$ such that $f = g + r$*

Proof. Since the division algorithm outputs $f = q_1g_1 + ... + q_sg_s + r$, the first property is already satisfied.
Suppose one can write f in two different ways: $f = g + r$ and $f = g' + r'$ for some $g, g' \in I$ and $r \neq r'$. Then,

$$g + r = g' + r' \iff g - g' = r - r'.$$

Since $g, g' \in I$ then $g - g' = r - r' \in I$, this implies that $LT(r - r') \in \langle LT(I)\rangle = \langle LT(G)\rangle$, meaning that there is some g_i such that $LT(g_i)|LT(r - r')$, which is a contradiction with property (i), so r and r' have to be equal, and therefore, r is unique. □

Example 2.4.1. *Going back to the previous Example 2.1.1, it is now possible to solve the problem. Recalling that the proof of the Circle Theorem of Apollonius was reduced to proving that a polynomial c is an element of the ideal generated by $\{p_1,...,p_8\}$, $I(p_1,...,p_8)$, where these are the polynomials presented in Table 4.1. For the computations in this example* Macaulay2 software*[34] was used.*

In the Macaulay2 software, the Gröbner basis of the ideal $I(p_1,...,p_8)$, regarding the variables $x, y, z, x_1,...,x_8$, was first computed. The result was the following 12 *generators:*

$$\{2x_4 - z, 2x_2 - z, 2x_1 - y, 4x_7y - y^2 - 4x_8z + z^2, x_6y + x_5z - yz,$$
$$x_5y - x_6z, x_3y, x_3^2 - 2x_3x_7, 4x_3x_8z - x_3z^2, 4x_6x_7z - 4x_5x_8z + 2x_5z^2 - yz^2,$$
$$4x_5x_7z + 4x_7x_8z - 2x_6z^2 - 4x_7z^2 + z^3, x_3x_6z, x_5^2z + x_6^2z - x_6z^2, x_3x_5z\}$$

In order to find if c is one of the ideal member, it is necessary to perform the polynomial division of c by the Gröbner basis. With the help of the Macaulay2, the division result interpreted by:

$$c = (-1/2x_1 + x_7 - 1/4y) * (2x_1 - y) + 1/4 * (4x_7y - y2 - 4x_8z + z2).$$

Hence, the remainder is zero, so c is an element of the ideal $(p_1, ..., p_8)$ *and the theorem is proven with the help of Gröbner Bases calculations.*

2.5 Buchberger Algorithm

It turns out that every ideal I in $k[x_1, ..., x_n]$ has a Gröbner basis, as can be seen in [8] in pages 76-81, and therefore, a method to compute them exists. The Buchberger's Algorithm was introduced in the 1970s, and more recent variants of the same algorithm appeared after in [13][14]. The Buchberger's Algorithm receives a generating set as an input and computes a Gröbner basis for the ideal generated by that set.

Firstly a monomial order is fixed, then the algorithm starts with an "approximation" of a Gröbner basis, which is the generating set where more elements are added until the Gröbner basis is completed.

This is achieved by repeatedly producing combinations of the basis elements. Sometimes, when computing this polynomial combinations one might get leading terms that are not in the ideal generated by the leading terms of the original generating set, F. This can occur when computing the linear combination with the leading terms, as they can cancel and present smaller terms. Since it is a linear combination, the smaller terms belong to the ideal generated by the initial basis. This cancellation is studied using S-polynomials (the S stands for subtraction).

The S-polynomial, with the notation $S(f_i, f_j)$, is a necessary tool to find a Gröbner basis. They are used to check if a basis is a Gröbner basis and are also a tool to construct a Gröbner basis.

Definition 2.5.1. *Consider the two polynomials from the initial basis f and g,*

$$S(f,g) = \frac{lcm(LM(f), LM(g))}{LT(f)} f - \frac{lcm(LM(f), LM(g))}{LT(g)} g$$

is the S-polynomial of f and g. Here, $lcm(p,q)$ *stands for the least common multiplier of the monomials p and q, while* $LM(f)$ *is the leading monomial of f, i.e., the leading term without the coefficient.*

In other words, the S-polynomials allow the creation of additional lower-order polynomials in the ideal by reducing the corresponding lead terms.

Example 2.5.1. *Using the lex order, the calculation of the S-polynomial* $S(f_1, f_2)$ *for* $f_1 = x + y$ *and* $f_2 = xy + z$ *goes as follows.*

$$S(f_1, f_2) = \frac{LCM(LT(f_1), LT(f_2))}{LT(f_1)}(f_1) - \frac{LCM(LT(f_1), LT(f_2))}{LT(f_2)}(f_2).$$

This means

$$S(f_1, f_2) = \frac{xy}{x}(x+y) - \frac{xy}{xy}(xy+z) = y^2 - z.$$

In the Buchberger algorithm, the set of generators will grow at each step by calculating the S-polynomials and then dividing them by the previous generator set, adding the remainder to the set. The process repeats itself until all the remainders of the division with the S-polynomials are zero. Meaning that in each step where a S-polynomial remainder is added to the basis then there will be more S-polynomials to be calculated, and more divisions to perform.

This condition comes from a key criterion to define if a given basis is a Gröbner basis, using the S-pairs to discover. This is called the *Buchberger's Criterion* which leads to the algorithm that allows to compute Gröbner basis.

Theorem 2.5.1. *A basis $G = \{g_1, ..., g_n\}$ of a polynomial ideal I is a Gröbner basis if and only if for all pairs $(g_i, g_j), i$, the remainder of the division of $S(g_i, g_j)$ by G is zero.*

Example 2.5.2. *Considering $F = \{f_1, f_2, f_3\}$ with $f_1 = x+y$, $f_2 = xy+z$ and $f_3 = y-z$, and using the results from Example 2.5.1, the first step of Buchberger's Algorithm would be to take a pair, for instance $\{f_1, f_2\}$, take its S-polynomial $S(f_1, f_2) = y^2 - z$, and divide it by F. In this case,*

$$y^2 - z = (0)(x+y) + (0)(xy+z) + (y+z)(y-z) + z^2 - z$$

As seen before, if the remainder is non-zero then the Gröbner basis needs to be updated, by adding $f_4 = z^2 - z$.

Theorem 2.5.2. *Consider an ideal $I = \langle f_1, ..., f_n \rangle \neq 0$, then a Gröbner basis for I can be computed using the following algorithm:*

Algorithm 2 Buchberger's Algorithm

Require: $F = f_1, ..., f_n$
Ensure: a Gröbner basis $G = (g_1, ..., g_m)$ for I, $F \subseteq G$
  $G \leftarrow F$
  $P \leftarrow \{(f_i, f_j) : 1 \leq i < j \leq s\}$ ▷ P is the set of all S-pairs from F
  while $P \neq \emptyset$ **do**
    $G' \leftarrow G$
    $(f_i, f_j) \leftarrow select(P)$
    $P \leftarrow P \setminus (f_i, f_j)$
    $r \leftarrow \overline{S(f_i, f_j)}^{G'}$ ▷ r is a remainder of the division of S-polynomial by G'
    if $r \neq 0$ **then**
      $P \leftarrow update(P, G, r)$
      $G \leftarrow G \cup r$
    end if
  end while

Proof. Following the Buchberger's criterion, the algorithm ends when the remainder of all the division of the pairs in the current G' by G is zero. That is, when all pairs have been considered and there are no more polynomials to be added, the algorithm returns a Gröbner basis.

To prove that the algorithm always terminates the while step is analysed. In every loop repetition, G is set to be equal to the previous G' plus non-zero remainders of the S-polynomials division. Thus having $\langle LT(G') \rangle \subseteq \langle LT(G) \rangle$, since $G' \subseteq G$. Also, if $G' \neq G$ then $\langle LT(G') \rangle$ is strictly smaller. From

successive iterations, a chain of monomial ideals is created and that chain can not stretch infinitely, from the Ascending Chain Condition theorem, in page 80 from [8], so, after a finite number of iterations the algorithm will reach to $\langle LT(G')\rangle = \langle LT(G)\rangle$, which implies that $G' = G$. Therefore, it must terminate. □

As seen, the Buchberger's algorithm begins with a fixed monomial order and a set of polynomials, F, as an input and outputs a Gröbner basis of the ideal generated by the set of polynomials. Certain aspects of it need further consideration.

In each iteration, one picks a pair $(f,g) \in P$: in this step a pair from a set of possible pairs P is chosen. This is called the "pair selection". Since it is choice driven, there are many strategies to select pairs in order to try to minimize runtime. There are some well-known selection strategies that select the pair (f_i, f_j).The following selection strategies are considered:

- First: Among pairs with minimal j, select the one with minimal i. In other words, the last pair to be added to the set is the "biggest".
- Degree: Select the pair with minimal total degree of $lcm(LM(f_i), LM(f_j))$. If needed, break ties with First.
- Normal: Select the pair with $lcm(LM(f_i), LM(f_j))$ minimal in the monomial order. If needed, break ties with First.
- Sugar: If one consider the input homogenized, it selects the pair with minimal sugar degree, which is the degree $lcm(LM(f_i), LM(f_j))$.
- Random: Select an element of the pair set uniformly at random.

Another detail to consider is that the output is a Gröbner basis is often much bigger than needed. When this happens, it is possible to eliminate generators from the basis. This is a result of the following property:

Proposition 2.5.1. *Suppose G is a Gröbner basis of an ideal $I \subseteq k[x_1,...,x_n]$. If $p \in G$ is such that $LT(p) \in \langle LT(G \setminus \{p\})\rangle$, then $G \setminus \{p\}$ is also a Gröbner basis of I.*

This reduction is done by seeing if any leading term of a generator divides the leading term of other generator from G. Also, usually the polynomials are divided so that the leading term has coefficient one. After this process, the final result is a *reduced Gröbner basis*.

Definition 2.5.2. *A reduced Gröbner basis for an ideal $I \subseteq k[x_1,...,x_n]$, is a Gröbner basis G with the following properties:*

- *The leading terms coefficients are equal to one, for all $p \in G$.*
- *For all $p \in G$, $LT(p) \notin \langle (G \setminus \{p\})\rangle$.*

Chapter 3

Machine Learning

The term machine learning was introduced by Arthur Samuel, a pioneer in artificial intelligence [21] and computer gaming [29], in 1959. Before this, the initial mathematical models aiming to map mathematically the thought process and decision making of the human brain were conceived only in 1943 by Walter Pitts [24]. A few years later, a simple model called the perceptron was presented by Frank Rosenblatt in 1957 [28], that learned through data and gradient-based learning rules. From this point on, single hidden-layer networks [32] began to emerge inspired by the previous researches and developed models, machine learning algorithms became more popular and began to be applied in various areas such as cancer diagnose [33], computer gaming [2] among others.

With the Buchberger Algorithm, a Gröbner basis for any ideal is always ensured. However, its performance depends on multiple factors seen before, such as the monomial ordering, selection strategy and the ideals features. The use of machine learning tools allows to better understand the algorithms complexity, learn patterns and make predictions based on generated data, that consist in ideals and their Gröbner bases, and seek improvements. Since for this problem there are no size limits for generated data, only time constraints, it is a good set up for supervised learning. Besides supervised learning there are other machine learning algorithms types, they are *unsupervised learning* and *reinforcement learning*. These differ in the type of data available to train the algorithm and the methods to analyse the training data and evaluate test data. Of the three categories mentioned above, only supervised learning is discussed since is the only one of relevance for this thesis.

3.1 Supervised learning

In general, supervised learning algorithms learn a function using training data composed by input-output pairs. After learning the function that mapped the input to an output, it can be used to map new examples. The model is evaluated in terms of a loss function, although most of the time the user is interested in some score as accuracy or precision. The loss function, that can be seen as some distance from the predicted values and correct values, is a function that is minimized during training. It is used for common problems such as *classification*, where it predicts qualitative outputs, and *regression* problems, where it predicts quantitative outputs.

The input is basically a selection of examples, each example is a collection of observations, *features*, from different cases, such as times or places. For example, if one wants to make a system

for speech recognition then the dataset will be a set of examples with different voice recordings of sentences, and the features could be the amplitude of the sound wave in a specific moment in time. A common way to express the dataset is with a *design matrix*. This matrix consists of a different example in each row, and each column corresponds to a different a feature. The input is denoted by X and the output by Y.

The learning task can be described as: given an input X (where X is a matrix $N \times p$, composed by a set of N p-vectors $x_i, i = 1, ..., N$), find a good estimate for the correct output Y, which is an array $N \times 1$ composed by the values to be predicted, denoted by $\hat{Y}$, with $N \times 1$ dimension. In order to train the algorithm it is necessary to have a considerable amount of data, called *training data*, denoted $(x_i, y_i), i = 1 \ldots N$.

A well-known learning algorithm, at least for mathematicians, is the Linear Regression. It has many practical uses, like forecasting and to understand the relationship between variables.

Linear Regression

The linear regression algorithm is commonly used for predictive analysis and to study the association between continuous variables. Like stated before, the algorithm is given an input, in this case a vector $X \in R^n$, and predict $\hat{Y} \in R$ as an output.

The output of this model will be a linear function that will return $\hat{Y}$, the value wanted to predict: $\hat{Y} = W^T X$, where $W \in R^n$ is a vector of parameters. These parameters are the *weight* that regulates how the features affect the prediction. The way weights affect the prediction is quite straightforward: if x_i gets a positive weight w_i, then it increases the value of the prediction $\hat{y}$, if w_i is negative it will decrease the value of the prediction.

The goal of the algorithm is to find the best fit line for the data, i.e., the error between predicted values and true values should be minimized. The *cost function* measures that error, which also defines the models performance. It optimizes the weights or the regression coefficients until a desired accuracy is reached. The *cost functions* can be designed in different ways to fit different expectations of the output.

Suppose one have a design matrix X for $N \times p$ for evaluating a models performance and a regression vector with the correct values of Y, denoted Y. In order to measure the performance of the model, in *linear regression algorithms*, the *mean squared error (MSE)* cost function is used, which is the average of squared error.

Then the mean squared error is designated by:

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (y_i - w^T x_i)^2.$$

In order to create a good model, it is necessary to improve the weights, for that it is necessary to minimize the mean squared error when working with a training set. The minimization process is done by using the least squares method, where the goal is to minimize the sum of the vertical distance between the true values and Y.

$$\min_w \frac{1}{N} \sum_{i=1}^{N} (y_i - w^T x_i)^2.$$

This problem is solved using matrix multiplication. Considering the linear regression equation $X\overrightarrow{W} = \overrightarrow{Y}$. The parameter estimates can be calculated with the equation:

$$\overrightarrow{W} = (X^T X)^{-1} X^T \overrightarrow{Y}.$$

If there are two or more independent variables and a dependent variable then it is the case of multiple linear regression. The estimated function is

$$Y = w_0 + w_1 x_1 + w_2 x_2 + ... + w_n x_n.$$

Similarly as in the simple linear regression, the goal is to choose the regression coefficients that returns the most accurate possible function and these values are obtained by minimizing the sum of the square errors, using the least squares method.

In this case it is,

$$\overrightarrow{W} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$

and from the linear model is $X\overrightarrow{W} = \overrightarrow{Y}$. The difference is in the number of w and x variables, but there are as many w as there are x. Therefore, the procedure goes similarly as for the linear case.

3.2 Neural Network

Neural network offer an alternative way to approximate an unknown function $f(x)$ that represents some kind of behaviour in nature, for example a time series of climate forecasting for prediction purposes.

A neural network approximates the function through a composition of multiple simple functions. Even though the function that one wishes to approximate might be complicated, there is a theorem, the *Universal Approximation theorem*, that states that, under certain conditions, any continuous function can be estimated by a shallow neural network [11][20]. In other words, a neural network is an universal function approximator, (more about this theorem and its proof can be read in [22][27]) which makes it a versatile tool for any problem solving that can be reduced to functions.

The architecture of a neural network is composed by stacked units, called layers, and there are three types:

- Input layer: which receives the input data and pass it forward to the rest of the network;
- Hidden layer: there can be more than one hidden layer in each network. For example, in the neural network above there are two hidden layers.
- Output layer: predicts our final output.

In order to better understand the mechanics of a neural network, consider the architecture in Figure 3.1 as a graph, more specifically a acyclic directed graph $G = (V, E)$ where each edge has a

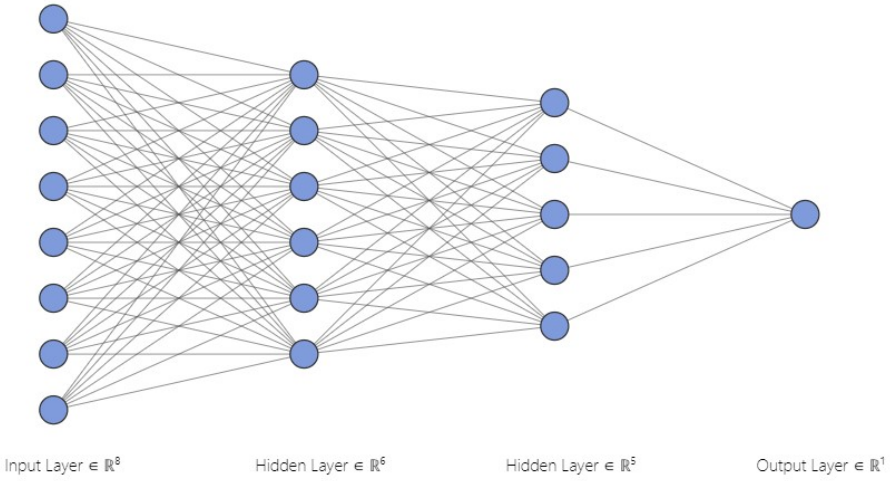


Fig. 3.1 A representation of a neural network

correspondent weight w and each node or vertex a bias b. The first layer has no input edges, this layer receives the input data x_0. The other nodes have input and output edges. Consider V_0 the set of vertices in the input layer and V_I the set of vertices with input edges. It is possible to extend x_0 to $x \in \mathbb{R}^{v_i}$, recursively for every $v \in V_I$, by the rule

$$x_v = \sigma\left(\sum_{(i,v)\in E}(W_{i,v}X_i) - b_v\right),$$

where the function represented by σ is called an activation function and it evaluates the weighted sum of the inputs and defines the vertex output. There are multiple commonly used activation functions [15] that do different types of transformations to their inputs. Notice that, by the Universal Approximation theorem, the activation functions need to be non-polynomial [23]. For example, the *Sigmoidal Function* [11] or the *Rectified Linear Units* (ReLu) [18], which is one of the most common used activation functions. The ReLu can be described very simply, it returns its input if it is positive or zero otherwise, in other words, $\sigma(x) = max(0,x)$.

The output layer, which will be denoted by V_1, is composed by the vertices without leaving edges, those nodes outputs the neural networks result. The function can then be represented by $F_G(x;w,b)$ with $F_G : \mathbb{R}^{V_0} \times \mathbb{R}^p \to \mathbb{R}^{V_l}$, a function that receives the input data x together with parameters (w,b) and outputs some values in the output layer.

For the regression case, the goal is to minimize the loss function

$$L(x,y;w,b) = \frac{1}{2}||y - F(x;w,b)||^2,$$

averaged over out training data. In other words, we want to solve the problem

$$\min_{w,b} \frac{1}{2N}\sum_{i=1}^{N}||y_i - F(x_i;w,b)||^2.$$

In order to do this a gradient-based optimization method is usually used. For large data, this is generally some variant of the stochastic gradient descent method.

In a basic model, at each iteration step, first a batch of k data points (x_i, y_i) is picked and the loss function considered is $L = \frac{1}{2k}\sum_{i=1}^{k} ||y_i - F(x_i; w, b)||^2$, and the weight update is made using the formula

$$w_l = w_l - \eta \frac{\partial L}{\partial w_i},$$

where η represents the learning rate, this is, the size of the iteration step, which is, consequently, an important parameter when training neural networks.

Training a neural network is therefore basically updating the weights w, b for each edge and vertex as new data is fed to the network. The graphical structure of this type of function allows to compute these updates very swiftly via what is called back-propagation.

3.2.1 Recurrent Neural Network

A recurrent neural network (RNN) can be described as an extension of a mainstream neural network model that handles sequenced and variable sized data and generate sequences. Another valuable characteristic of RNN is the concept of "memory" which stores past input data useful for generating the next output. They have multiple applications in various fields, such as video [30], music [3][12] and text [31].

Consider a sequence $x_1, x_2 ...$ of any size. A recurrent neural network processes each x_t sequentially at a time t. It consists of hidden states h that take in consideration the sequence data input and the previous state and outputs o_t. A simple way to describe the process is, for a time step t, a hidden state h_t is updated by

$$h_t = \sigma(f(h_{t-1}, x_t; w_t, b_t)),$$

where σ is an activation function [7].

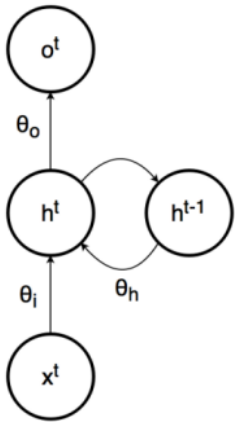


Fig. 3.2 A representation of an hidden layer in a RNN. [25]

The recurrent name comes from the repetition of the same task over all data sequence. Unfortunately, in the RNNs, long sequences might create a numerical issue, which is called the exploding or vanishing gradient problem. This problem happens when the derivatives get increasingly larger or

grown smaller and smaller, this affects the weights update step where either it does not converge or the weights updates are too small. This causes the possibility of the network losing important information. In order to deal with this problem, *Long Short-term Memory* (LSTM) and *Gated Recurrent Units* (GRU) were introduced into RNNs. They implement a gating mechanism that allows to regulate the information's flow by deciding how much past data it is not important and can be forgotten. In this report it is only of relevance to discuss GRUs, but one can read about LSTMs in [15][6].

The GRUs uses two gates: an update gate and a reset gate, the gates act as a threshold and together they choose what information is important enough to go through. To understand how this works first consider an update gate in a time step t, defined by the formula

$$z_t = \sigma(w_Z x_t + u_z h_{t-1} + b_Z),$$

where x_t is the input vector, h_{t-1} the previous hidden state and w_Z, u_Z, b are two weights and bias in vector format. The function proceeds to add the multiplication of the input data by its own weight w_Z and the h_{t-i} also multiplied by also its own weight, and then applies a sigmoid activation function which will return a value between zero and one. Then, there is the reset gate that informs the model of the amount of information to forget. It is calculated by

$$r_t = \sigma(w_R x_t + u_R h_{t-1} + b_R),$$

it works similarly to the update formula, the only changes are the weights and the gates usage. To know exactly the amount of information to forget first it is created a hidden state candidate

$$\hat{h_t} = tanh(w_h x_t + (r_t \circ h_{t-1}) + b_h)$$

where $\circ$ represents the elementwise product operator, in this case the output will be between $(-1,1)$. This function was initially designed with *tanh* function, but other functions can be used. The impact of the reset gate can be analysed from the elementwise multiplication where if it is close to zero then the result will not depend on the previous layer, behaving like a single hidden layer network. If it is close to one than it will behave like a plain RNN. The final update takes in the update gate that will determine how much of the new candidate hidden state is used, represented by the equation:

$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \hat{h_t}.$$

In this case, if z_t is close to zero then the updated hidden state will approach the candidate hidden state. Otherwise, if it is close to one then it will approach the majority of the previous information and discard most of the current information. By doing these four steps recursively between hidden states, a GRU is able to store relevant information, passing it through the other layers, for each data point consisting in a sequence x_i and an output y_i.

Chapter 4

Literature Review

This Chapter presents the contents of the article [26], from which the work in this thesis is derived. Since this paper uses binomial and toric ideals as numerical examples for the Buchberger algorithm, before introducing the results of the paper, it is necessary to introduce the background on those types of ideals.

4.1 Binomial Ideals and Toric Ideals

Definition 4.1.1. *Let k be a field and $R = k[x_1,...,x_n]$ a polynomial ring. A binomial in R is a polynomial with the format $u - v$, where u, v are monomials in R.*

A binomial ideal is an ideal of R generated by a finite set of binomials.

Within the binomial ideals, there are a class of binomials that respect certain properties called *toric ideals*.

Definition 4.1.2. *Consider the matrix $A = [a_1,...,a_m] \subseteq \mathbb{Z}^n$ with non-zero columns and consider the group homomorphism $\psi : \mathbb{Z}^n \to \mathbb{Z}^m$ determined by A. For example, given $v = (v_1,...,v_m) \in \mathbb{Z}^m$ then, $\psi(v) = v_1a_1 + ... + v_ma_m \in \mathbb{Z}^n$. Each column of A, $i = 1,...,n$ forms a monomial in the format $t^{a_i} = t_1^{a_{i1}}...t_n^{a_{in}}$.*

Now, consider the following polynomial rings over a field K, $K[x_1,...,x_m]$ and $K[t_1^{\pm 1},...,t_n^{\pm 1}]$. The toric ideal I_A associated to A is the kernel of the homomorphism ϕ:

$$\phi : K[x_1,...,x_m] \to K[t_1,...,t_n]$$

with $\phi(x_i) = t^{a_i} = t_1^{a_{i1}}t_2^{a_{i2}}...t_n^{a_{in}}$.

The monomial degrees are such that $deg(t_j) = 1, j = 1,...,m$ and $deg(t^{a_i}) = a_{i1} + a_{i2} + ... + a_{im}, i = 1,...,n$. This allows saving the degrees in the format $\tau = (\tau_1,...,\tau_m)$, where $\tau_i = deg(t^{ai}) = a_{i1} + a_{i2} + ... + a_{im}$.

For $\alpha = (\alpha_1,...,\alpha_m) \in \mathbb{Z}^m$ and $x^\alpha = x_1^{\alpha_1}...x_m^{\alpha_m}$ then $deg(x^\alpha) = x_1\alpha_1 + ... + x_m\alpha_m \in \mathbb{N}A$. Note that $\phi(x^\alpha) = t^{A\alpha)}$ hence $x^\alpha - x^\beta$ belongs to $I_A = ker(\phi)$ if and only if $\alpha - \beta$ belongs to $ker(\phi)$. In fact one can prove that the toric ideal associated to A is generated by:

$$x^u - x^v | u, v \in \mathbb{Z}^n, Au = Av$$

A toric ideal I can also be defined as a binomial prime ideal, where $I \subset K[x_1,...,x_n]$, the proof of this theorem can be seen in [19]. In the following example, a toric ideal is calculated given a monomial set.

Example 4.1.1. *Consider the monomial set $\{x^2y, xy^2, x^2z, xz^2, y^2z, yz^2\}$, what is the ideal toric associated to the set? The following matrix corresponds to the degrees of each monomial:*

$$A = \begin{bmatrix} 2 & 1 & 2 & 1 & 0 & 0 \\ 1 & 2 & 0 & 0 & 2 & 1 \\ 0 & 0 & 1 & 2 & 1 & 2 \end{bmatrix}$$

The toric ideal I_A is the kernel of the homomorphism ϕ:

$$\phi : K[x_1,x_2,x_3,x_4,x_5,x_6] \to K[t_1,t_2,t_3]$$

Where, for each x_i the function returns t^{a_i}, this is

$$x_1 \to t_1^2t_2; \quad x_2 \to t_1t_2^2; \quad x_3 \to t_1^2t_3; \quad x_4 \to t_1t_3^2; \quad x_5 \to t_2^2t_3; \quad x_6 \to t_2t_3^2.$$

Therefore, the toric ideal is: $I_A = (x_1x_6 - x_2x_4, x_1x_6 - x_3x_5, x_2^2x_3 - x_1^2x_5, x_2x_3^2 - x_1^2x_4, x_1x_5^2 - x_2^2x_6, x_1x_4^2 - x_3^2x_6, x_4^2x_5 - x_3x_6^2, x_1x_4x_5 - x_2x_3x_6, x_4x_5^2 - x_2x_6^2)$.

4.2 Learning a performance metric of Buchberger's algorithm

This section describes the contents of the article [26], on which this thesis is based.

When computing a Gröbner basis of a polynomial ideal with the Buchberger algorithm, the runtime of the algorithm can be evaluated by the number of additions of polynomials to the basis. Each addition enlarges the set of generators to be considered in the following steps. The goal of this article was to see if it is possible to predict the number of additions that occur in one run of the algorithm. With this information one would be able to understand and predict when and why the computations of Gröbner bases are hard.

The question this article wants to answer is: *For a given ideal I, how many polynomial additions are performed during one run of Buchberger's algorithm to compute a Gröbner basis of I?* Several prediction methods are employed to achieve this, including a multiple linear regression model and a recursive neural network with a gated recurrent unit.

For the numerical tests, two families of ideals are considered: binomial and toric ideals.

The Data

The question above was studied for binomial ideals of three unknown variables and generated by sets picked randomly with the following fixed parameters:

- 4 binomials of degree up to 20;
- 10 binomials of degree up to 20.

For toric ideals, four different distributions were considered, in eight variables. The generation of the toric ideal samples were with the following parameters with a non-negative integer matrices $A \in \mathbb{Z}^{D\times 8}$ format:

- $D = 2$ rows and integer entries up to 5;
- $D = 4$ rows and integer entries up to 5;
- $D = 6$ rows and integer entries up to 5;
- $D = 6$ rows and integer entries up to 10.

In the toric case, the number of generators is not given. The binomial instances are generated by polynomials, these are created fixing three parameters: the number of variables (denoted n), the maximum degree of a monomial in the polynomials (denoted d) and the number of polynomials generators of the ideals (denoted s). The binomials are constructed by sampling two monomials, of n variables and degree equal or less than d. The monomial sampling have two distributions: uniform and weighted. The uniform distribution samples a pair of monomials uniformly at random from the monomial set. The weighted distribution first picks a degree from one to d uniformly, and then from the monomial set with the chosen degree, it samples uniformly at random each monomial. The difference between the two distributions is that with the weighted distribution the sample has more binomials of low total degree, comparing to the uniform distribution sample, which has a mix but with more high total degrees.

As seen before, the toric ideal is the kernel of the monomial map φ, so that is exactly the way the toric ideals sample is generated. Firstly, a random monomial map is generated using a Macaulay2 function. This toric model was extended in order to allow negative exponents for the generation of the A matrix and it receives 4 fixed parameters: D the number of the lines (that corresponds to the number of variables), a minimum L (bound on the negative total) and maximum U (bound on the positive total degree) for the sum of each column of the matrix, and n as the number of columns, which represents the number of monomials.

The format of each distribution of binomial and toric ideals is denoted by n-d-s-uniform/weighted and $\tau(D,L,U,n)$, correspondingly.

The data was generated in the following formats described in table 3.1.

To study the diversity of the dataset many features were computed. For binomial ideals, the features are the minimum generator degree, mean generator degree, maximum degree, Krull dimension (length of the longest chain of ideals in the partially ordered set of prime ideals), regularity and number of pure powers. Where regularity studies relations between the generators and number of pure powers being the number of monomials that are a power of a single variable. In the toric ideals case, the same features were analysed except for number of pure powers and substituted by the number of ideal generators. Some of the features of the distributions are expensive to compute but they show the diversity of the dataset.

An important step is studying the Buchberger's algorithm performance with the dataset. From the Bruno Buchberger's thesis [4], one can see that there is a high variance in difficulty in many distributions, which shows to be a challenge for reinforcement learning training. For example,

Table 4.1 Characterizations of normal and extremely disconnected spaces

Model	Type of ideals	Sample size
3-20-10-weighted	binomial	1,000,000.
3-20-10-uniform	binomial	1,000,000.
3-20-4-weighted	binomial	1,000,000.
3-20-4-uniform	binomial	1,000,000.
$\tau(2,0,5,8)$	toric	429,093
$\tau(4,0,5,8)$	toric	314,688
$\tau(6,0,5,8)$	toric	325,927
$\tau(6,0,10,8)$	toric	151,532

analysing the number of polynomial additions, its distributions have high variance. But in some cases, such as S-pair selection, it displays improvements between the strategies.

Linear Regression

In order to predict the number of polynomial additions in Buchberger's algorithm, simple and multiple linear regression models were used, working with the data distributions in 4.1. To do this, firstly it is needed to find the function of *predictor variable*. This function is the one that computes the weight that certain features have in the output. This means, that it is necessary to fix the predictor variables first. Starting with the basic, for the binomial ideals they are: maximum, minimum, mean, standard deviation of generator degrees and the number of pure power leading terms of generators. For the toric ideals, they share mostly the same predictor variables with binomial ideals, except for the number of pure powers and the addition on number of generators (since it is not fixed for toric ideals). When a linear model is properly fit, one is able to collect the coefficients correspondent to the predictor variables.

	MaxDeg	MinDeg	MeanDeg	StdDeg	PurePowers	Deg	Dim	Reg	NumGens
3-20-10-w	-0.34	6.84	11.69	-9.75	-1.39	-0.03	-4.11	0.19	-
3-20-10-u	-1.69	-0.79	10.65	2.27	1.42	-0.03	-10.13	0.80	-
3-20-4-w	2.05	3.13	9.15	-3.14	-12.14	-0.18	-21.91	0.16	-
3-20-4-u	1.84	-0.02	6.21	-0.52	-2.82	-0.18	-30.79	1.65	-
T(2-0-5-8)	8.85	-30.39	147.44	-71.75	-	3.70	-	30.34	6.08
T(4-0-5-8)	-15.81	-66.45	76.58	-10.92	-	-1.02	-263.08	13.49	15.98
T(6-0-5-8)	-0.88	-0.55	-1.13	1.97	-	0.06	-52.76	0.44	8.57
T(6-0-10-8)	-0.16	-0.10	0.11	0.17	-	-0.00	-384.88	0.08	4.33

Table 4.2 Summary of regression analyses: coefficients of various predictors in multiple linear regression for each ideal dataset.

	3-20-10-w	3-20-10-u	3-20-4-w	3-20-4-u	T(2-0-5-8)	T(4-0-5-8)	T(6-0-5-8)	T(6-0-10-8)
MMMSDDeg	0.248	0.017	0.169	0.029	0.226	0.278	0.129	0.572
Regularity	0.041	0.010	0.035	0.036	0.264	0.156	0.051	0.148
Dimension	0.010	0.003	0.000	0.014	0.000	0.000	0.214	0.018
Degree	0.005	0.000	0.006	0.005	0.094	0.059	0.020	0.003

Table 4.3 Summary of regression analyses: R^2 statistics in multiple linear regression models for each ideal dataset.

The Table 4.2 shows a summary of how the linear model performs based how the predictors, meaning, how well they estimated the number of polynomial additions . Between them, the best predictors are chosen by analysing the correlation between the number of polynomial additions and possible predictors. In general, the predictors do not always have a stable correlation. It was observed that almost all of the predictors are statistically significant, with the exception of the minimum degree as it can be seen in the distribution for 3-20-4 uniform. This table is not enough to define which are the best predictors for polynomial additions.

More complex predictors are also analysed: Krull dimension, degree and Castelnuovo-Mumford regularity. These are costly to compute but they offer important information about the Gröbner basis complexity. The regression models were trained using 90% of each dataset and the remaining 10% was used to test the models predictions regarding the number of polynomial additions.

Learning via recursive neural networks

Recursive neural networks, denoted by RNNs, are a type of deep learning network. They are used to recognize patterns and structures within the data, learned from the past, and processes new data from experience.

Using RNNs, one is able to compare the machine learning method to the *naive human "guess"* and the regression model computed before. The naive human model is denoted as *uninformed* model, where only the mean value of the polynomial additions are computed from the training set and simply guess the mean value for every input. Then, a *linear* model is built using the features seen previously, for binomial ideals data are the maximum, minimum, mean, standard deviation of generator degrees

and the number of pure power leading terms of generators, and for the toric ideals are mostly the same predictor variables except for the number of pure powers and the addition on number of generators. Lastly, the *recursive network* algorithm is used to find a more suitable model for the problem.

Conclusion

In order to train the network, a matrix composed by vectors which entries are the exponents of the terms in each generator, which is a matrix of integers with the size of 10×6. The network does not receive its features, like in the linear models, it learns by itself.

	Mean Squared Error	Mean Absolute Error	R^2
Uninformed	2555058	39.46	0.0000
Linear	1946.40	33.71	0.2384
RNN	1499.37	29.25	0.4133

Table 4.4 Trained model performances on the dataset 3-20-10-weighted with uniformed model, linear regression model and recursive neural network model.

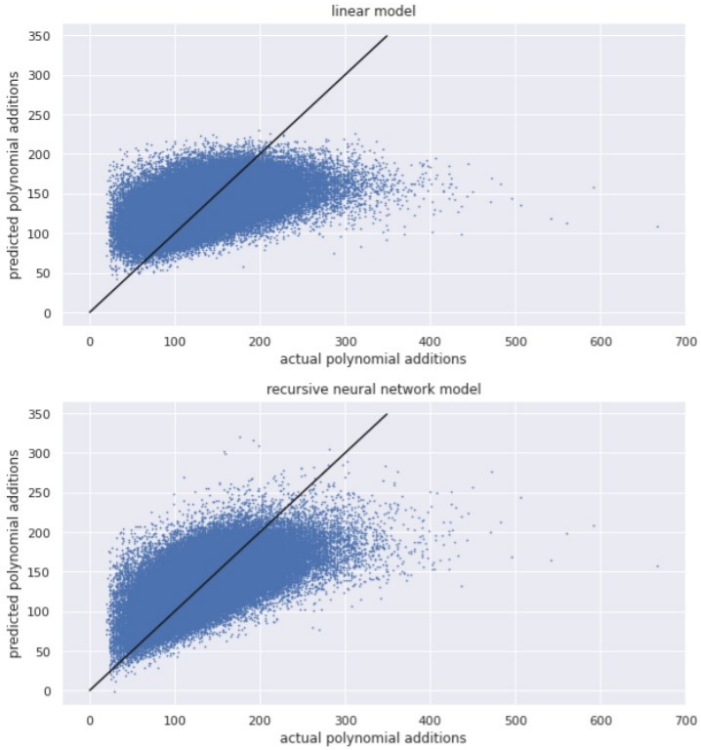


Fig. 4.1 Predicted versus actual polynomial additions in 3-20-10-weighted dataset.

Analysing the Table 4.4, one can conclude that the RNN model has a better performance compared to the other models in the binomial case. This shows that, for the binomial problem, some properties can be learned using machine learning methods.
In Figure 4.1, it is noticeable that both models struggle to predict polynomial additions in extreme values. Nevertheless, the RNN model still shows improvement over the uninformed model and linear

model, when looking at the 3-20-10-weighted distribution. In the following Table 4.5, it is presented the summary of other distributions trained in RNN models. This shows that the learning models made some improvements compared to the regression model, comparing to the Table 4.2. But, due to input format restrictions and the fact that RNNs models performed poorly when considering the generalization of other distributions, the predictions on others distributions weren't evaluated.

Train/Test data	3-20-10-w	3-20-10-u	3-20-4-w	3-20-4-u
3-20-10-w	0.41	0.06	-4.58	-6.36
3-20-10-u	0.23	0.13	-5.06	-6.62
3-20-4-w	0.06	-0.63	0.37	0.19
3-20-4-u	0.03	-0.36	0.29	0.22

Train/Test data	T(2-0-5-8)	T(4-0-5-8)	T(6-0-5-8)	T(6-0-10-8)
T(2-0-5-8)	0.39	-0.04	-15.43	-0.32
T(4-0-5-8)	-2.55	0.15	-78.20	-3.03
T(6-0-5-8)	-2.63	-0.00	0.65	0.04
T(6-0-10-8)	-3.31	-0.07	0.05	0.52

Table 4.5 Summary of neural network predictions: R^2 with different training and test datasets.

The goal was to prove that the Buchberger's algorithm performance metric can be predicted from data and that can be modeled as a machine learning problem. By evaluating linear regression models and recursive network models in random samples of binomial and toric ideal in different distributions, it was able to create an efficient approach that models polynomial additions properly. This model can be used as a reference for future work with RNNs.

4.3 Contributions of this thesis

As mentioned before, this thesis builds from the article reviewed in this chapter. It replicates the work done in the paper and explores some new possible paths of progress. More concretely it does the following:

- The results in [26] use some unspecified S-pair selection strategy. In order to see if this omission is justified, tests were conducted to measure the impact of the selection strategy on the predictability of the performance of the Buchberger's algorithm, and to identify the strategy used in the paper.

- A database was constructed with very large datasets, mimicking the datasets used in [26] but with a different selection strategy. This gives a playground to test different approaches, and allows to complement the results obtained in that paper with similar results obtained for a different strategy, for comparison purposes.

- The same linear regression tests as in [26] are performed in this new datasets, providing some insight on the robustness of the results obtained there. Moreover, some different features are introduced, to show that the linear approach can be further improved.
- Finally, neural networks are applied to the datasets. To do that the network models are chosen, and numerical experiments are run in order to fine tune the parameters. The results obtained complement the ones in the original paper and provide further insights on the learnability of the Buchberger's performances under several different data models.

Chapter 5

Data Generation

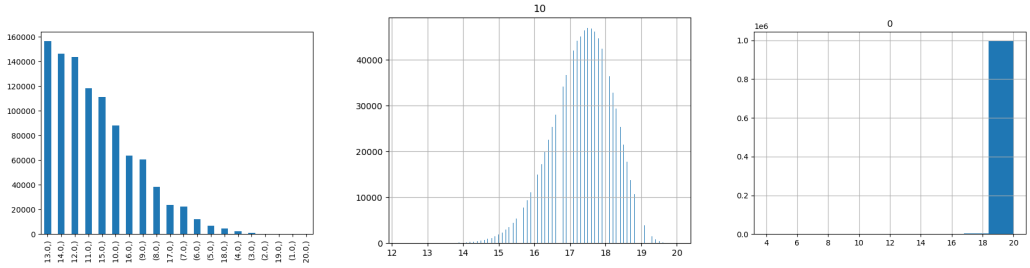
5.1 Dataset Structure

Machine learning algorithms depend heavily on data, since it needs to meet the requirements of each machine learning algorithm. Having a dataset structure helps having a clear vision of the underlying problem and a better performance. The dataset, composed by binomial and toric ideals, was generated using the software system Macaulay2 [16]. The ideals distributions 5.1 are defined similarly to [26] dataset distributions. The first four samples are binomial ideals, denoted by n-d-s-uniform/weighted, where n is the number of variables, d denotes the maximum degree of a monomial in the ideal generators and s is the number of each ideal polynomial generators. The last parameter refers to the polynomials generator monomials sampling, where the uniform distribution first samples a pair of monomials uniformly and randomly from a monomial set. Meanwhile, weighted distribution first randomly chooses a degree from one to d and creates a monomial set of the chosen degree, then it selects uniformly at random a pair of monomials from that set. The last four distributions are toric ideal samples. The used format to identify the distribution is $\tau(D,L,U,n)$, where D denotes the number of variables, L is the negative bound of the total degree, U is the positive bound of the total degree and n is the number of monomials. This was achieved by using the $Ideals.m2$ and $SelectionStrategies.m2$ packages created by Dylan Peifer. The ideals are generated using $Ideals.m2$ with the graded-reverse-lexicographic monomial ordering, by defining the ideal type and parameters, and the Gröbner basis of the ideals are calculated using the Buchberger algorithm from $SelectionStrategies.m2$, where the selection strategy is defined, and it returns the Gröbner basis and correspondent polynomial additions. The files can be found in the code repository of Dylan Peifer's Github account, https://github.com/dylanpeifer/deepgroebner.git. This script was created by modifying the $make\_stats.m2$ script in order to save additional information such as the ideals exponents, degree, dimension and regularity, which will be described ahead. The dataset is composed by one million of each ideal distribution:

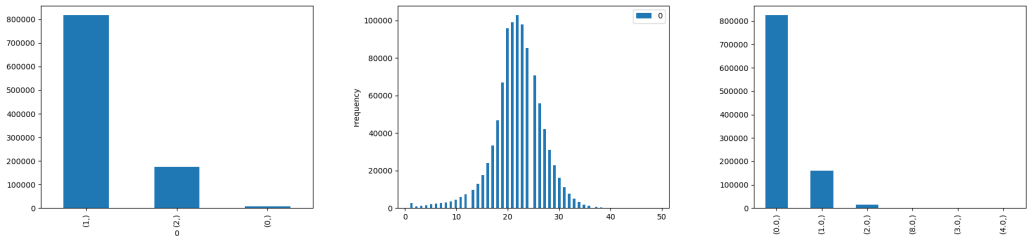
Model	Ideal Type	Sample size
3-20-10-uniform	binomial	1 000 000.
3-20-10-weighted	binomial	1 000 000
3-20-4-uniform	binomial	1 000 000
3-20-4-weighted	binomial	1 000 000
2-0-5-8	toric	1 000 000
4-0-5-8	toric	1 000 000
6-0-5-8	toric	1 000 000
6-0-10-8	toric	1 000 000

Table 5.1 Dataset distributions and sample size.

Some invariants of both binomial model 3-20-10 and toric model T(6-0-5-8) are illustrated in the Figures 5.1 and 5.2. The algebraic invariants used are, for the binomial samples, maximum generator degree, minimum generator degree, mean generator degree, ideal degree, Krull dimension, regularity and the number of pure powers, i.e, the leading term of a generator is a pure power of only one variable. As for the toric samples, the invariants are the same with the exception of the pure powers, which in this case is replaced by the number of generators of the ideal. The Krull dimension refers to the supremum of all the heights of the prime ideals from a polynomial ring R. Where if the polynomial ring R is written as a chain of prime ideals $p_0 \in p_1 \in ... \in p_n$ then its Krull dimension is equal to n, or, in other words, it is the dimension of the solution set. As for the regularity, or Castelnuovo-Mumford regularity, this parameter is a good complexity measurer for the Gröbner basis computation. The resemblance with the models used in [26] is evident, therefor the data see eye to eye as the one in [26].

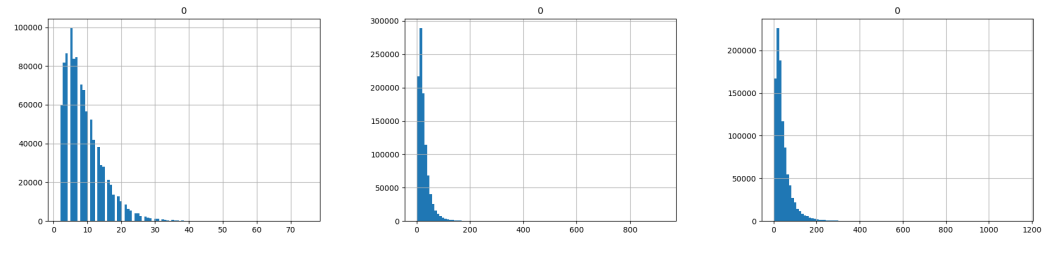


(a) Minimum generator degree (b) Mean generator degree (c) Maximum generator degree

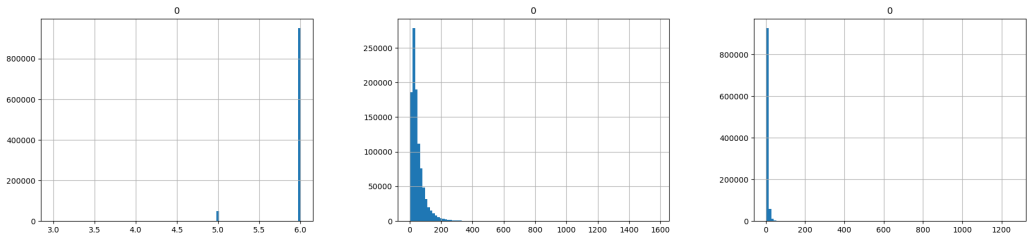


(d) Krull dimension (e) Regularity (f) Number of pure powers

Fig. 5.1 Histograms of some features from the 3-20-10-uniform from binomial ideals.



(a) Minimum generator degree (b) Mean generator degree (c) Maximum generator degree



(d) Krull dimension (e) Regularity (f) Number of ideal generators

Fig. 5.2 Histograms of some features from the toric-6-0-5-8 from toric ideals.

5.1.1 Selection Strategy evaluation

To generate the random samples of the ideal models some parameters need to be fixed, these are the monomial ordering and a selection strategy. The monomial ordering used is the default ordering in *Macaulay2*, the graded-reverse-lexicographic. Given that the selection strategy for the datasets used to study the prediction of polynomial additions was not mentioned in the article [26], three distributions were chosen, with a sample size of one hundred thousand, and computed the Gröbner basis of each ideals using four different selection strategies: Sugar, First, Normal and Degree. As previously described in 2.5, these datasets are useful for studying the behaviour of polynomial additions in different distributions. They will be later used to guess the selection strategy used for the generation of the datasets in 4.1.

Ideal Distributions	Sugar		First		Degree		Normal	
	mean	SD	mean	SD	mean	SD	mean	SD
3-20-10-weighted	159.14	66.12	187.34	74.01	135.50	50.73	135.61	51.40
3-20-10-uniform	265.40	88.15	352.84	117.14	196.60	56.89	197.22	57.85
toric-2-0-5-8	385.30	154.93	392.95	160.60	398.05	167.32	441.64	207.27

Table 5.2 Mean and standard deviation of the number of polynomial additions for two binomial distributions and a toric distributions of the same samples of 100 000 ideals. The table entries format are: mean standard deviation.

In the previous table, it is possible to compare the different impact that the selection strategies have when computing the Gröbner basis on the same samples. In the 3-20-10-weighted the "worst" selection strategy is First performing with the most polynomial additions and the best is the Normal selection strategy, very close to the Degree performance. The same happens with the 3-20-10-uniform distribution but in the toric case the best performing selection strategy is Sugar, while Normal is the worse.

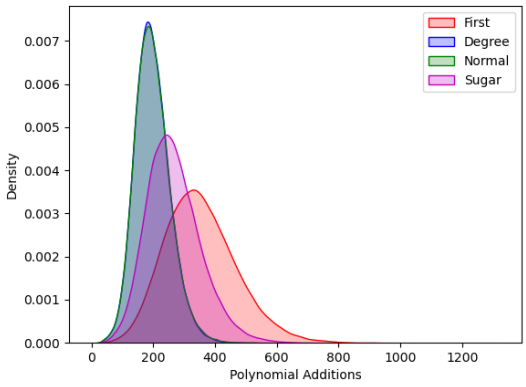


Fig. 5.3 Density estimations of number of polynomial additions in 3-20-10-uniform model with 100 000 samples.

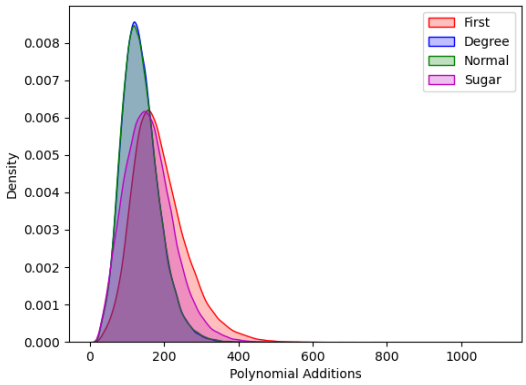


Fig. 5.4 Density estimations of number of polynomial additions in 3-20-10-weighted model with 100 000 samples.

Chapter 6

Linear Regression and Neural Network Models

6.1 Polynomial additions prediction by linear regression models

Before using complicated methods for polynomial addition prediction, one should first try simpler prediction methods such as linear regression. The goal is to predict the response variable which in this case is the number of polynomial additions when a Gröbner basis is calculated. The predictors used are features of the degrees from the generators, that are easy to calculate. Picking the maximum, minimum, mean and standard deviation seem like reasonable options for predictions. In addition, pure powers for binomial datasets is an easy statistic to calculate and the number of generators for toric ideals also have potential of being a predictor since it is not fixed, unlike binomial datasets. Each dataset used contains one million ideals and correspondent Gröbner basis samples. They are split into training and test sets, in this case, 90% of the data is for the training process, and 10% is for the test set.

The multiple linear regressions are performed by using Scikit-Learn [5], a Python library, which has the *LinearRegression* class. This method fits the data into a linear model and returns the regression coefficients. After this step, the model performance must be analysed, this is done by making predictions using the test data and see how accurate the algorithm is. Evaluation metrics used were mean absolute error, means squared error, root mean squared error and R^2.

6.1.1 Application

In the article [26], it is not clear what selection strategy for S-pairs was used in the Buchberger algorithm. In an attempt to discover the selection strategy used for the generation of the datasets, and to verify the consistency of the data, multiple linear regressions were performed. Using the same regression models but with three different datasets, 3-20-10-weighted, 3-20-10-uniform and toric-2-0-5-8, each with one hundred thousand samples.

The Gröbner basis were computed using four selection strategies: Sugar, First, Degree and Normal, and the number of polynomial addititions were recorded for each. The results, presented in the tables 6.1 and 6.2, are R^2 statistics for the multi-linear fit with the four selection strategies for each dataset

and four linear regression models that vary in the choice of the considered parameters: MMMSDDeg model which is built with multiple generator degrees (maximum degree, minimum degree, mean degree, standard deviation degree), and the remaining three models are built with regularity, dimension and degree.

Ideal Distributions	3-20-10-weighted				3-20-10-uniform			
	Sugar	First	Degree	Normal	Sugar	First	Degree	Normal
MMMSDDeg	19.02	22.48	23.56	23.98	5.69	9.54	1.64	1.62
Regularity	4.42	7.55	3.95	3.91	2.53	4.15	1.02	0.97
Dimension	0.67	1.06	0.96	1.01	0.41	0.49	0.35	0.38
Degree	0.60	0.98	0.43	0.41	0.07	0.16	0.08	0.07

Table 6.1 Summary of fitting the number of polynomial additions with four linear regression models in two binomial ideal datasets.

Ideal Distributions	toric-2-0-5-8			
	Sugar	First	Degree	Normal
MMMSDDeg	15.81	14.4	21.7	30.16
Regularity	28.34	28.68	26.23	19.54
Dimension	-0.0	-0.0	-0.0	-0.0
Degree	8.50	9.22	9.38	8.5

Table 6.2 Summary of fitting the number of polynomial additions with four linear regression models in a toric ideal dataset.

Between the binomial models, comparing both multiple and linear regression results with 4.3 from the article [26], the ones that resemble the most to ours correspond to the selection strategies Degree and Normal. The toric model results settles between both selection strategies, as the Degree strategy R^2 differs significantly from the Normal one, and approximates that of the article [26].

With the purpose of confirming this claim, an additional sample of 3-20-10-weighted was generated, in order to see the prediction behaviour with one million samples. The results maintain for the linear regression models. Having established that the original results from the article [26] were generated using the Degree strategy, the datasets generated for our research were generated using the selection strategy Sugar in order to study the selection strategy impact on the models.

When calculating the maximum, minimum, mean and standard deviation degree, two degree interpretations were used, which will be denoted by absolute difference degree (AbD) and polynomial degree. In the AbD degree, the degree considered was the sum of the absolute values of the exponents of the quotient between the two monomials. In other words, it is the sum of the entry wise absolute values of the difference between the vectors of exponents of each monomial. For example, if two monomials exponents were $(1,0,0,1),(0,2,0,0)$, then the subtraction result, considering the absolute values, is, $(1,2,0,1)$, and therefore, the polynomials degree is equal to 4. The polynomial degree is

just the usual degree: it computes the sum of the exponents from both monomials, and chooses the maximum value.

The following table exhibit the R^2 results from fitting the number of polynomial additions using four different linear regressions models: MMMSDDeg model is built with multiple generator degrees(maximum Degree, minimum degree, mean degree, standard deviation degree), where these features were calculated with AbD, degree and both degrees, and the remaining three models are built with regularity, dimension, degree, pure powers in binomial ideals and number of generators in toric ideals respectively. The datasets used for this fitting were all the datasets samples described in 5.1.

	3-20-10-w	3-20-10-u	3-20-4-w	3-20-4-u
MMMSDAbD	13.60	0.98	21.56	5.09
MMMSDDeg	19.07	1.37	23.58	1.86
MMMSDDeg+AbD	19.87	1.83	24.54	5.59
Regularity	4.46	2.52	8.95	5.29
Dimension	0.66	0.42	0.01	1.64
Degree	0.62	0.09	1.34	0.67
Pure Powers	1.15	0.01	0.98	0.0
Number of generators	-	-	-	-

	T(2-0-5-8)	T(4-0-5-8)	T(6-0-5-8)	T(6-0-10-8)
MMMSDAbD	16.9	25.57	5.07	15.27
MMMSDDeg	13.61	31.48	5.14	15.24
MMMSDDeg+AbD	17.78	32.59	5.24	15.31
Regularity	29.22	17.52	5.18	9.99
Dimension	-0.0	0.01	20.71	1.88
Degree	8.43	6.31	2.02	0.23
Pure Powers	-	-	-	-
Number of generators	4.38	27.65	19.54	15.26

Table 6.3 Summary of fitting the number of polynomial additions with four linear regression models, representing the corresponding R^2 statistics.

Analyzing the results of 6.3, and comparing the R^2 results of the binomial samples, there is an evident contrast between uniform and weighted distribution, where the samples with uniform distribution perform worse when compared to weighted. The ideals with a uniform monomial sampling distribution show less prospect for polynomial additions prediction. Also, the number of generators proves to have an impact in the prediction, where fewer ideal generators, in this case 4, shows better results than ideals generated with 10 generators. With toric ideals, toric-6-0-5-8 presents lower R^2 values for most of the regression models, with the exception of the linear regression model with Dimension feature.

Focusing on the variants MMMSD model with two different generator degree calculation types, between absolute difference degree and polynomial degree there is not one that stands out in the

eight ideal samples. However, when fitting the number of polynomial additions with the multiple linear regression model MMMSDDeg+AbD, there is a general improvement. Demonstrating how the regression models can improve with more features, in this case, using maximum Degree, minimum degree, mean degree, standard deviation degree with different ideal generator degree calculation. It also suggest that the linear regression approach used here is not optimal, and can still be improved by adding well-crafted predictors to the existing models.

Pure powers do not show to be good predictors for binomial ideals. The linear regression model using the number of generators have better results for toric-6-0-5-8, but low R squared in general for toric ideals.

As mentioned above, one could try to derive other features from the data, that better reflect the hardness of the problem, in order to get better prediction values. However, that is a time consuming task, that demands a profound knowledge of the algorithms and ideals that are working with. In the next section it is possible to see how the results can be improved automatically with recourse to machine learning.

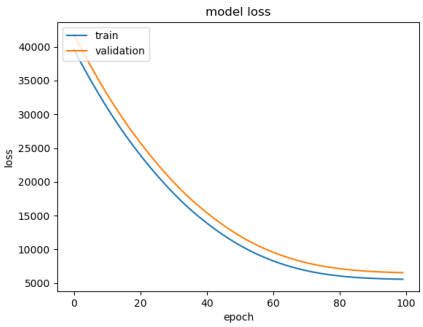
6.2 Neural Networks

With the aim to improve the prediction regression models results, it is necessary to use more sophisticated tools. In this section, two machine learn prediction models are used in order to study if the prediction of polynomials additions are learnable. The training of the neural network models are performed using Keras, the application programming interface from Tensorflow [1], an open-source software library for implementation of machine learning and artificial intelligence. The model is built according to the wanted architecture, and then it iterates the training data in batches. After the training, the model is evaluated with the test dataset, which is 10% of the dataset. In order to evaluate the model, the performance metrics used are mean absolute error, means squared error and R^2. As initial research, the neural network models used in [26] for polynomial addition prediction were implemented. They are a simple neural network model and a recursive neural network model. The first model is a neural network with two hidden layers with either 128 units or 300 units. The activation function of the nodes are *ReLU* and the *he_normal* distribution for the weights initialization. *ReLU*, or rectified linear unit, is the most popular activation function used. It transforms its input by using the formula $f(x) = max(0,x)$, this is, returns zero if the input is negative but if the input received is positive it returns that value back. This activation function requires no heavy computation, which helps on reducing the models computations time. This network was trained with hand-picked features, alike the linear regression models. The initialization of parameters used was *he_normal* initialization, where the weights are sampled from values of a normal distribution $w_i \sim N[0,\sigma]$ with $\sigma = \sqrt{(\frac{2}{fan\_in})}$, where fan_in represents the number of input units in the weight.

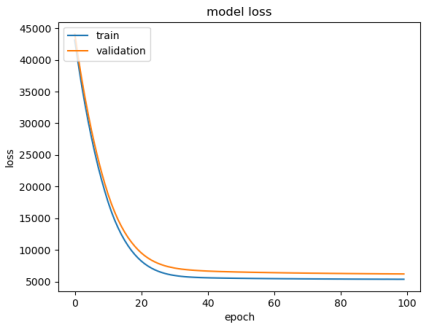
The second model is a recursive neural network (RNN) with gated recurrent unit cells (GRU). This RNN in particular, uses a gated state mechanism that manages the flow of information between the nodes of the RNN and have only one hidden state, with 128 hidden units or 300 units. This model was trained with a matrix composed by each ideals exponent vectors from each generator, for example, for the dataset 3-20-4-weighted, for each ideal there is a 4×6 matrix where each row represents each ideal generator exponents with the three variables for each monomial.

6.2.1 Models Architecture

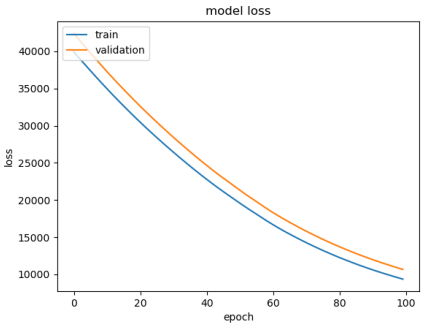
The models parameters were defined by experimenting different number of layers, nodes per layer, batch size and learning rate, where batch size is the number of training samples in one iteration and the learning rate defines the step size in each iteration when minimizing the loss function. These parameters are decided by analyzing the learning and validation curves of the training and evaluation metrics, in this case is the R^2, with the intention of avoiding learning underfit or overfit. The dataset used to measure the best fit for the data was considerably small, containing one thousand samples. This is because the training using the full dataset takes several hours to conclude. These tests were performed in all eight smaller datasets and, for each neural network model binomial and toric ideals, they have different architectures.



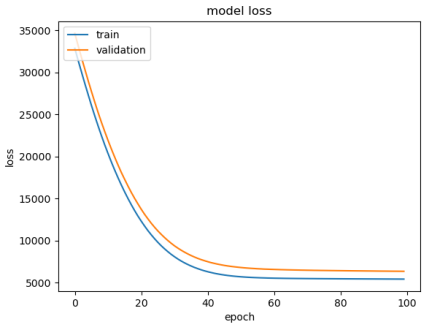
(a) Batch size=16, two dense layers, each with units 128 and learning rate = 10^{-5}



(b) Batch size=16, two dense layers, each with units 300 and learning rate = 10^{-5}



(c) Batch size=32, two dense layers, each with units 128 and learning rate = 10^{-5}

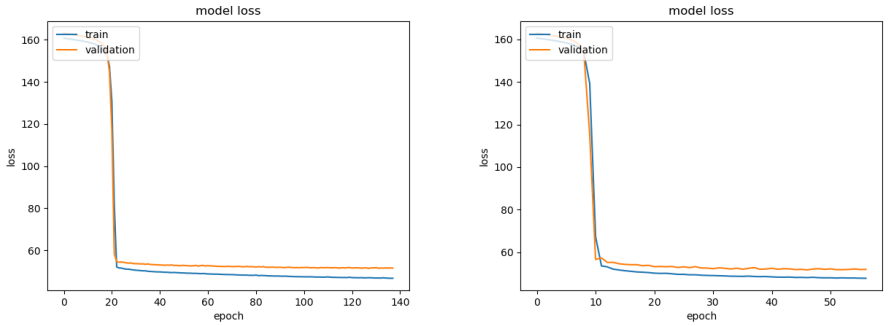


(d) Batch size=32, two dense layers, each with units 300 and learning rate = 10^{-5}

Fig. 6.1 Train and Validation learning curves during training of a recursive neural network for predicting the number of polynomial additions using already calculated statistic features, MMMSDDeg, for 3-20-4-weighted dataset. The parameters taken in consideration are the batch size, units per layer and learning rate.

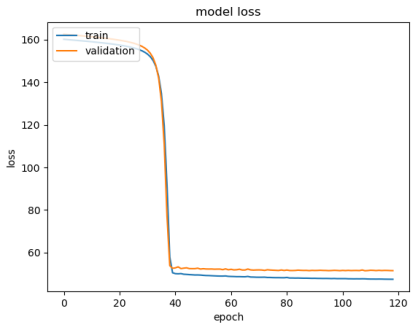
In Figure 6.1, each graph displays the learning and validation curves which describes the multiple models learning evolution. Comparing the results from the different models, one can see that the learning curve of the training improves for the smaller learning rate and there is a smaller gap between the curves. Considering the curves behaviour and the R^2, the best fitting model is (d).

Both of the models above were to predict the number of polynomial additions using a fixed set of features, the same for the multiple linear regression model. For the case where the input of the neural network model is the ideals exponents dataset the model was similarly modified in order to have the best results possible. Unlike the previous model with fixed features, this model learns its own features since the only input given are the exponents. Just like they were modified for the different input, the models parameters were also modified for the different ideal distributions: binomial and toric. This change is explained by the difference between the dimension of the binomial and toric dataset, since the binomial have a fixed number of generators, never bigger than ten, and the toric ideals do not have a fixed number of ideal generators, in many cases surpassing the number of generators of the binomials by a large amount, for example, over one thousand ideal generators.

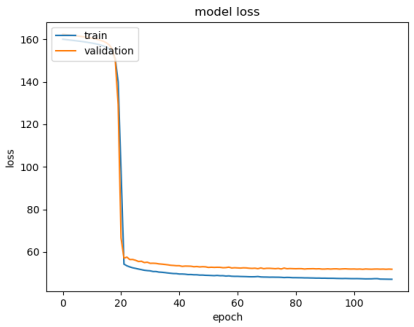


(a) Batch size=16, units per layer=128 and learning rate = 10^{-5}

(b) Batch size=16, units per layer=300 and learning rate = 10^{-5}



(c) Batch size=32, units per layer=128 and learning rate = 10^{-5}

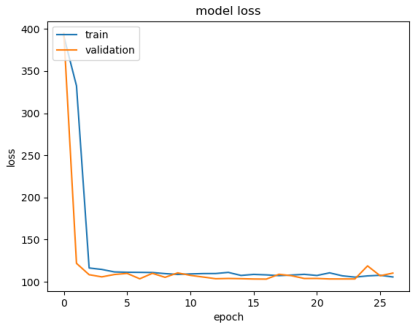


(d) Batch size=32, units per layer=300 and learning rate = 10^{-5}

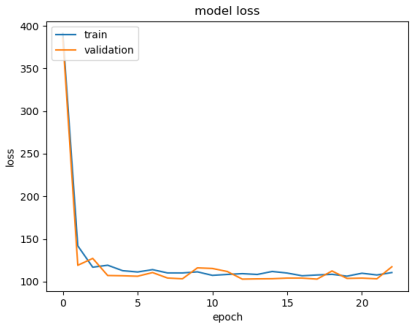
Fig. 6.2 Train and Validation learning curves during training of a recursive neural network for predicting the number of polynomial additions using the ideals exponents, for 3-20-10-weighted dataset. The parameters taken in consideration are the batch size, units per layer and learning rate.

Analysing the learning and validation curves from the graphs in Figure 6.2, it is clear that the smaller learning rate shows to be more underfit than the bigger learning rate. With the help of the model metrics, in this case R^2, the model with better results is the model (a), which shows that it is capable of further learning and improvements and the R^2 result is better than the other models.

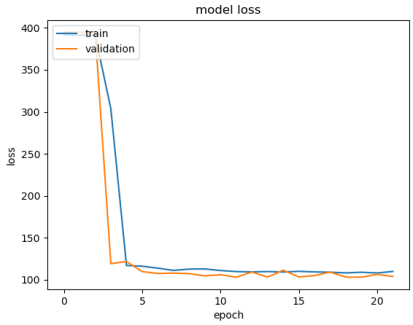
As for the toric case, the only difference is in the batch size, in which it was considered the sizes ten and twenty. Once again, the training using a bigger learning rate, this is 10^{-4}, showed to return a better fit for the toric data. The number of units in the GRU layer and batch size was ultimately decided



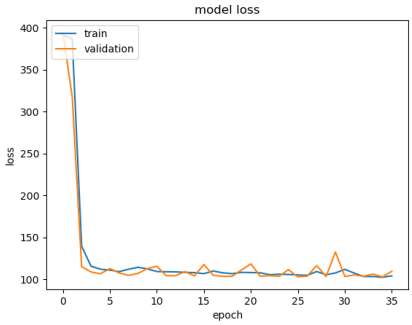
(a) Batch size=10, units per layer=128 and learning rate = 10^{-4}



(b) Batch size=10, units per layer=300 and learning rate = 10^{-4}



(c) Batch size=20, units per layer=128 and learning rate = 10^{-4}



(d) Batch size=20, units per layer=300 and learning rate = 10^{-4}

Fig. 6.3 Train and Validation learning curves during training of a recursive neural network for predicting the number of polynomial additions using the ideals exponents, for toric-2-0-5-8 dataset. The parameters taken in consideration are the batch size, units per layer and learning rate.

based on the R^2 metric, calculated after training. Therefore, the parameter choice that displayed more promising results was with a batch size of 20, 300 units per layer and a learning rate of 10^{-5}. But, when training with other toric datasets, due to the increasing ideals dimension, the batch size had to be also increased, as in for the other three datasets the batch size of doubled to 80 samples.

Each dataset was divided in training data, validation data and test data, the size of each sample was 80%, 10% and 10%, respectively. When working with the binomial datasets, the input dimensions were always fixed since the number of generators were also fixed. For the toric datasets, the ideals generators were mot a fixed parameter, so the ideals exponents vector size fluctuated, but always a multiple of sixteen. This means there were exponents vectors with an extremely big dimension or small dimension, and that also showed to be a problem when processing that data. To work around this problem, the input data was processed in batches and each sequence batch was padded to the same length. The padded value used was -10 due to it being an unlikely value for an exponent. This means, if the biggest length in a certain batch is 256, and an ideal exponent vector in the same batch is, for example, 240, the output of the padding function is a vector with a size of 256 where the last sixteen values are equal to -10. Doing so, might be seen has a forced improvement, given that the chosen value can prove to be an advantage with comparison if it was zero, as the value -10 is not a possible value for exponent.

6.2.2 Neural Network Models Training

In this section, the results from training each dataset from the Table 5.1 in a simple neural network and RNN are displayed in the following tables. These models are the multiple linear regression model with fixed features: maximum degree, minimum degree, mean degree, standard deviation degree and pure powers for binomial datasets and number of generators for toric datasets. With the neural networks, it is pursued improvement from the linear regression. To do so, a simple linear regression model to predict polynomial additions using the same fixed features in linear regression was performed. After, it was attempted to train neural network models using only the exponents, so that the algorithm would be free to choose its own features in order to predict polynomial additions. The model used for this was a recursive neural network with gated recurrent units. The datasets used for the fitting of the models represented in this table were computed using the selection strategy Sugar.

	3-20-10-w	3-20-10-u	3-20-4-w	3-20-4-u
Multiple Linear Regression	0.19	0.01	0.20	0.02
Simple Neural Network(MMMSDDeg+PurePowers model)	0.19	0.01	0.24	0.02
GRU RNN	0.31	0.15	0.47	0.21

	T(2-0-5-8)	T(4-0-5-8)	T(6-0-5-8)	T(6-0-10-8)
Multiple Linear Regression	0.17	0.31	0.22	0.15
Simple Neural Network(MMMSDDeg+NumGen model)	0.19	0.32	0.22	0.16
GRU RNN	0.39	0.27	0.68	0.72

Table 6.4 Summary of fitting the number of polynomial additions with linear regression and two neural network models. The results are represented by R^2 statistics. The calculation of the Gröbner basis in this dataset were computed with selection strategy Sugar.

Parallel to the table above, the following table represents the results from a multiple linear regression (maximum degree, minimum degree, mean degree, standard deviation degree, pure powers/number of generators), a simple neural network model with features maximum Degree, minimum degree, mean degree, standard deviation degree, pure powers/number of generators and a recursive neural network with GRU units model using the exponents. The dataset used for the fitting of the models represented in this table was computed using the selection strategy Degree, with a size of one million samples.

	3-20-10-w
Multiple Linear Regression	0.24
Simple Neural Network(MMMSDDeg model)	0.24
GRU RNN	0.40

Table 6.5 Summary of fitting the number of polynomial additions with linear regression and two neural network models. The results are represented by R^2 statistics. The calculation of the Gröbner basis in this dataset were computed with selection strategy Degree.

Chapter 7

Conclusion

In the previous chapter, the main result came from datasets with the same distributions as in the article [26] but with a different selection strategy. Comparing the results from the linear regression model using Degree in Table 4.3 and results using Sugar in Table 6.3 it is clear that, in general, the selection strategy Degree have better results than Sugar, this is, the prediction ability is better than Sugar. In an attempt to improve our linear regression models a different ideal degree calculation was considered. When using the MMMSDAbD model it performs worse than the MMMSDDeg, but together there is a considerable improvement, this comparison can also be seen from the results in Table 6.3. One can conclude from these results that the regression models can be improved greatly by choosing different prediction parameters. Another interesting detail comes from the different results between weighted and uniform binomials, weighted binomials are much easier to predict than uniform, this is most likely from the exponents degrees distribution, since weighted tends to generate binomials with lower degrees in comparison to uniform, where the uniform samples have higher degrees.

Using a more advanced model, a simple neural network, and feeding as an input the same MMMSDDeg model plus pure powers or number of generators, one should not expect any better result than the results of a linear regression. This comes from the fact that the features are fixed and are probably not the best for predicting polynomial additions. Therefore, the idea of implementing a neural network that takes as inputs the exponent matrix of each ideals comes naturally since it allows the model to learn by itself the best features for predicting polynomial additions. Observing the Table 6.4 and comparing to the results in Table 4.5 in the binomial dataset, the 3-20-10 uniform performed slightly better where with a Degree strategy, the R^2 is 13% and with Sugar increased to 15%, where for the 3-20-4-weighted it improved with Sugar, with a difference of 10%. The remaining binomial datasets performed worse. Although there were some better results for binomials, they don't show as much improvement as for toric datasets. In general the results are all similar, but the toric-6-0-10-8 with sugar selection strategy had much better results than with degree, with a difference of 20%. One should note that the fact that it is easier to predict the performance of some selection strategy does not, by any means, imply that it is a better strategy. The takeaway should only be that if one wants to study learnability of the performance of Buchberger's algorithm, in which the choice of selection strategy does have an impact on the results.

In order to also confirm that our data and models agree with the article, the dataset 3-20-10-weighted with Degree selection strategy was also used for the RNN, the results are very similar, with a difference of only 1%, this may be a consequence of the randomness from the datasets generation.

The toric ideals exhibit much more promising results than binomial ideals, in Tables 6.3 and 6.2. This may occur from the ideals generation, where the toric ideals appear more "natural". The models still have difficulty predicting when the number of polynomial additions are higher, as we can see in Figures 7.1 and 7.2, the improvement is visually noticeable, where in the recursive neural network model, for both distributions, the actual and predicted values are more condensed near the black line. This demonstrates how the predicted values from the RNN model are more accurate than with a simple regression model.

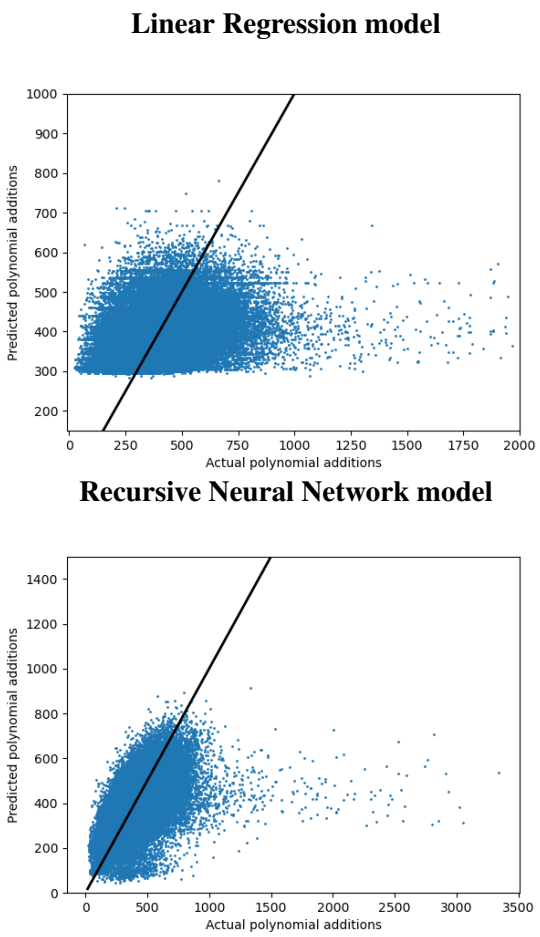


Fig. 7.1 Actual versus predicted polynomial additions of a 100 000 sized test set in toric-2-0-5-8. The black line represents the perfect prediction versus actual matches.

Evaluating the multiple ideal datasets in linear regression models and recursive neural network models demonstrated how different ideal characteristics can influence the number of polynomial additions and the prediction of the same. Uniform sampling in the binomial ideals are associated with highest number of polynomial additions, as we can see in Tables 5.2,5.3 and 5.4, also they perform worse both in linear regression models and neural network models. This can be associated with the models difficulty to predict higher number of polynomial additions. In the toric ideals datasets, it is

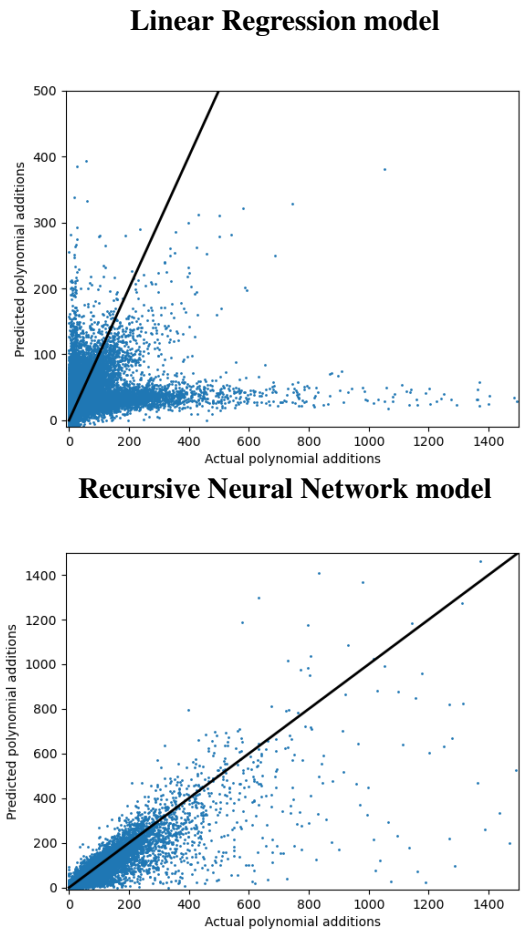


Fig. 7.2 Actual versus predicted polynomial additions of a 100 000 sized test set in toric-6-0-5-8. The black line represents the perfect prediction versus actual matches.

easier to predict distributions with six target variables, as can be seen in Table 6.3. But, can conclude that, in general, RNNs proves to be a useful tool for predicting the number of polynomial additions, and consequently, to be able to predict the performance of the Buchberger's algorithm for some ideal distribution.

For a future approach, generating bigger samples of datasets for the RNN training could improve the results. Also, regarding the results from toric ideals, a broader distribution of toric ideals datasets would be interesting to research, with bigger number of target variables and number of source variables, and also compute the correspondent Gröbner basis with more selection strategies. In this project, for both binomial and toric case, the ideals exponents were saved for the training, but a better approach for the toric ideals would be to save the correspondent matrices with integer entries of each ideal and utilize them for the training process. To be able to generate such datasets, with at least one million samples will require a considerable amount of time and hardware capacity. In addition, more complex neural network models can be designed to better fit the inputs, and see how they would perform. Additionally, trying to predict the final Gröbner basis size and not the overall number of polynomials additions could reveal other interesting results, and perhaps, could be more predictable. The creation of the data and its processing consisted the most laborious task during this dissertation, given that the generation of a large sample size for each ideal distribution took a considerable amount of time and had to be occasionally redone due to generation errors. The dataset used for this dissertation can be found in Zenodo at [10], and the correspondent code for the linear regression and machine learning models is available in Zenodo at [9].

References

[1] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

[2] Bernstein, A. and de V. Roberts, M. (1958). Computer v. chess-player. *Scientific American*, 198(6):96–107.

[3] Boulanger-Lewandowski, N., Bengio, Y., and Vincent, P. (2012). Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. *arXiv preprint arXiv:1206.6392*.

[4] Buchberger, B. (2006). *An Algorithm for Finding the Basis Elements in the Residue Class Ring Modulo a Zero Dimensional Polynomial Ideal*. PhD thesis.

[5] Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., VanderPlas, J., Joly, A., Holt, B., and Varoquaux, G. (2013). API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122.

[6] Calin, O. (2020). *Deep learning architectures*. Springer.

[7] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

[8] Cox, D., Little, J., and O'Shea, D. (2007). Ideals, varieties, and algorithms. an introduction to computational algebraic geometry and commutative algebra.

[9] Cruz, A. (2022a). Code files for 'predicting the performance of buchberger's algorithm'.

[10] Cruz, A. (2022b). Ideals dataset. Available at https://doi.org/10.5281/zenodo.6939734.

[11] Cybenko, G. (1992). Approximation by superpositions of a sigmoidal function. *Math. Control. Signals Syst.*, 5(4):455.

[12] Eck, D. and Schmidhuber, J. (2002). A first look at music composition using lstm recurrent neural networks. *Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale*, 103:48.

[13] Faugére, J.-C. (1999). A new efficient algorithm for computing gröbner bases (f4). *Journal of Pure and Applied Algebra*, 139(1):61–88.

[14] Faugére, J.-C. (2002). A new efficient algorithm for computing gröbner bases without reduction to zero f5. *Proceedings of ISSAC. ACM press*, pages 75–83.

[15] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

[16] Grayson, D. R. and Stillman, M. E. (2009). Macaulay2, a software system for research in algebraic geometry. Available at http://www.math.uiuc.edu/Macaulay2/.

[17] Gröbner, W. (1998). On elimination theory. *SIGSAM Bull.*, 32(2):40–46.

[18] Hanin, B. and Sellke, M. (2017). Approximating continuous functions by relu nets of minimal width. *arXiv preprint arXiv:1710.11278*.

[19] Herzog, J., Hibi, T., and Ohsugi, H. (2018). *Binomial ideals*, volume 279. Springer.

[20] Hornik, K. (1992). Multilayer feed-forward networks are universal approximators. *Artificial neural networks: Approximation and learning theory*.

[21] Kohavi, R. (1998). Glossary of terms. *Special issue on applications of machine learning and the knowledge discovery process*, 30(271):127–132.

[22] Leshno, M., Lin, V. Y., Pinkus, A., and Schocken, S. (1993a). Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):861–867.

[23] Leshno, M., Lin, V. Y., Pinkus, A., and Schocken, S. (1993b). Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6(6):861–867.

[24] McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.

[25] McGonagle, J., Williams, C., and Khim, J. (2015). Recurrent neural network. Available at https://brilliant.org/wiki/recurrent-neural-network/.

[26] Mojsilović, J., Peifer, D., and Petrović, S. (2021). Learning a performance metric of buchberger's algorithm. *arXiv preprint arXiv:2106.03676*.

[27] Pinkus, A. (1999). Approximation theory of the mlp model in neural networks. *Acta numerica*, 8:143–195.

[28] Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

[29] Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229.

[30] Sutskever, I., Hinton, G. E., and Taylor, G. W. (2008). The recurrent temporal restricted boltzmann machine. *Advances in neural information processing systems*, 21.

[31] Sutskever, I., Martens, J., and Hinton, G. E. (2011). Generating text with recurrent neural networks. In *ICML*.

[32] Widrow, B. and Hoff, M. E. (1962). Associative storage and retrieval of digital information in networks of adaptive "neurons". In *Biological Prototypes and Synthetic Systems*, pages 160–160. Springer.

[33] Wolberg, W. H., Street, W., and Mangasarian, O. (1994). Machine learning techniques to diagnose breast cancer from image-processed nuclear features of fine needle aspirates. *Cancer Letters*, 77(2):163–171. Computer applications for early detection and staging of cancer.

[34] Wolfram Research, Inc. (2010). Mathematica 8.0.