Ludwig-Maximilians-Universität München
Arnold Sommerfeld Center

Deep Learning Calabi-Yau Metrics

Master's Thesis
Theoretical and Mathematical Physics

Mathis Gerdes

First Referee:	Prof. Dr. Dieter Lüst
Second Referee:	Dr. Sven Krippendorf
Day of Defence:	24 September 2020

Abstract

No analytic expressions for the Ricci-flat metrics on compact Calabi-Yau manifolds are known, which has led to the development of multiple numerical approximation schemes. This thesis shows that a deep learning approach using the energy functionals introduced by Headrick and Nassar can replace existing methods to approximate Calabi-Yau metrics on projective varieties. Building on top of the algebraic metrics introduced for Donaldson's algorithm, the deep learning models introduced here can predict approximations to the Ricci-flat metric as a function of complex moduli parameters. A comparison with the benchmark of balanced metrics produced by Donaldson's algorithm indicates these approximations are of relatively higher accuracy, justifying it as an alternative, standalone approximation scheme. This approach is facilitated by modern machine learning frameworks, which provide efficient automatic differentiation that can be used to derive geometrical objects, and work with complicated, geometrically motivated loss functions.

Contents

1. Introduction

Calabi-Yau manifolds are compact complex Kähler manifolds with a Ricci-flat metric. Besides being of mathematical interest in their own right, they give a vacuum compactification for the heterotic string which may reproduce the physics of the Standard Model at low energies [1]. Unfortunately, there is no known analytic expression for the Ricci-flat metric on a compact Calabi-Yau manifold. While some physical properties can be deduced for the Calabi-Yau manifold in general, there are properties like the numerical value of the Yukawa couplings that depend on the metric [2]. The Yukawa couplings, in turn, determine the masses of the elementary particles that a top-down string theory would predict. Furthermore, the moduli of the Calabi-Yau manifold are a candidate for driving early universe inflation [3]. The exact form of the action of the four-dimensional effective field theory again depends on the metric, which motivates a search for numerical approximatinos to the Calabi-Yau metric. In the following, a new approach for finding moduli-dependent approximations to the Calabi-Yau metric using deep learning methods is explored.

The existing approaches to finding a Ricci-flat metric (or more generally one of constant scalar curvature) can be divided into two kinds. The first approach is to approximate the geometric objects of interest, such as the Kähler potential (these will be introduced in section 2 below), on a discretized lattice or a fixed set of points. This was first done by Headrick and Wiseman [4], using a relaxation method for the Monge-Ampère equation, which captures Ricci-flatness, on the K3 threefold. The second approach is based on an algebraic representation of the Kähler potential, developed by Donaldson in [5, 6, 7], and expanded on in [8, 9, 10].

The second approach has several advantages with regard to a machine learning, specifically deep learning, application. It provides a natural functional form for the Kähler potential characterized by a single Hermitian matrix h, without the need for explicit patches. These Hermitian matrices are defined for each integer value of the hyperparameter k, in which they grow polynomially. An algorithm introduced by Donaldson [5] provides an iterative scheme converging to the so called balanced metric for each fixed value of k. These balanced metrics can be shown to converge to the flat metric in the limit $k \to \infty$. The convergence can be quantified using several accuracy measures, introduced in section 2.4, which exploit properties of the Calabi-Yau manifold to evaluate how much a given metric differs from the Ricci-flat Calbi-Yau metric.

Computing Donaldson's algorithm for large values of k, however, quickly becomes computationally expensive, making cheaper or higher-accuracy approximations desirable. A first attempt to use machine learning to predict values of the balanced metrics on a fixed set of points on the

manifold was made in [11].

In the following, the algebraic Kähler potentials introduced above will be used, in the context of deep learning, to define an alternative approximation scheme that achieves better accuracies than Donaldson's algorithm in a similar amount of time. This will be done by producing mappings from a subset of moduli space to the aforementioned matrices h, in a way that the corresponding metrics minimize energy functionals introduced by Headrick and Nassar [10].

An overview of the mathematical prerequisites necessary to follow the numerical approach outlined here is given in section 2 below. Following a review of the definitions of Kähler and Calabi-Yau manifolds, Donaldson's algorithm is introduced, as well as the utilized integration scheme, and definitions of accuracy measures. These accuracy measures will not only be used the assess the quality of an approximation, but also provide the basis for the loss functions which are minimized in the final deep learning approach.

Section 3 presents an analysis of a new implementation of Donaldson's algorithm and the numerical values of the Hermitian matrices h it produces. The convergence of Donaldson's algorithm is well understood, which means that it gives a benchmark for accuracies that can be achieved in a certain amount of computational time. Any new approximation scheme must at least reproduce the accuracies achieved by Donaldson's algorithm in a similar amount of time in order to constitute an interesting alternative.

Section 4 lays out the deep learning approach. After a review of previous and possible future applications of machine learning, the optimal h matrix with respect to the accuracy measures are found using gradient descent for a fixed point in moduli space. This produces better approximations to the Ricci-flat metric than Donaldson's algorithm, according to the accuracy measures. Finally, the optimization is done with respect to a parametrized function from (a subset of) moduli space to h. This will lead to a high accuracy approximation of the Calabi-Yau metric, present as a function of moduli parameters. We will specialize to a subset of moduli space, which simplifies the following discussion. This analysis should therefore be seen as a first proof of principle of a deep-learning approach, laying the foundations for future improvements.

Finally, a summary of the results is given in section 5, including an outlook of possible future improvements and generalizations of the algorithm developed here.

Advances in modern machine learning frameworks, most centrally the feature of automatic differentiation, have proven invaluable for the practical implementation. A discussion of this can be found in section 3.1.1, in the context of Donaldson's algorithm, as well as in appendices A and B.

2. Mathematical Prerequisites

2.1. Calabi-Yau Manifolds

The subject of the investigation at hand are so called Calabi-Yau manifolds, which are compact Ricci-flat Kähler manifolds. The remainder of this section will give a brief review of the definitions and constructions of immediate relevance here[1]. This serves both as a reference and to clarify the choice of notation for the remainder of the analysis.

2.1.1. Projective Space and Algebraic Varieties

The maximum principle implies that all holomorphic functions $f : X \to \mathbb{C}^N$ defined on a compact complex manifold X are constant, which means there are no compact submanifold of $\mathbb{C}^N$. One is thus led to consider submanifolds of the complex projective space $\mathbb{CP}^N$, which is compact. It is obtained as the quotient of $\mathbb{C}^{N+1}$ with respect to multiplication with $\mathbb{C} \setminus \{0\}$. There are two ways to denote points in projective space. The first is in terms of the coordinates of a representative in $\mathbb{C}^{N+1}$, called homogeneous coordinates z, which are identified under multiplication with a complex number:

$$[z^0, \ldots, z^N] \sim [\lambda z^0, \ldots, \lambda z^N],\ \lambda \in \mathbb{C} \setminus \{0\}\,. \tag{2.1}$$

Although the homogeneous coordinates are numerically not unique for any given point (and are thus not strictly speaking coordinates), they will prove very useful for expressing globally defined objects. On each open patch

$$U_k = \left\{ z \in \mathbb{CP}^N : z^k \neq 0 \right\}, \tag{2.2}$$

we can use the above identification to set $z^k = 1$, which removes the scaling ambiguity and defines affine coordinates z_k (the subscript refers to the patch and is not an index) that are unique on U_k:

$$[z^0, \ldots, z^N] \sim \left[\frac{z^0}{z^k}, \ldots, \frac{z^{k-1}}{z^k}, 1, \frac{z^{k+1}}{z^k}, \ldots \frac{z^N}{z^k} \right] \widehat{=}\, (z_k^0, \ldots, z_k^{k-1}, z_k^{k+1}, \ldots, z_k^N)\,. \tag{2.3}$$

[1] A physically motivated introduction to Calabi-Yau manifolds can be found in [12, 13]. A rigorous introduction to complex differential geometry, sheaves, and line bundles not detailed here is given for example in [14].

To obtain submanifolds X of projective space we can consider the zero locus of a finite set of homogeneous polynomials $Q^r : \mathbb{CP}^N \to \mathbb{C}$,

$$X = \left\{ z \in \mathbb{CP}^N : Q^r(z) = 0 \; \forall r \right\}. \tag{2.4}$$

Submanifolds defined in this manner are called algebraic varieties. Indeed, we can limit ourselves to algebraic varieties [15].

Theorem 1 (Chow)**.**
Every analytic submanifold of $\mathbb{CP}^N$ is the zero-locus of some finite number of homogeneous polynomials.

In the following only one defining polynomial will be considered. However, following [8] it should be possible to extend the results to intersections.

Complex line bundles and their sections (maps from the manifold over which the bundle is defined into its fibers) will play an important role in Donaldson's algorithm. The trivial line bundle on projective space is just $\mathcal{O}_{\mathbb{CP}^N} = \mathbb{CP}^N \times \mathbb{C}$. Sections of the trivial line bundle over any complex manifold are the holomorphic functions on it. The tautological line bundle $\mathcal{O}_{\mathbb{CP}^N}(-1)$ is obtained by adjoining to each point in $\mathbb{CP}^N$ its corresponding equivalence class, which is a one dimensional subspace of $\mathbb{C}^{N+1}$. The dual of the tautological line bundle is denoted as $\mathcal{O}_{\mathbb{CP}^N}(1)$. The whole Picard group, i.e. the set of all line bundles over $\mathbb{CP}^N$, can be generated as $\mathcal{O}_{\mathbb{CP}^N}(k) = \mathcal{O}_{\mathbb{CP}^N}(1)^{\otimes k}$.

A technically very useful description follows from their treatment as invertible sheaves. The sections of $\mathcal{O}_{\mathbb{CP}^N}(m)$, $m \in \mathbb{Z}$, are holomorphic functions on each patch U_i with the transition functions

$$f^{ij} = \left(\frac{z^j}{z^i} \right)^m. \tag{2.5}$$

That is, a local section g_j in U_j corresponds to the local section $g_i(z) = f_{ij}(z) g_j(z)$ in U_i. The global sections of $\mathcal{O}_{\mathbb{CP}^N}(k)$ with $k > 0$ are the homogeneous polynomials of degree k in the homogeneous coordinates, $\mathbb{C}[z^0, \dots z^N]_k$. A basis is given by the degree k monomials

$$s^\alpha(z) = \prod_{i=0}^{N} z_i^{p_{i\alpha}}, \tag{2.6}$$

where $p_{i\alpha}$ is an appropriate matrix of powers and α is the index in the basis. The number of degree k monomials in $N+1$ coordinates, and thus the basis size of global sections, is given by

$$N_k\left(\mathbb{CP}^N\right) = \dim H^0\left(\mathbb{CP}^N, \mathcal{O}_{\mathbb{CP}^N}(k)\right) = \binom{N+k}{k}. \tag{2.7}$$

It is important to note that while the global sections are formally polynomials, they are not strictly functions on $\mathbb{CP}^N$, since due to the scaling ambiguity their value is undefined. Instead,

they represent local sections which are well defined functions in the local affine coordinates on each patch U_i, obtained by following the prescription of (2.3) to remove the scaling ambiguity:

$$s_i^\alpha(z_i) = \prod_{j\neq i} \left(z_i^j\right)^{p_{j\alpha}} = \prod_{j=0}^{N} \left(\frac{z^j}{z^i}\right)^{p_{j\alpha}} . \tag{2.8}$$

It can now be seen that the set of local sections obtained in this way transform under the aforementioned transition rule (2.5):

$$f_{li}(z) s_i^\alpha(z_i) = \left(\frac{z^i}{z^l}\right)^k \prod_{j=0}^{N} \left(\frac{z^j}{z^i}\right)^{p_{j\alpha}} = \prod_{j=0}^{N} \left(\frac{z^j}{z^l}\right)^{p_{j\alpha}} = s_l^\alpha(z_l)\,, \tag{2.9}$$

where it was used that $\sum_i p_{i\alpha} = k$ for all α (in other words, each element of the basis is a degree k monomial).

For an n-dimensional variety X of projective space $\mathbb{CP}^{n+1}$ defined by a degree $n+2$ homogeneous polynomial $Q(z) = 0$, the line bundle $\mathcal{O}_X(k)$ can be constructed from $\mathcal{O}_{\mathbb{CP}^{n+1}}(k)$ by removing all sections that vanish on X. This is achieved for $k \geq n+2$ by the quotient space of homogeneous polynomials

$$\mathbb{C}[z^0, \dots z^{n+1}]_k \,/\, \big\langle Q(z) \big\rangle \,, \tag{2.10}$$

where $\big\langle Q(z) \big\rangle = Q\,\mathbb{C}[z^0, \dots z^{n+1}]_{k-(n+2)}$ is the polynomial space that vanishes with Q. The basis size of $\mathcal{O}_X(k)$ is thus

$$\begin{aligned} N_k(X) &= \dim H^0\big(X, \mathcal{O}_X(k)\big) \\ &= N_k\Big(\mathbb{CP}^{n+1}\Big) = \binom{n+k+1}{k} && \text{if } k < n+2 \\ &= N_k\Big(\mathbb{CP}^{n+1}\Big) - N_{k-(n+2)}\Big(\mathbb{CP}^{n+1}\Big) = \binom{n+k+1}{k} - \binom{k-1}{k-n-2} && \text{if } k \geq n+2\,. \end{aligned} \tag{2.11}$$

How the reduction of the basis is done in practice, including a specific example, can be found in section B.1 of the appendix.

2.1.2. Kähler Manifolds

In the following, metrics on complex manifolds are taken to be symmetric positive-definite 2-tensors acting on the complexified tangent space. (A Riemannian metric from the point of view of a real manifold, acting on the real tangent space, can be extended by demanding $\mathbb{C}$-linearity.)

Consider a complex manifold X of dimension n. The metric g is called Hermitian if locally $g_{ij} = g_{\bar{i}\bar{j}}$, or equivalently $g = g_{i\bar{j}} dz^i \otimes d\bar{z}^{\bar{j}} + g_{\bar{i}j} d\bar{z}^{\bar{i}} \otimes dz^j$. To every Hermitian metric g a 2-form

$\omega \in \Lambda^{1,1} T_X^\star$ can be associated, called Kähler form:

$$\omega(v,w) = g(v,Jw), \text{ locally } \omega = i \sum_{i\bar{\jmath}} \left(g_{i\bar{\jmath}}\, dz^i \otimes d\bar{z}^{\bar{\jmath}} - g_{\bar{\imath}j}\, d\bar{z}^{\bar{\imath}} \otimes dz^j \right) = i \sum_{i\bar{\jmath}} g_{i\bar{\jmath}}\, dz^i \wedge d\bar{z}^{\bar{\jmath}}\,, \tag{2.12}$$

where J is the complex structure on X. The volume form induced by the metric can be expressed in terms of the Kähler form as

$$d\mathrm{Vol}_K = \frac{\omega^n}{i^n n!} \overset{\text{locally}}{=} \det g_{i\bar{\jmath}}\,. \tag{2.13}$$

If the Kähler form is closed, the corresponding element of the de Rham cohomology $[\omega] \in H^{1,1}(X)$ is called its Kähler class, and the manifold is called Kähler manifold.

Definition 1 (Kähler manifold)**.**
A complex manifold X with a Hermitian metric g whose corresponding Kähler form ω is closed, $d\omega = 0$ is called a Kähler manifold.

Kählerity $d\omega = (\partial + \bar{\partial})\,\omega = 0$ implies the following for the local form of the metric[2]:

$$\partial\omega = 0 \quad \text{and} \quad \bar{\partial}\omega = 0 \tag{2.14}$$

$$\Rightarrow \quad (\partial_k g_{i\bar{\jmath}} - \partial_i g_{k\bar{\jmath}})\, dz_k \wedge dz_i \wedge dz_{\bar{\jmath}} = 0 \quad \text{and} \quad (\partial_{\bar{k}} g_{i\bar{\jmath}} - \partial_{\bar{\jmath}} g_{i\bar{k}})\, dz_i \wedge dz_{\bar{\jmath}} \wedge d\bar{z}_{\bar{k}} = 0 \tag{2.15}$$

$$\Rightarrow \quad \frac{\partial g_{i\bar{\jmath}}}{\partial z_k} = \frac{\partial g_{k\bar{\jmath}}}{\partial z_i} \quad \text{and} \quad \frac{\partial g_{i\bar{\jmath}}}{\partial z_{\bar{k}}} = \frac{\partial g_{i\bar{k}}}{\partial z_{\bar{\jmath}}}\,. \tag{2.16}$$

This means that on a local patch, the metric is fully determined by some real, scalar function K called Kähler potential (to see this note, than on local patches $H^1(\mathbb{C}^d \cong \mathbb{R}^{2d}) = 0$ and for a local 1-form f, $\partial f_i/\partial x_j = \partial f_j/\partial x_i \Leftrightarrow df = 0$, thus $f = dh$ i.e. $f_i = \partial h/\partial x_i$):

$$g_{i\bar{\jmath}} = \frac{\partial^2 K}{\partial z^i \partial \bar{z}^{\bar{\jmath}}} \text{ and } \omega = i\partial\bar{\partial} K\,. \tag{2.17}$$

The Kähler potential is not uniquely defined since the metric is invariant under Kähler transformations

$$K'(z) = K(z) + f(z) + \bar{f}(\bar{z})\,, \tag{2.18}$$

where $f(z)$ is a holomorphic function. Furthermore, the Kähler potential cannot be globally defined because else the Kähler form would be exact, and thus the volume of the manifold would vanish. The Kähler potentials of two different patches thus differ by a Kähler transformation on their overlaps. The form of the Ricci curvature tensor simplifies significantly for Kähler metrics:

$$R_{i\bar{\jmath}} = R^{\bar{k}}{}_{i\bar{k}\bar{\jmath}} = -\partial_i \partial_{\bar{\jmath}} \log\det g\,. \tag{2.19}$$

[2]Making use of the Einstein summation convention.

2.1.3. Calabi-Yau Manifolds

The holonomy of the tangent bundle of a manifold is the group of all linear transformations generated by parallel transport (we consider manifolds with a metric here, and parallel transport with respect to the Levi-Civita connection). For a generic real oriented Riemannian manifold of dimension $2n$, the holonomy is a subgroup of $SO(2n)$. For a Kähler manifold, the Christoffel symbols do not mix holomorphic and anit-holomorhic parts of a tangent vector, so parallel transport respects the decomposition $T_X^{\mathbb{C}} = T_X^{(1,0)} \oplus T_X^{(0,1)}$. This limits the holonomy group to subgroups of $U(n)$.

Given the Riemann curvature tensor of a Kähler manifold, one can define a matrix valued curvature 2-form

$$\boldsymbol{\mathcal{R}} = i \sum_{i\bar{\jmath}} R^k{}_{li\bar{\jmath}}\, dz^i \wedge d\bar{z}^{\bar{\jmath}}\,. \tag{2.20}$$

Its trace $\mathcal{R} = \operatorname{tr} \boldsymbol{\mathcal{R}}$ is called Ricci form. Equation (2.19) implies the Ricci form is locally exact ($g_{i\bar{\jmath}}$ is only locally defined, so this does not imply globally exact), and thus closed. The k-th Chern class $c_k(X) \in H^k(X)$ is defined as the k-th term in the expansion

$$c(X) = \det(1 + \boldsymbol{\mathcal{R}}) = 1 + \mathcal{R} + \dots\,, \tag{2.21}$$

and only depends on topological properties of X. The first chern class can be read off as $c_1(X) = [\mathcal{R}]$, which clearly vanishes if X is Ricci-flat. In fact, the reverse also holds [16, 17].

Theorem 2 (Yau)**.**
Let X be a complex Kähler manifold with vanishing first Chern class $c_1(X) = 0$. Then each Kähler class has a unique representative ω such that the corresponding metric is Ricci-flat.

If one is searching for Ricci-flat manifolds, one is thus lead to the following definition.

Definition 2 (Calabi-Yau Manifold)**.**
A Calabi-Yau manifold is a compact, complex Kähler manifold with vanishing first Chern class.

More specifically, the problem is then to find (an approximation to) the Ricci-flat Calabi-Yau metric on a manifold with vanishing first Chern class, on which by theorem 2 we know it to exist. One can show that Ricci-flatness is equivalent to restricting[3] the holonomy group to $SU(n)$ [18].

It can be shown that algebraic varieties in $\mathbb{CP}^{n+1}$ defined by a homogeneous polynomial of degree $n + 2$ have vanishing first Chern class [19]. Varieties of dimension $n = 2$ are called K3 surfaces, those of dimension $n = 3$ are called quintics. For concreteness we will consider quintics

[3] The definition of Calabi-Yau manifolds often explicitly requires a holonomy group equal to $SU(n)$, not subgroups thereof, which is satisfied in the following examples.

with a single parameter ψ

$$Q_\psi(z) = \sum_{i=0}^{4} (z^i)^5 + \psi \prod_{i=0}^{4} z^i = 0\,. \tag{2.22}$$

The case $\psi = 0$ is called Fermat quintic, and $\psi = -5$ corresponds to the conifold (the derivative of Q_{-5} vanishes where all coordinates are equal which means the manifold becomes singular). In general, the set of all possible coefficients of degree 5 polynomials (modulo redefinitions of the coordinates, c.f. [8]) define the complex structure moduli space of the quintic. By restricting ourselves to equation (2.22), we thus consider a one dimensional subspace of the complex structure moduli space parametrized by ψ.

2.2. Donaldson's Algorithm

In [5, 6, 7], Donaldson introduced an iterative scheme to approximate the Ricci-flat metric by a sequence of unique algebraic metrics called balanced metrics. The balanced metric at a fixed degree $k \geq 1$ can be obtained by consecutive application of an integral T-operator. The series of balanced metrics are known theoretically and numerically to converge to the Ricci-flat metric like $O(k^{-2})$ with respect to the σ-accuracy introduced in section 2.4 [20, 8].

2.2.1. Algebraic Metric

The following construction of Kähler potentials, and thus metrics, is the basis for both Donaldson's algorithm described in section 2.2.2, and can serve as the starting point for machine learning models. The use of this kind of algebraic ansatz in the context of Donaldson's algorithm was introduced in [5, 7], and further developed in [8, 10, 20]. A synthesis of these, stressing the aspects relevant for the remaining analysis, is presented here.

The Projective space $\mathbb{CP}^N$ displays an $SU(N+1)$ symmetry of coordinate transformations. There is a unique metric (up to scaling[4]) which is invariant under these transformations, called Fubini-Study metric. Its Kähler potential is

$$K_{FS}(Z) = \frac{1}{\pi} \log \left(\sum_{\alpha=0}^{N} Z^\alpha \, \bar{Z}^{\bar{\alpha}} \right). \tag{2.23}$$

We can generalize this slightly by adding a Hermitian matrix h such that

$$K_{FS}(Z) = \frac{1}{\pi} \log \left(\sum_{\alpha,\bar{\beta}=0}^{N} Z^\alpha \, h_{\alpha\bar{\beta}} \, \bar{Z}^{\bar{\beta}} \right), \tag{2.24}$$

[4]The scaling is fixed to $1/\pi$ following [8] by requiring ω_{FS} to be an integer class.

which can be brought back to the canonical Fubini-Study potential by a coordinate transformation. The pullback of this defines a metric on any variety $X \hookrightarrow \mathbb{CP}^{n+1}$, which is unfortunately not Ricci-flat.

The geometric idea behind the algebraic metrics is to embed the manifold in some higher dimensional projective space, and define a metric on it by the pull back of the higher-dimensional Fubini-Study metric. One thus has to find an embedding such that this pullback is as close to Ricci-flat on the variety as possible. For larger projective spaces one gets, roughly speaking, more degrees of freedom for the embedding, and thus potentially more accurate approximations to the Ricci-flat metric.

To construct these embedding, consider a holomorphic line bundle $\mathcal{L}$ over a Kähler manifold X such that there are N_k global sections s^α forming a basis for $\mathcal{L}^{\otimes k}$. Given that by definition there is no point at which all s^α vanish, this gives an embedding into $\mathbb{CP}^{N_k-1}$ (no frame for the line bundle was fixed, which means the sections have the same scale equivalence as projective space, allowing the identification):

$$i_k : X \hookrightarrow \mathbb{CP}^{N_k-1}, \quad i_k(z) = \left(s^1(z), \ldots, s^{N_k}(z)\right). \tag{2.25}$$

For the application here, X is an n-dimensional algebraic variety in $\mathbb{CP}^{n+1}$ and $\mathcal{L} = \mathcal{O}_X(1)$. A basis of section on $\mathcal{O}_X(1)$ was introduced above as $s^\alpha(z)$, which are formally defined as homogeneous polynomials on projective space, restricted to evaluation on X. More generally, one would require $\mathcal{L}$ to be ample, which means that sections of $\mathcal{L}^{\otimes k}$ induce a Kodaira embedding into projective space in the above way, for large enough k. Note that for $k = 1$ we have $N_k = n + 2$, by equation (2.11). The embedding is therefore one into the ambient projective space $i_k : X \hookrightarrow \mathbb{CP}^{n+1}$, and for any $k \geq 1$ the embeddings are well-defined.

The pullback of the generalized Fubini-Study metric on $\mathbb{CP}^{N_k-1}$ can now be defined by the Kähler potential in which the homogeneous coordinates are substituted by the embedding introduced above:

$$K(z) = \frac{1}{\pi k} \log \left(\sum_{\alpha,\bar{\beta}=1}^{N_k} s^\alpha(z)\, h_{\alpha\bar{\beta}}\, \bar{s}^{\bar{\beta}}(\bar{z}) \right). \tag{2.26}$$

The metric on X defined with this potential is, by the chain rule, just the pullback via the embedding i_k of the generalized Kähler metric. Because of its algebraic construction, in the following this type of potential will be referred to as algebraic potential, and the corresponding metric as algebraic metric. Different values for h are now inequivalent since they cannot be removed by a simple coordinate transformation. In fact, a transformation on $\mathbb{CP}^{N_k-1}$ that brings the generalized Fubini-Study back to canonical form, $P^\top h \bar{P} = \mathbb{1}$, reveals that different h matrices just correspond to different embeddings:

$$i_k^h(z) = \left(P^1{}_\alpha\, s^\alpha(z), \ldots, P^{N_k-1}{}_\alpha\, s^\alpha(z)\right). \tag{2.27}$$

The prefactor $1/k$ in equation (2.26) was introduced to keep the Kähler class constant in k (indeed, substituting $\left(\sum |z_i|^2\right)^k$ into the logarithm gives back the Fubini-Study potential defined on the ambient projective space of X for any k). Note that since the sections s^α of $\mathcal{O}_X(k)$ are not themselves functions but represent functions on each patch, the above Kähler potential must be understood as representing different functions on each patch as well. The Kähler potential can be written as a global expression in terms of the homogeneous coordinates, because the overall scaling drops out when computing the derivatives of the logarithm. By restricting to affine coordinates, one automatically gets local Kähler potentials on each patch of projective space which are compatible via Kähler transformations. One can see this is explicitly by comparing the local Kähler potentials obtained by substituting with the local sections of (2.8). On the patch U_i of the ambient projective space the Kähler potential is

$$K_i(z) = \frac{1}{\pi k} \log \left(\frac{\sum_{\alpha,\bar{\beta}=1}^{N_k} s^\alpha(z)\, h_{\alpha\bar{\beta}}\, \bar{s}^{\bar{\beta}}(\bar{z})}{|z_i|^k} \right). \tag{2.28}$$

Between patches U_i and U_j the relation therefore is

$$K_i(z) = K_j(z) + \frac{1}{\pi} \log z_j - \frac{1}{\pi} \log z_i \,, \tag{2.29}$$

which is just a Kähler transformation. This makes clear another great advantage of using algebraic metrics. The Kähler potential is well defined on each patch of the ambient projective space, and it is therefore unnecessary to explicitly introduce patches on X. Note that the Kähler potential on X is defined by restriction, and derivatives to obtain the metric still have to be done in a local coordinate system on X. More detail on this can be found in section B.2 of the appendix.

The moduli space of Calabi-Yau metrics can be divided into Kähler moduli and complex structure moduli, whose dimensions are $h^{1,1}$ and $h^{2,1}$ respectively. As mentioned at the end of section 2.1.3, the complex structure moduli are the polynomial coefficients of the defining equation (after equivalent coefficients are identified). The Kähler class of the algebraic metric is fixed by a single number, namely the factor in front of the logarithm in equation (2.26) (here fixed to $1/\pi$; $1/k$ is needed to keep the Kähler class constant in k). This means that one is limited to a one dimensional subspace of the Kähler moduli space. However, for the quintic it is $h^{1,1} = 1$, which means using the ansatz of algebraic metrics does not amount to a restriction.

There is another useful interpretation of the algebraic metrics. The eigenfunctions of the first k eigenvalues of the Laplacian on $\mathbb{CP}^{n+1}$ can be expressed using the basis functions [21, 10]

$$\Phi^{\alpha\bar{\beta}} = \frac{s^\alpha(z) s^{\bar{\beta}}(\bar{z})}{\left(\sum_i \left|z^i\right|^2\right)^k} \,, \tag{2.30}$$

where the $s^\alpha(z)$ are defined as before. Two Kähler potentials that differ by a globally defined scalar function lie in the same Kähler class, since the difference between the induced Kähler forms is exact. As noted above, the Kähler class of K for all degrees k is fixed and the same as

that of the Fubini Study metric (2.23) given by

$$K_{FS} = \frac{1}{\pi} \log \left(\sum_i \left| z^i \right|^2 \right) . \tag{2.31}$$

Therefore, the difference $K - K_{FS}$ is a globally well defined function on X. One may therefore try to expand $e^{\pi k(K-K_{FS})}$ in terms of the $\Phi^{\alpha\bar{\beta}}$. This indeed gives the algebraic metric

$$K(z) = K_{FS}(z) + \frac{1}{\pi k} \log \left(\sum_{\alpha\bar{\beta}} h_{\alpha\bar{\beta}} \Phi^{\alpha\bar{\beta}}(z) \right) = \frac{1}{\pi k} \log \left(\sum_{\alpha\bar{\beta}} h_{\alpha\bar{\beta}} s^\alpha(z) s^{\bar{\beta}}(\bar{z}) \right) , \tag{2.32}$$

where h is the matrix of coefficients of the expansion. The ansatz of algebraic metrics for some fixed k can thus be interpreted as the truncation of a spectral expansion.

2.2.2. Balanced Metric

The matrix $h_{\alpha\bar{\beta}}$ that parametrizes the algebraic metrics (2.26) can be interpreted as a Hermitian metric on the line bundle $\mathcal{O}_X(k)$. Given two sections S, S' it is defined point-wise as

$$(S, S')(z) = \frac{S(z)\bar{S}'(z)}{\sum_{\alpha\bar{\beta}} s^\alpha(z) h_{\alpha\bar{\beta}} s^{\bar{\beta}}} = \frac{S(z)\bar{S}'(z)}{||s||_{h,k}} . \tag{2.33}$$

In turn, this induces a Hermitian metric on the space of sections as

$$\left\langle S, S' \right\rangle = \frac{N_k}{\mathrm{Vol}_{CY}} \int_X \frac{S\,\bar{S}'}{||s||_{h,k}} d\mathrm{Vol}_{CY} . \tag{2.34}$$

Evaluated on the basis of sections s^α this is $H^{\alpha\bar{\beta}} = \langle s^\alpha, s^\beta \rangle$. The T-operator is defined as the map from $h_{\alpha\bar{\beta}}$ to $H^{\alpha\bar{\beta}}$

$$T(h)^{\alpha\bar{\beta}} = \frac{N_k}{\mathrm{Vol}_{CY}} \int_X \frac{s^\alpha s^{\bar{\beta}}}{||s||_{h,k}} d\mathrm{Vol}_{CY} . \tag{2.35}$$

The matrices $h_{\alpha\bar{\beta}}$ and $H^{\alpha\bar{\beta}}$ are in general different. If they coincide, the metric $h_{\alpha\bar{\beta}}$ is called balanced [5].

Definition 3 (Balanced Metric)**.**
The metric $h_{\alpha\bar{\beta}}$ on the line bundle $\mathcal{O}_X(k)$ is called balanced if

$$h_{\alpha\bar{\beta}} = \left(H^{\alpha\bar{\beta}} \right)^{-1} . \tag{2.36}$$

By iteratively computing the metric on sections using the T-operator and taking its inverse to obtain a new metric on the line bundle, one obtains an approximation to the balanced metric [5, 7].

Theorem 3 (Donaldson)**.**
Iteratively applying

$$h_{\alpha\bar{\beta}} = \left(T(h)^{\alpha\bar{\beta}}\right)^{-1} \tag{2.37}$$

converges to the unique balanced metric for any $k \geq 1$ and initial metric h. Furthermore, the sequence of algebraic metrics on X (2.26) defined by the balanced metrics h converges to the unique Calabi-Yau metric in the given Kähler class and complex structure as $k \to \infty$.

2.3. Monte Carlo Integration

For both the accuracy measures defined in the next section and Donaldson's algorithm defined above, it is important to be able to numerically integrate on the manifold X. The easiest way to achieve this, since the integration space is relatively high dimensional, is by Monte Carlo integration [22]. The integration scheme used in the remainder of this analysis is the one laid out in [20]. For completeness, a short summary is given here.

Let X be an n-dimensional Calabi-Yau manifold defined as algebraic variety of $\mathbb{CP}^{n+1}$ by $Q(z) = 0$. Associated to X is a holomorphic n-form Ω. Without loss of generality assume z is in patch U_0 and $(z^1, \ldots, z^{l-1}, z^{l+1}, \ldots, z_n)$ are good coordinates on X. Then Ω can be written using the Poincaré residue map as [15]

$$\Omega(z) = (-1)^{l-1} \frac{dz^1 \wedge \ldots \wedge dz^{l-1} \wedge dz^{l+1} \wedge \ldots \wedge z^n}{\partial_l Q(z)} \,. \tag{2.38}$$

Using Ω, one can write the unique top form (up to scaling) of the Calabi-Yau manifold as

$$d\mathrm{Vol}_{CY} = (-i)^n \, \Omega \wedge \bar{\Omega} \,. \tag{2.39}$$

Samples for the Monte Carlo integration can be generated by uniformly sampling two points on $\mathbb{CP}^{n+1}$ and finding the intersections of the line they define with the manifold X. By Bezout's theorem the line always intersects X at $n+2$ points (solutions to an $n+2$ polynomial), which can be efficiently computed as the roots of a polynomial. This is outlined in more detail in section B.3.2 of the appendix. Assuming the sampling has a probability distribution according to some volume form dA, the Monte Carlo integration approximates the integral as a sum

$$\int_X f \, d\mathrm{Vol}_{CY} = \int_X f \, \frac{d\mathrm{Vol}_{CY}}{dA} dA \approx \frac{1}{N_p} \sum_{a=1}^{N_p} f(z_a) w(z_a) \,, \tag{2.40}$$

where N_p is the number of sample points z_a. The weights $w(z)$ are the ratio of the Calabi-Yau and probability measures evaluated at the sample points (implicit using the basis tangent

vectors given by the local coordinates):

$$w(z) = \left.\frac{d\text{Vol}_{CY}}{dA}\right|_z . \tag{2.41}$$

The Calabi-Yau volume of the manifold is then simply the mean over the weights w

$$\text{Vol}_{CY} = \int d\text{Vol}_{CY} = \frac{1}{N_p}\sum_{a=1}^{N_p} w(z_a) . \tag{2.42}$$

The volume Vol_K of a Kähler manifold with Kähler form ω_K given by a potential K can be similarly computed, using the volume form $d\text{Vol}_K = \omega_K^n/n!$.

Since the points are found as the intersections with a line in projective space that is chosen with uniform probability, their random distribution is invariant under the $SU(n+1)$ action of coordinate transformations on projective space. The measure dA is thus proportional to the pullback of the Fubini-Study Kähler form which respects this symmetry, [20]

$$dA \propto (i_Q^* \, \omega_{\mathbb{CP}^{n+1}}^{FS})^n , \tag{2.43}$$

where i_Q is the embedding into the ambient projective space defined by the homogeneous polynomial $Q(z) = 0$ that defines X. How the Monte Carlo approximations are calculated in practice is outlined in section B.3.3 in the appendix.

2.4. Accuracy Measures

Measures of how close a given metric on X is to the Ricci-flat Calabi-Yau metric serve two purposes. The first is that they facilitate statements about the convergence of, and comparison between, numerical approximations. The second is that they define a minimization problem which, if solved for a given parametrization of metrics, provides an alternative to Donaldson's algorithm. If these accuracy measures are interpreted as loss functions, one is naturally led to a formulation in the language of machine learning.

The most straight-forward accuracy measure to define is the $||R||$-measure, defined as the integral of the absolute value of the Ricci scalar[5]

$$||R|| = \frac{\text{Vol}_K^{1/n}}{\text{Vol}_{CY}} \int_X d\text{Vol}_K |R| , \tag{2.44}$$

where n is the dimension of the manifold X. The Ricci scalar is the full contraction of the Ricci curvature as defined in equation (2.19) with the metric. The main disadvantage is that it is computationally expensive to evaluate. The Ricci curvature is given by a second derivative of the metric, which in turn is given as the second derivative of the Kähler potential. If used

[5] the volume factor is chosen to cancel the scaling in k, as described in [23]

as the loss function in gradient descent, a fifth differentiation with respect to the parameters is needed. While still feasible for small values of k, the accuracy measure introduced next is significantly cheaper, especially if used as loss for gradient descent.

The top form on a Calabi-Yau manifold is unique up to scaling. This means that the volume form given in terms of the sought Ricci-flat Kähler form ω (or the corresponding metric) must be proportional to the known Calabi-Yau volume form of equation (2.39):

$$\frac{\omega^n}{n!} = \lambda\, \Omega \wedge \bar{\Omega}\,. \tag{2.45}$$

Any deviation from proportionality can be interpreted as a deviation from the Calabi-Yau metric. We can rewrite equation (2.45) in terms of the ratio

$$\hat{\eta} = \frac{\omega^n/n!}{\Omega \wedge \bar{\Omega}} = \lambda\,. \tag{2.46}$$

Integrating both sides of equation (2.45) implies the proportionality constant λ is given by the ratios of volumes. One is thus led to the normalized form

$$\eta = \hat{\eta}\, \frac{\mathrm{Vol}_K}{\mathrm{Vol}_{CY}}\,. \tag{2.47}$$

The σ-measure is now defined as the average deviation of the η form from being constant:

$$\sigma = \frac{1}{\mathrm{Vol}_{CY}} \int_X d\mathrm{Vol}_{CY} |1 - \eta|\,. \tag{2.48}$$

It therefore measures how much the given metric differs from the Calabi-Yau metric, and vanishes for the true metric.

As shown by Headrick and Nassar [10], there is a whole class of convex[6] energy functionals that are uniquely minimized by the Ricci-flat metric. Given a convex differentiable function $F : \mathbb{R}^+ \to \mathbb{R}$ that is bounded from below, one can define an energy functional

$$E_F = \int_X d\mathrm{Vol}_{CY} F(\eta)\,. \tag{2.49}$$

The minimum of this functional among a set of metrics, such as degree k algebraic metrics, is called optimal. Note that the σ-measure is a special case of this functional. Since the algebraic metrics can be interpreted as a spectral expansion truncated at some order k, the optimal algebraic metrics can, in analogy to a Fourier expansion, be expected to approximate the flat metric exponentially well in k with respect to the σ-measure [10]. More specifically, there is a sequence of balanced metrics ω_k such that

$$||\omega_{CY} - \omega_k|| = O(k^{-\nu}) \tag{2.50}$$

[6] Convexity here is in terms of the Kähler form and implies its uniqueness. Unfortunately, this property is not preserved once some parametrization for the metrics is chosen, and therefore does not guarantee good convergence for minimization with gradient descent.

for any ν and C^r-norm on X [7]. This would be a significantly more rapid convergence than that of the balanced metrics produced by Donaldson's algorithm, which converge like $O(k^{-2})$ in the σ-measure.

Underlying the energy functionals is the fact that the closer the metric is to Ricci-flat, the smaller the variance of $\hat{\eta}$. This can be seen qualitatively in Figure 2.1 which shows the narrowing of the distribution of $\hat{\eta}$ for Donaldson's balanced metrics for successively larger k, and thus successively better approximations to Ricci-flatness. As laid out in section 2.3, the integrals are approximated by Monte Carlo sums. Figure 2.4 compares the convergence behaviors of the $||R||$-measure, σ-measure, and the standard deviation of $\hat{\eta}$ computed over the same samples, for different number of sample sizes. The measures were computed for the balanced metric at $k = 7$, and at each sample size for 6 sets of points over which the standard deviation is shown. Based on these results, the relative error of the σ-measure can be expected to be less than 1 % for a sample size of about $N_p = 10\,000$. If not otherwise stated, this is the number of samples used to compute the accuracy measures in the following.

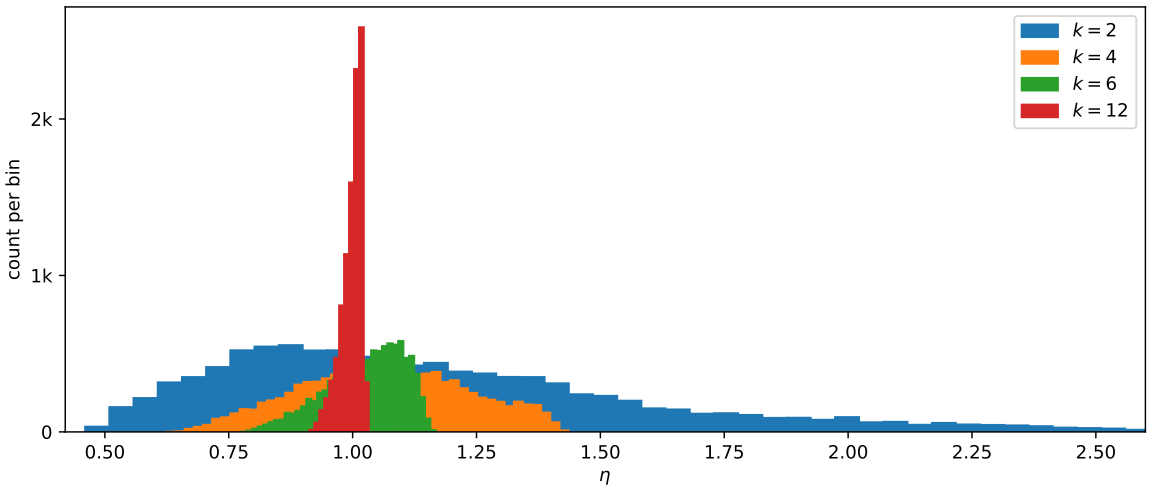

Figure 2.1.: Distribution of η evaluated on 10 000 points on the Fermat quintic (sampling as defined in section 2.3, no Monte Carlo weights applied) for balanced metrics of different degrees.

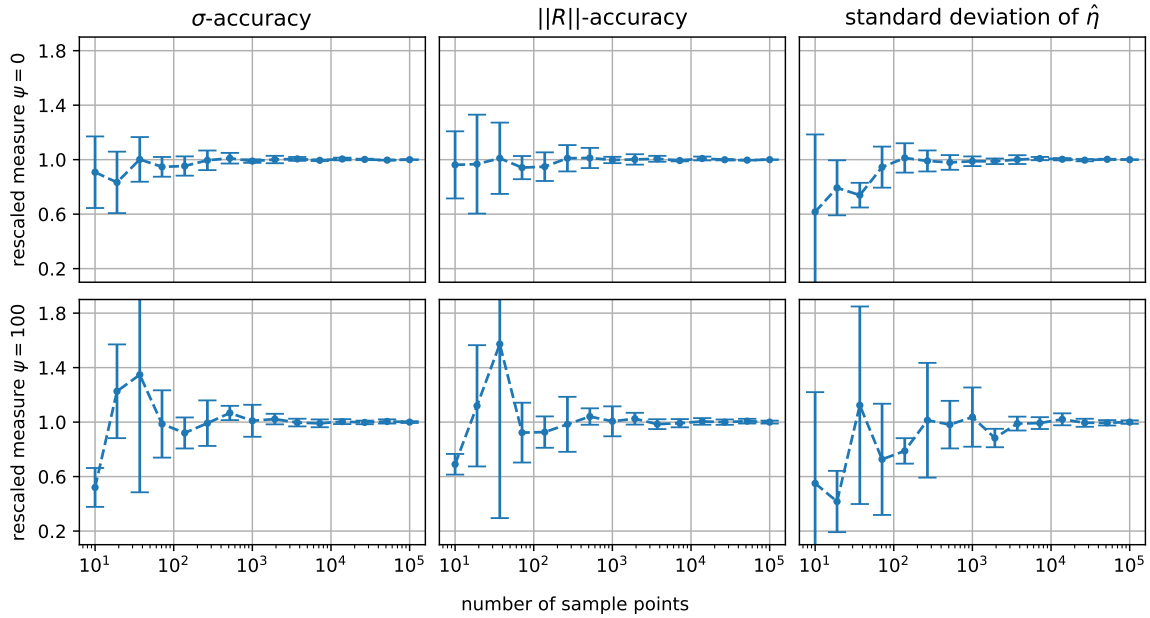

Figure 2.2.: Comparison of convergence behavior with respect to the number of sample points for σ- and $||R||$-measures, and the standard deviation of $\hat{\eta}$. Accuracies were computed for the balanced metrics produced by Donaldson's algorithm at degree $k = 7$ with $\psi = 0$ and $\psi = 100$ for the upper and lower plots respectively. The values shown are mean and standard deviations computed over 6 independent sets of points for each sample size. The standard deviations of $\hat{\eta}$ are computed with respect to the Monte Carlo measure (2.43) with no weights applied.

3. Numerical Analysis of Donaldson's Algorithm

This section lays out the results and a numerical analysis of a new implementation of Donaldson's algorithm. The balanced metrics approximated by Donaldson's algorithm are known to converge for large k to the Ricci flat metric, and thus provide a basic benchmark in terms of accuracy and computational cost, to which other approximation schemes can be compared. Besides that, it allows for a validation of implementations of algorithms that will be reused for the machine learning methods later. More details regarding the implementation can be found in section B of the appendix.

3.1. Implementation and Validation

This section presents the main results for the implementation of Donaldson's algorithm. Following a summary of implementation decisions, the accuracy's dependence on the degree k is analyzed for the Fermat quintic, as well as how much computation time this requires. After these principal results, further dependence on the complex structure moduli (in our case the parameter ψ), the random seed, and the numerical patterns of the produced h matrices will be examined.

3.1.1. Implementation Choices

Before analyzing the convergence behavior of Donaldson's algorithm in more detail, a few words must be said about its implementation. The numeric behavior of Donaldson's algorithm was analyzed previously, for example in [7, 20, 8, 11], using implementations in C/C++, Mathematica, and BASIC. Compiled languages like C/C++ have the advantage of generally smaller run-times at the cost of larger implementation complexity, while a high-level language like Mathematica is capable of symbolic differentiation, and may lead to a simpler implementation.

While deep learning has gained popularity in recent years, many new frameworks have been developed to facilitate it. At their core is the ability, called automatic differentiation, to generate a function that calculates the (partial) derivative of another given function to numerical accuracy. In contrast to symbolic differentiation the function whose derivative is computed does not have to be present as a single symbolic expression, but as an implementation in code which

can make use of most of the constructs of the programming language. This allows for much greater flexibility and ease of implementation. The way the derivative is computed also differs from symbolic differentiation, and is fundamentally numerical, which in practice makes it work for high dimensional spaces and higher derivatives.

The framework chosen for the following implementations is JAX [24], which is based on the Python programming language. There are several advantages to JAX when compared to other frameworks that were considered. In the most common applications of deep learning, only a first-order, real, high-dimensional derivative has to be computed, which means that some frameworks cannot easily be used for higher derivatives or complex numbers. JAX can be used to compute holomorphic and anti-holomorphic derivatives (see appendix A.1), as well as higher derivatives which naturally occur in the geometric setting outlined above (such as in computing the metric from the Kähler potential).

Python itself is an interpreted language, which means that code is not compiled to faster machine code before it is run, leading to comparatively slow run times. This can already be significantly ameliorated by using linear algebra libraries like NumPy, which internally use fast, pre-compiled implementations of algorithms. In many cases, code written in Python which calls these algorithms still has a large overhead cost. Another feature of JAX that has proven to be of great practical value is called just-in-time compilation (JIT). If applied to a Python function, it is translated into a sequence of compiled linear algebra operations at the time it is first called. This means that in consecutive runs the evaluation of the function is significantly faster. For an example that shows this, see Figure 3.2 below, and appendix A which gives a more detailed overview of the features of JAX and how they are used here.

While these properties of JAX already make it a good choice for implementing Donaldson's algorithm, it is important to keep in mind that the eventual goal is to apply deep learning techniques to approximating the Ricci-flat metric. A by-product of implementing Donaldson's algorithm in JAX is that it brings all involved geometric constructions into the context of deep learning. They can then, without modification, be automatically differentiated, JIT compiled, and thus immediately used later. Comparing the results achieved using Donaldson's algorithm here with previously published values can serve to validate the implementation, before reusing parts of it for the deep learning approach.

Finally, note that JAX allows code to be compiled, without modification, to run on a GPU or TPU. While this may be interesting to study further in the future, and was tested in principle, it is not further pursued here.

3.1.2. Validation and Convergence for the Fermat Quintic

For a first study of Donaldson's algorithm, we consider now the quintic with $\psi = 0$. The number of points required in the Monte Carlo approximation of the integral T-operator in

order to achieve convergence was studied in [8, 21], and found to be approximately

$$N_p = 10N_k^2 + 50\,000\,, \tag{3.1}$$

where N_k is the basis size of global sections of $\mathcal{O}_X(k)$. The scaling in N_k^2 can be understood by noting that this is the number of entries of the h matrix which parametrizes the algebraic metric, leading to $N_p \gg N_k^2$. If not otherwise stated, this is the number of sample points used to compute the T operator, and the initial h-matrix used in Donaldson's algorithm is the identity $h_{\alpha\bar{\beta}} = \delta_{\alpha\bar{\beta}}$. To validate the implementation of Donaldson's algorithm, Figure 3.1 compares the σ-measures for the quintic with $\psi = 0$ with those obtained in previous studies [11, 8]. The values are in close agreement.

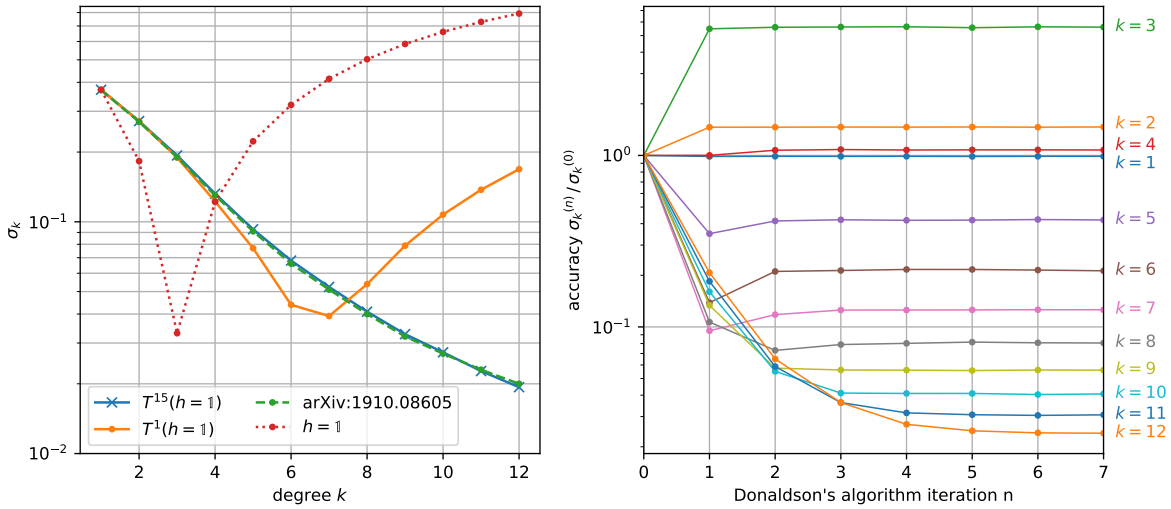

Figure 3.1.: Left: σ-accuracies of results obtained with the implementaion of Donaldson's algorithm used in the following for $\psi = 0$ and monomial degrees between 1 and 11, compared with previous results [11]. The identity was used as initial h-matrix, for which the σ-accuracy is shown as dotted line. Right: Convergence of Donaldson's algorithm over iterations of the T-operator, depending on degree k, for the Fermat quintic ($\psi = 0$). Accuracy σ_k is shown relative to initial value for $h = \mathbb{1}$.

A comparison of the σ-accuracy for the approximated balanced metric with that of the initial h-matrix displayed in the same plot shows that only for k greater than about 4 does the balanced metric lead to an improvement. This behavior is strongly dependent on the point in moduli space. For example at $\psi = 100$, the σ-accuracies achieved with $h_{\alpha\bar{\beta}} = \delta_{\alpha\bar{\beta}}$ are strictly worse than those achieved by Donaldson's algorithm (see Figure C.2 in the appendix). The comparison does help, however, in understanding the convergence behavior in the number of iterations of Donaldson's algorithm, shown in the right plot of Figure 3.1. Corresponding to the difference between initial and final σ-accuracies, one can see the value quickly converge after only a few iterations of the T-operator (as observed before [11, 8]), with slightly more iterations required for larger degrees k. Not too much should be read into the exact convergence behavior, as it is contingent on the chosen manifold. However, the cases where $h_{\alpha\bar{\beta}} = \delta_{\alpha\bar{\beta}}$ have better accuracies than the balanced metrics already show that the balanced metrics are not necessarily optimal with respect to the σ-measure for a given degree k.

3.1.3. Numerical and Theoretical Benchmark

The two primary attributes of any algorithm that approximates the Calabi-Yau metric are the achieved accuracy and its computational cost. There is no unique measure for the accuracy of a given metric to Ricci-flatness, and instead several accuracy measures can be introduced, as was outlined in section 2.4. To make all results produced in the present analysis comparable, the σ-measure is used as a reference[1].

The σ-accuracy achieved by Donaldson's algorithm in relation to the degree k was shown in Figure 3.1 for the quintic. Both theoretically and numerically, one finds a convergence of the order $O(k^{-2})$ [25, 7, 20] (both in terms of the σ-measure, and the difference between Kähler forms in any C^r norms on X [25]). A curve fit for $k \geq 3$ gives approximately

$$\sigma_k = \frac{3.18}{k^2} - \frac{4.30}{k^3} + O\left(\frac{1}{k^4}\right), \tag{3.2}$$

which is close to the values $3.1\,k^{-2} - 4.2\,k^{-3}$ found in [20].

The second question is how the time required for Donaldson's algorithm scales in the degree k, and by proxy how much time is needed to achieve a given accuracy. While there is still room for optimizing the implementation of Donaldson's algorithm, which among other things allows for a much greater parallelism than used here, the computational time can reveal the scaling in k and serve as comparison for other approximation schemes. All time benchmarks presented in the following are obtained in the same environment, running on 10 parallel threads on the CPU. Since there is in each case room for optimization, the values should be read as order of magnitude and relative to each other.

Figure 3.2 shows the time needed for one iteration of the T-operator depending on the degree k. In addition, it illustrates the advantage of using JIT compilation (used frequently in this and the following implementations) for computing the σ-measure, which reduces the overhead of the Python implementation but not the overall scaling of the computational complexity. It is obvious that the computational cost of Donaldson's algorithm rises steeply, making it unfeasible to run on a single standard workstation for $k > 12$ in a time span of days. An inspection of Donaldson's algorithm reveals that there are $O\left({N_k}^2\right)$ evaluations of the integrand, each containing an inner product with the $N_k \times N_k$ matrix h, leading to an overall scaling of

$$O\left({N_k}^4\right) = O\left(k^{4(n+1)}\right). \tag{3.3}$$

For the quintic this is $O\left(k^{16}\right)$. While not the focus here, note that it is possible to reduce the computational cost (although not the overall scaling in k) by specializing to set of defining equations and manually exploiting their symmetry to reduce the number of free parameters in

[1] The σ-measure has the additional advantage as a reference measure between published results, that it does not depend on the pre-factor in front of the logarithm in the definition of the algebraic metric, or on whether or not the Monte Carlo measures were properly normalized. In other words, it is independent of the volume of the manifold.

h, as was done in [10]. Since the Calabi-Yau metric is unique, it has to respect all symmetries of the defining equation, which can be translated to numerical conditions on h.

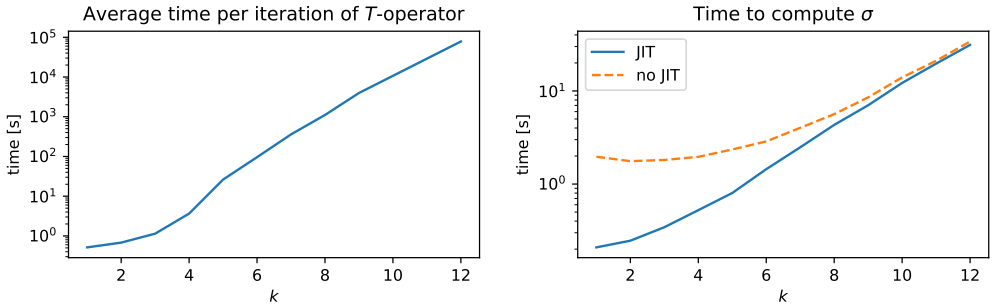

Figure 3.2.: Scaling in k of the computational time for one iteration of Donaldson's T-operator, and the evaluation of the σ-accuracy measure. The time for the T-operator is the average over 15 iterations for the quintic with $\psi = 0$. The σ accuracies were each computed with 10 000 sample points, once with and once without JIT-compilation enabled, which reduces the overhead.

3.2. Moduli and Random Seed Dependence

Donaldson's algorithm has two primary parameters, the degree k and the number of sample points used in the Monte Carlo integration. The convergence relative to the σ-measure in the degree k has been shown in Figure 3.1 for the Fermat quintic. In the same figure, convergence after just a few iterations of the T-operator is visible. This indicates that the number of sample points has been selected sufficiently large.

To confirm that the observed scaling of the accuracy measure in k is not unique to the Fermat quintic, Figure 3.3 shows the σ-measures achieved by Donaldson's algorithm for different values of ψ over a range of degrees k. All points in moduli space displayed in the figure seem to show similar convergence in the limit, with slightly different offsets (since shown on a logarithmic scale, the offset corresponds to different factors of k^{-2}). Since the conifold at $\psi = -5$ has local regions of large curvature, one may expect the balanced metrics to give relatively worse accuracies at that point in moduli space. The reason for this is that the algebraic metrics can be interpreted as a spectral expansion, with more localized fluctuations requiring larger values of k (exactly as for a Fourier expansion). The values shown in the figure are compatible with that being the case, however a scan to larger values of k would have to be done for a final conclusion to be drawn.

As mentioned previously, the σ-measure will be used as the main accuracy measure in order for the different results to be easily compared. The analysis of Donaldson's algorithm provides an opportunity to compare it to the more expensive $||R||$-measure, and check their consistency. The same figure as above, Figure 3.3, shows the correlation between σ- and $||R||$-measures for the balanced metric for different values of ψ. The two measures agree closely, warranting the use of the cheaper σ-measure over the $||R||$-measure. This is especially important for the machine

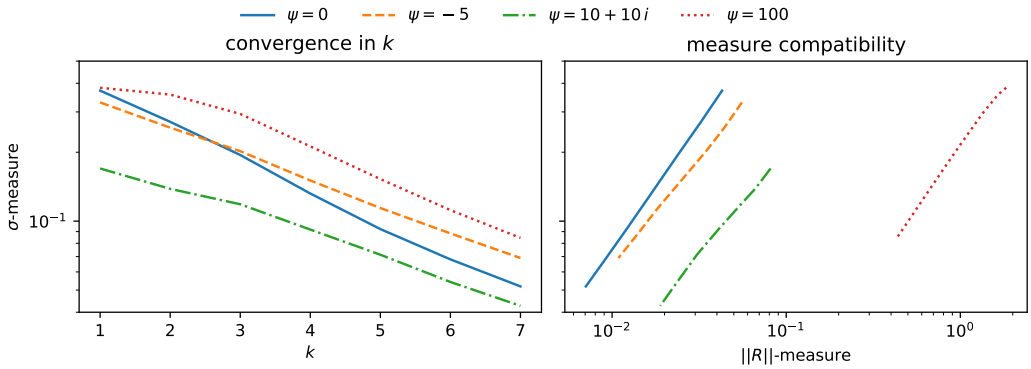

Figure 3.3.: Right: σ-measures achieved by Donaldson's algorithm in a range of degrees k for different ψ values. Left: Correlation of the σ- and $||R||$-measures computed over a fixed set of 10 000 sample points for the results of Donaldson's algorithm with k between 1 and 7 and different ψ values on a double-logarithmic scale.

learning application later, since the loss is used in every iteration of gradient descent.

Besides the ψ (i.e. complex structure moduli) dependence of the convergence of Donaldson's algorithm, one may wonder if the accuracies achieved depend on the random seed. Since random numbers are only used to compute the Monte Carlo approximation of the T-operator, significant variation would indicate insufficiently large sample sizes. Figure 3.4 shows the relative standard deviation of σ accuracies achieved by Donaldson's algorithm over 10 runs with different random seeds. The magnitude of variation is comparable to the accuracy with which the σ-measures are approximated (as analyzed in section 2.4), which indicates no significant seed dependence, and thus a sufficiently large sample size. Each of the σ-measures in this plot were computed using the same 10 000 points, so the variation comes only from the randomness in Donaldson's algorithm, not the approximation of the accuracy.

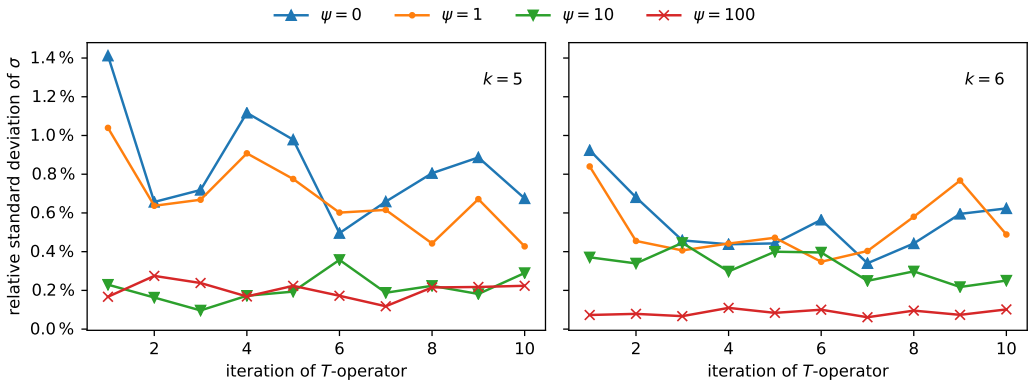

Figure 3.4.: Standard deviation divided by mean of the σ-measures obtained using Donaldson's algorithm for each iteration computed over 10 runs with different random seeds. The results are for degree $k = 5$ on the left and $k = 6$ on the right, each for 4 different values of ψ.

3.3. Numerical Pattern of h Matrices

So far, only the convergence of the σ-accuracies achieved by Donaldson's algorithm have been analyzed but not the numerical values of the h matrices themselves. A qualitative overview of h matrices produced by Donaldson's algorithm is shown in Figure 3.9, which displays the absolute value of entries of h as a logarithmic heat map. Besides consistently large values on the diagonal, it is difficult to recognize a pattern. Close inspection reveals however, that there seem to be a few distinct off-diagonal entries with relatively large magnitude. The remainder of this section analyses the structure in the h matrices produced by Donaldson's algorithm, and in how far only a few entries are significant in producing the balanced metrics.

The algebraic metrics are manifestly invariant under multiplications of the matrix h by a constant $0 \neq \lambda \in \mathbb{R}$

$$h_{\alpha\bar{\beta}} \sim \lambda\, h_{\alpha\bar{\beta}}\,, \tag{3.4}$$

since the potential (2.26) shifts by a constant which does not contribute to the derivative. Figure 3.5 shows the absolute values of diagonal entries of h at $k = 7$, as well as the eigenvalues (which are real since h is Hermitian) sorted by magnitude. It indicates that on the positive real strip of ψ values, the larger ψ is the more dominant a few of the entries of h become, and the larger the span of the spectrum. The absolute values and the spectrum are numerically close, which can be attributed to the fact that the h matrices are numerically dominated by entries on the diagonal. In fact, Figure 3.6 shows that for small values of k there is almost no difference in the σ-measure if off-diagonal entries are discarded (i.e. set to 0). However, for larger k, and especially for larger values of ψ, the accuracy is significantly reduced, which means that the off-diagonal entries must be included. To come to this conclusion, in Figure 3.6 the relative increase of the σ-measure defined as

$$\frac{\sigma(h) - \sigma(h_{\text{reduced}})}{\sigma(h)} \tag{3.5}$$

is shown, where h is the original matrix, h_{reduced} the modified one, and $\sigma(h)$ denotes the σ-measure computed for the algebraic metric given by the matrix h. Naturally, the relative decrease of the σ-measure is defined by inverting the sign.

The integral of the T-operator is approximated by a Monte Carlo sum over a finite number of sample points. This means that in each step the random choice of sample points leads to random fluctuations in the produced h matrix, even after the σ-measures no longer change meaningfully. Entries that would be zero if the integral was computed exactly may therefore continue to fluctuate. In order to determine which of the entries in h contribute significantly and which may constitute noise, consider the element-wise relative standard deviation over a

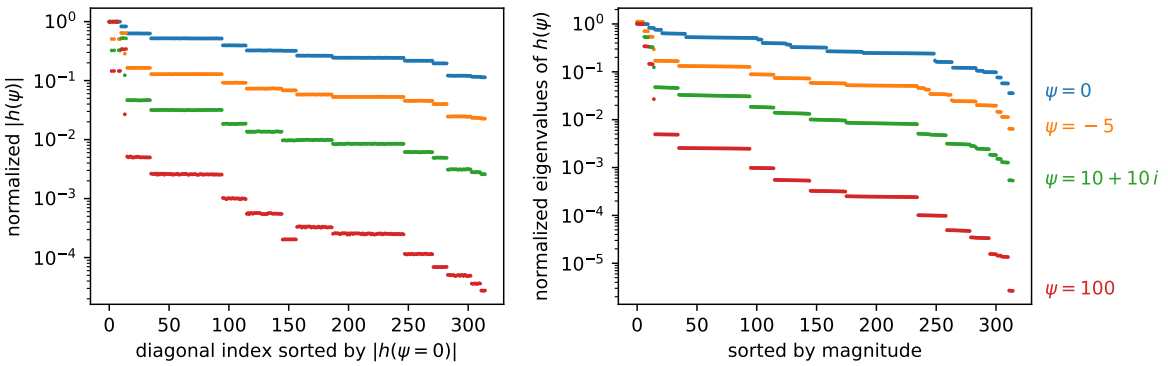

Figure 3.5.: Left: Absolute values of the diagonal entries of h matrices produced by Donaldson's algorithm after 10 iterations for $k = 7$ and different values of ψ. The indices are consistent between all ψ and were obtained by sorting the diagonal entries of h for $\psi = 0$. Right: Spectrum of eigenvalues of the same matrices, individually sorted by magnitude. For both plots, the h matrices were normalized such that $\max(|h|) = 1$.

set of matrices $\{h^m\}_{m=1}^M$ defined as

$$\text{rel-std}(h^m)_{\alpha\bar{\beta}} = \frac{1}{\left|\langle h_{\alpha\bar{\beta}}\rangle\right|}\sqrt{\frac{1}{M}\sum_{m=1}^{M}\left|h^m_{\alpha\bar{\beta}} - \langle h_{\alpha\bar{\beta}}\rangle\right|^2}, \tag{3.6}$$

where the angle brackets denote the mean

$$\langle h_{\alpha\bar{\beta}}\rangle = \frac{1}{M}\sum_{m=1}^{M} h^m_{\alpha\bar{\beta}}\,. \tag{3.7}$$

Since it removes the dependence on the absolute scaling of elements of h, this is a good measure for their fluctuation over the given set.

Figure 3.7 shows the relative standard deviation over iterations of Donaldson's algorithm for multiple values of k and ψ in relation to their absolute value. In other words, it shows how much each entry fluctuates between applications of the T-operator, and how this relates to how large the absolute value (on average over the iterations) of the entry is. Crucially, the standard deviation is computed over a number of iterations of Donaldson's algorithm after convergence has already been reached. There are visibly two clusters of points, whose relative standard deviations differ by about two orders of magnitude. The relative standard deviation of the larger-fluctuation cluster (shown in blue) is of the order $O(1)$, which means their fluctuation is of about the same size as their magnitude, strongly suggesting they are noise. From the perspective of the Monte Carlo approximation, the entries in the large fluctuation cluster may therefore vanish if the integral was computed exactly, but due to the introduced randomness fluctuate around that value. This defines a prescription for reducing the h matrices to only those entries which are statistically significant. Figure 3.10 shows again a qualitative overview of h matrices produced by Donaldson's algorithm but only keeping the small-fluctuation entries. Now a distinct pattern is visible, which seems to show some resemblance between different values

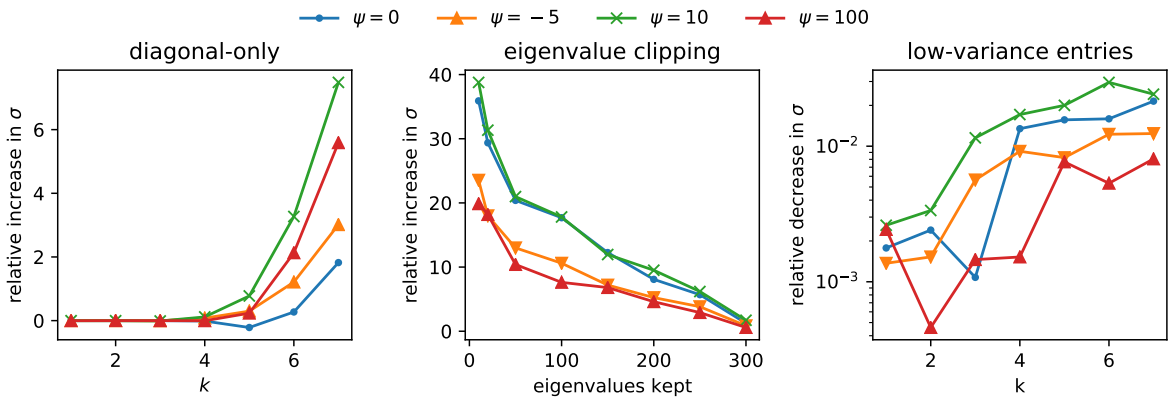

Figure 3.6.: Left: Relative decrease of the σ-measure once h matrices produced by Donaldson's algorithm for various ψ and k values are reduced to the small-fluctuation entries. Middle: Relative increase in σ-measures obtained for h matrices produced by Donaldson's algorithm for $k = 7$ when keeping different numbers of eigenvalues and setting the rest to 0. In total, h has 315 eigenvalues. Right: Relative increase of the σ-measure obtained when setting all off-diagonal entries of the h matrices produced by Donaldson's algorithm to 0.

of k and ψ.

Another notable observation about Figure 3.7 is that the difference in fluctuation between the large- and the small- fluctuation clusters do not change significantly depending on ψ or k. It seems justified, at least for the quintic studied here, to set a threshold for the low-fluctuation cluster at a relative standard deviation of 10^{-1}. Since we seem to have established that the large-fluctuation cluster corresponds to noise around zero values, one may suspect that elements in the large-fluctuation cluster must be small and those in the small-fluctuation cluster large. While the figure does show that small-fluctuation entries tend to have a larger magnitude, it is not possible to separate the clusters based on magnitude alone. That is especially so for larger values of ψ and k, for which the two clusters overlap in magnitude.

The reduction of the h matrices to the small-fluctuation elements can, through a purely numerical analysis, only be justified if the σ-accuracies do not change significantly. Figure 3.6 shows that by setting all large-fluctuation elements to 0, the σ measure changes by at most about 1 %. For all tested cases, the reduction in fact leads to a decrease in the σ-accuracy.

The number of elements in the low-fluctuation cluster increases with the value of ψ. In fact, the elements of h kept for a smaller ψ is always a subset of those kept for a larger ψ. This seems to show a reflection of the geometry in the h matrices, corresponding to the symmetry breaking that happens with the introduction of $\psi > 0$. The discarded large-fluctuation elements for $\psi = 0$ contain entries that are not required for the variety at $\psi = 0$, but become continuously more significant as ψ increases and part of the symmetry of the homogeneous polynomial gets broken. The number of entries in the low-fluctuation cluster in relation to the value of ψ can seen in Figure C.3 in the appendix.

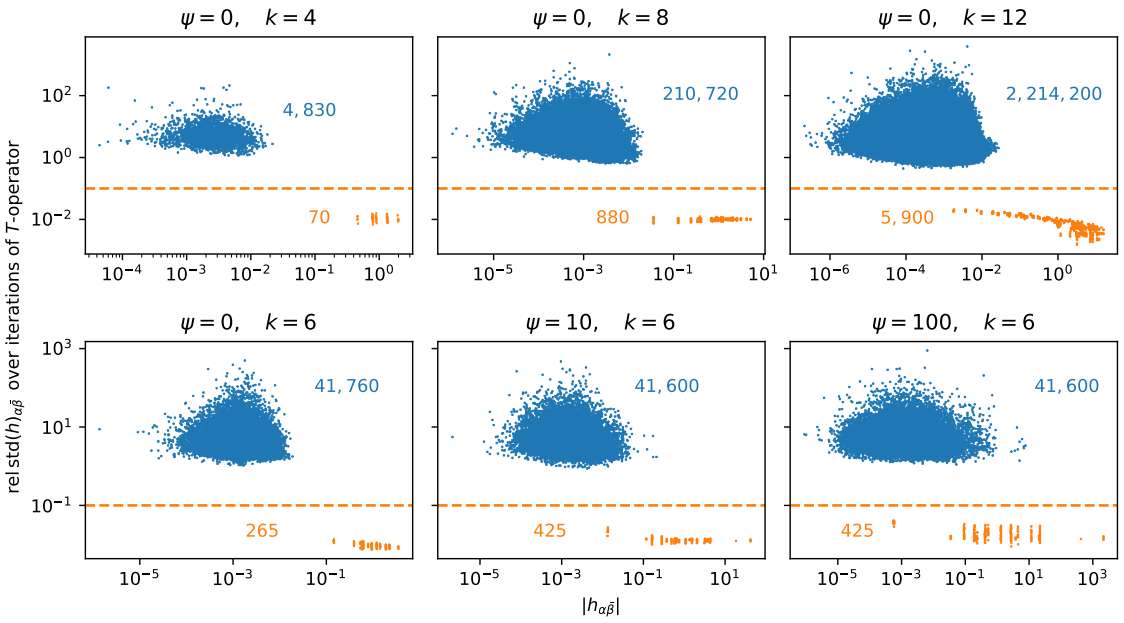

Figure 3.7.: Relative standard deviation (3.6) of entries in h computed over iterations 10 to 25 of Donaldson's algorithm for different values of ψ and k, in relation to the absolute of their mean. The differently colored clusters correspond to a cut-off value for the relative standard deviation of 10^{-1}, shown as a dashed line. The small numbers inside the plot indicate the number of entries in each cluster.

So far, the complex angle of entries in h have not been analyzed. Figure 3.8 shows the fluctuation of elements of h over iterations of Donaldson's algorithm in relation to their complex angle. It is notable that for $\psi = 0$, all small-fluctuation entries are real. In the case of $|\psi| > 0$ there seem to be only a few different complex angles in the small-variance cluster, whose value changes depending on the complex angle of ψ. The mirror symmetry visible in both figures is due to the Hermiticity of h, which requires that for every entry of h with complex angle α there is one with angle $-\alpha$.

3.3.1. Implications

In the above we have seen that, just based on an analysis of the numerical values produced by Donaldson's algorithm, it is possible to reduce the h matrices to only a few entries. One may suspect that these entries are not only the significant ones for Donaldson's balanced metric, but for any approximation of the Ricci-flat metric at the given degree k of algebraic metrics. Limiting oneself to only these few entries would speed up solving the minimization problem introduced in the following section 4.

Since the Ricci-flat metric is unique, it has to respect the symmetries of the defining homogeneous polynomial. The defining equation Q_ψ of the quintic studied here displays a large symmetry group (such as permutations of coordinates), making it possible to manually reduce the number of parameters required to parametrize h. Neither the reduction of entries based on

a numerical study of Donaldson's algorithm, nor the manual reduction of parameters of h based on symmetry leads to a reduction of the basis of $\mathcal{O}_X(k)$. The middle plot of Figure 3.6 shows the increase of the σ-measure that results from discarding eigenvalues of h. The σ-measure immediately increases once eigenvalues are discarded, which indicates also numerically that one cannot hope to find only a few homogeneous polynomials to which the construction of the algebraic metric can be limited.

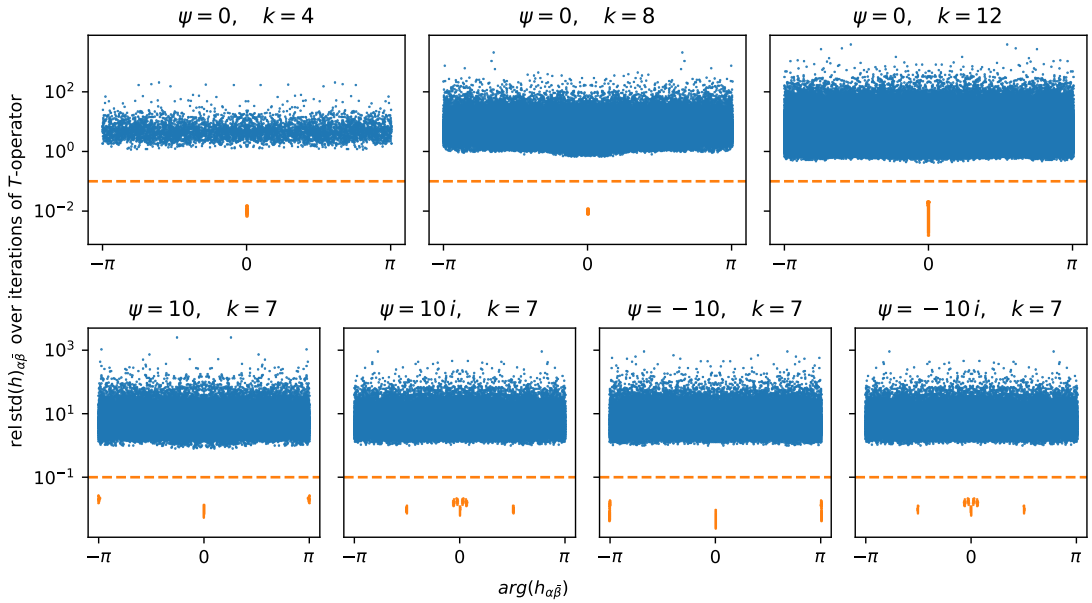

Figure 3.8.: Relative standard deviation of entries in h computed over iterations 10 to 25 of Donaldson's algorithm for different values of k and ψ, in relation to their complex angle. Similarly to Figure 3.7, the cut-off line separating the two clusters is shown as a dashed line.

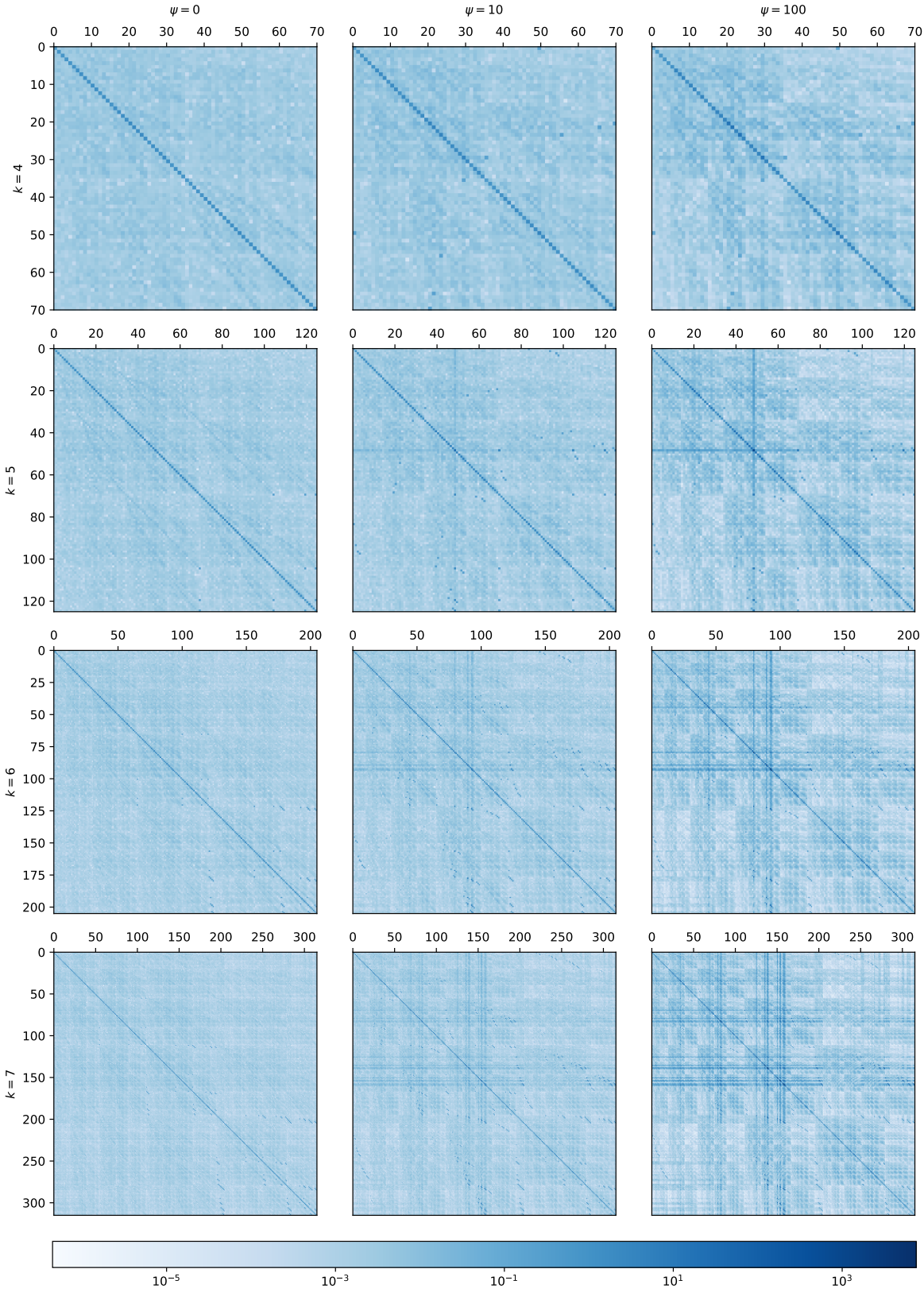

Figure 3.9.: Qualitative overview of h matrices computed by 15 iterations of Donaldson's algorithm for different values of ψ and degrees k. The color corresponds to the absolute value of entries of h, shown on a logarithmic scale.

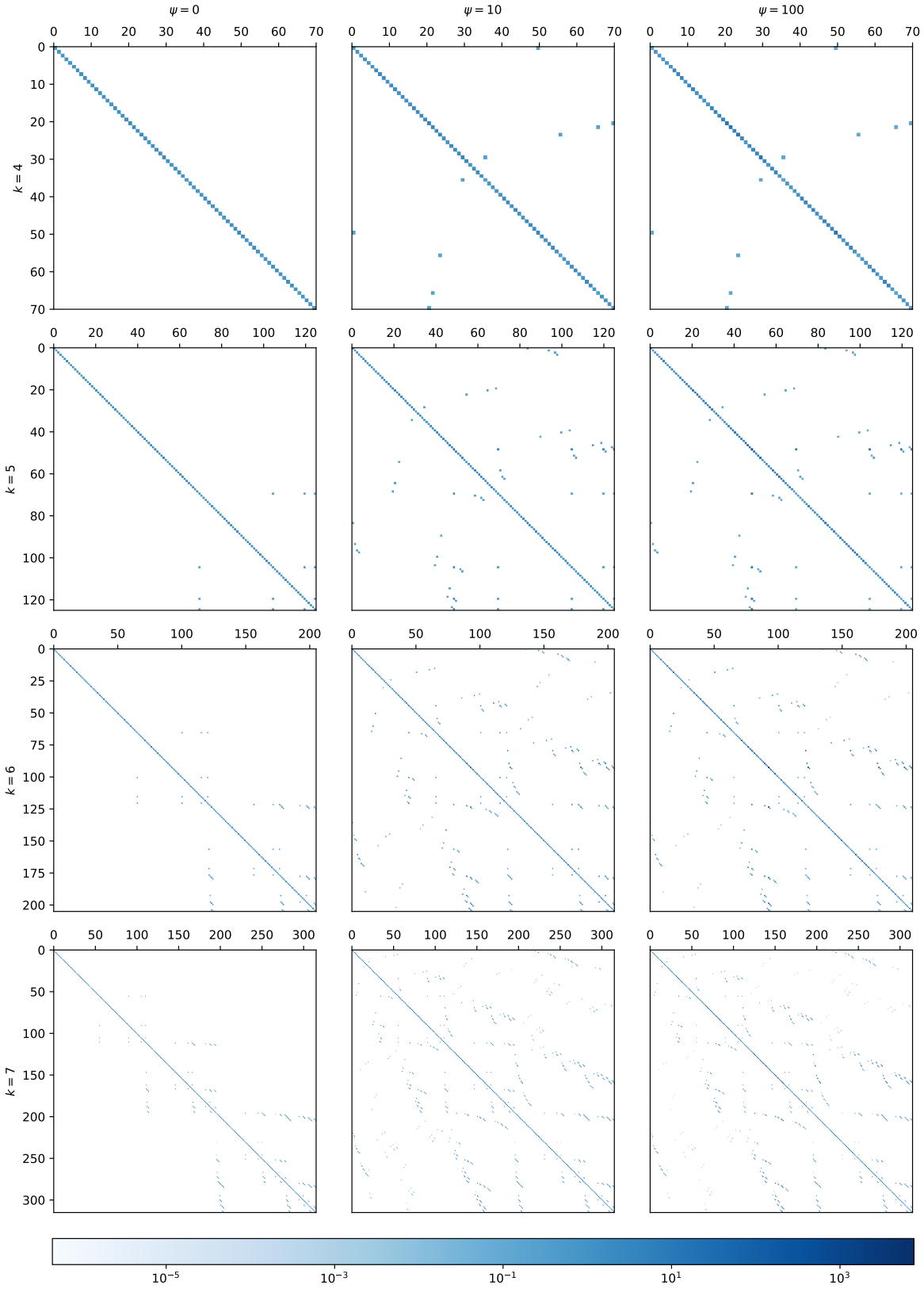

Figure 3.10.: The same as Figure 3.9, except with h matrices reduced to the entries with small relative variance. The same color map was used.

4. Machine Learning Approach to Calabi-Yau Metrics

In the previous section, we have seen how Donaldson's algorithm can be used to approximate the Ricci-flat metric on a Calabi-Yau manifold using the algebraic metrics represented by Hermitian matrices h. The quality of these approximations was measured using one of the integral accuracy measures introduced in section 2.4. Changing the perspective slightly, the problem of finding a good approximation corresponds to finding a metric which makes these accuracy measures as small as possible. This is justified in principle by the investigation of Headrick and Nassar [10], which showed that the η-based energy functionals of equation (2.49) are convex in the Kähler form and uniquely minimized by the Calabi-Yau metric. We are thus justified in claiming a metric with a smaller accuracy measure is a better approximation.

Fully committing to this point of view, the problem can be put into the context of deep learning, in which one aims to find a parametrized set of functions, and optimize the parameters by minimizing a specified loss (which will be chosen to correspond to the accuracy measures). The results presented in section 4.6 will show that this approach is capable of producing approximations to the Ricci-flat metric of better accuracy than Donaldson's algorithm. Not only will a better accuracy than Donaldson's algorithm be achieved in a similar amount of time, but the approximate Calabi-Yau metrics are simultaneously found on a range of moduli space. The metrics are present in functional form built on the algebraic potential, and can therefore be differentiated and cheaply computed at different values of the moduli parameters.

Section 4.1 gives a basic overview of deep learning and the related vocabulary that is important to understand the following approach. Before moving towards the final formulation of the machine learning optimization scheme, section 4.2 outlines the different possible and previous applications of machine learning to the problem at hand. After an overview of the loss functions derived from the accuracy measures in section 4.4, and a more narrow definition of which deep learning approach is taken here, in section 4.3, section 4.5 reproduces the results in [10] of minimizing the energy functionals (2.49) for the algebraic metrics at a fixed point in moduli space. This will be done in the context of machine learning and gradient descent, laying the foundation for section 4.6 in which approximations for a range of complex moduli space are found simultaneously.

4.1. Deep Learning

Deep learning[1] is a subfield of machine learning, roughly concerned with finding the optimal parameters of a parameterized function which is defined by the composition of algebraic operations ("deep" refers to multiple chained layers of operations). Optimality is typically defined as the minimum with respect to some loss function, mapping the output of the model and its input to a real number, such that optimal parameters can be approximated using variants of gradient descent[2]. The specific parametrized function is called a model, and the combination of algebraic operations that define the mapping from parameters and inputs to the outputs is called its architecture. Because the models in deep learning are usually defined by composition of layers of multivariate linear and non-linear functions, they are also called networks, or neural networks. Each dimension of the inputs and outputs of these intermediate multivariate functions can be interpreted as a node in a graph, whose connections are given by the dependencies as defined by the functions that map between them. In the case here, the output of one multivariate function is the input to the next, before eventually giving the final output. For this reason they are classified as so-called feed-forward neural networks. The process of approximating optimal parameters using methods based on gradient descent is called training. The output of a model is sometimes called a prediction, in analogy to the language used in regression.

To formulate a deep learning problem based on these definitions, one has to specify what the inputs and outputs of the model are, and what loss function is to be minimized. Based on the geometric setting summarized in section 2, one may try to find models predicting any of the objects describing the Calabi-Yau manifold, such as the metric itself, its determinant or other derived objects, or its Kähler potential. This could be done either as a function of the point on the manifold, or as its values on a fixed set of points. Candidates for the input of the model are therefore points on the manifold, and points in moduli space (e.g. the parameter ψ). The input and output dimensions of a model do not typically depend on the value of inputs or parameters. For a model that is based on algebraic metrics, the degree k is therefore not an input but rather a hyperparameter. A hyperparameter is a parameter that is not changed in the optimization process but rather parametrizes a set of models or aspects of the training procedure.[3]

To make the usage of models and losses in the following sections clearer, consider the following example. Assume we are looking to fit a polynomial using gradient descent to reproduce data that was generated using $y = f(x) = 2x^2$ (the functional form would in practice not be known, of course). Given a set of example inputs x_i and example outputs y_i, we can devise a model $f_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$ and minimize the mean-squared-error loss

$$\mathrm{MSE}(y_i, \hat{y}_i) = \frac{1}{N} \sum_{i=1} N (y_i - \hat{y}_i)^2 \,. \tag{4.1}$$

[1] For an extensive introduction to deep learning see for example [26].
[2] For the numerical results in the following, mostly the Adam optimization algorithm will be used [27].
[3] Sometimes hyperparameters only refer to parameters that change the optimization process (not the model), but it will be used in this broader sense in the following.

For a given set of values (x_i, y_i), called a batch, we can substitute $\hat{y}_i = f_\theta(x_i)$ to obtain the loss as function of θ, by which one can differentiate and thus minimize via gradient descent. Because both example inputs and outputs are available, this setting is called supervised learning. The degree of the polynomial ansatz which defines the model constitutes a hyperparameter, and defines how many scalar parameters θ contains. Assume now we have a functional $F(f) = \partial f/\partial x$, which for the sought function has the form $4x$. This may seem somewhat contrived, but the language in this example is closely mirrored the following application. Given again a model f_θ, we can define a new model as $\tilde{f}_\theta = F(f_\theta)$. We can then use gradient descent to find the target function (of course not all polynomial coefficients will be fixed) as the minimum of

$$MSE(x_i, \hat{y}_i) = \frac{1}{N} \sum_{i=1} N(4x_i - \hat{y}_i)\,, \tag{4.2}$$

where again to obtain a function of the parameters one substitutes $\hat{y}_i = F(f_\theta)(x_i)$. Note that in this example no pairs of input and output values is given. Instead, one may generate sample points x_i freely to construct the loss function. This process of computationally deriving a new model from a given one is not common in machine learning, but will turn out to be exactly what is needed here. The ability to automatically compute the derivative of the model, and still be able to differentiate the loss with respect to the parameter (thus effectively leading to a second order derivative) was one of the main criteria for which machine learning framework was used for the following approach. We will come back to this in the text below, and appendix A and B detail this on the level of implementation.

4.2. Overview of Machine Learning Approaches

Before laying out the approach taken here to approximate the Calabi-Yau metric, the following gives an overview of approaches that have been taken in previous studies, and possible other directions. For each, some basic advantages and disadvantages are noted. The various approaches differ in multiple ways, most importantly by which object the models aim to predict. A second difference mirrors the distinction between the two kinds of approximation schemes outlined in the introduction. The models may either take the point on the manifold as input and predict the corresponding local object, or the points on the manifold may be fixed such that the model predicts the desired objects on the chosen points simultaneously.

Extrapolating to Larger Degrees of the Balanced Metrics

Donaldson's algorithm as discussed previously gives a convergent algorithm approximating the Calabi-Yau metric. However, to achieve higher accuracies the degree k has to be increased, which quickly makes Donaldson's algorithm prohibitively expensive. It would, therefore, be useful if one could predict the balanced metric at $k+1$ based on the results from Donaldson's algorithm at k. One approach may be to find a model that maps from one h matrix produced

by Donaldson's algorithm to one at a higher degree, as a function also of ψ. This falls under the category of supervised learning, where Donaldson's algorithm would be used to compute h for multiple values of ψ, and the loss would be a function of the difference between h at $k+1$ and the output of the model given h at k and ψ. An immediate disadvantage is that the input and output sizes depend on k so that a model that maps from k to $k+1$ cannot be used to map from $k+1$ to $k+2$. Additionally, to train the model one has to evaluate Donaldson's algorithm for both k and $k+1$. This approach could therefore only be justified if the model somehow generalizes very well to values of ψ outside the set on which it was trained. Otherwise, nothing would be gained over Donaldson's algorithm.

A related approach is to predict not h of Donaldson's algorithm but rather the corresponding values of the metric or derived objects at a fixed number of points on the manifold. These do not scale in k, since they are defined as geometric objects independent of the algebraic ansatz. A first step in this direction was taken in [11], where decision trees are used to predict the determinant of the metric at larger k based on results from Donaldson's algorithm at smaller k. For some fixed Kähler manifold (e.g. fixed ψ), Donaldson's algorithm is evaluated for several small values of k which is then extrapolated with the help of machine learning to higher values of k. Even if extended to the full metric on each point, this approach has several drawbacks. Firstly, if the output of the model is the metric, Kählerity is no longer guaranteed. Instead one could predict the Kähler potential on a set of points, however then the metric has to be computed using a finite elements approximation. Consequently, in order to achieve higher accuracies the number of points has to be increased, which reverses the advantage that the model does not grow quickly in k. In addition to this, just as in the previous approach, it has to be trained with results from Donaldson's algorithm at larger values of k, which makes it questionable whether any meaningful improvement over it would be achieved.

Predicting Balanced Metrics at a Fixed Degree

Instead of a model that extrapolates from some k to $k+1$, one may try to find a model that predicts the balanced metric as a function of the point z and potentially ψ for a fixed k. If the model has an arbitrary form, one has to introduce multiple patches on the ambient projective space over which the metric has a well defined numerical value. It is in general still not guaranteed that the learned metric (which is now present in functional form in z) would be Kähler. Since the local functions representing the metric are not globally defined, a different model has to be trained for each patch, and on each overlap a compatibility condition has to be satisfied (namely that after a coordinate transformation the predicted metrics are the same). The first issue can be addressed by using a model that predicts the local Kähler potential, but one still has to enforce boundary conditions. Both issues can be avoided by using the approach of algebraic metrics. Since they are defined by a Kähler potential, the derived metrics are guaranteed to be Kähler. As discussed in section 2.2.1, a given matrix h implicitly defines Kähler potentials in each patch whose metrics are automatically compatible.

Predicting The Optimal Algebraic Metric

So far, all formulated deep learning approaches are based on some kind of supervised learning with the results from Donaldson's algorithm. It will prove useful, however, to remember that the problem is not fundamentally to approximate Donaldson's algorithm, but rather to approximate the Ricci-flat metric. Stepping away from Donaldson's algorithm and searching for another loss function makes the deep learning approach a lot more ambitious, as it no longer falls under classic supervised learning. At the same time, however, it leads to a whole class of approaches that can be used in place of Donaldson's algorithm, and that hold the potential for many future improvements.

As was already noted by Donaldson [7], the balanced metrics are not the best possible approximation given by the algebraic metrics to the Ricci-flat metric for a given k. Instead, as demonstrated in [10], a better approach is to find the optimal algebraic metric defined as the minimum of the energy functional introduced in equation (2.49). Possible loss functions based on these functionals are explored in the next section. Since the algebraic metrics for a given k can be interpreted as a truncated spectral expansion, in analogy to Fourier expansions one would expect exponential convergence in k, much faster than what can be observed for Donaldson's balanced metrics.

We thus arrive at a very powerful formulation of the problem. Using the algebraic metrics as basis for the architectures of models, the predicted metrics are guaranteed to be Kähler and are automatically compatible between patches. Using loss functions inspired by Headrick and Nassar they can be trained to approximate the Calabi-Yau metric in functional form to potentially higher accuracy than Donaldson's algorithm at a given degree k. Even more ambitiously, the network may take ψ (or more moduli parameters) as input and thus simultaneously predict Calabi-Yau metrics on a subset of moduli space.

Predicting the Metric Using an Arbitrary Parametrization

A possible disadvantage of using the spectral expansion of algebraic metrics as basis for machine learning models is that they may not perform well on manifolds that are not geometrically uniform, since large values of k are required to describe localized areas of high curvature [10]. It may still be possible to use a similar approach as outlined here using a different expansion of the Kähler potential, since the energy functionals of [10] are still applicable.

One may generally use an arbitrary network predicting either the metric, or the Kähler potential in functional form, depending on the point on the manifold, and optimize it using the energy functionals. There are, however, two issues which make this approach potentially harder. One has to manually enforce overlap conditions between the local networks, since neither the potential nor the metric is globally well-defined. If the model predicts the metric directly, one additionally has to worry about whether or not the outputs are actually Kähler (unless one is not interested in the metric being Kähler). At the cost of additional complexity, it may be

possible to address both issues by adding more terms to the loss.

4.3. Algebraic Networks

Motivated by the advantages of the algebraic ansatz for the Kähler potential, the following outlines an approach to using its mathematical structure to define models for deep learning.

The algebraic potential can be split into two components, the Hermitian matrix h, and the basis $S^\alpha(z)$ (which in the definition is given by the full monomial basis $s^\alpha(z)$). This leads to two possible candidates of models, one kind that outputs a Hermitian matrix h, and another kind that parametrizes polynomials defining $S^\alpha(z)$. We have seen that in general it is not possible to reduce the number of elements in the polynomial basis. Models that output a polynomial basis must therefore in general amount to a linear superposition of all elements of the monomial basis. This can be absorbed, however, into a redefinition of the Hermitian matrix h. It thus seems that we can limit ourselves to models that produce Hermitian matrices h. A further investigation into models that compute superpositions may still be of interest in the future, as it is possible that this approach leads to better convergence behavior and stability in gradient descent.

The primary difference that is left between models (besides their specific architecture) is whether or not they take moduli parameters as input or not. Since the structure of the algebraic metrics already defines the dependence on the point z on the manifold, models that do not take moduli parameters as input are just parametrizations of the h matrix. Models that do depend on moduli can make use of the same basic parametrization, and can be seen as a generalization.

There are multiple objects of eventual interest, including the h matrices themselves, the metric, and derived geometric objects. A unifying way to think of the different models collectively is using the concept of model composition and transformation. If eventually we are interested in geometric objects, any model defining a part of the algebraic potential can be composed with a function that computes the Kähler potential or one of its derivatives. For example, a model that outputs an h matrix can be thought of as part of a second model that additionally takes the coordinate z and outputs the algebraic metric at the given points. Similarly, a model that predicts a basis $S^\alpha(z)$ can be combined with a (potentially constant) model that outputs a matrix h into a model that computes the the algebraic metric. The output of these final composite models can be used to compute the accuracy measures. The models described in the following replace parts of the algebraic Kähler potential, which thus inherit its properties. No matter what the input and output of a specific model itself is, it can always be extended to take a point on the manifold as input, and return the algebraic metric, the Kähler potential, or the Ricci curvature as output. The key principle is that a model, or even multiple models, composed with a function is again a model.

In order to understand how that works and how the models in the following are defined in

practice (i.e. in the implementation), it is important to appreciate the versatility of automatic differentiation that is provided by machine learning frameworks. Given any function, defined programmatically by the composition of algebraic operations, automatic differentiation can produced a new function that defines its derivative up to numerical accuracy. This derivative function can in turn be composed into a new function which can, again, be automatically differentiated, and so on. Given the setting of differentiable manifolds, it is clear that the ability to differentiate algebraically defined functions is a very useful one. Indeed, one could define a Kähler potential of any form and derive functions describing the metric, its determinant, and the Ricci curvature purely using automatic differentiation. Since here we consider the special form of algebraic potentials it is, however, slightly more efficient to implement parts of the derivative from Kähler potential to metric and Ricci curvature by hand, as detailed in appendix B (by using a manual implementation some repeated calculations can be avoided that automatic differentiation cannot possibly recognize as such). A description of how complex functions can be dealt with using automatic differentiation is given in section A.1 of the appendix. Automatic differentiation is still used to compute the holomorphic derivative (locally in each patch) of the sections $s^\alpha(z)$, which would therefore allow them to be substituted by an arbitrary model.

The most significant and most powerful use of automatic differentiation, however, appears for the training of the model. A loss function which takes the output of a model as input can, by composition, instead be considered as a function that takes the parameters of the model as input. Good parameters are then found by taking the gradient of the loss function with respect to the model parameters and minimizing it by gradient descent. The fact that this gradient can be automatically derived in a sufficiently efficient manner is the reason that the algebraic metric can be extended to more complex machine learning models, such as ones that take as an input a point in moduli space (in the case here the parameter ψ). It is also crucial that derivatives can be chained, since the loss function itself contains derivatives. Lastly, a model that takes moduli as input could in turn be differentiated with respect to moduli space, which may be useful in a future, physically motivated exploration of moduli space.

4.3.1. Models Predicting the Hermitian Matrix h

The most basic model is one that outputs the Hermitian matrix h with respect to the fixed monomial basis s^α on X. We can start by fixing the moduli (i.e. ψ), leading to a classical optimization problem. The model can be defined as input-free and just returns its parameters as the Hermitian matrix. Because no complexity is added to the algebraic metric, and because the energy functionals that describe the loss function is convex in the Kähler form (although not necessarily with respect to the parameters), one can expect good convergence in this case. The only thing left to the model architecture is how exactly the Hermitian matrix is parametrized. Although formulated in a different way, the resulting problem is effectively the same as was studied in [10]. This approach will serve as a starting point for understanding the use of gradient descent methods in the present geometric setting.

Instead of solving the problem for a single point in moduli space, one can attempt to train a model that returns h as a function of a point in moduli space. As opposed to the optimization problem for a fixed point in moduli space, many possible network architectures may be used to predict h given the values of the moduli. Finding a good network, which converges on some selected subset of moduli space, is the central difficulty. The advantage of this approach is that if a good architecture for the network is found and trained, approximations for the Calabi-Yau metric are defined in functional form on a continuous range of moduli space. A schematic overview of the setup can be found in Figure 4.1 below.

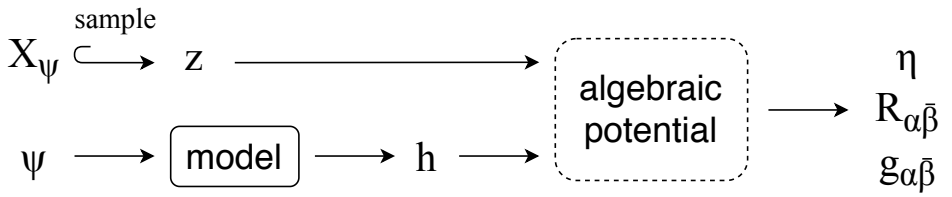

Figure 4.1.: Schematic overview of how models predicting the Hermitian matrix h are used. Since the Hermitian matrix parametrizes the algebraic metric, the output of the model implicitly defines geometric objects for each point on the manifold.

4.4. Calabi-Yau Losses

The basic idea for constructing loss functions is to take an integral accuracy measure that defines how close a given metric is to Ricci-flat, and approximate it using Monte Carlo integration on a batch of points. The way a machine learning model is then trained is as follows:

1. If the model depends on moduli parameters, sample them randomly from some predefined set.

2. Randomly generate a set of points $\{z_a\}_{a=1}^M$ on the manifold X using the line-intersection algorithm outlined in section 2.3.

3. Compute the loss function.

   a) Evaluate the model at the sampled values (moduli, point z).

   b) Construct the geometric objects required for the loss using the output of the model, the point z, and the value of the moduli parameters.

   c) Evaluate the Monte Carlo approximation defining the loss over the set of points $\{z_a\}_{a=1}^M$ and the geometric objects computed in the previous step.

4. Differentiate through step 3 with respect to the model parameters and update them via gradient descent. This is done using automatic differentiation.

One could cycle through a set of precomputed points on the manifold, however the majority of

the computational cost (by orders of magnitude) is needed to compute the geometric objects based on the output from the model, and to compute the derivative with respect to the parameters. In the following, randomly sampled points are thus used in each step. It is also possible to replace the sampling algorithm, even with one where the random measure is not explicitly known (the Monte Carlo approximations are then with respect to some implicit random measure). In how far this impacts the convergence of gradient descent has to be analyzed in each case.

The basis for a variety of loss functions that are minimized by the Ricci-flat Calabi-Yau metric is given by the energy functionals of Headrick and Nassar as defined in equation (2.49). Using as convex bounded function the square, the functional is

$$E = \int_X d\mathrm{Vol}_{CY} (\eta - 1)^2 \,. \tag{4.3}$$

Recall from section 2.4 that η is normalized with respect to the volumes

$$\eta = \hat{\eta} \frac{\mathrm{Vol}_K}{\mathrm{Vol}_{CY}} \,. \tag{4.4}$$

Unfortunately the volume Vol_K depends on the Kähler potential which is determined by the parameters of the model and thus changes in each step. A first loss can be derived using the fact that the sought Kähler potential leads to an $\hat{\eta}$ with vanishing variance:

$$E_1 = \frac{1}{M} \sum_{a=1}^{M} \left(\frac{\hat{\eta}(z_a)}{\langle \hat{\eta} \rangle} - 1 \right)^2 , \tag{4.5}$$

where $\hat{\eta}$ is normalized using the batch mean

$$\langle \hat{\eta} \rangle = \frac{1}{M} \sum_{a=1}^{M} \hat{\eta}(z_a) \,. \tag{4.6}$$

The loss E_1 is clearly just the normalized variance of $\hat{\eta}$. One can get a step closer to the energy functional by weighing the deviation of the normalized measure with the Monte Carlo weights $w(z)$,

$$E_2 = \frac{1}{M} \sum_{a=1}^{M} w(z_a) \left(\frac{\hat{\eta}(z_a)}{\langle \hat{\eta} \rangle} - 1 \right)^2 . \tag{4.7}$$

Finally, a third loss is obtained by normalizing with the mean computed using the Monte Carlo weights

$$\langle \hat{\eta} \rangle_w = \frac{\frac{1}{M} \sum_{a=1}^{M} w(z_a)\, \hat{\eta}(z_a)}{\frac{1}{M} \sum_{a=1}^{M} w(z_a)} \approx \frac{1}{\mathrm{Vol}_{CY}} \int_X d\mathrm{Vol}_{CY} \hat{\eta} = \frac{\mathrm{Vol}_K}{\mathrm{Vol}_{CY}} \tag{4.8}$$

which is just the Monte Carlo approximation of (4.3):

$$E_3 = \frac{1}{M}\sum_{a=1}^{M} w(z_a)\left(\frac{\hat{\eta}(z_a)}{\langle\hat{\eta}\rangle_w} - 1\right)^2 = \left\langle\left(\frac{\hat{\eta}}{\langle\hat{\eta}\rangle_w} - 1\right)^2\right\rangle_w . \tag{4.9}$$

The computational cost of adding Monte Carlo weights is negligible compared to the rest of the loss function.

As can be seen in Figure 4.2, the above measures are all well correlated with the σ-measures when computed using a batch of size $M = 1\,000$. The fact that E_1 seems to show the same correlation for all shown values of ψ should not be confused to mean it is superior, as the σ measure uses an absolute value while the losses approximate an integral of the squared term (4.3). For the models analyzed in section 4.5 all three losses were tested and no difference in convergence could be ascertained. The η-based loss function used in the following is E_3, since it most closely resembles the well-defined energy functional. If not otherwise stated, the batch size is $M = 1\,000$.

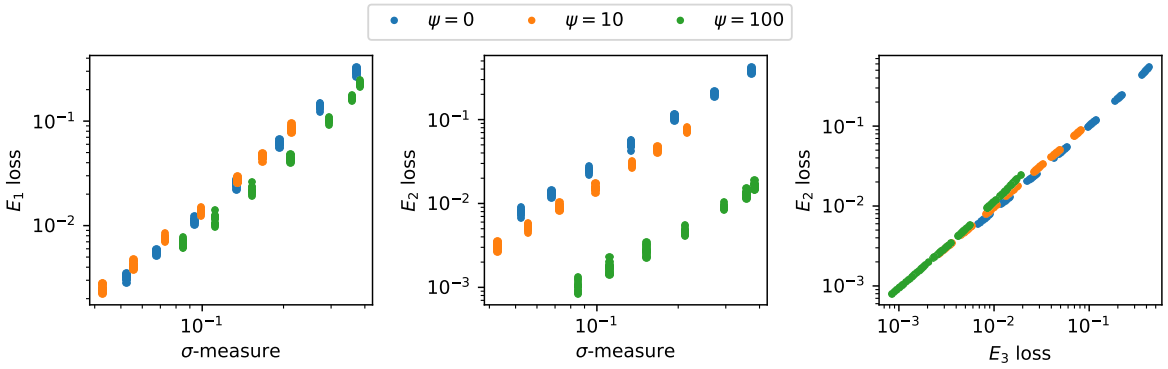

Figure 4.2.: Scatter plot relating the losses E_1 and E_2 computed on 1 000 sample points to the σ-measures computed on 10 000 sample points for balanced metrics at a range of values for k and multiple values of ψ. The plot on the right shows the correlation of the E_2 and E_3 loss. All losses are compatible with each other and with the σ-accuracy.

The η loss functions defined above depend on the second derivative of the Kähler potential. This was possible because of the simple form the Ricci curvature takes for Kähler metrics, which also led to the energy functionals. More obvious than the σ-measure is the $||R||$-measure, which is the integral of the absolute of the Ricci scalar over the manifold. One can similarly define a loss

$$R_0 = \frac{1}{M}\sum_{a=1}^{M} w(z_a)\,|R(z_a)|^2 . \tag{4.10}$$

The main disadvantage in the Calabi-Yau case is that it is significantly more expensive to evaluate than the losses before, since it depends on the fourth derivative of the Kähler potential. However, it has the advantage that it can be extended to the case of constant Ricci scalar, which

is not further investigated here,

$$R_c = \frac{1}{M} \sum_{a=1}^{M} w(z_a) \left| R(z_a) - c \right|^2 . \tag{4.11}$$

4.5. Optimizing h for Fixed Moduli

The most basic application of gradient descent is to find a matrix h such that the corresponding algebraic metric minimizes the losses defined in section 4.4 for a fixed value of ψ. This will serve as the starting point for the deep learning approaches, from the perspective of which it can be interpreted as an input-free model which parametrizes a Hermitian matrix.

Formulated as a classical minimization problem, one fixes a number of points on the manifold and minimizes the loss E_3 by changing the value of h. This has been done in [10] outside the context of deep learning. There are two significant differences that make the gradient descent approach different to merely replicating previous results. As opposed to [10], the symmetry of the defining polynomial will not be manually exploited to reduce the monomial basis on X. This means that the computational complexity will not increase significantly if more than one complex structure modulus is introduced, and one may expect the results here to generalize. Indeed, the implementation used for all following numerical studies does not require much modification in order to work for any defining homogeneous polynomial. Secondly, the formulation in terms of gradient descent provides a starting point for many possible deep learning extensions that go beyond the classical optimization problem. The advantage of gradient descent, and the reason for its predominant use in deep learning, is its relative robustness with respect to an expansion of the dimensionality of parameter space. After the following initial study, summarized in section 4.5.4, the parameters defining the single matrix h will be replaced with ones that parametrize a function mapping from moduli space to matrices h.

4.5.1. Parametrization

The analysis of Donaldson's algorithm, specifically the patterns of the h matrices explored in section 3.3, seem to suggest only a few entries are non-zero for good approximations to the Calabi-Yau metric. Because of random fluctuations in the loss function (due to the random sample of points over which they are computed), gradient descent sometimes has trouble setting values to zero. One effectively needs a mechanism for networks to learn to "turn off" some of the entries of h.

This can be done using the so-called sigmoid function (unfortunately colliding with the notation for the σ-measure; it should be clear from context which one is meant), defined as

$$\sigma(x) = \frac{e^x}{1 + e^x} . \tag{4.12}$$

If it is applied to a vector or matrix, it is calculated element-wise. Consider a value y which may be zero. Replacing the value with $y = \sigma(\tilde{y})\hat{y}$ using two auxiliary variables, it is easier to set it to zero by gradient descent. Whereas for the raw parameter y gradient descent may oscillate around zero, the re-parametrization can be arbitrarily suppressed by making $\tilde{y}$ smaller.

There are two simple ways to parametrize the Hermitian matrix h using real values. One method, which is used for the fixed-moduli optimization in the next section, is just in terms of the real and imaginary parts:

$$h = \begin{pmatrix} h^{\mathrm{d}}_1 & & h^{\mathrm{r}} + ih^{\mathrm{i}} \\ & \ddots & \\ h^{\mathrm{r}} - ih^{\mathrm{i}} & & h^{\mathrm{d}}_{N_k} \end{pmatrix}. \tag{4.13}$$

Here, h^{d} represents the real entries on the diagonal, while h^{r} and h^{i} respectively represent the real and imaginary entries on the upper and lower diagonals.

Since the h matrix represents a metric on the line bundle, it is a positive definite Hermitian matrix. A second possibility is therefore to parametrize it using the Cholesky decomposition, with the same base variables:

$$L = \begin{pmatrix} h^{\mathrm{d}}_1 & & h^{\mathrm{r}} + ih^{\mathrm{i}} \\ & \ddots & \\ 0 & & h^{\mathrm{d}}_{N_k} \end{pmatrix}, \quad h = LL^\dagger, \tag{4.14}$$

where now the diagonal entries are positive. This prevents negative or zero eigenvalues, which may lead to non-definite metrics on the manifold X.

The parametrization used in the following fixed-ψ optimization combines the former complex decomposition of h with sigmoid-suppression:

$$h^d = \sigma(\tilde{h}^d)\,\hat{h}^d, \quad h^r = \sigma(\tilde{h}^r)\,\hat{h}^r, \quad h^i = \sigma(\tilde{h}^i)\,\hat{h}^i. \tag{4.15}$$

4.5.2. Minimizing the η-Variance

The value of the matrix h for multiple fixed points ψ and degrees $1 \leq k < 7$ were optimized using gradient descent with the E_3 loss, and again using the minimization algorithm BFGS-L [28] (an abbreviation for limited-memory Broyden–Fletcher–Goldfarb–Shanno algorithm) which uses an approximation to the inverse of the Hessian matrix. Figure 4.3 shows the σ-accuracies achieved using either method after convergence was reached, and also the accuracies produced by Donaldson's algorithm at each k for $\psi = 0$. The first notable thing is that both methods consistently produce metrics that are significantly closer to Ricci-flat than Donaldson's algorithm, in a similar amount of time at each value of k. Indeed, the accuracies achieved using gradient descent for $\psi = 0$ for values $k > 4$, produced in minutes, are better than the one pro-

duced by Donaldson's algorithm for $k = 12$ in days (both implementations may be subject to optimization so the comparison should be taken as relative order of magnitude). Furthermore, the convergence in k is faster than that of Donaldson's algorithm. The results achieved by both optimization algorithm are similar to those produced in [10], where it was demonstrated that the convergence is exponential up to even larger values of k except around the singular point $\psi = -5$.[4] The fact that gradient descent seems to converge to similarly small values as the minimal values found by BFGS (in some cases even outperforming it) shows that it is a valid method for minimizing the loss functions and approximating the Ricci-flat metric.

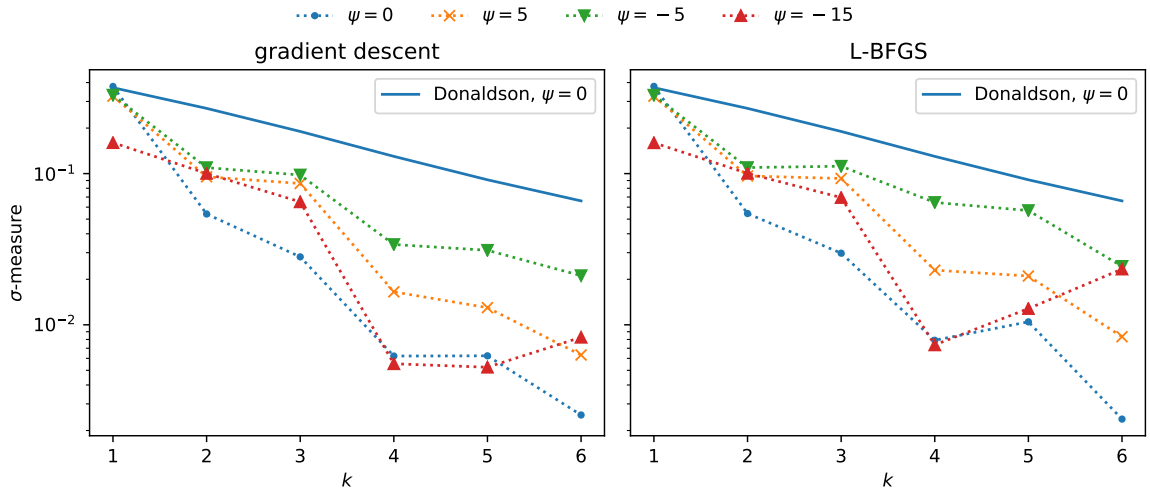

Figure 4.3.: σ-accuracies achieved by algebraic metrics where the h matrix was found using gradient descent in one case and the BFGS-L minimization algorithm in the other to minimize the E_3 loss, for multiple values of ψ and over a range of degrees k. The shown σ-measures were computed using the same set of sample points for both instances. Training using gradient descent was done using a batch size of 1 000, while the BFGS-L minimization was done over a set of 10 000 points. Accuracies achieved by Donaldson's algorithm are shown as a reference.

Hyperparameter Tuning

Another reason for studying this basic application of gradient descent, besides as proof of concept, is to find a good optimization algorithm and associated hyperparameters that can be reused later, where no direct comparison of the convergence with other optimization algorithms (from outside the field of deep learning) is available. The gradient descent optimization was tried using all three η losses defined in section 4.4 and different variations of optimization algorithms. All produced a similar convergence behavior and no significant difference in computational cost could be observed. Since it is closest to the integral energy functional, the η based loss used in the following is the E_3 loss of equation (4.9). The best gradient descent algorithm found, which is the one used to produce Figure 4.3, is Adam [27] with an initial learning rate of 0.01 that has to be decreased close to the point of convergence depending on the batch size. Batch sizes between 500 and 10 000 all led to a similar accuracy in the limit, with 1 000 points being

[4]Figure C.1 in the appendix reproduces Figure 4.3 using the same accuracy measure as was used in [10].

a good compromise between speed of convergence and computational cost.

As mentioned before, new random points on the manifold are sampled for each step of gradient descent. This does not lead to a significant increase in time, since it is fast in comparison to computing the loss function and its derivative. It additionally allows for smaller batch sizes to be used, since it avoids biasing the optimization with respect to the specific points used for training. If a fixed set of 10 000 points are used for each step of gradient descent, the accuracy as determined on the same set can be decreased to several orders of magnitude below the values reported above. However, when measured on a different set of points the accuracy is significantly worse. This is a well known observation in machine learning called over-fitting, with the former value known as training loss and the latter as validation loss. In many applications limited data makes a discrepancy between training and validation loss inevitable, however a nice property of geometric applications, such as this one, is that the amount of independent training data (here points on the manifold) can be freely generated as needed. The fact that using new samples in each step prevents over-fitting is also an explanation for the slightly better values achieved by gradient descent compared to BFGS, which uses a fixed set of points. The difference is more notable as k gets larger since then the specific set of 10 000 points become increasingly insufficient to determine all entries of h.

Comparison of Optimal and Balanced h Matrices

A direct comparison of the h matrices produced using the above optimization with the ones produced by Donaldson's algorithm previously shows a close similarity. In both cases, the matrix is dominated by the diagonal entries, with a few off-diagonal terms of similar magnitude. An example of this can be found in Figure C.5 in the appendix. In fact, reducing the optimal matrices to the small-fluctuation cluster as found using Donaldson's algorithm does not significantly change the σ-accuracies in any of the tested cases. This suggests that one could have limited the parameters of the optimization scheme to the small-fluctuation entries, if they are available. A comparison of the accuracies achieved by reducing the optimized h matrices to the low-fluctuation entries can be found in Figure C.4 in the appendix.

4.5.3. Minimizing the Ricci-Scalar

The second type of loss introduced in section 4.4 is one based on minimizing the absolute value of the Ricci scalar. Because it depends on the Kähler potential in its fourth derivative, it is significantly more expensive to compute than the η based loss studied above. Where the h matrix converged within minutes for the η based loss with $k \leq 6$, the Ricci based loss converged within tens of minutes.

Figure 4.4 shows the σ accuracies achieved by a similar gradient descent setup as in the previous section, using instead the Ricci scalar loss defined in equation (4.10). Convergence is similar to the one achieved using the η loss. This shows that gradient descent is in principle also possible

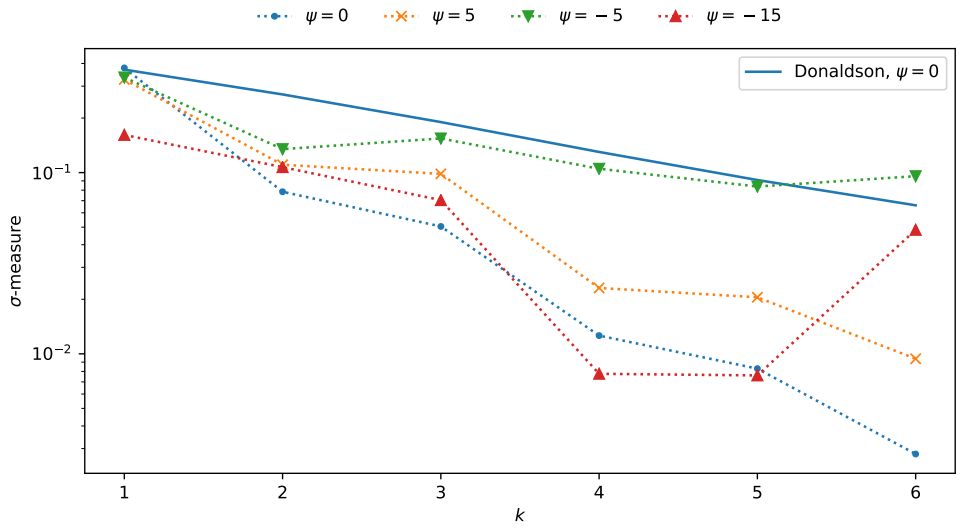

Figure 4.4.: σ-accuracies achieved by an optimization of the h matrix using gradient descent with respect to the Ricci loss of equation (4.10).

for more complicated loss functions, depending on higher derivatives of the Kähler potential. Due to its higher complexity, the Ricci loss will, however, not be used for the more advanced models studied in the following.

4.5.4. Summary

The results outlined here have shown that a deep learning approach using gradient descent is in principle possible. Convergence for larger values of k than shown Figure 4.3 and Figure 4.4 requires a more careful tuning of the learning rate, as well as a larger batch size to achieve convergence. A study of that, and an optimization of the hyperparameters to achieve convergence most quickly, requires a further scan of the hyperparameters and possibly exploring more variations of gradient descent and training schemes. Optimization could also be improved if the polynomial basis is reduced manually given each defining equation. The goal here, although an interesting one, is not to find an algorithm that produces the best possible approximation to the Ricci-flat metric for a given point in moduli space as fast as possible. Rather, the goal is a broad exploration of the application of deep learning models to the problem. In the following the models will be maps from moduli space to algebraic metrics, which is fundamentally a different approach. If one is interested in multiple points of moduli space, simultaneously training a model over a range of moduli space may prove advantageous with respect to computation time. More than that, the map from moduli space to Kähler potential will be available as a continuous and differentiable function which may outperform simple interpolation.

4.6. Moduli Dependent Learning of the Hermitian Matrix h

The full potential of the machine learning perspective on finding the Calabi-Yau metric only becomes clear once moduli-dependent networks are considered. We will use the single parameter ψ of equation (2.22) as an example for moduli parameters, and the implementation will not have to be modified significantly for a future study that expands the number of complex structure parameters considered. As a first proof of concept, section 4.6.2 investigates how networks can be trained to reproduce h matrices previously computed using Donaldson's algorithm at a fixed degree k. This effectively produces an interpolation between the ψ values of the training set, and serves as a first test for networks that map from ψ to h. As a compromise between accuracy and speed of running the optimizations, the degree will be fixed to $k = 6$ in the following.

As a next step, the networks trained to approximate the balanced metrics are further optimized with respect to the loss functions introduced above, which is outlined in the first part of section 4.6.3. We will see that this leads to better accuracies than Donaldson's algorithm in the range of ψ it is trained on. Computing Donaldson's algorithm for many values of ψ needed to train the network is expensive. In the final step of this initial analysis, detailed in the second part of section 4.6.3, networks are trained directly using the loss functions without previously learning the balanced metrics. Once again, accuracies below Donaldson's algorithm are reached on the range of training values, in a similar time as a single computation of Donaldson's algorithm. This amounts to a significant advancement, as approximations are simultaneously found for a range of the moduli space, to a better accuracy than previous algorithms.

In the remaining subsection 4.6.4, the optimization scheme using just the loss functions is reproduced with different network architectures. The following analysis presents a first exploration into moduli-dependent networks to approximate Calabi-Yau metrics. The final conclusion in section 5 reviews what the implications of the following results are, and what they mean for possible future improvements.

4.6.1. Architecture of Dense Layers

The main question for networks of the kind we are interested in now is how to map from ψ to the matrix h. One of the most common architectures of networks in machine learning is a linear superposition of a set of input variables to a set of output variables by matrix multiplication. This is typically followed by an element-wise non-linear function, called activation function, such as the sigmoid function introduced before. Since multiple of these transformations can be chained, and each input value is connected through the matrix multiplication to each output value, they are called dense layers. In the limit of arbitrarily large intermediate output spaces, one can show that any function can be approximated in this manner. A schematic overview of model architecture based on dense layers outlined below can be found in Figure 4.5.

The input to the network in our case is a single complex number ψ. Typically, deep learning

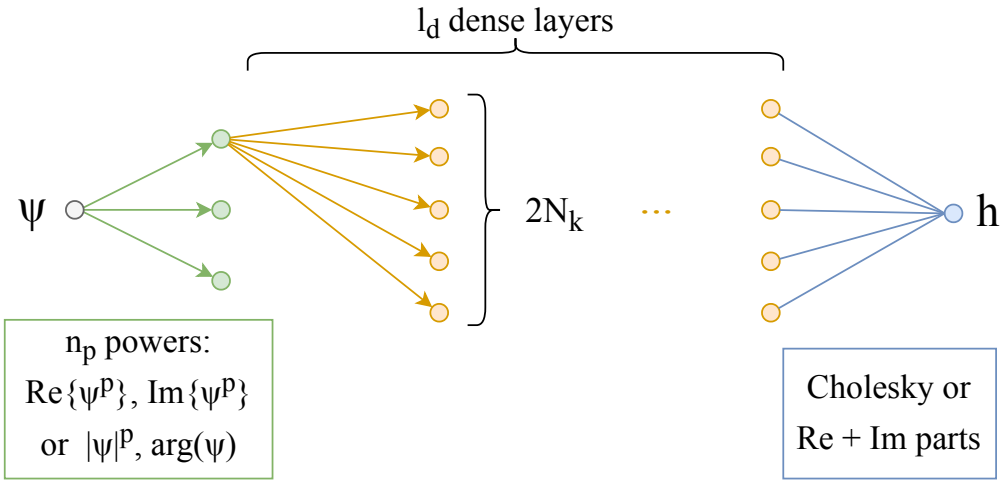

Figure 4.5.: Schematic overview of the dense layer architecture which maps from the single moduli parameter ψ to the Hermitian matrix h.

models take multiple values as input. As a first step of the network, on may thus try to generate multiple input values by taking powers of ψ[5],

$$P_i = \psi^{p_i} \,, \tag{4.16}$$

where p_i is some vector of real parameters, which may be fixed or trainable. The number of powers n_p, $i = 1, \ldots, n_p$, can be adjusted. The dense layer architecture is designed for real parameters, which means the values of the first layer will be split into real and imaginary parts

$$R = \left[\mathrm{Re}\{P_i\}, \mathrm{Im}\{P_i\}\right] , \tag{4.17}$$

where the notation in brackets represents concatenation.

Alternatively to taking powers of ψ directly, one may first separate it into absolute value and angle:

$$R = \left[|\psi|^{p_i}, \arg\psi\right] . \tag{4.18}$$

Let the number of values in R be denoted as n_R. One can now chain an arbitrary number of hidden dense layers (so called, because their outputs are intermediate), represented by matrices $M^t \in \mathrm{Mat}(\mathbb{R}, n_{t-1}, n_t)$ with $t = 1, \ldots, l_d$. l_d is the number of these dense layers, and $n_0 = n_R$. In general all layers may have different sizes. Motivated by the sparseness of h matrices found in the analysis of Donaldson's algorithm, and to reduce the number of hyperparameters, they will now be fixed to $n_t = 2N_k$. The heuristic for this choice is that N_k is the number of diagonal entries of h, and since most off-diagonal entries effectively vanished, there are likely much fewer

[5] In practice the networks are built to compute the output for multiple inputs, called a batch, simultaneously. This additional batch index is suppressed for the sake of simplicity.

independent values than the maximal N_k^2 real parameters[6]. If each dense layer was just a linear transformation, they could be collapsed into a single linear transformation. They are therefore followed by an element-wise non-linear activation function, in our case the sigmoid function. Concretely, the layers are computed as follows:

$$D_i^t = \sigma\left(\sum_j M^t{}_{ij}\, D_j^{t-1} + B^t{}_i\right). \tag{4.19}$$

The real vector $B^t{}_i$ is called bias, and the index t is the number of the layer ranging from $t = 1$ to $t = l_d$. The initial input is given by the real array of numbers constructed in the beginning, $D^0 = R$.

So far, the layers of the model produce an array of $n_{l_d} = 2N_k$ real numbers. In order construct a Hermitian matrix h as output, the last dense layer D^{l_d} is followed by another dense layer with N_k^2 number of outputs. This is exactly the number of real parameters of the h matrix, which means that either of the parametrization schemes introduced in section 4.5.1 can be used to produce the final Hermitian matrix. The magnitude of h matrix entries is not constrained, which means the final dense layer that produces the real parameters of h is not followed by a sigmoid activation function (whose output would be in $[0, 1]$).

The set of hyperparameters that have to be chosen are the number n_p of initial powers of ψ, the number of dense layers l_d, whether the angle and absolute values of ψ are separated, if the powers are fixed or trainable, and how the final real outputs are used to reconstruct the h matrix.

To a person unfamiliar with machine learning, this construction of functions, meant to bet fitted to predict $h(\psi)$, may seem arbitrary. In fact it is arbitrary, in the sense that it differs from a classical statistical approach where the model is carefully chosen such that its parameters have a concrete interpretation. The premise of machine learning is that one is less interested in the precise form of the sought function, than in the ability to predict, approximately, its value. The basic problem of finding a good network architecture is therefore to expand the range functions included in the parametrization enough such that good approximations are contained, but keep it as simple as possible so gradient descent converges. Combinations of linear and element-wise non-linear transformations have proven to be capable of representing a wide range of unknown target functions, when optimized using gradient descent. This is the motivation for using the above construction as a first approach. Further, more mathematically motivated network architectures will be presented in later sections.

4.6.2. Supervised Learning of Balanced Metrics

For a first study of the feasibility of predicting h depending on the value of ψ, the aim in this section is to learn the h matrices produced by Donaldson's algorithm. This kind of application

[6]The number of real parameters of a Hermitian $N \times N$ matrix is N^2.

is called supervised learning, referring to the fact that the network is trained to reproduce a known set of inputs ψ and outputs h produced by Donaldson's algorithm.

To produce the training data, Donaldson's algorithm was run for 20 equally spaced values in $0 < |\psi| \leq 10$, and in each case 4 random complex angles, as well as for $\psi = 0$. If only the h matrix of the last iteration is taken, one gets 81 examples to train with. This number can be increased by using the h matrices of the last few iterations, after the T-operator has already converged. Since the h matrices mostly vary in the zero-value entries between iterations (see section 3.3), no systematic difference would be expected. However, by computing the loss function over a subset of all target h matrices simultaneously[7], the two choices of training values may lead to different stochasticity in gradient descent.

Loss Between h Matrices

We want to train a model to reproduce some known set of h matrices over a range of values of ψ. A commonly used loss for this kind of situation is the mean-squared error

$$L_{\text{naive}}(h, h^{\text{target}}) = \frac{1}{N_k^2} \sum_{\alpha\bar{\beta}} \left| h_{\alpha\bar{\beta}} - h^{\text{target}}_{\alpha\bar{\beta}} \right|^2 . \tag{4.20}$$

The potential problem of this loss is that it restricts the value of the learned matrices h more than necessary. As was discussed in section 3.3, the final metric on X is invariant under re-scalings of h, $h \sim \lambda h$. Restricting the networks to reproduce the exact numerical value of h^{target} given by Donaldson's algorithm may therefore make it harder for the network to converge, as it additionally has to learn the arbitrary scaling. The following loss function takes the scaling ambiguity into account, letting the model reach minimal loss by attaining the value of any h matrix in the equivalence class:

$$L_{\text{inv}}(h, h^{\text{target}}) = \frac{1}{N_k^2} \sum_{\alpha\bar{\beta}} \left| \frac{h_{\alpha\bar{\beta}}}{\langle |h| \rangle} - \frac{h^{\text{target}}_{\alpha\bar{\beta}}}{\langle |h^{\text{target}}| \rangle} \right|^2 , \tag{4.21}$$

where

$$\langle |h| \rangle = \frac{1}{N_k^2} \sum_{\alpha\bar{\beta}} \left| h_{\alpha\bar{\beta}} \right| . \tag{4.22}$$

Numerical Results

The model hyperparameters with the largest number of possible values are the number of dense layers l_d and the number of initial powers n_p. To make the possible model space to be explored manageable, in this section we only consider models which take powers of ψ directly (no decomposition into angle and magnitude), trainable powers, and using the composition of

[7] In the machine learning literature this is called stochastic gradient descent with mini-batches.

h via real and imaginary parts. The choices left are then which of the above losses are used in training, whether the last 5 iterations (after convergence was reached) of Donaldson's algorithm are used for training or just the last, and the hyperparameters n_p and l_d. The results of a scan over both possible choices of training data, $n_p = 2, 5, 10, N_k$, and $l_d = 0, 1, 2$ can be found in Figure 4.6 and Figure 4.7 using the naive and the invariant loss for training, respectively. They show in both cases the invariant loss that is reached over the training data (with respect to only the last h matrix produced by Donaldson's algorithm in each case). The reason is that we are in the end not interested in the scaling of h, although using it for the comparison does potentially favor the models for which it was used as loss during training.

The figures indicate that adding more powers that are initially taken of ψ does not lead to better results, suggesting that this first step may even be superfluous. Making the network deeper, that is adding more layers and increasing l_d, does tend increase the ability to learn the balanced metrics.

It is always better to compare the performance of models using an independent loss over values not seen during training, rather than using the loss over the training data itself. The reason to compute the loss over unseen inputs, as was mentioned in section 4.5.2, is that the network may over-fit and perform well on data used in training, while generalizing poorly to other values. Taking a step back, we are ultimately not interested in numerically approximating the outputs of Donaldson's algorithm as closely as possible. Rather, the aim is to approximate the Calabi-Yau metrics. Figure 4.8 and Figure 4.9 show the σ-accuracies achieved by the trained models (trained with the naive and invariant loss, respectively) over a range of real values of ψ. This comparison addresses both of the above issues. The real values of ψ are not the ones seen during training, and the σ-measure favors neither of the losses used during training. Furthermore, by comparing larger values of ψ than used for training, one can compare how well the models extrapolate.

The comparison of the different architectures in Figure 4.8 and Figure 4.9 confirms that the invariant loss in general leads to better results. It also strengthens both of the previous observations for the network architectures, suggesting that more hidden dense layers and fewer initial powers lead to better results. Interestingly, while the comparison of the losses on the training data indicated that training with a single h matrix for each value of ψ leads to better results, the overall best σ-measures were obtained by training with the last 5 matrices.

Summary

The most important conclusion that can be drawn from the above results is that it is in principle possible to find moduli-dependent networks, and train them using gradient descent to approximate Calabi-Yau metrics. The focus in the following is to train models without previously computing Donaldson's algorithm. Since the optimal and balanced h matrices are relatively similar, as noted in section 4.5.2, insights into which models perform well to predict Donaldson's balanced metrics are likely to generalize. The above results suggest that deep networks of

multiple dense layers with few, or maybe no, initial powers are a good approach.

Even without moving on to minimizing the accuracy measures, one may want to use this approach of machine learning Donaldson's balanced metrics to extrapolate and interpolate between available data. The performance for interpolation within the range of ψ of the training data already looks promising. Looking at the results for the architectures that have achieved the best accuracies in Figure 4.9, however, it seems that they show no better extrapolation than maintaining the last value of h matrix at $\psi = 10$ for larger values $\psi > 10$. One may achieve better results by modifying the network architecture, possibly with theoretical insights into how the ψ dependence may look.

4.6.3. Training with η-Loss

Having established some preliminary results for moduli dependent networks approximating balanced metrics, the remainder of this section will present the final and most powerful application of machine learning. Instead of building on top of Donaldson's algorithm, the networks will be directly trained using the η-based loss defined in equation (4.9). The results will show that this approach can replace Donaldson's algorithm, as it produces better accuracies during similar amounts of time, simultaneously over a range of ψ values. Some technical details, specific to training ψ-dependent networks is discussed in section B.5 of the appendix.

Starting with a Pre-Trained Network

Before making the optimization scheme wholly independent of Donaldson's algorithm, we can take the network trained using data from Donaldson's algorithm, and continue training with respect to the η-loss of equation (4.9).

Figure 4.10 shows the σ-accuracy achieved by a network with $l_d = 2$ and $n_p = 2$, that was previously trained using the balanced metrics. This is the architecture that performed best in approximating the balanced metrics. One can see a visible improvement in the σ accuracies after training with the η-loss, compared to the previous accuracy when the network was only trained with data from Donaldson's algorithm. The figure shows the accuracy achieved by Donaldson's algorithm at $\psi = 0$ and $k = 12$ as a reference, which is notably the same order of magnitude as the accuracies achieved by the network here for a much smaller degree $k = 6$. Assuming the σ-measures are in fact a good measure for Ricci-flatness, this means that the machine learning approach can simultaneously predict very good approximations over a range of moduli space, in significantly less time than Donaldson's algorithm. The optimization with respect to the η-loss at $k = 6$ took only a time on the order of minutes, whereas Donaldson's algorithm at $k = 12$ takes a time on the order of days.

Before claiming that this machine learning approach can in fact produce better results than Donaldson's algorithm in smaller time, we have to confirm the initial training with balanced

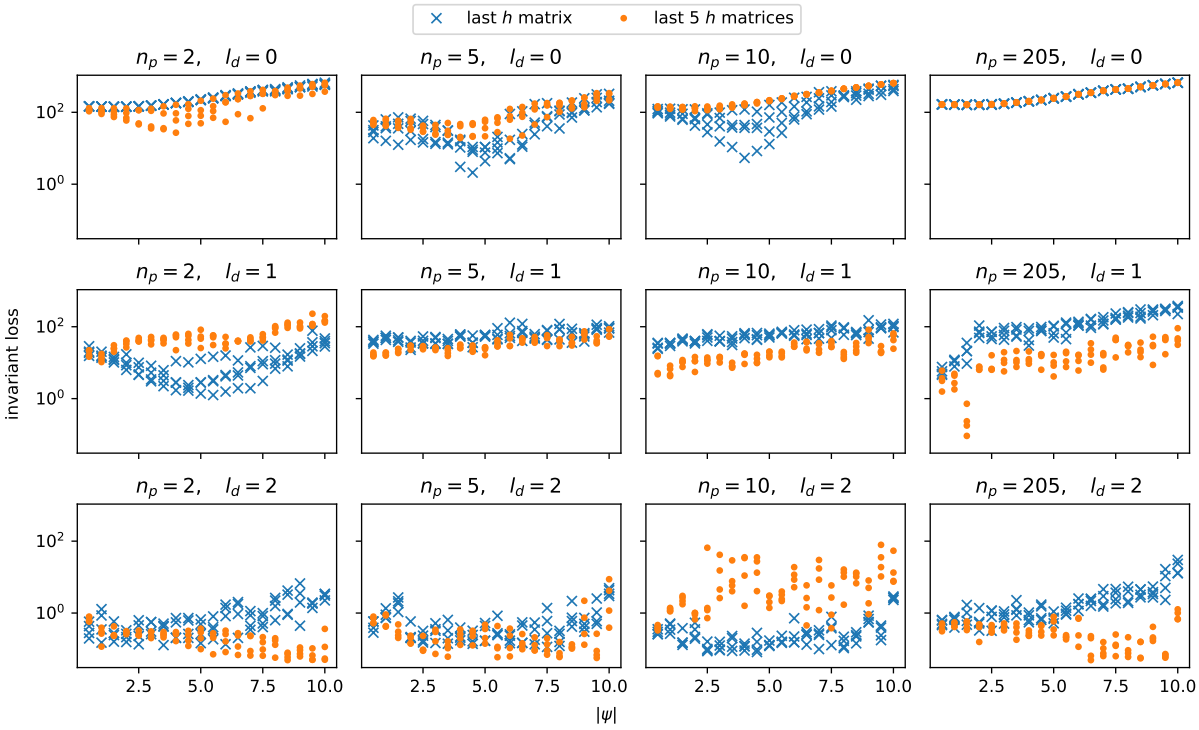

Figure 4.6.: Invariant loss of equation (4.21) computed over all last h-matrices produced by Donaldson's algorithm for different networks as specified in section 4.6.2, trained using the naive loss of equation (4.20). The loss shown is with respect to the same target values of h as were used for training, so one cannot see whether over-fitting has occurred. See Figure 4.8 for a validation loss.

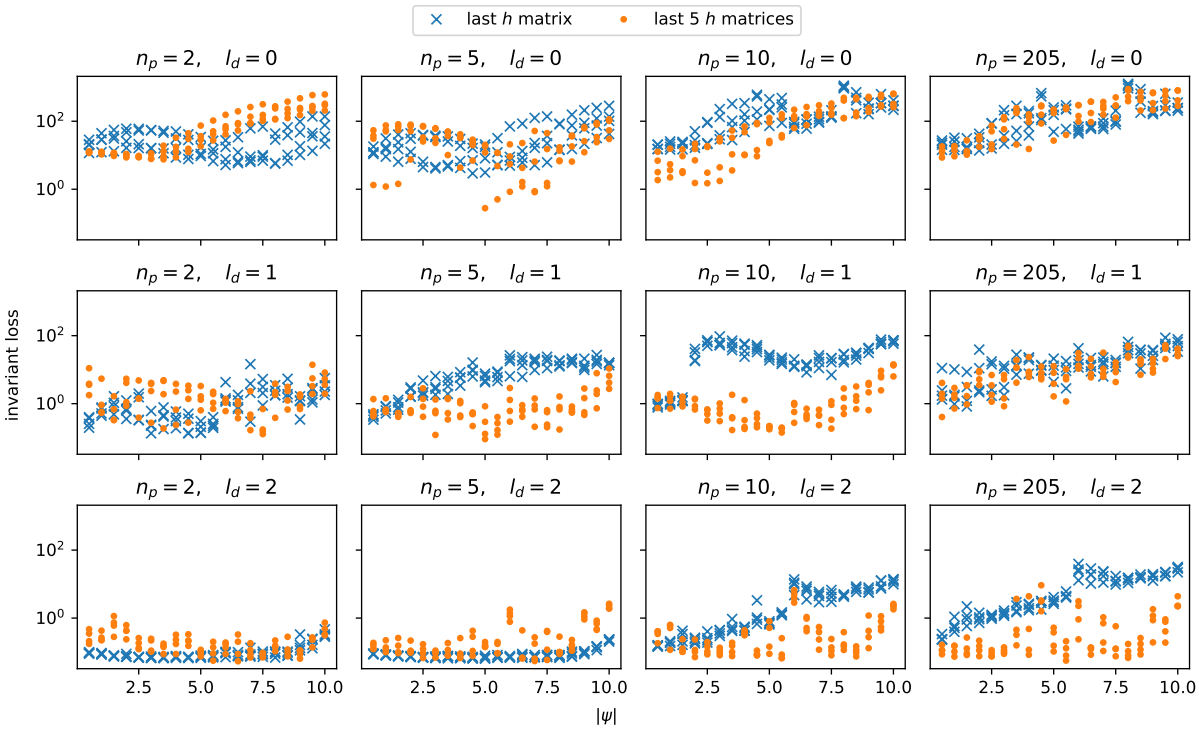

Figure 4.7.: Same as Figure 4.6, except the models were trained using the invariant loss of equation (4.21). For a validation loss, see Figure 4.9.

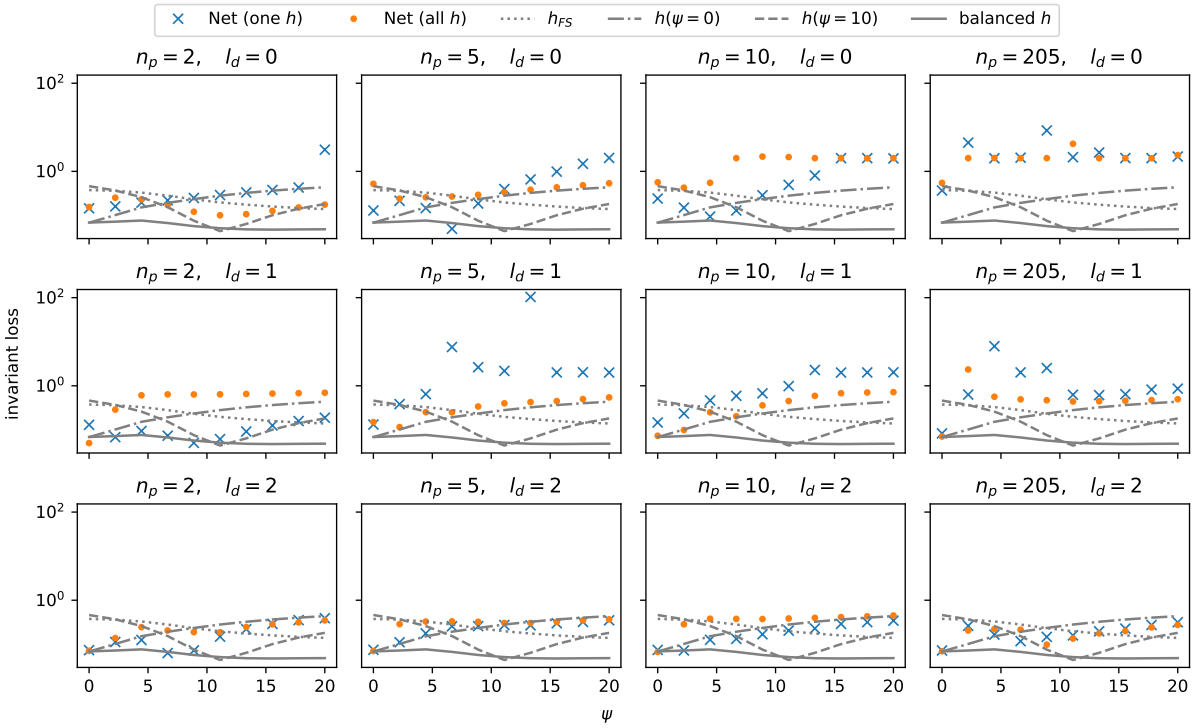

Figure 4.8.: σ-accuracy over a range of real values of ψ not seen during training for the same models as in Figure 4.6. The figure therefore shows a validation loss, and how well the models extrapolate. Additionally, the figure shows accuracies attained by using the h matrices of Donaldson's algorithm for $\psi = 0$ and $\psi = 10$ over the whole range, as well as for the respective values, and the Fubini-Study metric. This serves as a reference, to assess the quality of the model accuracies.

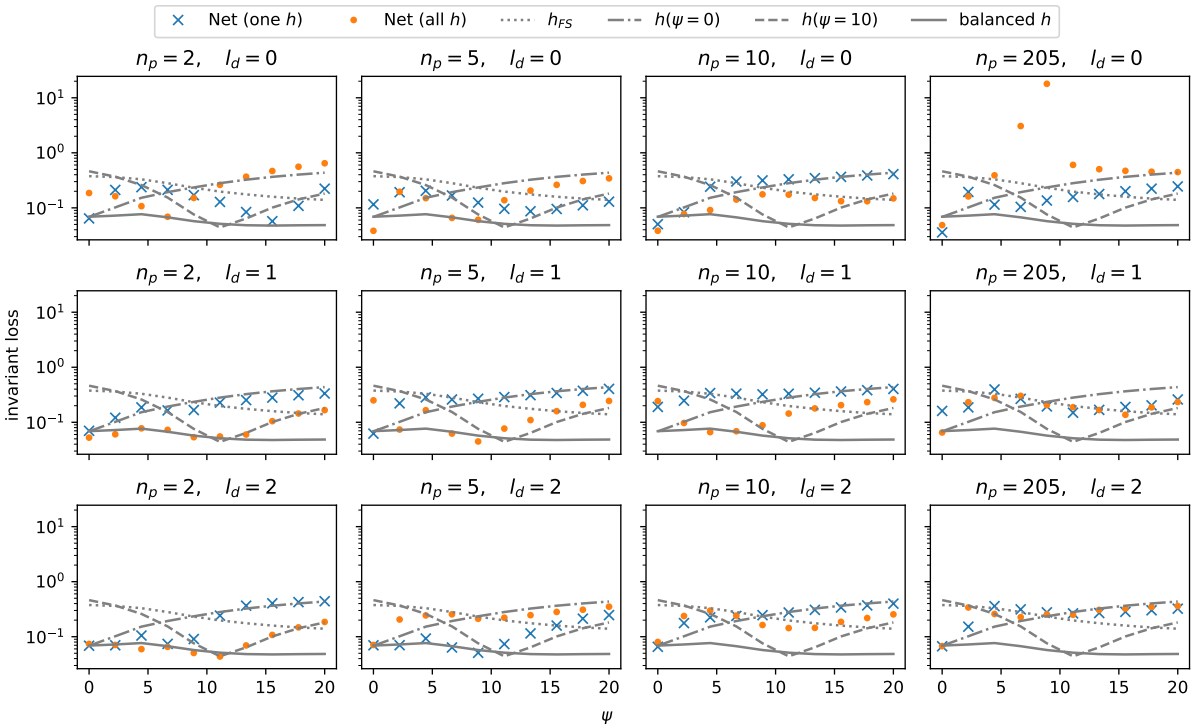

Figure 4.9.: Same as Figure 4.8, except the models were trained using the invariant loss. Close to ideal performance can be seen in the case $n_p = 2, l_d = 2$, trained using all last h matrices.

metrics is unnecessary. This would make it an independent approximation scheme. The next section will explore other variants of the dense layer architecture introduced above, and skip the initial training with data from Donaldson's algorithm.

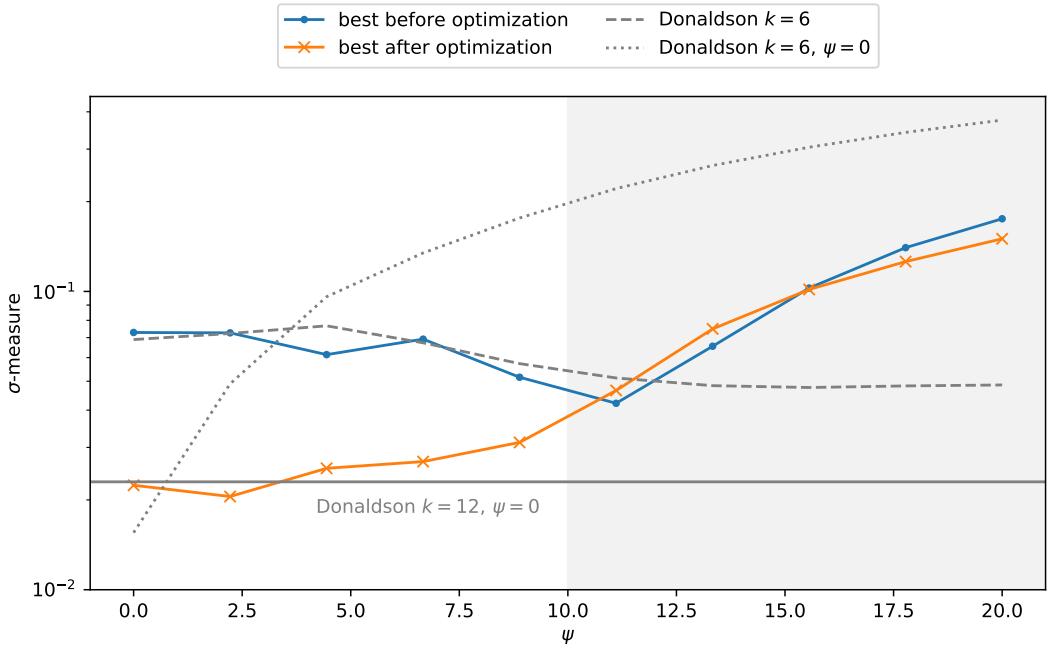

Figure 4.10.: σ-accuracies at $k = 6$ achieved by the dense network with $l_d = 2, n_p = 2$ before and after optimization using the η-loss. The network was initially trained using the last 5 iterations of Donaldson's algorithm. The shaded area indicates the range of $|\psi|$ that was not used during training, and thus shows the extrapolation behavior of the networks. For reference, the σ accuracy achieved by Donaldson's algorithm for each real value of ψ is shown, as well as the accuracy obtained by using h computed at $\psi = 0$ for all values of ψ. The horizontal line shows the accuracy achieved by Donaldson's algorithm at $k = 12$ and $\psi = 0$.

Starting With Untrained Networks

A scan over the various network architectures outlined in section 4.6.1 has been conducted, where the networks were directly trained using the η-loss of equation (4.9) and $|\psi| \leq 10$. A comprehensive overview of the results can be found in Figure C.6 in the appendix. Figure 4.11 shows the σ-accuracies of the best-performing networks, in comparison with the network of the previous section, which was first trained with the balanced metrics. The figure shows that it is not in principle necessary to pre-train the networks, as the networks directly trained with the loss which measures how close the approximation is to Ricci-flat produce equally good, and potentially better accuracies.

All networks were trained until gradient descent has converged (the loss no longer decreased over several iterations). The comprehensive scan over the variations of the networks shows that, as opposed to the supervised training with balanced metrics, increasing the number of layers did not improve the performance. This may be a problem that can be addressed by changing

the training scheme, for example by increasing the sample size in each batch or changing the initialization of the network[8]. There is no obvious choice for the best network architecture, with several combinations leading to similarly good results. Overall, the Cholesky decomposition seems like a better choice than decomposition into real and imaginary parts.

It is notable that one network using a separation into the absolute value and angle of ψ, without a following application of powers seems to show good extrapolation for values $|\psi| > 10$ outside the training range. None of the networks reach an accuracy quite as good as the ones obtained for the optimization procedure of section 4.5 at a fixed value of ψ, which is to be expected since the fact that the network is moduli dependent introduces additional complexity (and depending on the network a constraint on possible outputs). However, this does indicate that it may be possible to improve the convergence of the networks further in the future.

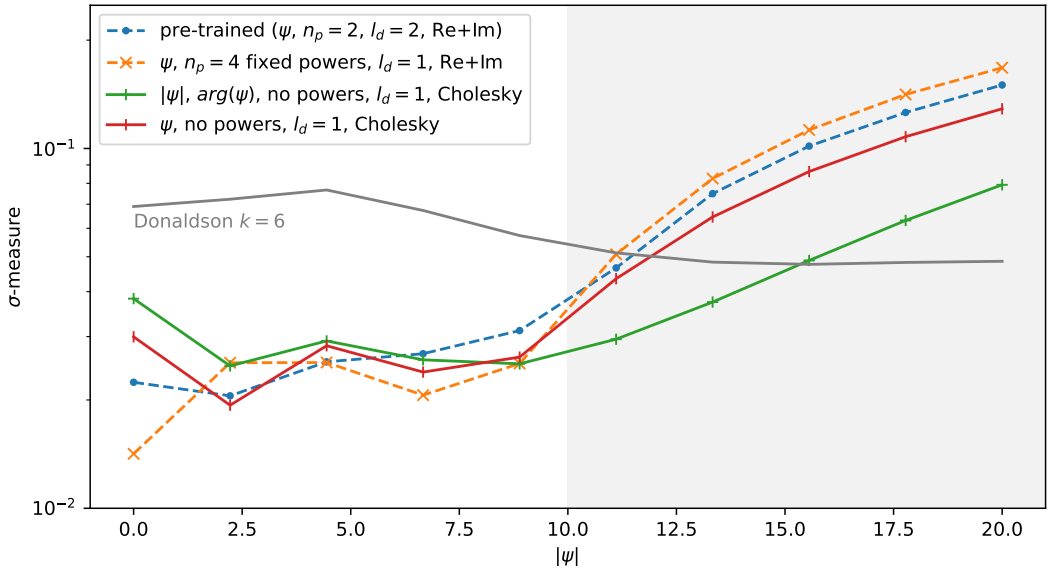

Figure 4.11.: σ-accuracies at $k = 6$ achieved by the best network architecture of the kind introduced in section 4.6.1 after optimization using the η-loss for $|\psi| \leq 10$. In the case where the powers are fixed and not parameters changed during gradient descent, they were chosen to be $1/2, 1, 2$, and 3. The accuracies shown are the mean over four equally spaced complex angles at each absolute value of ψ. For reference, the σ accuracy achieved by Donaldson's algorithm for each real value of ψ is shown, as well as the results for the network that was previously trained using balanced metrics.

4.6.4. Other Network Architectures

So far, only a single basic kind of network (although multiple variants of it) has been explored to produce h matrices depending on the parameter ψ. While the design using a stack of dense layers has proven to be versatile, it has the disadvantage that one cannot immediately extract an analytical expression that represents the network. To obtain an expression that can be worked

[8]It may also be possible to tune the learning rate, however a range of learning rates was already tested in producing the present results.

with by hand (instead of the numerical representation given by the network), one would have to fit the outputs of the network using appropriately chosen functions.

In the following sections, two more approaches to the network architecture are presented, which can be directly translated into an analytical expression. This is motivated by an intuition that algebraic expression may exist which give good approximations to the moduli dependence. If this is true and the right parametrization is found, it may lead to a significantly better extrapolation than was found for the networks using dense layers. The idea of the first kind of network, referred to as "power networks" in the following, is to combine iterations of summation and raising to a power. Using two layers, the network can for example learn to compute an expression like $\sqrt{1+x^2}$. In the first layer, this network would compute the powers 0 and 2 of the input variable x and superpose them to $1+x^2$. The next layer takes again a power, which may be $1/2$, leading to the final expression. It is here crucial that the input variables are complex, which means that any real power is defined, including for negative values. The powers have to be real because gradient descent requires continuous parameters. Depending on the number of layers of this kind of network, arbitrarily complex nestings of sums and powers can be represented.

Another approach to constructing networks that more closely resemble analytical expressions is using the architecture introduced in [29] as NALU, which stands for "neural arithmetic logic unit", and is roughly a trainable superposition of adding and multiplying inputs. The aim of this architecture is to make it easier for a network to learn the basic algebraic operations of addition, subtraction, multiplication, and division. This is achieved by biasing the powers and summation coefficients to $-1, 0, 1$. By chaining multiple layers of NALU, once again the network can learn to represent expressions such as $(x+xy)^2$. The maximal power is now dependent on the number of layers used, where this expression requires at least three layers: one to produce x and xy from the inputs x and y, one to combine it into $x+xy$, and a final one multiplying this with itself. For the last step the second layer has to produce $x+xy$ twice, which shows that the final power is not solely determined by the number of layers but also by the intermediate dimension of outputs.

Below, the mathematical definition of power networks is outlined, followed by a summary of the NALU layer and the architecture of a network based thereon. Finally, the accuracy achieved by variations of both architecture when trained using the η-based loss is shown. Note that it has proved advantages to carefully choose the initialization of both networks, as explained in appendix B.5.

Power Networks

The following outline of the architecture of power networks, based on iterations of dense linear combinations and raising to a power, is schematically summarized in Figure 4.12. As the first

step, once again powers of the variable ψ are taken,

$$P_i^0 = \psi^{p_i^0} , \tag{4.23}$$

where the number of powers is denoted as n_p, and each is a real, trainable parameter.

Afterwards, the following two steps are repeated l times. First, take a linear superposition of the previous outputs (the initial one being P_i^0),

$$D_i^k = \sigma\left(\sum_j M_{ij}^k P_j^{k-1} + B_i^k\right) , \tag{4.24}$$

which is now a dense layer with complex parameters $M_{ij}^k, B_i^k \in \mathbb{C}$. The number of values in the vector P_i^k is chosen to be $2N_k$, as in section 4.6.1. Next, each output value is raised to a real power p_i^k,

$$P_i^k = \left(D_i^k\right)^{p_i^k} . \tag{4.25}$$

The real values of the diagonal are computed using

$$h_i^{\text{diag}} = \sum_j \left| M_{ij}^{\text{diag}} P_j^l + B_i^{\text{diag}} \right|^2 . \tag{4.26}$$

The complex values for the upper triangle are computed using another dense layer

$$h_i^{\text{upper}} = \sum_j M_{ij}^{\text{upper}} P_j^l + B_i^{\text{upper}} . \tag{4.27}$$

Finally, the Hermitian matrix is reconstructed using the Cholesky decomposition introduced in section 4.5.1.

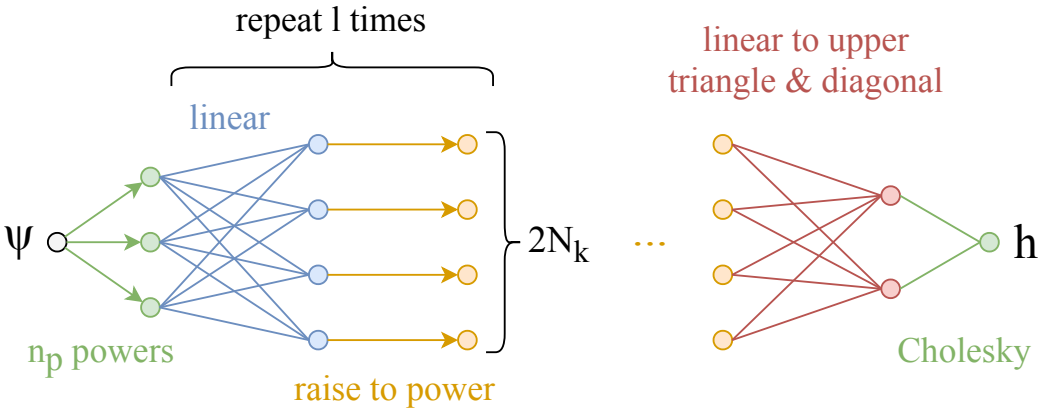

Figure 4.12.: Schematic overview of the power network architecture, based on iterations of linear combination and rasing to a power.

The NALU Layer

The idea behind NALU is that traditional network architecture, such as the dense layers with sigmoid activation, do not generalize well for functions that have an unknown but simple analytic expression. A slightly modified version of the NALU layer will now be described.

The NALU layer is composed of two parts, one for addition/subtraction and one for multiplication/division. The addition and subtraction part would ideally amount to a binary choice of exclusion and inclusion in a sum (and the respective sign). However, a binary choice between including or excluding an input value cannot be learned using gradient descent which requires continuous parameters. Instead, the network will be formulated in a way that biases the factors to values close to $1, -1$, or 0. Given the input x_j, the first part of NALU is computed as

$$A_i = \sum_j W_{ij} x_j \,, \tag{4.28}$$

where W_{ij} is a real value parametrized as

$$W_{ij} = \sigma(\tilde{W}_{ij}) \, \tanh\Big(\hat{W}_{ij}\Big) \,. \tag{4.29}$$

This parametrization achieves the bias in favor of values $1, -1$, and 0. In the two limits of real space ($\pm\infty$), the sigmoid function assumes values 0 and 1 respectively, while tanh becomes -1 and $+1$.

The second part of the layer is multiplication. Again, gradient descent requires the parametrization to be continuous. We will therefore need powers whose values are biased to $1, -1$, and 0. This can be done using the same parameters W_{ij} as

$$M_i = \exp\left(\sum_j W_{ij} \log\big(x_j\big)\right) . \tag{4.30}$$

In the original paper, the input variable is real which means the logarithm has to be restricted to the absolute value of the inputs. For the application here, the inputs will be complex, which means no such restrictions needs to be made[9].

The two parts are finally combined to a single output by introducing a parameter C_i that "chooses" between the additive and multiplicative operations:

$$N_i = A_i \, \sigma(C_i) + M_i \left(1 - \sigma(C_i)\right) . \tag{4.31}$$

In summary, the NALU layer chooses between a learned expression of addition/subtraction or multiplication/division of the input variables for each output dimension independently. Because this choice again has to be parametrized by a continuous parameter, the output is actually a

[9] Infinite values can be represented as flaot values and reproduce the desired behavior, so that in fact numerically $\exp(\log 0) = \exp(-\infty) = 0$.

superposition of the additive and multiplicative operation, although the sigmoid activation function is chosen to bias towards a binary choice.

NALU Networks

If the input dimension to the NALU layer is one, the outputs correspond to superpositions of ψ and powers of it. This means NALU can be taken as the first layer, and it is not necessary to manually construct a multi-variate input. The NALU networks analyzed in the following are l layers of repeated applications of the NALU layer, each with a dimension of $2N_k$.

The construction of the Hermitian matrix h from the complex outputs of the last layer is the same as was introduced for the power networks above.

Numerical Results

Figure 4.13 shows an overview of σ-accuracies achieved after reaching convergence in gradient descent with the η-loss of equation (4.9). As for the previous analysis, the degree is fixed to $k = 6$. It shows that both architecture seem to be feasible, in principle, reaching similar accuracies as Donaldson's algorithm at the same degree within the training range of ψ. The results are, however, not quite as good as the ones achieved using the dense layer architecture. Overall, the results seem to suggest that shallower networks (fewer number of layers l) do better. This may be due to a general difficulty in gradient descent that is present for networks that include taking powers[10].

In neither case do the networks display a good extrapolation behavior for values of $|\psi| > 10$. This means that while the two alternative approaches introduced here seem like possible directions of further exploration, they do not presently capture the parameter dependence of the h matrices better than the dense layers.

[10]The difficulty can be illustrated by considering a loss $|x^p - x^4|^2$. In the parameter p, x^p traces a spiral in the complex plain around the origin, which means that starting from $p = 1$ the required continuous transformation of the parameter p includes sections in which the network value x^p moves farther away from the target. Depending on the loss function and its stochasticity, gradient descent may get stuck at local minima at even values of p.

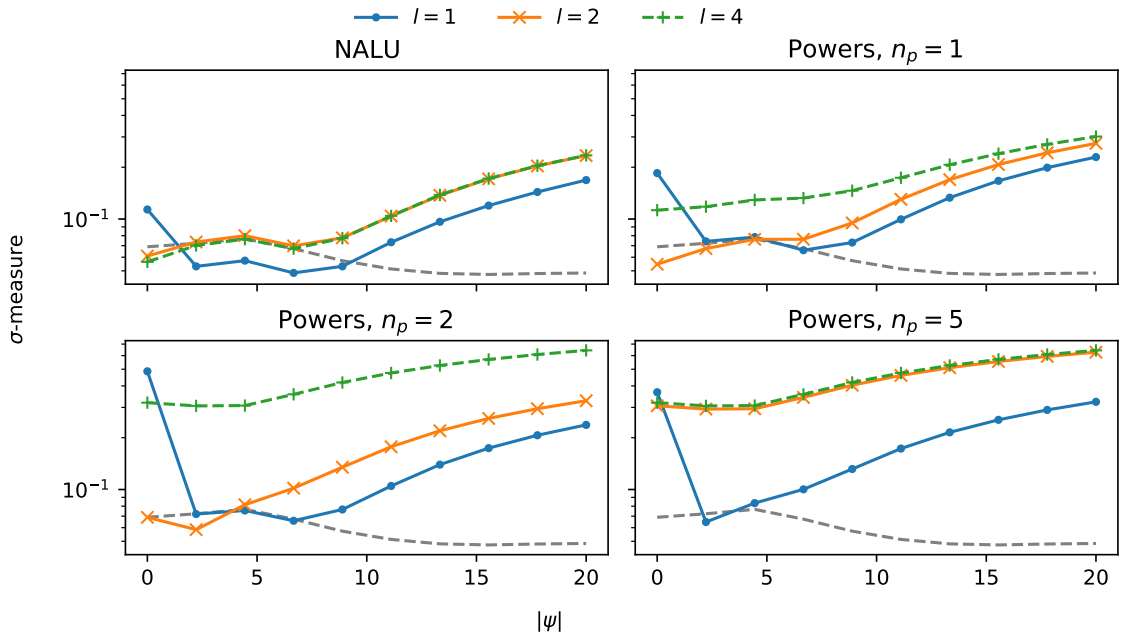

Figure 4.13.: Comparison of σ-accuracies reached by models at $k = 6$ discussed in sections 4.6.4 and 4.6.4, computed over 4 angles for each value of $|\psi|$. The networks were trained using the η-loss for $|\psi| < 10$. The gray dashed line are the σ-accuracies achieved by Donaldson's algorithm at $k = 6$.

5. Conclusion

In the above analysis we have seen that deep learning constitutes a viable approach that can replace Donaldson's algorithm, as measured by the σ-accuracies. This can be seen in Figure 5.1, which gives an overview of the best accuracies achieved by models of the different kinds.

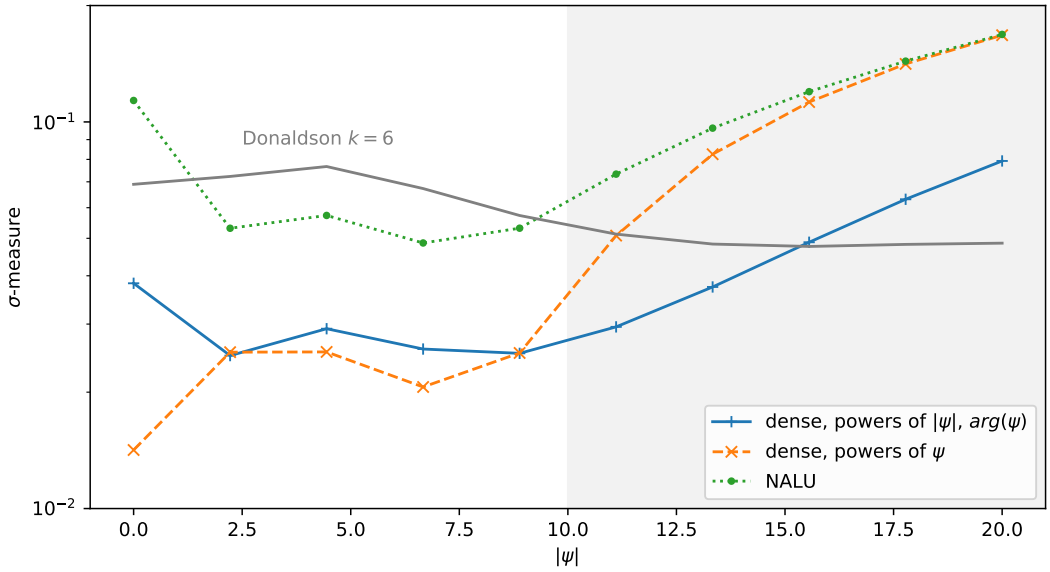

Figure 5.1.: Overview of the σ-accuracies achieved by the best network architectures devised here. The measures are the average over four angles for each value of $|\psi|$, and the gray shaded area indicates where the extrapolation of the networks can be seen. Both dense layer architectures use $l_d = 1$ layers. For the dense architecture using powers of ψ directly, the powers were fixed to $1/2, 1, 2, 3$, and the real and imaginary decomposition was used to construct h. The other dense architecture using angle and norm of ψ instead made use of the Cholesky decomposition, and the initial taking of powers is skipped. The last architecture is composed of a single NALU layer. In gray, the accuracy achieved by Donaldson's algorithm for each real value of ψ is shown. Both the models and Donaldson's algorithm are computed at a degree $k = 6$.

After an initial study of Donaldson's algorithm whose accuracies provide a benchmark for the deep learning approach, several deep learning models mapping from moduli space (here limited to the single parameter ψ) to the Hermitian matrix h, which defines an algebraic metric, have been explored. The dense layer architecture studied in section 4.6.1 has been seen to be able to simultaneously approximate the Calabi-Yau metric on a range of moduli space to a significantly better accuracy than Donaldson's algorithm. This can be achieved in a similar amount of time as

it takes to compute Donaldson's algorithm for a single point in moduli space at the same degree k. It thus seems like no overstatement to say that the machine learning approach developed here presents a full replacement of Donaldson's algorithm. In addition to that, the approximations are present in functional form, which in future studies would allow one to explore the moduli space of Calabi-Yau manifolds without having to run Donaldson's algorithm multiple times.

The present investigation of the deep learning approach was specialized to $k = 6$ and a subset of the quintics, defined as varieties parametrized by a single complex number ψ. A future study may build on these results to extend to more than one complex structure parameter. Large parts of the implementation may be reused for this, with only some small adjustments having to be made to allow multiple parameters for the variety. The principle components that define the optimization scheme are the network architecture and the specific gradient descent optimization algorithm. Both, as is often the case in machine learning, have a large range of possible solutions, only a small number of which were explored in the present analysis. In summary, the results shown here should thus be seen as laying the groundwork for many possible improved algorithms that can approximate the Calabi-Yau metrics.

The cost of producing samples on the manifold was not found to be the computational bottleneck in the deep learning approach. It may still be advantageous to investigate more advanced methods of sampling points, because they change the loss function and therefore the convergence behavior of gradient descent. For the present analysis, the line sampling algorithm introduced in section 2.3 was used, including a weighing of the loss using the Monte Carlo weights as shown in section 4.4. It may lead to better convergence of the networks if relatively more samples are generated in regions of large curvature. This could be achieved by temporarily fixing an approximation to the true Ricci-flat metric, and using Markov Chain methods to produce the desired samples. Gradient descent neither requires uncorrelated samples nor the exact probability distribution to be known, which means a large range of algorithms could be tested.

There are several steps that may be taken to improve and extend the initial machine learning approached outlined and studied here. The preceding investigation was limited to the degree $k = 6$ of the algebraic networks. This was sufficient to show that one can find moduli-dependent approximations of better accuracy than Donaldson's algorithm, however one can likely achieve even better accuracies by going to larger values of k. The following are observations that may be taken account for a future study.

- Preliminary experiments have shown that if the learning rate is chosen too small relative to the batch size, gradient descent fails to minimize the parameters of the network.

- This must be addressed not only by carefully choosing a batch size, but also how samples are chosen on which the loss is computed, including the random moduli parameters. Convergence seems more stable if multiple moduli parameters are considered in each step of gradient descent, although enough points on the manifold for each are required so the η-variance is well defined. This is discussed for the present application in appendix B.5.

If the number of complex structure parameters is increased beyond the single number ψ, it is likely that more entries of the h matrix become relevant, and in general the choice of network may become more critical.

- In the present case, the networks could be made more stable by including parameters that suppress entries independent of the inputs (see section B.5 of the appendix). Once several moduli are included in the range over which the network is supposed to predict, it may become increasingly advantageous to both further study how the subset of significant entries depends on them, and to make the suppression input-dependent.

- Another approach which could, unfortunately, not be further investigated here, is to use high-dimensional splines (more specifically a network architecture based upon them) to map from the set of moduli parameters to the Hermitian matrix h. The reason that this may be a viable approach is that the space the matrices lie in is in general significantly larger than the number of moduli parameters. We can thus interpret the networks as parametrizing a hyper-surface in terms of the moduli in the space of matrices.

The mathematical structure of algebraic metrics has proven to be a useful basis for the design of models, both as it solves the problem of overlap conditions, and as one knows of possible approximations which converge for large degrees k. As noted previously, the approach of using algebraic metrics may, however, not be the most advantageous one depending on what points of moduli space one is interested in (specifically, it may not perform well for geometrically inhomogeneous manifolds). Future studies may attempt to find other parametrizations of the Kähler potentials, which may include giving up the automatic compatibility of the metric between the patches of projective space. In that case, one has to introduce overlap conditions to the loss functions that assert they differ by a Kähler transformation. Because of the versatility of automatic differentiation, these potentials (which would be be defined in affine coordinates on each patch) can have an arbitrary differentiable form. Although encoding the potential has the advantage that Kählerity is guaranteed, and potentially fewer parameters are needed, one may also attempt to find networks that parametrize the metric in each patch directly.

A. JAX as Computational Framework

The following is a short overview of the main features of JAX [24] pertinent to the implementations used in the analysis above. The aim is not to give a thorough introduction to JAX, but rather to highlight in more detail why it is a good fit for this application, and to mention aspects in which its use is slightly different in the present geometric context than in a typical machine learning setting. It may also be helpful for understanding the implementation details laid out in the next section.

A.1. Complex Differentiation with JAX

Because of the rise in popularity of machine learning with gradient descent, many frameworks have been developed to automatically differentiate algebraically defined functions. In almost all cases, these frameworks primarily focus on differentiable real functions (at most containing internal complex operations, such as Fourier transformations). It is clear that a general complex function ($\mathbb{C} \to \mathbb{C}$) does not have a well-defined derivative in the usual sense (it would in general require a real 4×4 matrix). However, for the special class of holomorphic functions the derivative can be defined as a single complex function. Any complex function $f : \mathbb{C} \to \mathbb{C}$ can be written as

$$f(z) = f(x + iy) = u(x, y) + v(x, y)\,. \tag{A.1}$$

Holomorphic functions are complex functions that satisfy the Cauchy-Riemann equations

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y} \quad \text{and} \quad \frac{\partial u}{\partial y} = -\frac{\partial v}{\partial x}\,. \tag{A.2}$$

Its complex derivative can thus be computed by just looking at its real part

$$2\frac{\partial f}{\partial z} = \frac{\partial f}{\partial x} - i\frac{\partial f}{\partial y} = \frac{\partial u}{\partial x} + i\frac{\partial v}{\partial x} - i\frac{\partial u}{\partial y} + \frac{\partial v}{\partial y} \overset{\text{C.R.}}{=} 2\left(\frac{\partial u}{\partial x} - i\frac{\partial u}{\partial y}\right). \tag{A.3}$$

This is exactly the derivative that JAX provides:

$$\nabla_{\text{hol}}\, f(z) = \nabla \operatorname{Re}\big\{f(z)\big\}\,. \tag{A.4}$$

It cannot detect whether or not a given function is in fact holomorphic, which means one has to assure this in the implementation. The following is an example of how the holomorphic

derivative is computed.

```
>>> def f(z):
>>>   return z ** 2

>>> df = grad(f, holomorphic=True)
>>> z = 4.0 + 2.0j
>>> df(z)
8.0 + 4.0j

>>> ddf = grad(df, holomorphic=True)
>>> ddf(z)
2.0
```

Implementation A.1: An example of computing holomorphic derivatives with JAX.

The holomorphic and anti-holomorphic derivatives if defined as

$$\frac{\partial}{\partial z} = \frac{\partial}{\partial x} - i\frac{\partial}{\partial y}\,, \quad \frac{\partial}{\partial \bar{z}} = \frac{\partial}{\partial x} + i\frac{\partial}{\partial y} \tag{A.5}$$

can be applied to any complex function. The condition for holomorphic functions can then be rewritten as $\partial_{\bar{z}} f = 0$. The way to make holomorphic and anti-holomorphic derivatives computable using automatic differentiation is to formally expand the input space to two independent complex variables, z and $\bar{z}$. As long as the the implementation is holomorphic in each variable, the derivative generated by JAX when differentiated with respect to z and $\bar{z}$ are respectively the holomorphic and anti-holomorphic derivatives. These derivatives are exactly the ones required for the geometric constructions introduced in section 2. For an example of how this can be used to compute the metric from the Kähler potential, see section A.3 below. Note that in the above, f and z scalar functions to simplify the notation, but just as for real derivatives no such restriction exists for automatic differentiation with JAX.

Gradient descent is used to minimize a real loss function. In its most basic form this is done by iteratively moving the parameters in the direction of steepest descent given by the gradient. For a parameter θ and a loss function $l(\theta)$, a gradient descent step has the form

$$\theta_{i+1} = \theta_i - \alpha \nabla_\theta l(\theta_i)\,, \tag{A.6}$$

with some learning rate $0 < \alpha < 1$. Assume now that $\theta = \theta_r + i\theta_i$ is a complex parameter. Writing out the minimization step for the real parameters one naturally gets

$$\begin{pmatrix} \theta_r \\ \theta_i \end{pmatrix} - \alpha \begin{pmatrix} \partial_{\theta_r} l \\ \partial_{\theta_i} l \end{pmatrix}. \tag{A.7}$$

As a complex expression this is

$$(\theta_r + i\theta_i) - \alpha(\partial_{\theta_r} l + i\partial_{\theta_i} l) = \theta - \alpha\overline{\nabla_\theta l}\,. \tag{A.8}$$

If the parameter is complex, gradient descent is thus in the direction given by the conjugate of the gradient. Practically, in order avoid mistakes and to reuse algorithms written without complex parameters in mind, it is useful to rewrite the complex parameters in terms of their real components, and minimize with respect to them.

A.2. Just-In-Time Compilation

Just-in-time compilation compiles a Python function at the time it is first called. Afterwards, if the function is called again, this faster compiled version is run. This has the usual benefits of compilation. For example, any objects that do not explicitly depend on the function input are statically computed at compilation time, speeding up the run time. While this simplifies the implementation in many occasions, the most important benefit is is that it removes the drawback of an interpreted language like Python by removing the large computational overhead compared to compiled languages like C++. There are a few things one has to be careful about for JIT to work, notably all arrays are immutable (the general programming style of JAX is functional), and indices into an array cannot depend on the input values of the function (see the documentation of JAX for more detail). The following is an example showing how the monomials are calculated given a power matrix (more on this below), exemplifying how functions are JIT compiled. Note that JIT compilation can also be arbitrarily composed with computing the gradients.

```
def compute_monomials(z, patch, degree):
  """Compute monomials s^α for the affine coordinates z in the given patch."""
  # Produce the power matrix defining the monomial basis.
  # In general this would also depend on the defining equation.
  # The point of including it here is to highlight that for each value
  # of the degree, this matrix is only computed once at compile time.
  # This can simplify the implementation, since one does not have to manually
  # manage frequently used objects and pass them to each function that uses them.
  pows = monomial_basis(degree)  # indices: (α, affine coordinate index i)

  # For a compiled function, indices to arrays cannot depend on input values.
  # A workaround used here is to cyclically shift the arrays.
  # The following is necessary because z are affine coordinates, while the
  # power matrix is given with respect to holomorphic coordinates.
  # This means that all powers referring to the coordinate z_patch = 1 have
  # to be removed.
  pows = roll(pows, -patch, axis=1)[:, 1:]
  z = roll(z, -patch, axis=0)

  return exp(pows @ log(z))
```

```
21
22 # Now apply JIT. Note that the actual value and input shape dependent compilation
23 # only occurs once the function is called. The function here statically depends
24 # on the degree, which means for each value a new compiled version is produced.
25 compute_monomials_jit = jit(compute_monomials, static_argnums=(2,))
```

Implementation A.2: An example of applying JIT compilation to a Python function. This is close to the code used to compute the monomial basis for the algebraic metrics.

A.3. Computing the Metric from a Kähler Potential

In order to see how JAX is used specifically for the geometric context here, consider now the Fubini-Study metric on projective space. Both the Kähler potential and the metric are known in analytic form. This means we can compare the metric that is derived via JAX's automatic differentiation, both in time and in numerical accuracy, to a direct implementation of the analytic expression. The following code outlines how this can be implemented in practice (here and in the following the code is not literal python code; where appropriate simpler and more obvious mathematical expressions replace the actual python functions).

```
1 @jit
2 def fs_potential(z, z̄):
3   # z and z̄ are affine coordinates
4   return log(1 + ∑ z^i z̄^ī)
5
6 @jit
7 def fs_metric(z, z_c):
8   norm^2 = 1 + ∑ z z̄
9   g_iȷ̄ = (δ^iȷ̄ norm^2 − z̄^ī z^j) / (norm^2)^2
10  return g
11
12 # The Kähler metric can be computed by applying the Jacobian twice
13 fs_metric_autodiff = jacfwd(jacrev(fs_potential, 0, holomorphic=True), 1,
       holomorphic=True)
14 fs_metric_autodiff = jit(fs_metric_autodiff)
```

Implementation A.3: Analytic and generated functions for the Fubini-Study metric on projective space.

Figure Figure A.1 shows the run time of computing the metric using the explicit expression and using automatic differentiation, as well as their numerical difference. The values indicate that

the metric using automatic differentiation is exact up to a small numerical error. Furthermore, and this is a great advantage of JAX and its JIT feature, computing the gradient using automatic differentiation takes insubstantially longer than using the analytic expression.

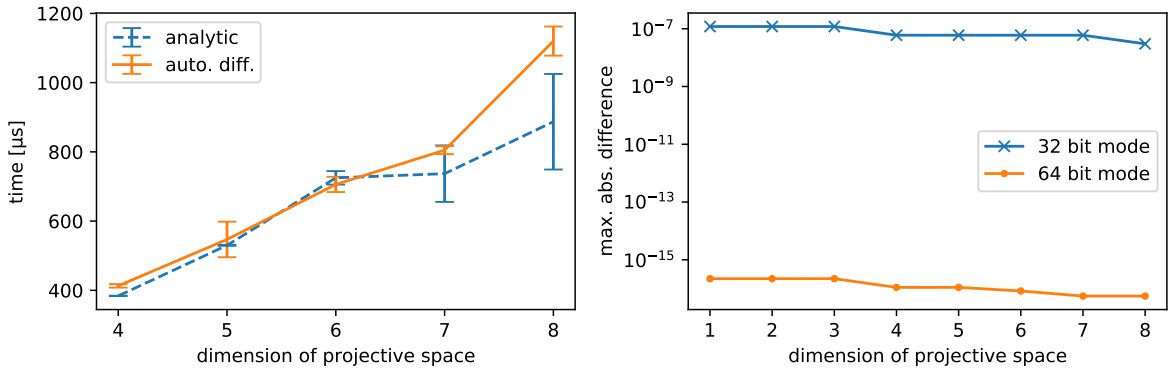

Figure A.1.: Left: Run times of computing the Fubini-Study metric once with the analytic expression and once using automatic differentiation of the potential, for 10 000 points each time. The mean value and standard deviations shown are computed over 7 runs. Right: Maximal numerical difference between the metrics computed with the explicit expression and automatic differentiation over 10 000 points. The difference is shown once using 32- and once using 64-bit mode. All results presented elsewhere are obtained in 64-bit mode.

B. Implementation Details

B.1. Constructing the Monomial Basis

When the basis of $\mathcal{O}_{\mathbb{CP}^{n+1}}(k)$, given by all homogeneous monomials defined in the homogeneous projective coordinates, is restricted to X, the basis has to be reduced for $k \geq n+2$. The reason is that on X the defining polynomial vanishes, $Q|_X = 0$, which means that all polynomials containing Q (a degree $n+2$ polynomial) must be removed to obtain a basis. Formally, the basis is defined, as in section 2.1.1, as

$$\mathbb{C}[z^0, \dots z^{n+1}]_k \,/\, \big\langle Q(z) \big\rangle \,, \tag{B.1}$$

where

$$\big\langle Q(z) \big\rangle = Q\, \mathbb{C}[z^0, \dots z^{n+1}]_{k-(n+2)} \,. \tag{B.2}$$

Another perspective on this is that each linearly independent polynomial in $\big\langle Q(z) \big\rangle$ can be rewritten to express one of the constituent monomials in terms of the remaining monomials. This leads to the number of basis elements on $\mathcal{O}_X(k)$ given in equation (2.11): from $N_k(\mathbb{CP}^{n+1})$ the number of basis elements of $\mathbb{C}[z^0, \dots z^{n+1}]_k$, which is the number of basis elements in $\langle Q \rangle$.

To make this clearer, consider $k = 6$, $n = 3$, and $Q_\psi(z) = \sum_i \left(z^i\right)^5 + \psi \prod_i z^i$. A basis of $\langle Q(z) \rangle$ is then given by multiplying Q with the basis z^0, z^2, z^2, z^3, z^4 of $\mathbb{C}[z^0, \dots, z^4]_1$. Since Q vanishes, the following relations are generated

$$z^j \left(\sum_i \left(z^i\right)^5 + \psi \prod_i z^i \right) = 0 \quad \forall j \,. \tag{B.3}$$

Each of these 5 equations can be used to eliminate one monomial. One choice is to remove all monomials $z^j \big(z^0\big)^5$, $j = 0, \dots, 4$.

So far, the discussion of how the reduced monomial basis is obtained was on a mathematical level. In practice, as was seen in the implementation example A.2 for evaluating the monomials, the sections can be represented using a matrix of integers. For example, the monomial

$$s(z) = z^0 \left(z^2\right)^3 z^3 \tag{B.4}$$

of $\mathcal{O}_{\mathbb{CP}^3}(4)$ corresponds to the row vector

$$[1, 0, 3, 1]\,. \tag{B.5}$$

Going to patch U_2 corresponds to setting $z^2 = 1$, which means the vector reduces to

$$[1, 0, 1]\,, \tag{B.6}$$

relative to the affine coordinates. The full basis of monomials of, for example, $\mathcal{O}_{\mathbb{CP}^1}(2)$ can be written as a matrix with each row representing a monomial:

$$\begin{bmatrix} 2 & 0 \\ 1 & 1 \\ 0 & 2 \end{bmatrix}. \tag{B.7}$$

Given a defining equation such as $Q(z) = \left(z^0\right)^2 + \left(z^1\right)^2$, each summand corresponds to a row in the matrix. Solving for either and removing it from the basis thus corresponds to deleting a row in the matrix. For the current example, either of $[0, 2]$ and $[2, 0]$ could be removed to obtain a basis on X. Both the generation of the monomial basis on projective space, and the reduction given a defining polynomial can be done algorithmically. This allows the defining equation to be replaced without adding significant implementation work.

B.2. Computing Geometric Objects from the Algebraic Potential

As mentioned in section 4.3, all holomorphic and anti-holomorphic derivatives can be done using automatic differentiation. This means one could technically describe the Kähler potential using any function and train it for Ricci-flatness using the schemes described above. For the specific case of models based on the algeraic metric, a significant speed up can be achieved by manually writing out the derivatives. The reason for this is that many derivatives occur multiple times which is not detected by JAX. For example, the holomorphic derivative of $\bar{S}^{\bar{\alpha}}(\bar{z})$ is the same as the complex conjugate of the anti-holomorphic derivative of $S^{\alpha}(z)$ and doesn't have to be calculated again. Since z and $\bar{z}$ have to practically be independent variables for the holomorphic derivatives to be computable (see section A.1), automatic differentiation would compute all derivatives twice. Using autodiff this way is not categorically unfeasible, however using the analytic expressions for the derivatives outlined below leads to faster code and significantly faster just-in-time compilation. In fact, the implementation of the equations below was tested by comparing it with the derivatives computed using automatic differentiation. The geometric objects of interest are the metric, its determinant, the Ricci-curvature, and the Ricci-scalar.

The Kähler potential is defined in terms of homogeneous coordinates on projective space which are much easier to handle than local coordinates on X. Instead of defining local coordinates, it is numerically simpler to compute the metric and Ricci curvature with coordinates on projective

space and compute the pullback to X using the Jacobian. All geometrically defined objects do not depend on the coordinates using which they were computed (the metric itself depends on it is defined with respect to a coordinate system, but its determinant and the Ricci scalar do not). For a given point z, the derivatives are therefore done in the numerically most advantageous patch of projective space U_k given by

$$k = \underset{i}{\operatorname{argmax}} \left| z^i \right| . \tag{B.8}$$

This leads to numerically bounded affine coordinates,

$$\left| {z_k}^i \right| \leq 1 \quad \forall\, i\,. \tag{B.9}$$

All following derivatives are implicitly with respect to affine coordinates chosen this way. Given a variety X defined by a homogeneous polynomial $Q(z) = 0$, the pullback is defined by the Jacobian with respect to some local coordinates x

$${J_a}^i = \frac{\partial x^i}{\partial x^a}, \quad \bar{J}_{\bar{b}}{}^{\bar{\jmath}} = \frac{\partial x^{\bar{\jmath}}}{\partial x^{\bar{b}}}\,. \tag{B.10}$$

If $\hat{g}$ is the metric with respect to affine coordinates, computed by differentiating the Kähler potential, the metric on X is

$$g_{a\bar{b}} = {J_a}^i \bar{J}_{\bar{b}}{}^{\bar{\jmath}} \hat{g}_{i\bar{\jmath}}\,. \tag{B.11}$$

Since on the manifold Q is constant, $dQ|_{TX} = 0$, which implies the Jacobian can be computed as

$$\frac{\partial z^i}{\partial x^a} = -\frac{G_a}{G_i} \quad \text{where} \quad G_a = \frac{\partial Q}{\partial x^a},\, G_i = \frac{\partial Q}{\partial z^i}\,. \tag{B.12}$$

Practically, coordinates on X are therefore defined by choosing a dependent affine coordinate z^δ for which the gradient is maximal,

$$\delta = \underset{i}{\operatorname{argmax}} \left| G_i \right| . \tag{B.13}$$

The local coordinates on X are then $\left\{ z^i \right\}_{k \neq i \neq \delta}$. This leads to all but one row of the Jacobian being the identity. Auxiliary objects defined with respect to affine coordinates, as above for the metric, will appear marked with a hat.

The reason for choosing the projective patch and the depending coordinate as in the above is to avoid numerical divergences. If the patch was chosen at random, one may happen to choose U_r with the corresponding coordinate $z^r \approx 0$. Since the affine coordinates are obtained via division with z^r, this may lead to arbitrarily large values. It has proven numerically advantageous to avoid these situations in principle. Choosing δ such that the derivative is maximal similarly avoids division by very small values.

It is useful to introduce the z-dependent function

$$\psi = S^\alpha h_{\alpha\bar{\beta}} S^{\bar{\beta}}\,, \tag{B.14}$$

so the Kähler potential can be written as

$$K(z) = \frac{1}{k\pi}\log\big(\psi(z)\big)\,. \tag{B.15}$$

Since the potential is the second derivative of the potential, we need the derivatives of ψ,

$$\psi_i = \frac{\partial\psi}{\partial z^i}\,,\quad \psi_{ij} = \frac{\partial^2\psi}{\partial z^i\partial z^j}\,,\quad \psi_{ij\bar{k}} = \frac{\partial^3\psi}{\partial z^i\partial z^j\partial\bar{z}^{\bar{k}}}\,,\quad \psi_{ij\bar{k}\bar{l}} = \frac{\partial^4\psi}{\partial z^i\partial z^j\partial\bar{z}^{\bar{k}}\partial\bar{z}^{\bar{l}}}\,. \tag{B.16}$$

Other combinations of holomorphic and anti-holomorphic derivatives are obtained by complex conjugation, for example $\psi_{\bar{i}} = \bar{\psi}_i$. Since the h matrix does not depend on z, these derivatives depend only on the derivatives of $S^\alpha(z)$, which are computed using automatic differentiation. There are at most two holomorphic or anti-holomorphic derivatives, so only the first and second holomorphic derivatives of S^α need to be generated. This can be be achieved by two applications of the holomorphic Jacobian computable using JAX.

In terms of the above derivatives, the metric and its derivatives,

$$\hat{g}_{i\bar{\jmath}} = \frac{1}{k\pi}\frac{\partial^2\log\left(S^\alpha h_{\alpha\bar{\beta}}S^{\bar{\beta}}\right)}{\partial z^i\partial\bar{z}^{\bar{\jmath}}} \tag{B.17}$$

$$\hat{g}_{i\bar{\jmath},\alpha} = \frac{\partial\hat{g}_{i\bar{\jmath}}}{\partial z^\alpha} \tag{B.18}$$

$$\hat{g}_{i\bar{\jmath},\alpha\bar{\beta}} = \frac{\partial^2\hat{g}_{i\bar{\jmath}}}{\partial z^\alpha\partial\bar{z}^{\bar{\beta}}}\,, \tag{B.19}$$

can be written as

$$k\pi\,\hat{g}_{i\bar{\jmath}} = \frac{\psi_{i\bar{\jmath}}}{\psi} - \frac{\psi_i\psi_{\bar{\jmath}}}{\psi^2} \tag{B.20}$$

$$k\pi\,\hat{g}_{i\bar{\jmath},\alpha} = \frac{1}{\psi}\,\psi_{\alpha i\bar{\jmath}} - \frac{1}{\psi^2}\left(\psi_\alpha\,\psi_{i\bar{\jmath}} + \psi_i\,\psi_{\alpha\bar{\jmath}} + \psi_{\bar{\jmath}}\psi_{\alpha i}\right) + \frac{2}{\psi^3}\,\psi_\alpha\,\psi_i\,\psi_{\bar{\jmath}} \tag{B.21}$$

$$\begin{aligned} k\pi\,\hat{g}_{i\bar{\jmath},\alpha\bar{\beta}} =& \frac{1}{\psi}\,\psi_{\alpha i\bar{\beta}\bar{\jmath}} \\ &- \frac{1}{\psi^2}\left(\psi_{\alpha i\bar{\jmath}}\,\psi_{\bar{\beta}} + \psi_{i\bar{\jmath}\bar{\beta}}\,\psi_\alpha + \psi_{\alpha\bar{\beta}}\,\psi_{i\bar{\jmath}} + \psi_{i\bar{\beta}}\,\psi_{\alpha\bar{\jmath}} + \psi_i\,\psi_{\alpha\bar{\beta}\bar{\jmath}} + \psi_{\bar{\beta}\bar{\jmath}}\,\psi_{\alpha i} + \psi_{\bar{\jmath}}\,\psi_{\alpha i\bar{\beta}}\right) \\ &+ \frac{2}{\psi^3}\left(\left(\psi_\alpha\,\psi_{i\bar{\jmath}} + \psi_i\,\psi_{\alpha\bar{\jmath}} + \psi_{\bar{\jmath}}\psi_{\alpha i}\right)\psi_{\bar{\beta}} + \psi_{\alpha\bar{\beta}}\,\psi_i\,\psi_{\bar{\jmath}} + \psi_\alpha\,\psi_{i\bar{\beta}}\,\psi_{\bar{\jmath}} + \psi_\alpha\,\psi_i\,\psi_{\bar{\beta}\bar{\jmath}}\right) \\ &- \frac{6}{\psi^4}\,\psi_\alpha\,\psi_i\,\psi_{\bar{\beta}}\,\psi_{\bar{\jmath}}\,. \end{aligned} \tag{B.22}$$

Using the Jacobian, the metric in terms of local coordinates can be computed as defined in equation (B.11). However, if one is only interested in the determinant of the metric, one can

derive the following equation that does not explicitly involve the Jacobian:

$$\det(g) = \frac{|G|^2}{|G_\delta|^2} \det(\hat{g})\,. \tag{B.23}$$

Here two new objects were used, the derivative of the defining equation $G_i = \partial_i Q$, and the following contraction:

$$|G|^2 = \hat{g}^{\bar{\eta}} G_i \bar{G}_{\bar{\jmath}} = G^\dagger \hat{g}^{-1} G\,. \tag{B.24}$$

The Ricci curvature is a derivative of the logarithm of the determinant

$$\hat{R}_{\alpha\bar{\beta}} = \frac{\partial}{\partial z^\alpha}\frac{\partial}{\partial \bar{z}^{\bar{\beta}}} \log\det g = \frac{\partial}{\partial z^\alpha}\frac{\partial}{\partial \bar{z}^{\bar{\beta}}} \log\left(\frac{|G|^2}{|G_\delta|^2}\det\hat{g}\right). \tag{B.25}$$

To make the following explicit expression for the Ricci curvature more readable, it is written in matrix notation, where the indices i and $\bar{\jmath}$ in $\hat{g}_{i\bar{\jmath}}$ and G_i are suppressed. Objects with subscripts following a comma denote the derivative, e.g. $G_{,\alpha} = \partial_\alpha G$. Using the definitions

$$\Lambda = \frac{\hat{g}^{-1} G G^\dagger \hat{g}^{-1}}{|G|^2} \quad \text{and} \quad \tilde{g} = \hat{g}^{-1} - \Lambda\,, \tag{B.26}$$

the Ricci curvature can be computed as

$$\begin{aligned} R_{\alpha\bar{\beta}} = {} & \mathrm{tr}\Big(\tilde{g}\hat{g}_{,\alpha\bar{\beta}}\Big) - \left(\frac{G^\dagger\hat{g}^{-1}G_{,\alpha}}{|G|^2} - \frac{\mathrm{tr}\,\Lambda\hat{g}_{,\alpha}}{|G|^2}\right)\left(\frac{G_{,\bar{\beta}}{}^\dagger\hat{g}^{-1}G}{|G|^2} - \frac{\mathrm{tr}\,\Lambda\hat{g}_{,\bar{\beta}}}{|G|^2}\right) - \mathrm{tr}\left(\tilde{g}\,\hat{g}_{,\alpha}\,\hat{g}^{-1}\,\hat{g}_{,\bar{\beta}}\right) \\ & + \frac{1}{|G|^2}\left(G_{,\bar{\beta}}{}^\dagger\hat{g}^{-1}G_{,\alpha} - G^\dagger\hat{g}^{-1}g_{,\bar{\beta}}\,\hat{g}^{-1}G_{,\alpha} - G_{,\bar{\beta}}{}^\dagger\hat{g}^{-1}\hat{g}_{,\alpha}\,\hat{g}^{-1}G + \mathrm{tr}\Big(\Lambda\hat{g}_{,\bar{\beta}}\,\hat{g}^{-1}\hat{g}_{,\alpha}\Big)\right). \end{aligned} \tag{B.27}$$

This is the Ricci curvature with indices corresponding to all affine coordinates. The Ricci curvature on X is obtained by contracting with the Jacobian,

$$R_{a\bar{b}} = J_a{}^i \bar{J}_{\bar{b}}{}^{\bar{\jmath}} \hat{R}_{i\bar{\jmath}}\,. \tag{B.28}$$

From this, the Ricci scalar can be computed by contracting with the inverse of the local metric:

$$R = R_{a\bar{b}} g^{a\bar{b}}\,. \tag{B.29}$$

B.3. Monte Carlo Integration and Sampling

The ability to sample points that lie on a given variety X, defined by a homogeneous polynomial $Q(z)$ as a subspace of projective space, is crucial for all numerical algorithms presented here. When the probability distribution is known, random sampling can be used to approximate integrals as a Monte Carlo sum. At the heart of Donaldson's algorithm is the integral T-operator, which means that an efficient sampling algorithm is required to achieve convergence

in a reasonable amount of time. The loss functions for machine learning can also be seen as Monte Carlo approximations to some functional. Since the exact meaning of the loss function (beside it being a loss function with respect to the desired target) is of less importance, it is not as significant as for Donaldson's algorithm to know the probability distribution.

Before a review of how the sampling methods are used for Monte Carlo integration, two sampling algorithms are presented. The former is a relatively simpler algorithm, but has the drawback that no probability density is, at present, available. Its advantage is that it can be used to sample points that specifically lie on the overlap between two patches of projective space of a predefined size, without discarding too many values. This is not presented here, although it was implemented, since the networks analyzed in the previous sections automatically satisfy the overlap conditions, such that no explicit training on the overlaps to enforce it is required.

B.3.1. Sampling by Solving for The Dependent Coordinate

The most basic approach to generate points on the n-dimensional variety X is to sample n complex numbers representing coordinates on X, $(z^1, \dots, z^n)$, and solve for z^{n+1} to obtain affine coordinates on the ambient space $\mathbb{CP}^{n+1}$ using $Q(z) = 0$. This uniquely defines a point on X. If the defining polynomial is not symmetric under coordinate permutation, one also has to randomly choose a patch and assign a coordinate index to the missing dependent coordinate. Depending on the manifold, one may have to restrict the sampling of the initial coordinates, so that the equation $Q(z) = 0$ has a solution for the last coordinate.

For example, in the case of the Fermat quintic in the affine patch U_0 (this is no restriction since the Fermat quintic is invariant under coordinate permutations)

$$Q_0(z_0) = \sum_{i=1}^{4} \left(z_0^i\right)^5 , \tag{B.30}$$

one can solve for z_0^4 given $(z_0^2, \dots, z_0^3)$:

$$z_0^4 = \sqrt[5]{\sum_{i=1}^{3} \left(z_0^i\right)^5} . \tag{B.31}$$

Since there are in general five fifth roots, one gets for each choice of initial complex values five points on X.

The crucial step is to find the solutions of the single-variable (all but one affine coordinate fixed) complex polynomial equation $Q(z) = 0$. A fast method to do this is by computing the eigenvalues of the polynomial. The starting point for this is the array of polynomial coefficients, which represents the polynomial $Q(z)$ with all but one coordinate fixed. The following algorithm shows how this computation can be done[1].

[1]The method was not yet available in JAX, and therefore had to be newly implemented for this application.

```
def roots(p):
  # p is an array of coefficients, must start with non-zero entry.
  # p = [1, 2, 0] corresponds to x^2 + 2x

  # construct companion matrix
  A = diag(ones((p.size - 2,), p.dtype), -1)
  A = A[0, :] = -p[1:] / p[0]

  # the roots are now given by the eigenvalues
  roots = eigvals(A)
  return roots
```

Implementation B.1: Simplified algorithm for finding roots of a polynomial.

A specific example of the difference between the two sampling algorithms defined above can be found in Figure C.7.

B.3.2. Homogeneous Sampling in Projective Space

The way to sample points on the manifold X using intersections with a line is as follows.

1. Uniformly sample two $a, b \in \mathbb{CP}^{n+1}$, defining a complex line.
2. Compute the polynomial coefficients for the complex variable t that define the equation

$$Q(a + tb) = 0\,, \tag{B.32}$$

   where $Q(z)$ is the defining homogeneous polynomial of X. This can either be done manually given a specific defining equation, or using a library for symbolic manipulations (this was done for the implementation here using SymPy [30], making it more easily extendable to other defining equations).
3. Solve the defining equation for t using the implementation of B.1.
4. Because of the multiplicity of roots, for each chosen line one finds $n + 2$ points $z = a + tb$ on the manifold.

One can uniformly sample points on $\mathbb{CP}^{n+1}$ by first sampling real numbers from $S^{2(n+2)}$ and combining them into complex numbers representing homogeneous coordinates. There are multiple algorithms for sampling points on a real sphere, an efficient one is to independently sample coordinates from a normal distribution and then divide by their norm. A simplified version of the sampling algorithm is presented below.

The result has been contributed to the original project where it is now available in the main distribution as `jax.numpy.roots`.

```python
def sample_sphere(count, dim):
  """Random points on the real unit sphere in ℝ^(dim+1)."""
  points = normal(shape=(count, dim))
  return points / norm(points, axis=1)

def line_sample(variety, ψ):
  """Sample points on the variety via line intersections."""
  a, b = sample_sphere(2, 2 * (variety.dimension + 2))
  a = a[0] + i a[1]
  b = b[0] + i b[1]

  coeffs = intersection_poly_coefficients(variety, a, b, ψ)
  t = roots(coeffs)
  # return the n + 2 solutions
  return a.reshape(1, -1) + t * a.reshape(1, -1)
```

Implementation B.2: Simplified algorithm for sampling points on a variety using line intersections.

Since the line is chosen uniformly in projective space, this sampling algorithm leads to points on the manifold that are not uniform to its volume form, but on projective space with respect to the Fubini-Study metric.

B.3.3. Monte Carlo

If one was able to directly sample according to the density given by the Calabi-Yau volume form $d\text{Vol}_{CY}$, the integrals could be approximated as an unweighted mean:

$$\int f d\text{Vol}_{CY} \approx \frac{1}{M} \sum_{m=1}^{M} f(z_m) \,. \tag{B.33}$$

Unfortunately, no such sampling algorithm is available. The next best solution, as it was introduced in section 2.3, is to use a sampling algorithm with some known probability density dA. One then obtains an approximation to the integral by computing the weighted mean of the integrand.

$$\int f d\text{Vol}_{CY} \approx \frac{1}{M} \sum_{m=1}^{M} f(z_m)\, w(z_m) \,. \tag{B.34}$$

The weights are defined by the ratio between the desired density and the actual density of the sampling algorithm

$$w = \frac{d\text{Vol}_{CY}}{dA} \,. \tag{B.35}$$

For the homogeneous sampling algorithm outlined above, the sampling density is proportional to the pull back of the Fubini-Study metric to the variety X, as defined in equation (2.43). The approximations of integrals to a mean in equations (2.40) and (2.42) are only valid if dA is a properly normalized probability distribution, which is not the case for the definition above. As long as one is interested in scale independent objects (such as integrals divided by the volume), the proportionality factor drops out, and one can directly use the determinant of the pullback of the Fubini-Study metric. The unnormalized weights are then, using the explicit expression for the holomorphic top form of the varieties defined in equation (2.38),

$$\tilde{w}(z) = \left.\frac{(-i)^n \Omega \wedge \bar{\Omega}}{\left(i_Q^* \, \omega_{\mathbb{CP}^{n+1}}^{FS}\right)^n}\right|_z = \frac{1}{\left|\partial_\delta Q(z)\right|} \frac{1}{\det g_X^{FS}} \tag{B.36}$$

where δ is the index of the dependent coordinate defining a local coordinate system, as defined in the previous section. The pullback of the Fubini-Study metric is computed as in equation (B.11) above, given its definition in affine coordinates z_k of projective space (in patch U_k)

$$\hat{g}_{i\bar{j}} = \frac{\left(1 + |z_k|^2\right)\delta_{i\bar{j}} - \bar{z}_k^{\bar{i}} z_k^j}{\left(1 + |z_k|^2\right)^2} \,. \tag{B.37}$$

The object $|z_k|^2$ denotes the norm $\sum_i \left|z_k^i\right|^2$.

To obtain the correct numerical values for scale-dependent expressions, the weights $\tilde{w}$ have to be multiplied by the mean of the densities dA,

$$w = \tilde{w} \sum_{m=1}^{M} \det g_X^{FS} \,, \tag{B.38}$$

which effectively normalizes the density.

B.4. Donaldson's Algorithm

Using the method laid out in A.2 for computing the monomials, the power matrix forming a basis of monomials from section B.1, the line sampling algorithm of section B.3.2, and Monte Carlo integration introduced in the previous section, it is now straight-forward to implement Donaldson's algorithm. Below is simplified Python pseudo-code illustrating how a single iteration of Donaldson's algorithm is computed (just as in previous examples, the shown code differs from the real implementation for performance reasons, non-central features, specific constraints of JAX, and in some instances for the sake of readability).

```
def donaldson_step(variety, h, k, pows, ψ, vol_cy, count):
  # Approximate the T operator for the `h`-matrix of degree `k` by a Monte Carlo
  # sum over `count` sample points.

  # accumulate the integral in this variable
  T = zeros_like(h)

  for i in range(count):
    # pretend this returns a single point now, for simplicity
    z = line_sample(variety, ψ)
    z, patch = to_affine(z)

    weight = mc_weight(variety, z, patch, ψ)
    s = compute_monomials(z, patch, k)
    s̄ = conj(s)

    numerator = s^α s̄^β̄
    denominator = s^α h_{αβ̄} s^β̄

    dT = numerator / denominator * weight
    T = T + dT / count

  T = T * basis_size(variety, k) / vol_cy
  new_h = invert(T).transpose()
  return new_h
```

Implementation B.3: Simplified algorithm for computing a single iteration of Donaldson's algorithm.

B.5. Training Moduli Dependent Networks

B.5.1. Training Networks for Multiple Values of ψ Simultaneously

For networks that depend on an additional moduli parameter ψ, additional thought has to go into how the batch sampling of points on the manifold is combined with sampling values for ψ. In all cases discussed here, the values of ψ are uniformly sampled from the disk of complex space given by $|\psi| < \psi_{\text{max}}$.

Computing an η-based loss such as the one defined in (4.9) requires the batch mean over $\hat{\eta}$ values. For this, it is important that all $\hat{\eta}$ values the mean is computed over corresponds to the same value of ψ. The most basic solution to this is to use a single, randomly sampled value of ψ in each iteration of gradient descent. It has proven effective, however, to compute multiple values of ψ in each step. This can be done by sampling enough points for each value of ψ that a statistically stable enough mean of $\hat{\eta}$ can be computed for each. Practically, sampling 4

values of ψ and 500 points on the manifold for each has proven to be a good choice, although no thorough search over different combinations has been conducted.

B.5.2. Better Convergence by Suppressing Entries

It has proven useful to introduce an additional set of parameters, the same number as real parameters for h, to introduce an additional sigmoid suppression as in section 4.5.1. If the final output of the network above is $\hat{h}_{\text{real}}$, an additional set of parameters $\tilde{h}_{\text{real}}$ is introduced, so the value passed to the reconstruction of the Hermitian matrix is

$$h_{\text{real}} = \sigma(\tilde{h}_{\text{real}})\, \hat{h}_{\text{real}}\,. \tag{B.39}$$

This is done for all models whose output is an h matrix. The suppression parameters are not dependent on the input, which means this is more of a numerical trick for better convergence, and is thus not included in the network architectures above. If more parameters than ψ are introduced, it may become useful to make the suppression input-dependent.

B.5.3. Initialization

Convergence using gradient descent can be achieved significantly more quickly if the initialization of the networks is close to the identity. This can be achieved in the above networks by tuning the range of values the biases of the final dense layer are chosen from, and by tuning the input-independent sigmoid suppression introduced in the above. If the initialization is done poorly, and especially if the learning rate is large, the learned metric may become indefinite. This can be prevented to some extent by using the Cholesky decomposition, instead of parametrizing the real and imaginary entries of h directly.

C. Additional Figures

This part of the appendix contains several figures with additional results that are not necessary to follow the discussion of the main part of this document.

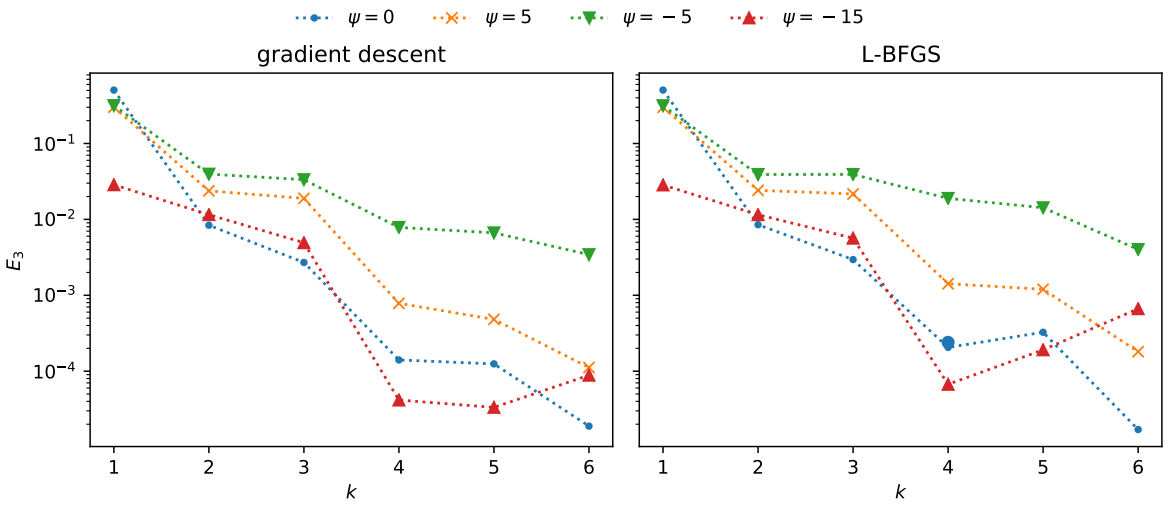

Figure C.1.: Same as Figure 4.3, except the measure used to assess convergence is the loss E_3 computed using 10 000 sample points which is the same as Figure 10 in [10]. The convergence is qualitatively similar and of the same order of magnitude.

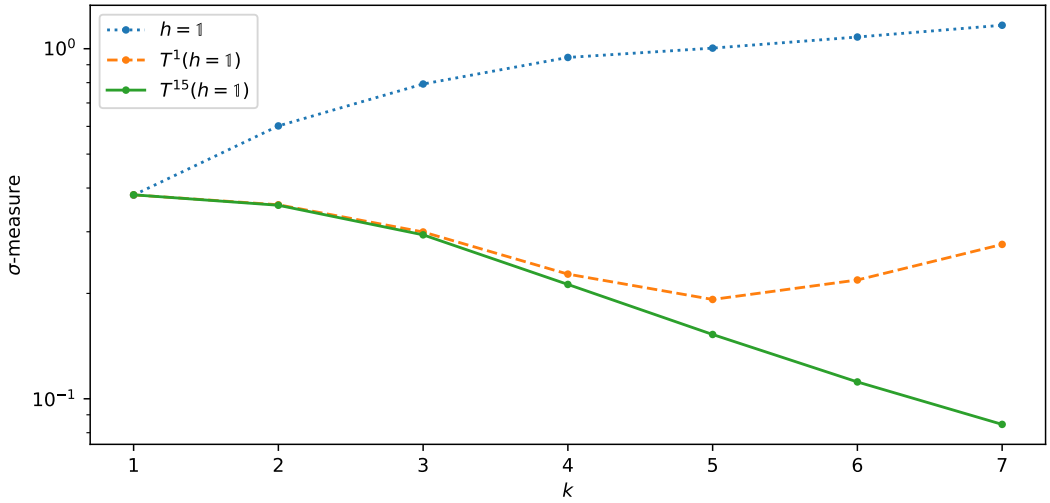

Figure C.2.: Comparison of σ-measures achieved by Donaldson's algorithm for $\psi = 100$ after 1 and 15 iterations (convergence) with those at the initial value $h = \mathbb{1}$. In contrast to $\psi = 0$, the initial value does not give a good approximation to Ricci-flatness, and the first iteration never yields better accuracies than the converged result.

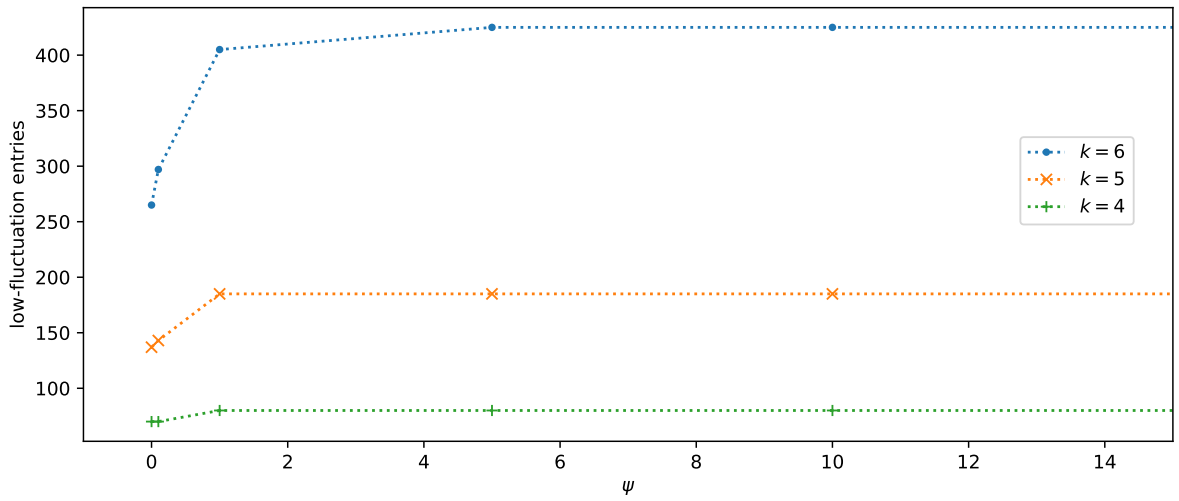

Figure C.3.: Number of entries of h in the small-fluctuation cluster as found in section 3.3 for multiple real values of ψ. In all cases the line is continued towards the value at $\psi = 100$, which is not included in the visible range to make the increase around $\psi = 0$ more obvious. The number of entries at $\psi = 100$ was found to be the same as that at $\psi = 10$.

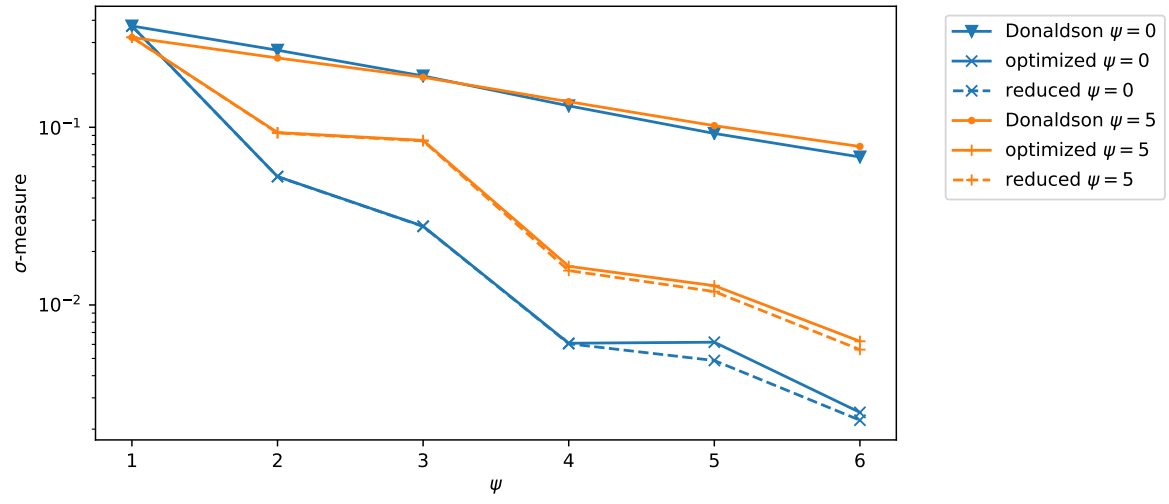

Figure C.4.: Comparison between σ-accuracies achieved by the optimized h matrices of section 4.5 before and after they are reduced to the small-fluctuation entries as found by analysing Donaldson's algorithm in section 3.3. The accuracies achieved by Donaldson's algorithm are shown as a reference.

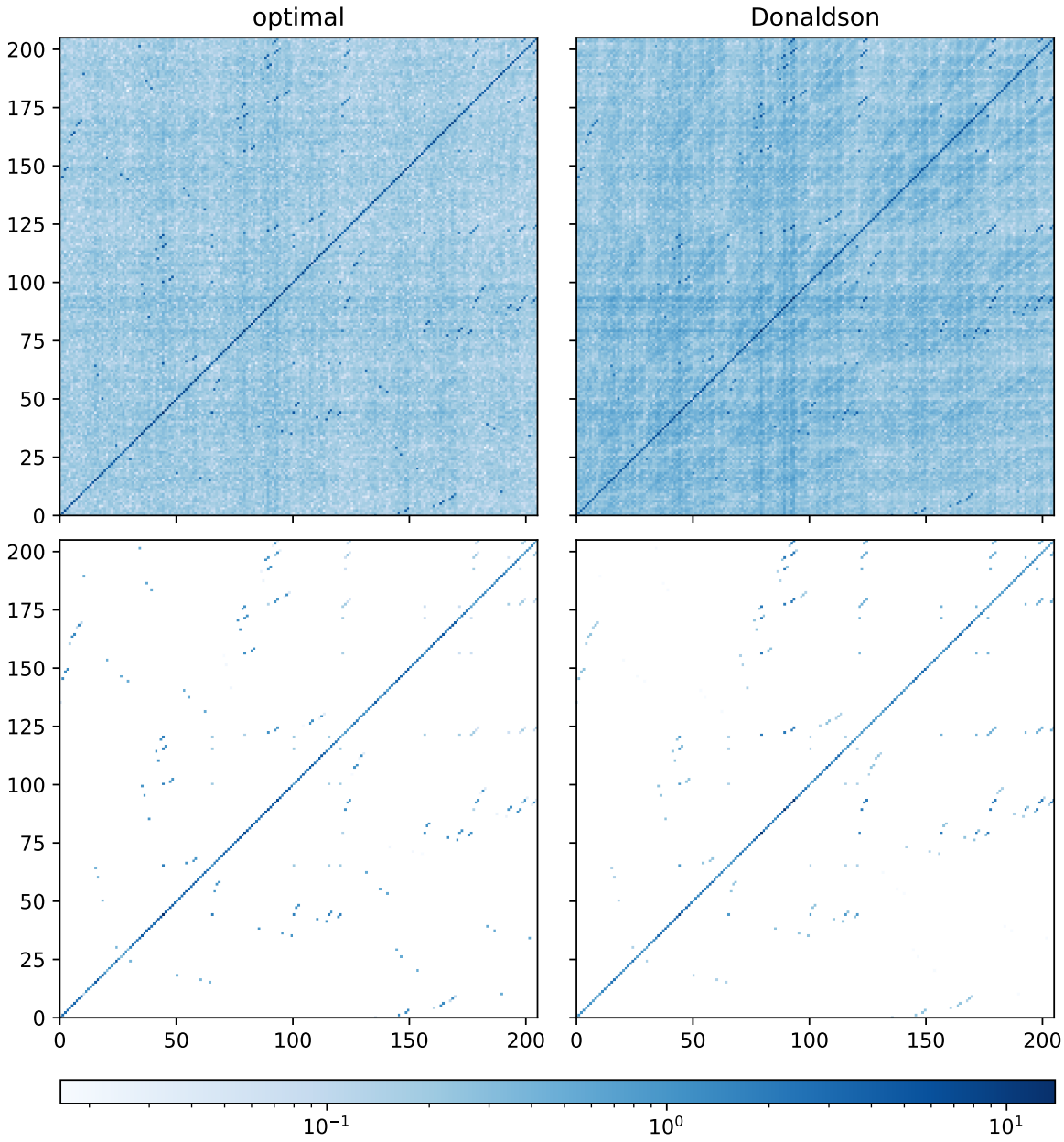

Figure C.5.: Visual comparison of h matrices produced by Donaldson's algorithm and by optimizing the η-based loss of section 4.5, for $k = 6$ and $\psi = 10$. The lower two plots show only the low-fluctuation entries as found in section 3.3.

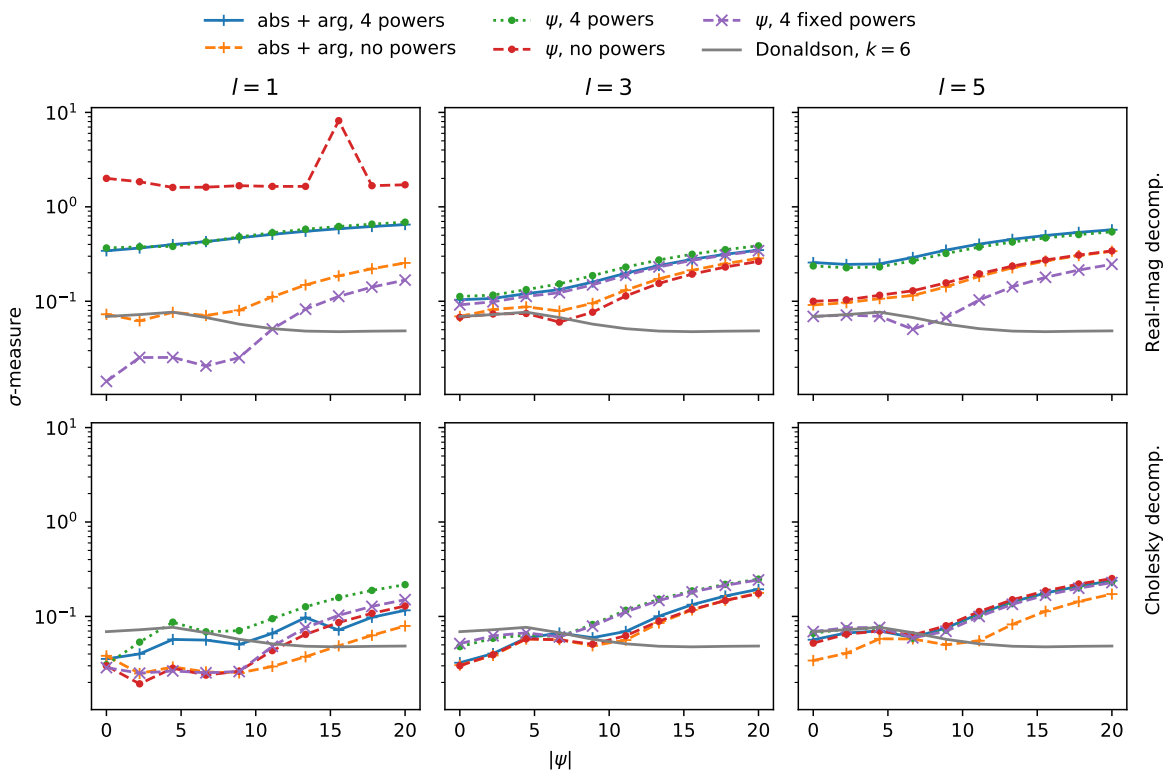

Figure C.6.: σ-accuracies at $k = 6$ achieved by different variations of the dense-layer network architecture introduced in section 4.6.1, after optimization using the η-loss. In the case where the powers are fixed and not parameters changed during gradient descent, they were chosen to be $1/2, 1, 2$, and 3. The accuracies shown are the mean over four equally spaced complex angles at each absolute value of ψ. For reference, the σ accuracy achieved by Donaldson's algorithm for each real value of ψ is shown, as well as the results for the network that was previously trained using balanced metrics.

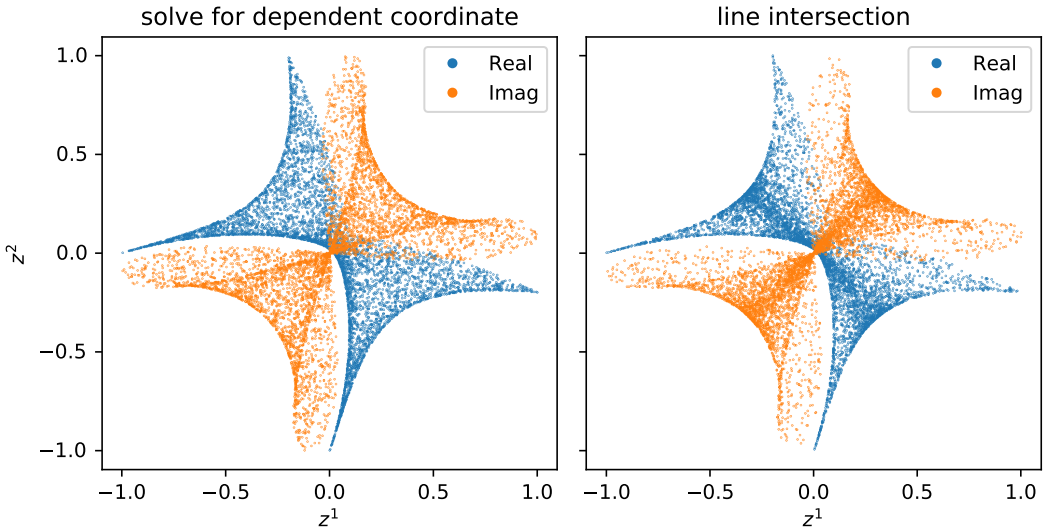

Figure C.7.: Scatter-plot comparing real and imaginary parts of the affine coordinates of the variety in $\mathbb{CP}^2$ defined by $(z^0)^3 + (z^1)^3 + (z^2)^3 + 10\, z^0 z^1 z^2 = 0$, generated using either sampling algorithm introduced in section B.3. The values all lie in the affine patch $\hat{U}_0$ defined by $|z_0^1|, |z_0^2| \leq 1$.

Bibliography

[1] P. Candelas et al. "Vacuum Configurations for Superstrings". In: *Nucl. Phys. B* 258 (1985), pp. 46–74. DOI: 10.1016/0550-3213(85)90602-9.

[2] Volker Braun, Yang-Hui He, and Burt A. Ovrut. "Yukawa couplings in heterotic standard models". In: *JHEP* 04 (2006), p. 019. DOI: 10.1088/1126-6708/2006/04/019. arXiv: hep-th/0601204.

[3] Daniel Baumann and Liam McAllister. *Inflation and String Theory*. Cambridge Monographs on Mathematical Physics. Cambridge University Press, May 2015. ISBN: 978-1-107-08969-3, 978-1-316-23718-2. DOI: 10.1017/CBO9781316105733. arXiv: 1404.2601 [hep-th].

[4] Matthew Headrick and Toby Wiseman. "Numerical Ricci-flat metrics on K3". In: *Class. Quant. Grav.* 22 (2005), pp. 4931–4960. DOI: 10.1088/0264-9381/22/23/002. arXiv: hep-th/0506129.

[5] S.K. Donaldson. "Scalar Curvature and Projective Embeddings, I". In: *J. Differential Geom.* 59.3 (Nov. 2001), pp. 479–522. DOI: 10.4310/jdg/1090349449.

[6] S. K. Donaldson. "Scalar curvature and projective embeddings, II". In: *The Quarterly Journal of Mathematics* 56.3 (Sept. 2005), pp. 345–356. ISSN: 0033-5606. DOI: 10.1093/qmath/hah044.

[7] S. K. Donaldson. *Some numerical results in complex differential geometry*. 2005. arXiv: math/0512625 [math.DG].

[8] Volker Braun et al. "Calabi-Yau Metrics for Quotients and Complete Intersections". In: *JHEP* 05 (2008), p. 080. DOI: 10.1088/1126-6708/2008/05/080. arXiv: 0712.3563 [hep-th].

[9] Michael R. Douglas et al. "Numerical solution to the Hermitian Yang-Mills equation on the Fermat quintic". In: *JHEP* 12 (2007), p. 083. DOI: 10.1088/1126-6708/2007/12/083. arXiv: hep-th/0606261.

[10] Matthew Headrick and Ali Nassar. "Energy functionals for Calabi-Yau metrics". In: *Adv. Theor. Math. Phys.* 17.5 (2013), pp. 867–902. DOI: 10.4310/ATMP.2013.v17.n5.a1. arXiv: 0908.2635 [hep-th].

[11] Anthony Ashmore, Yang-Hui He, and Burt Ovrut. *Machine learning Calabi-Yau metrics*. 2019. arXiv: 1910.08605 [hep-th].

[12] Tristan Hubsch. *Calabi-Yau manifolds: A Bestiary for physicists*. Singapore: World Scientific, 1994. ISBN: 978-981-02-1927-7.

[13] Brian R. Greene. "String theory on Calabi-Yau manifolds". In: *Theoretical Advanced Study Institute in Elementary Particle Physics (TASI 96): Fields, Strings, and Duality*. June 1996, pp. 543–726. arXiv: hep-th/9702155.

[14] J.P. Demailly. *Complex Analytic and Differential Geometry*. Université de Grenoble I, 1997. URL: https://www-fourier.ujf-grenoble.fr/~demailly/manuscripts/agbook.pdf.

[15] Phillip Griffiths and Joseph Harris. *Principles of algebraic geometry*. John Wiley & Sons, 1994.

[16] Shing-Tung Yau. "Calabi's Conjecture and some new results in algebraic geometry". In: *Proc. Nat. Acad. Sci.* 74 (1977), pp. 1798–1799. DOI: 10.1073/pnas.74.5.1798.

[17] Eugenio Calabi. "On Kähler manifolds with vanishing canonical class". In: *Algebraic geometry and topology. A symposium in honor of S. Lefschetz*. Vol. 12. 1957, pp. 78–89.

[18] Michael B. Green, John H. Schwarz, and Edward Witten. *Superstring Theory Vol. 2: 25th Anniversary Edition*. Cambridge Monographs on Mathematical Physics. Cambridge University Press, Nov. 2012. ISBN: 978-1-139-53478-9, 978-1-107-02913-2. DOI: 10.1017/CBO9781139248570.

[19] Tristan Hubsch. *Calabi-Yau manifolds: A Bestiary for physicists*. World Scientific, 1994. ISBN: 978-981-02-1927-7.

[20] Michael R. Douglas et al. "Numerical Calabi-Yau metrics". In: *J. Math. Phys.* 49 (2008), p. 032302. DOI: 10.1063/1.2888403. arXiv: hep-th/0612075.

[21] Volker Braun et al. "Eigenvalues and Eigenfunctions of the Scalar Laplace Operator on Calabi-Yau Manifolds". In: *JHEP* 07 (2008), p. 120. DOI: 10.1088/1126-6708/2008/07/120. arXiv: 0805.3689 [hep-th].

[22] Stefan Weinzierl. "Introduction to Monte Carlo methods". In: (June 2000). arXiv: hep-ph/0006269.

[23] Lara B. Anderson et al. "Numerical Hermitian Yang-Mills Connections and Vector Bundle Stability in Heterotic Theories". In: *JHEP* 06 (2010), p. 107. DOI: 10.1007/JHEP06(2010)107. arXiv: 1004.4399 [hep-th].

[24] James Bradbury et al. *JAX: composable transformations of Python+NumPy programs*. Version 0.1.72. 2020. URL: http://github.com/google/jax.

[25] Gang Tian. "On a set of polarized Kähler metrics on algebraic manifolds". In: *J. Differential Geom.* 32.1 (1990), pp. 99–130. DOI: 10.4310/jdg/1214445039.

[26] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[27] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: (Dec. 2014). arXiv: 1412.6980 [cs.LG].

[28] Ciyou Zhu et al. "Algorithm 778: L-BFGS-B: Fortran Subroutines for Large-Scale Bound-Constrained Optimization". In: *ACM Trans. Math. Softw.* 23.4 (Dec. 1997), pp. 550–560. ISSN: 0098-3500. DOI: 10.1145/279232.279236. URL: https://doi.org/10.1145/279232.279236.

[29] Andrew Trask et al. *Neural Arithmetic Logic Units*. 2018. arXiv: 1808.00508 [cs.NE].

[30] Aaron Meurer et al. "SymPy: symbolic computing in Python". In: *PeerJ Computer Science* 3 (Jan. 2017), e103. ISSN: 2376-5992. DOI: 10.7717/peerj-cs.103.

Acknowledgements

I would like to express my deep gratitude to Dr. Sven Krippendorf for our many interesting discussions, without which this thesis would not have been possible.

I thank Dr. Robert Helling for his extraordinarily engaged management of the TMP programme, and Prof. Dr. Dieter Lüst for being my tutor.

I am grateful to the developers of the JAX library, for their quick response to suggestions and software issues.