DEEP LEARNING FOR MODULI SPACE OF ALGEBRAIC CURVES

ELIRA CURRI AND TONY SHASKA

Abstract. In this project we investigate the moduli space of genus two curves defined over $\mathbb{Q}$ using methods of deep learning. We will use several machine learning models such as sequential model, equivariant neural networks, transformer neural networks to determine several properties of genus two curves. Our experiments lead us to some interesting observations. For example, we prove that the only genus two curve with weighted moduli height $\mathcal{S}_{\rm k} < 3$ defined over $\mathbb{Q}$ is the single curve $y^2 = x(x^5 - 1)$ with automorphism group isomorphic to the cyclic group C_{10}. Equivalently this implies that the well-known 2-dimensional locus of genus 2 curves with extra involutions, has no rational point with height $\mathcal{S}_{\rm k} \leq 3$.

This is a very rough draft. Check with us for the latest update.

1. Introduction

Artificial Intelligence and Machine Learning are some of the most active and exciting branches of science of the last few decades. These new technologies have made their way into economy, including engineering, medical science, finance, cybersecurity, etc. Can they be used for mathematical research?

The question is not new. After all science is all about collecting data and deducing conclusions. Machine Learning is about gathering data, training the data, and drawing conclusions. Depending on the kind of data we use different methods for machine learning: supervised learning, unsupervised learning, or a combination of the two.

So the first step is to gather the data. There have always been databases in mathematics, but the most famous databases of the XX-century were the Atlas of Finite Simple Groups, Cremona tables of elliptic curves, database of elliptic curves compiled by Birch and Swinnerton-Dyer which led to the famous Birch and Swinnerton-Dyer conjecture; see [4, 5]. With the development of computer algebra toward the last quarter of the XX-century we saw different databases which had a huge impact on mathematics, for example the Small Library of Groups in Gap, the list of Calabi-Yau hypersurfaces, etc.

The goal of this work is to use new tools of machine learning to study the moduli space $\mathcal{M}_g$ of genus $g \geq 2$ curves defined over a field k. The moduli space of algebraic curves has been the focal point of algebraic geometry for the last few decades. With the development of the new computational tools it became necessary in the last few decades to reconsider the theory of invariants, in its classical form or in the framework of the theory developed by Mumford [11] with the intention of studying the arithmetic of moduli spaces. Naturally some of the first attempts focused on $\mathcal{M}_2$; see for example [9] and attempts to generalize to $g > 2$ [10]. From these attempts the concept of weighted Weil height was born; see [1, 6, 12].

1.1. The moduli space $\mathcal{M}_2$ as a case study.

The moduli space $\mathcal{M}_2$ of genus 2 curves is the most understood moduli space among all moduli spaces. This is mostly due to two main facts; first all genus two curves are hyperelliptic and therefore studying them it is easier than general curves, secondly even among hyperelliptic curves the curves of genus two have a special place since they correspond to binary sextics which, from the computational point of view, are relatively well understood compared to higher degree binary forms.

One of the main questions related to $\mathcal{M}_2$ has been to recover a nice equation for any point $\mathfrak{p} \in \mathcal{M}_2$. Since $\mathcal{M}_2$ is a coarse moduli space, such equation is not always defined over the field of moduli of $\mathfrak{p}$. Can we find a universal equation for genus two curves over their minimal field of definition? Can such equation provide a minimal model for the curve? Does the height of this minimal model has any relation to the projective height

Date: November 29, 2023.

1991 *Mathematics Subject Classification.* Primary: 68T07; Secondary: 68T20, 68Q32.

Key words and phrases. Stability, binary forms, weighted heights.

of the corresponding moduli point $\mathfrak{p} \in \mathcal{M}_2$? What is the distribution in $\mathcal{M}_2$ of points $\mathfrak{p}$ for which the field of moduli is not a field of definition? The answers to these questions are still unknown.

In [2] we provide a database of genus 2 curves which contains all curves with height $\mathrm{h} \leq 5$, curves with moduli height $\mathfrak{h} \leq 20$, and curves with automorphism and height ≤ 101. They are organized in three Python directories $\mathcal{L}_i$, $i = 1, 2, 3$. The database is build with the idea of better understanding $\mathcal{M}_2$, the distribution of points in $\mathcal{M}_2$ based on the moduli height, the distribution of points for which the field of moduli is not a field of definition.

Let $\mathcal{X}$ be a genus two curve defined over $\mathbb{Q}$. The moduli point in $\mathcal{M}_2$ corresponding to $\mathcal{X}$ is given by $\mathfrak{p} = (i_1, i_2, i_3)$, where i_1, i_2, i_3 are absolute invariants as in [2]. Since i_1, i_2, i_3 are rational functions in terms of the coefficients of $\mathcal{X}$, then $i_1, i_2, i_3 \in \mathbb{Q}$. The converse isn't necessarily true. Let $\mathfrak{p} = (i_1, i_2, i_3) \in \mathcal{M}_2(\mathbb{Q})$. The universal equation of a genus 2 curve corresponding to $\mathfrak{p}$ is determined in [9], which is defined over a quadratic number field K. The main questions we want to consider is what percentage of the rational moduli points are defined over $\mathbb{Q}$? How can we determine a minimal equation for such curves?

For every point $\mathfrak{p} \in \mathcal{M}_2$ such that $\mathfrak{p} \in \mathcal{M}_2(k)$, for some number field K, there is a pair of genus-two curves $\mathcal{C}^{\pm}$ given by $\mathcal{C}^{\pm}: \ y^2 = \sum_{i=0}^{6} a_{6-i}^{\pm} x^i$, corresponding to $\mathfrak{p}$, such that $a_i^{\pm} \in K(d)$, $i = 0, \ldots, 6$; see [9].

In [2] were created three Python dictionaries: $\mathcal{L}_1$: curves with height ≤ 10, $\mathcal{L}_2$: curves with extra involutions, $\mathcal{L}_3$: curves with small moduli height. In [14] it is discussed when such curves have minimal height and how a reduction as in [3] is easier to perform in this case. There are 20 697 curves in $\mathcal{L}_2$, such that for each h we have roughly 4 h curves. So it is expected that the number of curves of height $\leq h$, defined over $\mathbb{Q}$ is $\leq 4^{\frac{\mathrm{h}(\mathrm{h}+1)}{2}}$; see [14] for more details. For $\mathfrak{p} \in \mathcal{M}_2(\mathbb{Q})$ be such that $\mathrm{Aut}(\mathfrak{p}) \cong V_4$ there is a genus 2 curve $\mathcal{X}$ corresponding to $\mathfrak{p}$ with equation $y^2 z^4 = f(x^2, z^2)$. We pick $f \in \mathbb{Z}[x, z]$, such that $f(x, z)$ is a reduced binary form. From 20 292 such curves we found only 57 which do not have minimal absolute height. $\mathcal{L}_3$ is a list of all moduli points $[x_0 : x_1 : x_2 : x_3]$ of projective height $\leq \mathfrak{h}$ in $\mathbb{P}^3(\mathbb{Q})$, for some integer $\mathfrak{h} \geq 1$.

Problem 1. *What percentage of rational points $\mathfrak{p} \in \mathcal{M}_2(\mathbb{Q})$ with a fixed moduli height $\mathfrak{h}$ have $\mathbb{Q}$ as a field of definition, when $\mathfrak{h}$ becomes arbitrarily large?*

We confirm, as expected, that for large moduli height $\mathfrak{h} \in \mathcal{M}_2(\mathbb{Q})$, the majority of genus 2 curves not defined over $\mathbb{Q}$ and they don't have extra automorphisms.

Problem 2. *Can we train a neural network which detects with some confidence if for a given moduli point $\mathfrak{p} \in \mathcal{M}_2(\mathbb{Q})$ there is a curve $\mathcal{X}$ defined over $\mathbb{Q}$, corresponding to $\mathfrak{p}$.*

1.2. **Choosing the right machine learning model.** For distinguishing points in a 3-dimensional space with properties A and B, there are various machine learning models you could consider based on the nature of your data and the problem you're solving:

1.2.1. *Support Vector Machines (SVM).* SVMs are powerful for binary classification tasks. They work well for both linearly separable and non-linearly separable data. They find the hyperplane that best separates the data into two classes. We consider them in **??**

1.2.2. *Random Forests or Decision Trees.* Decision trees or ensembles like Random Forests can be effective for classification tasks. They handle both numerical and categorical data, are relatively easy to interpret, and can handle complex relationships between features.

1.2.3. *K-Nearest Neighbors (KNN).* KNN is a simple and effective algorithm for classification. It classifies points based on the majority vote of their neighbors. It might work well for spatial data as points close together might share similar properties.

1.2.4. *Neural Networks.* Specifically, you could use feedforward neural networks or deep learning models. They are highly flexible and can learn complex patterns in the data. For spatial data, convolutional neural networks (CNNs) might be particularly useful. We consider them in Section 2.3

1.2.5. *Gradient Boosting Models.* Models like XGBoost or LightGBM can be effective for classification tasks. They work by combining multiple weak learners to create a strong model and can handle complex relationships in the data.

1.2.6. *Naive Bayes Classifier.* This is a probabilistic classifier that assumes the independence of features. It can work well for high-dimensional data and is relatively simple and fast.

The choice of the best model depends on various factors like the size of your dataset, the nature of the data, the presence of noise or outliers, and computational resources. Experimenting with a few models and evaluating their performance using metrics like accuracy, precision, recall, F1 score, and area under the ROC curve (AUC-ROC) would be a good approach to determine the most suitable model for your specific problem.

1.3. **Higher moduli:** Can we generalize the approach above to $\mathcal{M}_g$ for $g > 2$? Moreover, can we train a machine learning model to obtain reliable results for $g \geq 2$?

The moduli space $\mathcal{M}_2$ is a very good model for the hyperelliptic moduli $\mathcal{H}_g$. Many of the results of $g = 2$ have been realized to higher genus hyperelliptic curves already and we now know many general theorems for $\mathcal{H}_g$; see [7], [10], etc.

Moreover, generalizing from hyperelliptic curves to superelliptic curves gives a very important tool in understanding $\mathcal{M}_g$; see [10] for details. Using results from [8] and previous work of these authors we can determine fully the list of automorphisms groups and inclusions among the loci for any genus, hence obtaining a full stratification of the moduli space $\mathcal{M}_g$. About 75-80% of all cases come from superelliptic curves, for which we know a great deal.

A very important development in understanding $\mathcal{M}_g$ is the discovery of the weighted height on the weighted projective spaces. Hence, the most efficient way to create a database of points in $\mathcal{M}_g$ is to consider the corresponding weighted moduli space $\mathcal{W}_g$ and sort the points in this space via their weighted heights.

A great learning example is the case $g = 3$ for many reasons. It is the first case that we have non-hyperelliptic curves, so it is more general than $g = 2$, but also it is still a case that we fully understand. For example, we explicitly know invariants of binary octavics, which classify hyperelliptic genus 3 curves, and invariants of ternary quartics which classify non-hyperelliptic genus 2 curves. We have a full understanding of the list of groups of automorphisms and in each case we can write an explicit parametric equation for the corresponding family. There has been work in the last decade by several authors on the field of moduli of genus 3 curves and we can recover the equation of the curve over a minimal field of definition.

It needs to be pointed out that in this general approach the biggest difficulty comes from arithmetic invariant theory in the sense that we don't know an explicit way of describing a moduli point $\mathfrak{p} \in \mathcal{M}_g$. While GIT provides an elegant theoretical framework, explicit results are missing even for genus g as small as 4 or 5.

Our general philosophy is to build the skeleton of $\mathcal{M}_g$ using the superelliptic curves. After all, the majority of points in $\mathcal{M}_g$ with nontrivial $\mathrm{Aut}(\mathfrak{p})$ are superelliptic points. We can say a lot on these superelliptic points on the problem of field of moduli versus field of definition, determine if they have complex multiplication, and write down explicit equations for them.

In this paper we will describe what can be achieved and what are the challenges for fully understanding the arithmetic of the moduli space. Our goal is to bring this topic to the attention to mathematicians specialized on machine learning and artificial intelligence techniques and hopefully involve more people in this ambitious but exciting project.

2. Preliminaries on neural networks

Let k be a field and for any integer $n \geq 1$ denote by $\mathbb{A}^n_k$ (resp. $\mathbb{P}^n_k$) the affine (resp. projective) space over k. When k is an algebraically closed field, we will drop the subscript. A fixed tuple of positive integers $\mathfrak{w} = (q_0, \ldots, q_n)$ is called **set of weights**. The weight of $\alpha \in k$ will be denoted by $\mathbf{wt}(\alpha)$. The set

$$\mathbb{V}^n_{\mathfrak{w}}(k) := \{(x_1, \ldots, x_n) \in k^n \mid \mathbf{wt}(x_i) = q_i,\ i = 1, \ldots, n\}$$

is a graded vector space over k. An element $\mathbf{x} \in \mathbb{V}^n_{\mathfrak{w}}(k)$ is denoted by $\mathbf{x} = (x_0, \ldots, x_n)$ and its i-th coordinate by $x_i(\mathbf{x})$.

2.1. **Neural networks.** A **neuron** is a function $f : \mathbb{V}^n_{\mathfrak{w}}(k) \to k$ such that

$$\alpha_{\mathfrak{w}}(\mathbf{x}) = \sum_{i-0}^{n} w_i x_i + \mathbf{b},$$

where $\mathbf{b} \in k$ is a constant called **bias**. We can generalize neurons to tuples of neurons via

$$\begin{aligned} \phi := \mathbb{V}^n_{\mathfrak{w}}(k) &\to \mathbb{V}^n_{\mathfrak{w}}(k) \\ \mathbf{x} &\to g\left(\alpha_0(\mathbf{x}), \ldots, \alpha_n(\mathbf{x})\right) \end{aligned}$$

for any gives set of weights $\mathfrak{w}_0, \ldots, \mathfrak{w}_n$. Then ϕ is a k-linear function with matrix written as

$$\phi(\mathbf{x}) = W \cdot \mathbf{x} + \mathbf{b},$$

for some $\mathbf{b} \in k^{n+1}$ and W an $n \times n$ matrix with integer entries.

Definition 1. *A function $g : \mathbb{V}^n_{\mathfrak{w}} \to \mathbb{V}^n_{\mathfrak{w}}$ is called an **activation function** while a **network layer** is a function*

$$\begin{aligned} \mathbb{V}^n_{\mathfrak{w}}(k) &\to \mathbb{V}^n_{\mathfrak{w}}(k) \\ \mathbf{x} &\to g\left(W \cdot \mathbf{x} + \mathbf{b}\right) \end{aligned}$$

*for some g some activation function. A **neural network** is the composition of many layers. The l-th layer*

$$\begin{aligned} \cdots \longrightarrow \mathbb{V}^n_{\mathfrak{w}}(k) &\xrightarrow{\phi_l} \mathbb{V}^n_{\mathfrak{w}}(k) \longrightarrow \cdots \\ \mathbf{x} &\longrightarrow \phi_l(\mathbf{x}) = g_l\left(W^l \mathbf{x} + \mathbf{b}^l\right), \end{aligned}$$

where g_l, W^l, and $\mathbf{b}^l$ are the activation, matrix, and bias corresponding to this layer.

After m layers the output (predicted values) will be denoted by $\hat{\mathbf{y}} = [\hat{y_1}, \ldots, \hat{y_n}]^t$, where

$$\hat{\mathbf{y}} = \phi_m\left(\phi_{m-1}\left(\ldots\left(\phi_1(\mathbf{x})\right)\ldots\right)\right),$$

while the true values by $\mathbf{y} = [y_1, \ldots, y_n]^t$.

2.2. **Loss or cost function.** For *regression problems* we define a **loss functions** as the **mean absolute error** (MAE) or the **mean square error** (MSE) as

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i|, \qquad MSE = \frac{1}{n}\sum_{i=1}^{n} (y_i - \hat{y}_i)^2, \tag{1}$$

where y_i and $\hat{y}_i$ are the true and predicted values respectively.

For *classification problems* the loss function will be **cross entropy** (CE)

$$CE = -\sum_{i=1}^{n} y_i \log\left(\hat{y}_i\right), \tag{2}$$

for multiclass classification.

2.3. **Equivariant Neural Networks.** Let G be a group acting on a set X via

$$\begin{aligned} G \times X &\to X \\ (g, x) &\to gx \end{aligned}$$

We say G *acts from the left* and will call X a G-set. Let $f : X \to Y$ be a map among G-sets X and Y. We call f G-**equivariant** if

$$f(gx) = gf(x)$$

for all $x \in X$, and $g \in G$. A function is called G-**invariant** if

$$f(gx) = f(x)$$

An equivariant map induces a map between quotient spaces

$$\bar{f} : X/G \to Y/G$$

Notices that sometimes G/X is used to denote the quotient space of a left G-action.

Theorem 1 (G-Equivariant Convolution Theorem)**.** *A neural network layer (linear map) ψ is G-equivariant if and only if its form is a convolution operator $\star$*

$$\psi(f) = (f \star w)(u) = \sum_{g \in G} f g(ug^{-1}) w(g)$$

where $f : U \to \mathbb{R}^n$ and $w : V \to \mathbb{R}^n$ are lifted functions of subgrouos U and V of G. On the first layer of a neural network, f is usually is defined on the quotient space $U = G/H$. If the group G is locally compact (infinite elements), then the convolution operator is

$$\psi(f) = (f \star w)(u) = \sum_{g \in G} f g(ug^{-1}) w d\mu(g)$$

where μ is the group Haar measure.

3. Algebraic geometry background

Let k be a field, $k[x, y]$ be the polynomial ring in two variables and V_d denote the $(d+1)$-dimensional subspace of $k[x, y]$ consisting of homogeneous polynomials

$$f(x, y) = a_d x^d + a_{d-1} x^{d-1} y + \cdots + a_0 y^d \tag{3}$$

of degree d. Elements in V_d are called **binary forms** of degree d. $\mathrm{GL}_2(k)$ acts as a natural group of automorphisms on $k[x, y]$. Denote by $f \to f^M$ this action. It is well known that $\mathrm{SL}_2(k)$ leaves a bilinear form (unique up to scalar multiples) on V_d invariant; see [13] for details.

Consider $a_0, a_1, \ldots, a_d$ as transcendentals over k (coordinate functions on V_d). Then the coordinate ring of V_d can be identified with $k[a_0, \ldots, a_d]$. We define an action of $\mathrm{GL}_2(k)$ on $k[a_0, \ldots, a_d]$ via

$$\begin{aligned} &\mathrm{GL}_2(k) \times k[a_0, \ldots, a_d] \to k[a_0, \ldots, a_d] \\ &(M, F) \to F^M := F(f^M), \quad \text{for all } f \in V_d. \end{aligned}$$

Thus for $F \in k[a_0, \ldots, a_d]$ and $M \in \mathrm{GL}_2(k)$, define $F^M \in k[a_0, \ldots, a_d]$ as

$$F^M(f) := F(f^M),$$

for all $f \in V_d$. Then $F^{MN} = (F^M)^N$. The homogeneous degree in $a_0, \ldots, a_d$ is called the **degree** of F, and the homogeneous degree in x, y is called the **order** of F. An **invariant** is usually referred to an $\mathrm{SL}_2(k)$-invariant on V_d. Hilbert's theorem says that the ring of invariants $\mathcal{R}_d$ of binary forms of degree d is finitely generated. Thus, $\mathcal{R}_d$ is finitely generated, and $\mathcal{R}_d$ is a graded ring.

Let $\{\xi_0, \ldots, \xi_n\}$ be a minimal generating set for $\mathcal{R}_d$. Since $\xi_i \in k[a_0, \ldots, a_d]$ are homogenous polynomials we denote $\deg \xi_i = q_i$ and assume that

$$q_0 \le q_1 \le \cdots \le q_n. \tag{4}$$

Degrees $q_0, \ldots, q_n$ are called **weights**. If $f, g \in V_d$, $M \in \mathrm{GL}_2(k)$, $\lambda = (\det M)^{\frac{d}{2}}$, then $f = g^M$ if and only if

$$(\xi_0(f), \ldots \xi_i(f), \ldots, \xi_n(f)) = (\lambda^{q_0} \xi_0(g), \ldots, \lambda^{q_i} \xi_i(g), \ldots, \lambda^{q_n} \xi_n(g)). \tag{5}$$

Example 1 (Quadratics)**.** *Let $f(x, y) = a_2 x^2 + a_1 xy + a_0 y^2$. $\mathcal{R}_2$ is generated by the discriminant $\Delta = a_1^2 - 4a_0 a_2$. For any $M = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \in \mathrm{GL}_2(k)$, $f^M(x, y)$ has*

$$\Delta(f^M) = (ad - bc)^2 \cdot (a_1^2 - 4a_0 a_2) = (\det M)^2 \cdot \Delta(f).$$

The exponent of $\det(M)$ is precisely $\frac{d}{2} \cdot q_i$, where q_i the degree of the invariant ξ_i.

If $k = \mathbb{Q}$ we can choose $\xi_0, \ldots, \xi_n$ with integer coefficients and primitive polynomials in $\mathbb{Z}[a_0, \ldots, a_d]$.

3.1. **Weighted greatest common divisors.** Let $\mathbf{x} = (x_0, \dots x_n) \in \mathbb{Z}^{n+1}$ be a tuple of integers, not all equal to zero. Their greatest common divisor, denoted by $\gcd(x_0, \dots, x_n)$, is defined as the largest integer d such that $d|x_i$, for all $i = 0, \dots, n$. Let $q_0, \dots, q_n$ be positive integers. A set of weights is called the ordered tuple $\mathfrak{w} = (q_0, \dots, q_n)$. Denote by $r = \gcd(q_0, \dots, q_n)$ the greatest common divisor of $q_0, \dots, q_n$. A *weighted integer tuple* is a tuple $\mathbf{x} = (x_0, \dots, x_n) \in \mathbb{Z}^{n+1}$ such that to each coordinate x_i is assigned the weight q_i. We multiply weighted tuples by scalars $\lambda \in \mathbb{Q}$ via

$$\lambda \star (x_0, \dots, x_n) = (\lambda^{q_0} x_0, \dots, \lambda^{q_n} x_n)$$

For an ordered tuple of integers $\mathbf{x} = (x_0, \dots, x_n) \in \mathbb{Z}^{n+1}$, whose coordinates are not all zero, the **weighted greatest common divisor with respect to the set of weights** $\mathfrak{w}$ is the largest integer d such that

$$d^{q_i} \mid x_i, \quad \text{for all } i = 0, \dots, n.$$

We will call a point $\mathfrak{p} \in \mathbb{P}^n_{\mathfrak{w}}(\mathbb{Q})$ **normalized** if $\operatorname{wgcd}(\mathfrak{p}) = 1$. The **absolute weighted greatest common divisor** of an integer tuple $\mathbf{x} = (x_0, \dots, x_n)$ with respect to the set of weights $\mathfrak{w} = (q_0, \dots, q_n)$ is the largest real number d such that

$$d^{q_i} \in \mathbb{Z} \quad \textit{and} \quad d^{q_i} \mid x_i, \quad \text{for all } i = 0, \dots n.$$

3.2. **Weighted projective spaces and moduli space of binary forms.** For any integer $m \geq 1$, let μ_m denote the group of m-th roots of unity generated by ξ_m, which is assumed to be contained in k. Consider the action of $k^* = k \setminus \{0\}$ on $\mathbb{A}^{n+1}_k \setminus \{(0, \cdots, 0)\}$ given by

$$\lambda \star (x_0, \dots, x_n) = (\lambda^{q_0} x_0, \dots, \lambda^{q_n} x_n), \text{ for } \lambda \in k^*. \tag{6}$$

Define the **weighted projective space**, denoted by $\mathbb{WP}^n_{\mathfrak{w}}(k)$, to be the quotient space $\mathbb{V}^{n+1}_k / k^*$ of this action, which is a geometric quotient since k^* is a reductive group. An element $\mathbf{x} \in \mathbb{WP}^n_{\mathfrak{w}}(k)$ is denoted by $\mathbf{x} = [x_0 : \cdots : x_n]$ and its i-th coordinate by $x_i(\mathbf{x})$.

Sometimes $\mathbb{P}^n_{\mathfrak{w},k}$ will be used instead of $\mathbb{WP}^n_{\mathfrak{w}}(k)$. A weighted space $\mathbb{P}^n_{\mathfrak{w},k}$ is called **reduced** if $\gcd(q_0, \cdots, q_n) = 1$. It is called **normalized** or **well-formed** if

$$\gcd(q_0, \dots, \hat{q}_i, \dots, q_n) = 1, \quad \text{for each } \ i = 0, \dots, n.$$

Any weighted projective space is isomorphic to a reduced and well-formed one. Given any tuple of weights $\mathfrak{w} = (q_0, \dots, q_n)$, the following hold ([12, Prop. 4]):

(i) Any weighted projective space $\mathbb{P}^n_{\mathfrak{w},k}$ is isomorphic to $\mathbb{P}^n_{\mathfrak{w}',k}$, where $\mathfrak{w}'$ is a reduced tuple of weights.
(ii) If $\mathbb{P}^n_{\mathfrak{w},k}$ is reduced and $d_i = \gcd(q_0, \cdots, \hat{q}_i, \cdots, q_n)$ for $0 \leq i \leq n$, then $\mathbb{P}^n_{\mathfrak{w},k} \cong \mathbb{P}^n_{\mathfrak{w}',k}$ with $\mathfrak{w}' = \left(\frac{q_0}{d_i}, \dots, \frac{q_{i-1}}{d_i}, q_i, \frac{q_{i+1}}{d_i}, \dots, \frac{q_n}{d_i}\right)$.
(iii) Any weighted projective space is isomorphic to a reduced and well-formed one.
(iv) If $\mathfrak{w}$ is reduced and all of m/q_i are coprime, where $m = \operatorname{lcm}(q_0, \cdots, q_i)$, then $\phi_m : \mathbb{P}^n_{\mathfrak{w},k} \to \mathbb{P}^n_k$, via

$$\phi_m([x_0, \dots, x_n]) = [x_0^{m/q_0}, x_1^{m/q_1}, \dots, x_n^{m/q_n}]. \tag{7}$$

is an isomorphism. The map in Eq. (7) is called the **Veronese map**.

3.3. **Heights on weighted projective spaces.** For any point $\mathfrak{p} = [x_0 : \cdots : x_n] \in \mathbb{P}^n_{\mathfrak{w},k}$ we can assume, without loss of generality, that $\mathfrak{p} = [x_0 : \cdots : x_n] \in \mathbb{P}^n_{\mathfrak{w},k}(\mathcal{O}_k)$. The height for weighted projective spaces will be defined in the next section.

Let $\mathfrak{w} = (q_0, \dots, q_n)$ be a set of weights and $\mathbb{P}^n_{\mathfrak{w},k}$ the weighted projective space over a number field k. Let $\mathfrak{p} \in \mathbb{P}^n_{\mathfrak{w},k}$ a point such that $\mathfrak{p} = [x_0, \dots, x_n]$. We define the **weighted multiplicative height** of $\mathfrak{p}$ as

$$\mathcal{S}_{\mathrm{k}}(\mathfrak{p}) := \prod_{v \in M_k} \max\left\{ |x_0|_v^{\frac{n_v}{q_0}}, \dots, |x_n|_v^{\frac{n_v}{q_n}} \right\}. \tag{8}$$

The **logarithmic height** of the point $\mathfrak{p}$ is defined as follows

$$\mathfrak{s}_{\mathrm{k}}(\mathfrak{p}) := \log \mathcal{S}_{\mathrm{k}}(\mathfrak{p}) = \sum_{v \in M_k} \max_{0 \leq j \leq n} \left\{ \frac{n_v}{q_j} \cdot \log |x_j|_v \right\}. \tag{9}$$

$\mathcal{S}_{\mathrm{k}}(\mathfrak{p})$ is well defined and $\mathcal{S}_{\mathrm{k}}(\mathfrak{p}) \geq 1$ for any $\mathfrak{p} \in \mathbb{P}^n_{\mathfrak{w},k}$, see [12].

Let $\mathbb{P}_{\mathfrak{w},k}$ be a well-formed weighted projective space and $\mathbf{x} = [x_0 : \cdots : x_n] \in \mathbb{P}_{\mathfrak{w},k}(k)$. Assume $\mathbf{x}$ normalized (i.e. $\operatorname{wgcd}_k(\mathbf{x}) = 1$). Clearly $\operatorname{wgcd}(\mathbf{x}) | \gcd(x_0, \ldots, x_n)$ and therefore $\operatorname{wgcd}(\mathbf{x}) \leq \gcd(x_0, \ldots, x_n)$.

Remark 1. *Let $\mathbb{P}_{\mathfrak{w},k}$ be a well-formed weighted projective space and $\mathbf{x} = [x_0 : \cdots : x_n] \in \mathbb{P}_{\mathfrak{w},k}(k)$ such that $\mathbf{x}$ is absolutely normalized. Then $\gcd(x_0, \ldots, x_n) = 1$.*

Proof. Let $\mathfrak{w} = (q_0, \ldots, q_n$, $m = \operatorname{lcm}(q_0, \ldots, q_n)$, $\phi : \mathbb{P}_{\mathfrak{w},k} \to \mathbb{P}^n$ Veronese embedding. Assume $\gcd(x_0, \ldots, x_n) = d > 1$. Then $\gcd(\phi(x_0), \ldots, \phi(x_n)) = d$ and $\overline{\operatorname{wgcd}}(\mathbf{x}) = d^{\frac{1}{m}} > 1$, which is a contradiction. □

If $\mathbf{x} = [x_0 : \ldots, x_n]$ is a normalized point then by definition of the height

$$\mathcal{S}_{\mathrm{k}}(\mathbf{x}) = \max_{i=0}^{n} \{|x_i|^{\frac{1}{q_i}}\}$$

Assume now that $\mathbf{x} = [\lambda^{q_0} x_0 : \cdots : \lambda^{q_n} x_n]$ such that $\lambda = \operatorname{wgcd}(\lambda^{q_0} x_0 : \cdots : \lambda^{q_n} x_n)$. Denote by s the index where $\min_j \{|\lambda^{q_i} x_j|^{\frac{1}{q_j}}\} = \lambda \min_j \{|x_j|^{\frac{1}{q_j}}\}$ is obtained. Then

$$\frac{1}{\lambda (x_s)^{1/q_s}} \star \mathbf{x} = \left[\frac{x_0}{x_s^{q_0/q_s}} : \cdots : 1 : \cdots : \frac{x_n}{x_s^{q_n/q_s}} \right] =: \mathbf{y} \tag{10}$$

where 1 is in the s position. Simplify all coordinates in Eq. (10). Multiplying $\mathbf{y}$ by $(x_s)^{\frac{1}{q_s}}$ we have

$$(x_s)^{\frac{1}{q_s}} \star \mathbf{y} = [x_0 : \cdots : x_n],$$

which is now a normalized point. Hence

$$\mathcal{S}_{\mathrm{k}}(\mathbf{x}) = \frac{\max_{i=0}^{n} \{|\lambda^{q_i} x_i|^{\frac{1}{q_i}}\}}{\min_{i=0}^{n} \{|\lambda^{q_i} x_i|^{\frac{1}{q_i}}\}}$$

Remark 2. *Notice that $\mathcal{S}_{\mathrm{k}}(\mathbf{x})$ is given by*

$$\mathcal{S}_{\mathrm{k}}(\mathbf{x}) = \frac{\max_i |x_i|^{\frac{1}{q_i}}}{\min_j |x_j|^{\frac{1}{q_j}}}, \tag{11}$$

Theorem 2. *Let $\mathbf{x} = [x_0 : \cdots : x_n] \in \mathbb{P}_{\mathfrak{w}}(\mathbb{Q})$ be an absolutely normalized point. Then*

$$\gcd(x_0, \ldots, x_n) \leq \mathcal{S}_{\mathrm{k}}(\mathbf{x})$$

Proof. Let $\mathbf{x} = [x_0 : \ldots, x_n]$ be a normalized point. Denote by s the index where $\min_j |x_j|^{\frac{1}{q_j}}$ is obtained and by r the index where $\max_i |x_i|^{\frac{1}{q_i}}$ is obtained. Hence

$$\mathcal{S}_{\mathrm{k}}(\mathbf{x}) = \frac{|x_r|^{\frac{1}{q_r}}}{|x_s|^{\frac{1}{q_s}}}$$

Let $d := \gcd(x_0, \ldots, x_n)$. Then

$$\gcd(x_0, \ldots, x_n) := \prod_{p \in \mathbb{Z}} p^{\min\{\nu_p(x_0), \ldots, \nu_p(x_n)\}}$$

Assume $d > \mathcal{S}_{\mathrm{k}}(\mathbf{x})$. There must be at least one $l \in \{0, \ldots, n\}$ such that

$$d \cdot |x_s|^{\frac{1}{q_s}} < |x_l|^{\frac{1}{q_l}},$$

otherwise $\operatorname{wgcd}(x_0, \ldots, x_n) > 1$. Hence we have

$$\frac{1}{\lambda} |x_i|^{\frac{1}{q_i}} < d < |x_s|^{\frac{1}{q_s}},$$

So

$$|x_s|^{\frac{1}{q_s}} > \frac{1}{\lambda} |x_i|^{\frac{1}{q_i}} \implies \frac{1}{\lambda} |x_s|^{\frac{1}{q_s}} > \frac{1}{\lambda} \mathcal{S}_{\mathrm{k}}(\mathbf{x})$$

□

3.4. **Stability.** Let G be an algebraic group acting rationally on a variety $\mathcal{X}$ (that is, through a morphism $G \times \mathcal{X} \to \mathcal{X}$, say $(g, x) \to g.x$. We write $G.x$ for the orbit

$$G.x = \{y \in \mathcal{X} : y = g.x \text{ for some } g \in G\}$$

of x. Assume that G is a **reductive** group.

Let $\mathcal{X} \subset \mathbb{P}^d_k$ and G act linearly on $\mathcal{X}$. Hence we can assume $G \leq \mathrm{GL}_2(k)$ acting on $\mathcal{X}$ in the natural way and $I \in k[a_0, \ldots, a_d]$ a G-invariant polynomial. By $\mathcal{X}_I \subset \mathbb{P}^d(k)$ we denote the set $\mathcal{X}_I := \{\mathbf{b} \in \mathcal{X} \,|\, I(\mathbf{b}) \neq 0\}$.

Definition 1. *A point $\alpha \in \mathcal{X}$ is called **stable under the G-action** if α has a finite stabilizer G_α and there exist a G-invariant $I \in k[a_0, \ldots, a_d]$ such that $\alpha \in \mathcal{X}_I$.*

*If we drop the condition that the stabilizer G_α is finite then $\alpha \in \mathcal{X}$ is called **semistable under the G-action**.*

A binary form $f(x, y) \in k[x, y]$ of degree $\deg f = d$ is stable if and only if all roots of f are of multiplicity $< \frac{d}{2}$ and semistable if and only if all roots are of multiplicity $\leq \frac{d}{2}$.

Definition 2. *If a degree $d \geq 2$ binary form $f(x, y)$ has roots of multiplicity $\frac{d}{2}$ we say that f is **strictly semistable**.*

A binary form $f(x, y)$ of degree $\deg f = d$ is unstable if and only if $\xi(f) = \mathbf{0}$ in $\mathbb{P}^n_{\mathfrak{w},k}$. Moreover, if d is even there is only one strictly semistable point in the moduli space and there are no such points when d is odd. The following was shown in [6]. Let $d \geq 3$, k be a number field, $f \in V_d$ an integral form defined over k, and $\mathfrak{p} = \xi(f) \in \mathbb{P}^n_{\mathfrak{w},k}$ the moduli point in the corresponding weighted projective space. If f is semistable, then $\mathfrak{s}(\xi(f)) \geq 0$. Moreover, for every $d \geq 4$, there exist exactly one integral binary form $g \in V_d$, defined over k, such that $\mathfrak{s}(\xi(g)) = 0$. If f is strictly semistable then $d = \deg f$ is even and its absolute weighted moduli height $\mathcal{S}(\xi(f)$ and absolute logarithmic weighted height $\mathfrak{s}(\xi(f))$, for $d = 4, 6, 8, 10$ are determined in [6, Table 1].

4. Generating invariants

Our goal is to find numerical evidence on the number of binary forms of degree $d > 2$ with bounded weighted moduli height. Since weighted moduli height is defined in terms of the generating invariants of the ring of invariants $\mathcal{R}_d$, we display such invariants for $d \leq 10$; see [6] for details.

Let $f(x, y)$ be a degree $d > 2$ binary form as in Eq. (3). The ring of invariants of degree > 2 binary forms is denoted by $\mathcal{R}_d$ and $(f, g)_r$ denotes the r-transvection between two binary forms f and g. Below we list generating sets $\xi = (\xi_0, \ldots, \xi_n)$, for $\mathcal{R}_d$, $d = 3, \ldots, 10$.

4.0.1. *Cubics.* A generating set for $\mathcal{R}_3$ is $\xi = \{\xi_0\}$,

$$\xi_0 = \frac{1}{2}\left((f, f)_2, (f, f)_2\right)_2 = \frac{2}{3}a_1a_3a_0a_2 - \frac{4}{27}a_1^3a_3 - \frac{4}{27}a_2^3a_0 + \frac{1}{27}a_2^2a_1^2 - a_0^2a_3^2$$

4.0.2. *Quartics.* A generating set for $\mathcal{R}_4$ is $\xi = [\xi_0, \xi_1]$ with $\mathfrak{w} = (2, 3)$, where

$$\xi_0 = a_4a_0 - \frac{a_1a_3}{4} + \frac{a_2^2}{12},\ \xi_1 = a_2a_4a_0 - \frac{3}{8}a_1^2a_4 - \frac{3}{8}a_0a_3^2 + \frac{1}{8}a_2a_1a_3 - \frac{1}{36}a_2^3$$

4.0.3. *Quintics.* A generating set for $\mathcal{R}_5$ is $\xi = [\xi_0, \xi_1, \xi_2]$ with $\mathfrak{w} = (4, 8, 12)$, where

$$c_1 = (f, f)_4, \qquad c_2 = (f, f)_2, \qquad c_3 = (f, c_1)_2, \qquad c_4 = (c_3, c_3)_2$$
$$\xi_0 = \frac{1}{2}(c_1, c_1)_2, \quad \xi_1 = (c_4, c_1)_2, \quad \xi_2 = (c_4, c_4)_2,$$

4.0.4. *Sextics.* Let $c_1 = (f, f)_4$, $c_3 = (f, c_1)_4$, $c_4 = (c_1, c_1)_2$. A generating set for $\mathcal{R}_6$ is $\xi = [\xi_0, \xi_1, \xi_2, \xi_3]$ with weights $\mathfrak{w} = (2, 4, 6, 10)$, where

$$\xi_0 = \frac{1}{2}(f, f)_6,\ \xi_1 = \frac{1}{2}(c_1, c_1)_4,\ \xi_2 = \frac{1}{2}(c_4, c_1)_4,\ \xi_3 = (c_4, c_3^2)_4.$$

4.0.5. *Septics.* A generating set of $\mathcal{R}_7$ is given by $\xi = [\xi_0, \xi_1, \xi_2, \xi_3, \xi_4]$ with weights $\mathfrak{w} = (4, 8, 12, 12, 20)$. We define them as $c_1 = (f, f)_6$, $c_2 = (f, f)_4$, $c_4 = (f, c_1)_2$,

$$c_5 = (c_2, c_2)_4, \quad \xi_2 = \frac{1}{16}((c_5, c_5)_2, c_5)_4, \quad c_7 = (c_4, c_4)_4, \quad \xi_0 = \frac{1}{2}(c_1, c_1)_2,$$
$$\xi_1 = (c_7, c_1)_2, \quad \xi_3 = \left((c_4, c_4)_2, c_1^3\right)_6, \qquad \xi_4 = \frac{1}{64}\left([(c_2, c_5)_4]^2, (c_5, c_5)_2\right)_4$$

4.0.6. *Octavics.* A generating set of $\mathcal{R}_8$ is given by $\xi = [\xi_0, \xi_1, \xi_2, \xi_3, \xi_4, \xi_5]$ with weights $\mathfrak{w} = (2, 3, 4, 5, 6, 7)$. We define them as follows. Let $c_1 = (f, f)_6$, $c_2 = (f, c_1)_4$, $c_3 = (f, f)_4$, $c_5 = (c_1, c_1)_2$. Then invariants are:

$$\xi_0 = \frac{1}{2}(f, f)_8, \qquad \xi_1 = (f, c_3)_8, \qquad \xi_2 = \frac{1}{2}(c_1, c_1)_4,$$
$$\xi_3 = (c_1, c_2)_4, \qquad \xi_5 = \frac{1}{2}(c_5, c_1)_4, \qquad \xi_6 = ((c_1, c_2)_2, c_1)_4.$$

4.0.7. *Nonics.* A generating set of $\mathcal{R}_9$ is given by $\xi = [\xi_0, \xi_1, \xi_2, \xi_3, \xi_4, \xi_5, \xi_6]$ with weights $\mathfrak{w} = (4, 8, 10, 12, 12, 14, 16)$; see [6] for their definitions.

4.0.8. *Decimics.* A generating set of $\mathcal{R}_{10}$ is given by $\xi = [\xi_0, \xi_1, \xi_2, \xi_3, \xi_4, \xi_5, \xi_6, \xi_7, \xi_8]$ with weights $\mathfrak{w} = (2, 4, 6, 6, 8, 9, 10, 14, 14)$; see [6] for their excplicit definitions.

Problem 3. *Determine an equivariant neural network which will determine a minimal set of invariants for any given d.*

5. Number of binary forms with bounded weighted moduli height

Let $\xi = (\xi_0, \dots, \xi_n)$ be a generating set of $\mathcal{R}_d$ and with weights $\mathfrak{w} = (q_0, \dots, q_n)$. Recall that $\mathbb{P}^n_{\mathfrak{w}}$ denotes the weighted projective space over $\mathbf{C}$ (say $\mathbb{P}^n_{\mathfrak{w},\mathbf{C}}$) and $\mathbb{P}^n_w(\mathbb{Q})$ the set of points with rational coordinates. Notice that for each $\mathfrak{p} \in \mathbb{P}^n_{\mathfrak{w}}(\mathbb{Q})$ we can assume $\mathfrak{p} = [x_0, \dots, x_n]$, where $x_i \in \mathbb{Z}$, for all $i = 0, \dots, n$.

Fix $h \in \mathbb{R}^{\geq 0}$ and let

$$B_h := \{\mathfrak{p} \in \mathbb{P}^n_{\mathfrak{w}} \mid \mathcal{S}_{\mathrm{k}}(p) \leq h)\}, \quad C_h := \{\mathfrak{p} \in \mathbb{P}^n_{\mathfrak{w}} \mid h - 1 < \mathcal{S}_{\mathrm{k}}(p) \leq h)\},$$

Let $F_d : [0, \infty) \to \mathbb{Z}^{\geq 0}$ be the function which denotes the cardinality of B_h and $G_d : [1, \infty) \to \mathbb{Z}^{\geq 0}$ the cardinality of C_h. Then,

$$G_d(h) = F_d(h) - F_d(h - 1).$$

From Northcott's theorem for weighted heights (see [1, Theorem 1]) $F(h)$ is well defined.

Theorem 3. *Let $\mathfrak{w} = (q_0, \dots, q_n$ and $\mathbb{WP}^n_{\mathfrak{w},\mathbb{Q}}$ a well-formed weighted projective space. The number of points in $\mathbb{WP}^n_{\mathfrak{w},\mathbb{Q}}$ with height less or equal to a number h is*

$$F_d(h) \leq \sum_{i=0}^{n} \left(h^{q_{n-i}} \cdot \prod_{j=0}^{n-i} (2h^{q_j} + 1) \right)$$

Proof. Assume $\mathcal{S}_{\mathrm{k}}(p) \leq h$. For each $i = 0, \dots n$ we have

$$|x_i|^{\frac{1}{q_i}} \leq h \implies |x_i| \leq h^{q_i}$$

Hence, there are $2h^{q_i} + 1$ choices for x_i. Moreover, we can normalize one of the coordinates so it is always positive. We can do that for the highest power, which by our ordering in Eq. (4) is q_n. Then there will be only $h^{q_n} + 1$ choices for that coordinate. Hence, our total number is bounded by

$$(h^{q_n} + 1) \cdot \prod_{i=0}^{n-1} (2h^{q_i} + 1)$$

Since we are counting stable points in the moduli space of binary forms, then at least one of the coordinates must be nonzero; see [6]. Assume $x_n \neq 0$. Then there are h^{q_n} choices for x_n and the number of such forms is less than

$$h^{q_n} \cdot \prod_{j=0}^{n-1} (2h^{q_j} + 1).$$

If $x_n = 0$ then by the same argument there are $h^{q_{n-1}} \cdot \prod_{j=0}^{n-2} (2h^{q_j} + 1)$. and so on. Adding up all the cases we have

$$F_d(h) \leq \sum_{i=0}^{n} \left(h^{q_{n-i}} \cdot \prod_{j=0}^{n-i} (2h^{q_j} + 1) \right)$$

This completes the proof. □

Remark 3. *Notice that the above method counts as different moduli points tuples $\lambda \star (x_0, \ldots, x_n)$ as long as $\max\{|\lambda^{q_i} x_i|^{1/q_i}\} \leq h$. Formula is precise only for $h = 1$ as it will be seen in computations with binary sextics. Consider for example $[1:0:0:0]$ and $[2:0:0:0]$ in $\mathbb{P}_{(1,2,3,5)}$. They are the same point but counted separately by the above formula when $h > \sqrt{2}$.*

Next we determine such bound for binary forms of degree ≤ 10.

Lemma 1. *For $d = 4, \ldots, 10$ the moduli space of binary forms as a well formed space and values of $F_d(h)$ are given in Table 1.*

TABLE 1

d	$\mathfrak{w} = (q_0, \ldots, q_n)$	$\mathfrak{w}'$	$F_d(1)$
4	$\mathfrak{w} = (2,3)$	$\mathfrak{w} = \mathfrak{w}'$	3
5	$(4,8,12)$	*(1,2,3)*	3^2
6	$w = (2,4,6,10)$	*(1,2,3,5)*	3^3
7	$(4,8,12,12,20)$	*(1,2,3,3,5)*	3^4
8	$(2,3,4,5,6,7)$	$\mathfrak{w} = \mathfrak{w}'$	3^5
9	$(4,8,10,12,12,14,16)$	$(2,4,5,6,6,7,8)$	3^6
10	$(2,4,6,6,8,9,10,14,14)$	$\mathfrak{w} = \mathfrak{w}'$	3^8

Proof. Cases $d = 4, 8, 10$ are already well formed spaces. Let $d = 5$. Following the definition of the Veronese map from Eq. (7),

$$d_0 = \gcd(8,12) = 4,\ d_1 = \gcd(4,12) = 4,\ d_2 = \gcd(4,8) = 4,$$

and $\alpha_0 = \operatorname{lcm}(d_1, d_2)$, $\alpha_1 = \operatorname{lcm}(d_0, d_2)$, $\alpha_2 = \operatorname{lcm}(d_0, d_1)$, are

$$\alpha_0 = \alpha_1 = \alpha_2 = 4, \quad \text{and} \quad a = \operatorname{lcm}(d_0, d_1, d_2) = \operatorname{lcm}(4,4,4) = 4.$$

The new set of weights is $\mathfrak{w}' = (q_0', q_1', q_2')$, where $q_i' = \frac{q_i}{\alpha_i}$. Hence, $\mathfrak{w}' = (1,2,3)$. The morphism $\mathbb{P}^3_{(4,8,12),k} \to \mathbb{P}^3_{(1,2,3),k}$, given by $[x_0 : x_1 : x_2] \to [y_0 : y_1 : y_2] = \left[x_0^4 : x_1^4 : x_2^4\right]$ is an isomorphism. Then $q = 2 \cdot 3 = 6$ and the Veronese embedding is $[J_4 : J_8 : J_{12}] \longrightarrow \left[J_4^6 : J_8^3 : J_{12}^2\right]$.

Let $d = 6$ and consider the weighted projective moduli space of binary sextics. It is isomorphic to $\mathbb{P}^3_{\mathfrak{w},k}$ for $\mathfrak{w} = (2,4,6,10)$. Following the definition of the Veronese map from Eq. (7), we let

$$d_0 = \gcd(4,6,10) = 2,\ d_1 = \gcd(2,6,10) = 2,$$
$$d_2 = \gcd(2,4,10) = 2,\ d_3 = \gcd(2,4,6) = 2$$

and $\alpha_0 = \operatorname{lcm}(d_1, d_2, d_3)$, $\alpha_1 = \operatorname{lcm}(d_0, d_2, d_3)$, $\alpha_2 = \operatorname{lcm}(d_0, d_1, d_3)$, $\alpha_3 = \operatorname{lcm}(d_0, d_1, d_2)$, are

$$\alpha_0 = \alpha_1 = \alpha_2 = \alpha_3 = 2, \quad \text{and} \quad a = \operatorname{lcm}(d_0, d_1, d_2, d_3) = \operatorname{lcm}(2,2,2,2) = 2.$$

The new set of weights is $\mathfrak{w}' = (q_0', q_1', q_2', q_3',)$, where $q_i' = \frac{q_i}{\alpha_i}$. Hence, $\mathfrak{w}' = (1,2,3,5)$.

Let $d = 7$. Then $d_0 = \ldots = d_4 = 4$. Then $\alpha_0 = \ldots = \alpha_4 = 4$, and $a = 4$. The set of new weights is $q_i = \frac{q_i}{\alpha_i}$, so we have $\mathfrak{w}' = (1,2,3,3,5)$.

Let $d = 9$. Then $d_0 = \ldots = d_6 = 2$ and $\alpha_0 = \ldots = \alpha_6 = 2$. Then the set of weights is $\mathfrak{w}' = (2,4,5,6,6,7,8)$. This completes the proof. □

Define the function $f_d(h) := F_d'(h) + 1$, then $f_d(h)$ represents the number of binary forms for a given height h.

Remark 4. *How good is the above bound for $F_d(h)$? For what h the formula is precise?*

TABLE 2. Number of binary sextics up to equivalence

h	# of points in B_6
1	40
2	24 862
3	1 781 202
4	39 251 668
5	440 104 780
6	3 195 496 050
7	17 146 927 462
8	73 657 853 512
9	266 816 523 888
10	844 626 323 110

6. Moduli space $\mathcal{M}_2$ of genus 2 curves

We showed in the previous section that the moduli space of binary sextics is isomorphic to $\mathbb{P}_{(1,2,3,5)}$. Moreover the invariant J_{10} of degree 10 is the discriminant of the sextic and therefore $J_{10} \neq 0$. Thus, the morphism $\mathbb{P}^3_{(2,4,6,10),k} \to \mathbb{P}^3_{(1,2,3,5),k}$, given by

$$[x_0 : x_1 : x_2 : x_3] \to [y_0 : y_1 : y_2 : y_3] = \left[x_0^2 : x_1^2 : x_2^2 : x_3^2\right] \tag{12}$$

is an isomorphism.

Since we want to design a model where the incoming features will be a genus two curve, then equivalently this means a point $[x_0 : x_1 : x_2 : x_3] \in \mathbb{P}_{(1,2,3,5)}$. Equivalently the input could be the equation of the curve, but this poses no issue for $g = 2$ since we can compute invariants. In general this would be a major issue for higher genus g.

There is also an issue to address when it comes to finding the "smallest" representatives for the equivalence class $[x_0 : x_1 : x_2 : x_3]$. Theoretically this is handled in [1], but that would require computing weighted greatest common divisors and that could be very costly for large coordinates $x_0, \ldots, x_3$.

Since $q = 1 \cdot 2 \cdot 3 \cdot 5 = 30$, the Veronese embedding is

$$[J_2 : J_4 : J_6 : J_{10}] \longrightarrow \left[J_2^{30} : J_4^{15} : J_6^{10} : J_{10}^6\right] = \left[\frac{J_2^{30}}{J_{10}^6} : \frac{J_4^{15}}{J_{10}^6} : \frac{J_6^{10}}{J_{10}^6} : 1\right] \tag{13}$$

So the triple $i_1 = \frac{J_2^{30}}{J_{10}^6}$, $i_2 = \frac{J_4^{15}}{J_{10}^6}$, $i_3 = \frac{J_6^{10}}{J_{10}^6}$ uniquely determines the equivalence class. Ideally we would create a dictionary with keys (i_1, i_2, i_3), but these numbers blow up very quickly which makes and significant computations impossible. If the rational numbers have a significant number of decimal places, their exact representation might be lost when converted to floating-point format.

We will use a very simplistic model for now and see how things work out.

```
in_features:   Igusa invariants
out_features:  weighted height, Fine/Coarse, Automorphism group,
```

An entry in the dictionary looks like:

```
(x,y, z) :  ( [J_2, J4, J6, J10], weighted_height, obstruction, AutGroup)
```

where

```
# M_dict data with data types
M_dict = {
    # (x, y, z): ([a, b, c, d], wh, label, [m,n] ), where
    # x, y, z are float32,
    # a, b, c, d are int,
    # wh is float32,
    # label is bool,
```

```
    # m,n are int
    (x1, y1, z1): ([1, 2, 3, 4], 1.23,  True, [2,1] ),
    (x2, y2, z2): ([5, 6, 7, 8], 2.54,  False, [4,2] )
    # ... other entries
}
```

LISTING 1. M_dict data with data types

In a second stage we intend to include other properties of genus two curves as whether or not the curve is of CM-type, splitting (n, n) of the Jacobian, torsion subgroup, etc.

6.1. **Another look on the number of moduli points with bounded moduli height.**

Lemma 2. *The number of points in $\mathcal{M}_2$ with weighted moduli height $\leq h$*

$$\begin{aligned} F_6(h) &\leq h^5\big(2\,h^3+1\big)\big(2\,h^2+1\big)(2\,h+1)+h^3\big(2\,h^2+1\big)(2\,h+1)+h^2(2\,h+1)+h \\ &= h\big(8\,h^{10}+4\,h^9+4\,h^8+6\,h^7+2\,h^6+6\,h^5+3\,h^4+2\,h^3+3\,h^2+h+1\big) \end{aligned}$$

Moreover, there are exactly 27 genus 2 curves with weighted moduli height $\mathcal{S}_{\rm k}=1$.

Proof. The invariant with the highest degree is the discriminant of the binary form. Since this binary form correspond to a hyperelliptic curve, the discriminant is not zero. Hence, $x_{q_n} \neq 0$. Then, there are only h^{q_n} choices for x_{q_n}; see the proof of Theorem 3. This completes the proof. □

In Table 3 we display all points $\mathfrak{p} \in \mathbb{WP}^3_{(1,2,3,5)}(\mathbb{Q}) \setminus \{J_{10}=0\}$ with weighted moduli heights h = 1.

TABLE 3. Moduli points $\mathfrak{p}=[J_2 : J_4 : J_6 : J_{10}]$ with weighted moduli height h = 1

#	$\mathfrak{p}$	#	$\mathfrak{p}$	#	$\mathfrak{p}$
1	[0, -1, 0, 1]	10	[1, 0, 1, 1]	19	[0, 1, 1, 1]
2	[0, 1, 0, 1]	11	[1, -1, -1, 1]	20	[1, 0, 1, -1]
3	[0, -1, 1, 1]	12	[1, 1, -1, 1]	21	[1, -1, -1, -1]
4	[0, 0, 0, 1]	13	[1, 1, 1, -1]	22	[1, 1, -1, -1]
5	[0, 0, 1, -1]	14	[1, -1, 1, -1]	23	[1, -1, 0, -1]
6	[0, 0, 1, 1]	15	[1, 1, 1, 1]	24	[1, 1, 0, -1]
7	[1, 0, -1, 1]	16	[1, 0, -1, -1]	25	[1, 1, 0, 1]
8	[1, 0, 0, -1]	17	[0, -1, 1, -1]	26	[1, -1, 0, 1]
9	[1, 0, 0, 1]	18	[0, 1, 1, -1]	27	[1, -1, 1, 1]

Let N_h denote the number of $\mathbb{Q}$-points $\mathfrak{p} \in \mathbb{WP}^n_{\mathfrak{w}}$ with height $h-1 < \mathcal{S}_{\rm k}(\mathfrak{p}) < h$. Thus, $N_h := B_h - B_{h-1}$. Notice that

$$N_h(0)=8, N_h(1)=5208, N_h(2)=160360$$

Notice that as the weighted moduli height h increases, the number N_h is not precise but only an estimate since our database is not complete.

6.2. **Distribution of fine points in $\mathcal{M}_2$.** There are two types of points in $\mathbb{WP}_{(1,2,3,5)}$, namely *fine points* and *coarse points*. Fine points are those points such that their field of moduli is a field of definition, while the rest of points are called coarse points.

We also classify fine points in two classes, those with extra automorphisms and those which have automorphism group isomorphic to the cyclic group of order 2.

6.3. **Genus two curves with extra involutions.** The set of points with extra automorphisms is a 2-dimensional irreducible subvariety of the moduli space corresponding exactly to points $p \in \mathbb{WP}_{(1,2,3,5)}$ which satisfying $J_{30}(p) = 0$. This locus has two 1-dimensional loci corresponding to points with automorphism group D_4 and D_6; for details see [10].

Problem 4. *Given a genus 2 curve in the form of a moduli point $p=[J_2, J_4, J_6, J_{10}]$, determine the properties of this curve such as the i) weighted height, ii) Fine or coarse, ii) automorphism group, iv) CM-type, and possibly others.*

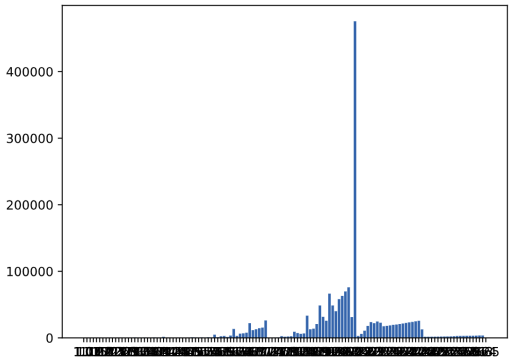

FIGURE 1. The number of points in the moduli for a given height. Data represents 850 000 points and the height is computed numerically

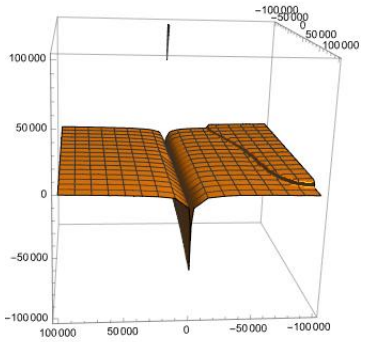

FIGURE 2. Curves with extra involutions

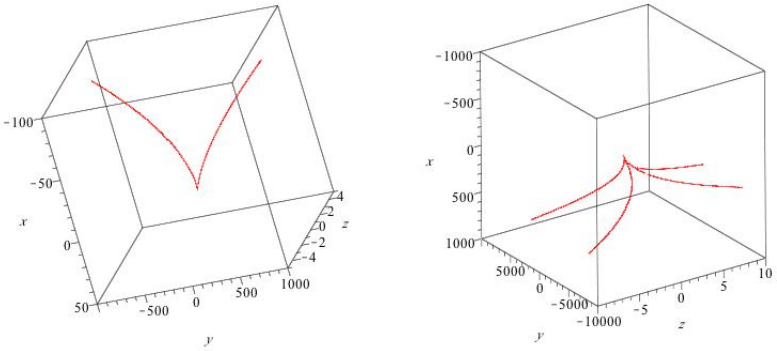

FIGURE 3. Curves with automorphism group D_4 and D_6

7. Determining a suitable machine learning model for our problem

Since our problem is outside of the typical problems studied by machine learning we experimented with several different models. The code is provided in the Appendix and also in the separate file.

7.1. **Sequential Model.** See Appendix A for details. This model has an accuracy of 99% in our tests and performed very well.

1. **Data Preparation:**
   - Extracts input features and labels from a dictionary `M_dict`.
   - Converts boolean labels to integers (0 or 1).
   - Converts the lists of features and labels into NumPy arrays.
2. **Normalization:**
   - Normalizes the input features using Min-Max scaling.
3. **Neural Network Definition:**
   - Defines a simple neural network model using Keras Sequential API.
   - The model has one hidden layer with 32 units and ReLU activation, and an output layer with 1 unit and sigmoid activation.
4. **Model Compilation:**
   - Compiles the neural network model using the Adam optimizer with a specified learning rate and binary cross-entropy loss.
5. **Training Loop:**
   - Trains the model for a total of 5 epochs (specified in the loop header).
   - In each epoch, the model is trained for one epoch using the training data.
   - Loss and accuracy values are recorded after each epoch.
6. **Loss and Accuracy Visualization:**
   - Plots two graphs:
     - Training loss over epochs.
     - Training accuracy over epochs.

7.2. **Equivariant Neural Network.** See Appendix B for details.

The equivariant neural network provides probably the most efficient method to study this type of problem. Unfortunately, we were not able to run it with the full data because there were problems with over floating. The big size of our coordinates makes this method rather challenging to run. When we cleaned the data (deleted all entries with coordinates $> 10^3$) we had 66% accuracy.

1. **Custom Dataset Definition:**
   - Defines a custom dataset class `CustomDataset` inheriting from PyTorch's `Dataset`.
   - Initializes the dataset with a dictionary and converts dictionary items into a list.
2. **Neural Network Definition:**
   - Defines a simple neural network class `EquivariantNet` inheriting from PyTorch's `nn.Module`.
   - The neural network includes three linear layers with ReLU activations, considering coordinates.
3. **Data Loading and Model Instantiation:**
   - Creates an instance of `CustomDataset` named `custom_dataset` using a dictionary `M_dict`.
   - Sets up a data loader (`data_loader`) to load data in batches from the custom dataset.
   - Instantiates the neural network model (`EquivariantNet`) based on the features of the first sample in the dataset.
4. **Model Training:**
   - Defines a mean squared error loss (`criterion`) and an Adam optimizer for training the model.
   - Runs a training loop for a specified number of epochs.
   - Within each epoch, iterates over batches of data from the data loader, performs forward and backward passes, clips gradients, and updates the model parameters.
   - Prints the average loss for each epoch.
5. **Loss Plotting:**
   - Plots the training loss over time using matplotlib.

7.3. **Transformers.** See Appendix C for the code. The model worked when we run it with a few hundred data points, but then very quickly would run out of memory for anything significant. It was the same even after we got the professional version of Colab.

1. **Data Preparation:**
   - Extracts input features and labels from a dictionary `M_dict` containing coordinates, input features, labels, and other information.
   - Converts the extracted features and labels into PyTorch tensors.
2. **Feature Mapping:**
   - Maps the original features to unique indices using `torch.unique`.
3. **Model Definition:**
   - Defines a transformer-based neural network model (`TransformerModel`) using PyTorch's `nn.Module`.
   - The model consists of an embedding layer, a transformer layer, global average pooling, and two fully connected layers with ReLU and sigmoid activation functions.
4. **Model Training:**
   - Instantiates the model with specified input size, hidden size, and output size.
   - Defines a binary cross-entropy loss (`criterion`) and an Adam optimizer for training the model.
   - Trains the model for a specified number of epochs using a loop.
   - Within each epoch, it performs forward and backward passes, updates the model parameters, and prints the loss.

8. Results

Let us now see what is the distribution of fine points with automorphisms in out database (red points). From a computational point of view it is quite hard to do this simply by brute force for our database which has about 500 000 points. Instead we will use the above models to see what information we can gather and then prove our results (if any) via brute force computationally.

By taking a random sample and graphing all red points we get the following picture.

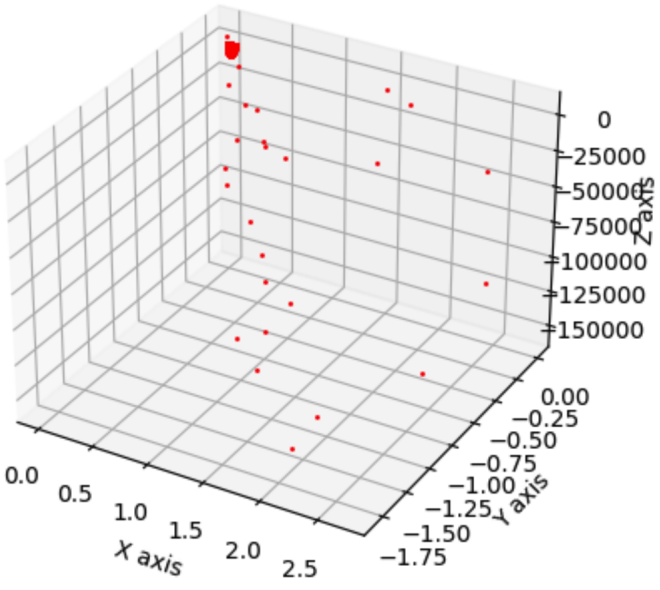

Figure 4. Distribution of points with automorphisms

Remark 5. *There are two quick points to be noticed:*

First, the red points seem to be very scarce around the origin of the coordinate system and secondly there are no green points (fine points with trivial automorphism group). Somewhat to be expected by people who have extensive computational experience with the moduli space of genus 2 curves, but not any obvious theoretical reason for it.

8.1. **Coarse moduli points in $\mathcal{M}_2$.** Note that in Table 3 the points 1 to 15 are fine moduli points, i.e. with methods that we will explain in the upcoming subsection we can compute their equations defined over $\mathbb{Q}$. We graph below all points for weighted height $\mathcal{S}_{\rm k}(p) \leq 3$.

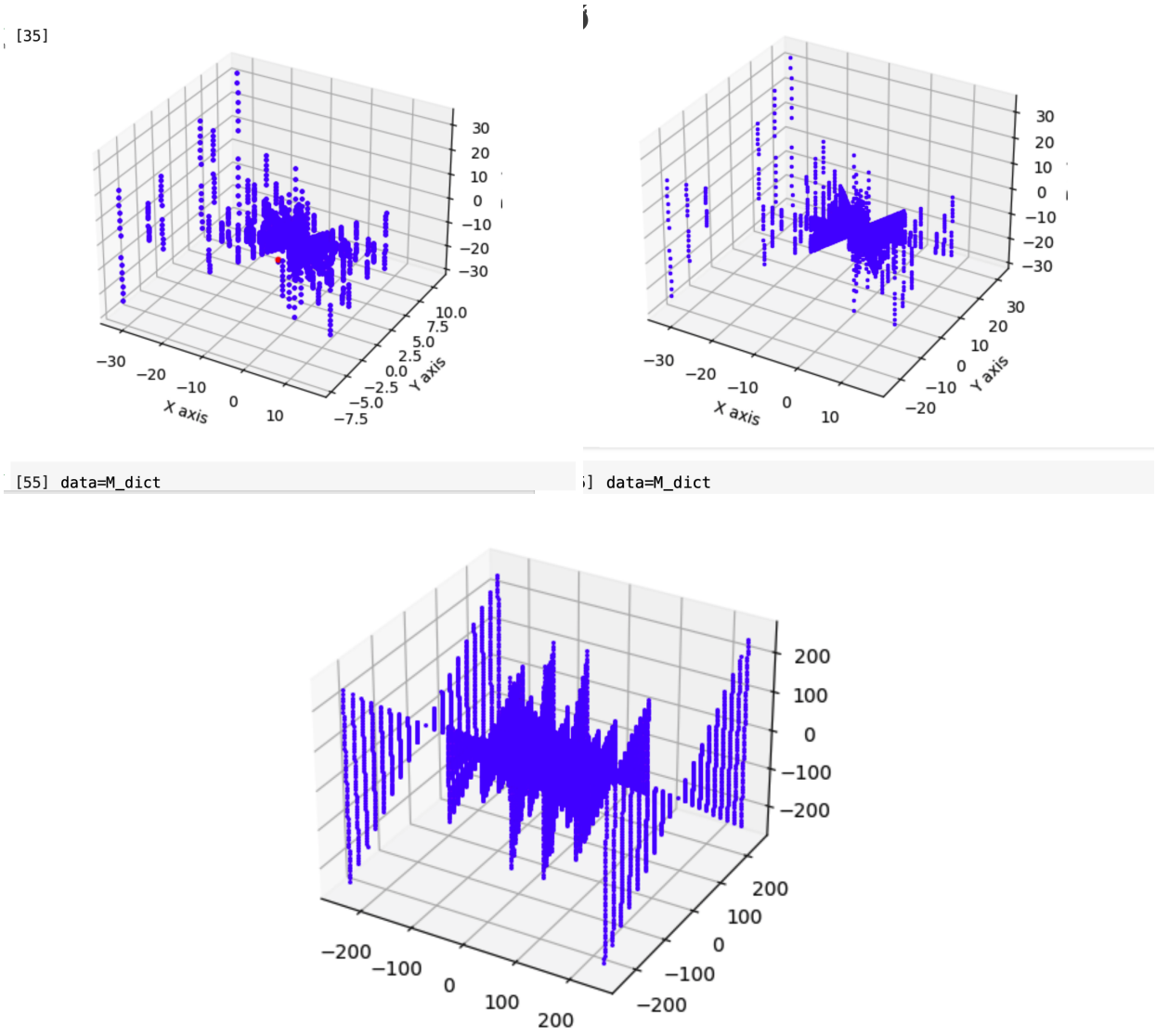

FIGURE 5. Graph of rational points of weighted height $\mathcal{S}_{\rm k} \leq 3$

Surprisingly there is only one red dot in all these graphs. These graphs were obtained using the sequential method and the existing red dot was no surprise because that is a very special genus 2 curve and well known. However, these graphs were a strong enough reason for us to go through all the cases for $\mathcal{S}_{\rm k}(p) \leq 3$ and check computationally.

Lemma 3. *There is only one genus two curve with weighted moduli height $\mathcal{S}_{\rm k} < 3$ defined over $\mathbb{Q}$, namely $y^2 = x(x^5 - 1)$, which is the single curve with automorphism group isomorphic to the cyclic group of order 10.*

Proof. Computationally compiling a list of all points with weighted height $\mathcal{S}_{\rm k}(p) \leq 3$ and checking each one of them.

□

The following is an immediate consequence of the above Lemma.

Corollary 1. *There are no rational points with weighted moduli height $\mathcal{S}_{\rm k} < 3$ in the locus $J_{30} = 0$ in $\mathbb{WP}_{(1,2,3,5)}$.*

9. Concluding remarks

With necessary adjustments, these results can be extended to a field of positive characteristic. The reader planning to attempt this must be aware that the system of generating invariants will change in certain characteristics. For example, an invariant of order eight, usually denoted by J_8, is needed for binary sextics.

Our models didn't always give us what we expected in terms of efficiency and at this point it is unclear to us if this is due to our limitations in computing power or limitations of the architectures chosen. This remains to be further investigated.

Our main goal was to test whether machine learning techniques can be used in theoretical areas of mathematics and we can definitely say that this modest project gave a positive answer to that question.

References

[1] L. Beshaj, J. Gutierrez, and T. Shaska, *Weighted greatest common divisors and weighted heights*, J. Number Theory **213** (2020), 319–346. MR4091944

[2] L. Beshaj, R. Hidalgo, S. Kruk, A. Malmendier, S. Quispe, and T. Shaska, *Rational points in the moduli space of genus two*, Higher genus curves in mathematical physics and arithmetic geometry, 2018, pp. 83–115. MR3782461

[3] Lubjana Beshaj, *Reduction theory of binary forms*, Advances on superelliptic curves and their applications, 2015, pp. 84–116. MR3525574

[4] B. J. Birch and H. P. F. Swinnerton-Dyer, *Notes on elliptic curves. I*, J. Reine Angew. Math. **212** (1963), 7–25. MR146143

[5] ———, *Notes on elliptic curves. II*, J. Reine Angew. Math. **218** (1965), 79–108. MR179168

[6] Elira Curri, *On the stability of binary forms and their weighted heights*, Albanian J. Math. **16** (2022), no. 1, 3–23. MR4448533

[7] J. Gutierrez and T. Shaska, *Hyperelliptic curves with extra involutions*, LMS J. Comput. Math. **8** (2005), 102–115. MR2135032

[8] K. Magaard, T. Shaska, S. Shpectorov, and H. Völklein, *The locus of curves with prescribed automorphism group*, 2002, pp. 112–141. Communications in arithmetic fundamental groups (Kyoto, 1999/2001). MR1954371

[9] A. Malmendier and T. Shaska, *A universal genus-two curve from Siegel modular forms*, SIGMA Symmetry Integrability Geom. Methods Appl. **13** (2017), Paper No. 089, 17. MR3731039

[10] ———, *From hyperelliptic to superelliptic curves*, Albanian J. Math. **13** (2019), no. 1, 107–200. MR3978315

[11] David Mumford and John Fogarty, *Geometric invariant theory*, Second, Ergebnisse der Mathematik und ihrer Grenzgebiete [Results in Mathematics and Related Areas], vol. 34, Springer-Verlag, Berlin, 1982. MR719371

[12] S. Salami and T. Shaska, *Local and global heights on weighted projective varieties and Vojta's conjecture*, Houston J. Math. (2023).

[13] T. Shaska, *Reduction of superelliptic Riemann surfaces* **776** (2022), 227–247. MR4375119

[14] T. Shaska and L. Beshaj, *Heights on algebraic curves*, Advances on superelliptic curves and their applications, 2015, pp. 137–175. MR3525576

Appendix A. Sequential Model

```python
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
# Data Preparation
coordinates = []
features = []
labels = []

for coords, (input_features, wh, label, _) in M_dict.items():
    coordinates.append(coords)
    features.append(input_features)
    labels.append(int(label))  # Convert boolean to int
features = np.array(features, dtype=np.float32)
labels = np.array(labels, dtype=np.float32)
scaler = MinMaxScaler()
features_normalized = scaler.fit_transform(features)
# Neural Network Architecture
model = Sequential()
model.add(Dense(32, input_shape=(4,), activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# Model Compilation
model.compile(optimizer=Adam(learning_rate=0.123), loss='binary_crossentropy', metrics=['
    accuracy'])
# Training Loop
losses = []
accuracies = []

for epoch in range(5):
    history = model.fit(features_normalized, labels, epochs=1, batch_size=32, verbose=0)
    losses.append(history.history['loss'][0])
    accuracies.append(history.history['accuracy'][0])
    print(f'Epoch [{epoch+1}/5], Loss: {losses[-1]:.4f}, Accuracy: {accuracies[-1]:.4f}')
# Visualization
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(losses, label='Training Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training Loss over Epochs')
plt.legend()
plt.subplot(1, 2, 2)
plt.plot(accuracies, label='Training Accuracy', color='orange')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training Accuracy over Epochs')
plt.legend()
plt.show()
```

Listing 2. Sequential Neural Network

Appendix B. Equivariant model

```python
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
import matplotlib.pyplot as plt

class CustomDataset(Dataset):
    def __init__(self, data_dict):
        self.data = list(data_dict.items())

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        key, value = self.data[idx]
        coordinates = torch.tensor(key, dtype=torch.float32)
        in_features = torch.tensor(value[0], dtype=torch.float32)
        out_features = torch.tensor(value[2], dtype=torch.float32)

        return coordinates, in_features, out_features

# Create an instance of the CustomDataset
custom_dataset = CustomDataset(M_dict)

# Use a larger batch size
batch_size = 64
data_loader = DataLoader(custom_dataset, batch_size=batch_size, shuffle=True)

# Define a simple equivariant neural network without e3nn
class EquivariantNet(nn.Module):
    def __init__(self, in_features, out_features):
        super(EquivariantNet, self).__init__()

        self.fc1 = nn.Linear(in_features + 3, 16)  # Add 3 for coordinates
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(16, 8)
        self.fc3 = nn.Linear(8, out_features)

    def forward(self, coordinates, in_features):
        # Concatenate coordinates and in_features
        x = torch.cat([coordinates, in_features], dim=-1)

        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        x = self.relu(x)
        x = self.fc3(x)

        return x

# Instantiate the model
in_features = len(next(iter(custom_dataset))[1])  # Use the first sample to determine
    in_features
out_features = len(next(iter(custom_dataset))[2])  # Use the first sample to determine
    out_features
model = EquivariantNet(in_features, out_features)

# Set up the loss function and optimizer
```

```
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

# Training loop
num_epochs = 10
losses = []  # to store the loss values for plotting

for epoch in range(num_epochs):
    model.train()
    total_loss = 0.0

    # Inside your training loop
    for coordinates, in_features, out_features in data_loader:
        optimizer.zero_grad()
        outputs = model(coordinates, in_features)
        loss = criterion(outputs, out_features)
        loss.backward()

        # Clip gradients before optimization step
        torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)

        optimizer.step()
        total_loss += loss.item()

    # Print average loss for the epoch
    average_loss = total_loss / len(data_loader)
    losses.append(average_loss)
    print(f'Epoch {epoch + 1}/{num_epochs}, Average Loss: {average_loss:.4f}')

# Plot the loss over time
plt.plot(losses, label='Training Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training Loss Over Time')
plt.legend()
plt.show()
```

Listing 3. Equivariant Neural Network

Appendix C. Transformer model

```
import torch
import torch.nn as nn
import torch.optim as optim

# Assuming M_dict is your data dictionary

# Extract input features and labels from the dictionary
coordinates = []
features = []
labels = []

for coords, (input_features, label, _) in M_dict.items():
    coordinates.append(coords)
    features.append(input_features)
    labels.append(int(label))  # Convert boolean to int

# Convert lists to PyTorch tensors
```

```
features = torch.tensor(features, dtype=torch.float32)
labels = torch.tensor(labels, dtype=torch.float32)

# Map the original features to indices
unique_features, inverse_indices = torch.unique(features, dim=0, return_inverse=True)

# Define the transformer model
class TransformerModel(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(TransformerModel, self).__init__()
        self.embedding = nn.EmbeddingBag(num_embeddings=len(unique_features), embedding_dim=
    hidden_size)
        self.transformer = nn.Transformer(d_model=hidden_size, nhead=4, num_encoder_layers=3)
        self.global_pooling = nn.AdaptiveAvgPool1d(1)  # Global average pooling
        self.fc1 = nn.Linear(hidden_size, 64)
        self.fc2 = nn.Linear(64, output_size)
        self.relu = nn.ReLU()
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.embedding(x, torch.zeros(x.size(0), dtype=torch.long))  # Use zeros as offsets
        x = x.view(1, x.size(0), -1)  # Add batch dimension and rearrange features
        x = self.transformer(x, x)  # Use x as both source and target sequences
        x = self.global_pooling(x.permute(0, 2, 1))  # Global average pooling
        x = x.squeeze(dim=-1)
        x = self.relu(self.fc1(x))
        x = self.sigmoid(self.fc2(x))
        return x

# Instantiate the model
model = TransformerModel(input_size=len(unique_features), hidden_size=32, output_size=1)

# Define loss and optimizer
criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Ensure labels is a 1D tensor
labels = labels.view(-1)

# Train the model
num_epochs = 10
for epoch in range(num_epochs):
    optimizer.zero_grad()
    outputs = model(inverse_indices)
    outputs = outputs.squeeze(dim=-1)  # Remove singleton dimension
    loss = criterion(outputs, labels)
    loss.backward()
    optimizer.step()
    print(f'Epoch [{epoch + 1}/{num_epochs}], Loss: {loss.item():.4f}')
```

Listing 4. TransformerModel

Department of Computer Science, Oakland University, Rochester, MI, 48309
Email address: elirashaska@oakland.edu

Department of Mathematics and Statistics, Oakland University, Rochester, MI, 48309
Email address: tanush@umich.edu