Topic 3: String Theory Compactifications, Calabi-Yau Manifolds and Ricci-flat Metrics

Lecture 4:CY metrics from NNs

What to learn, how to do it (efficiently)?

Magdalena Larfors, Uppsala University
ML in Maths and Physics 2023

AI4Research

Vetenskapsrådet

Brief summary of Lara’s & James’ Lectures

- Calabi-Yau manifolds: compact, complex, Kähler
- Admit unique Ricci-flat metric
  [for fixed Kähler and complex structures]
- Large databases of example manifolds
- The Ricci-flat metric, g_{CY} is something we want to compute
- This requires solving a PDE on a compact, curved space

Aim of these lectures

- Show how ML can be used in learning CY metrics
- Partial overview of past work…
- … with emphasis on open-source packages

- Goals
  Know which packages exist and what they are designed to do
  Develop familiarity with (some) methods and packages
  …

Outline: ML of CY metrics

Lecture 1: overview

- Intro ML implementations
  Available packages
- Point sample recap:
  measures and patches
- Training and loss functions
- Accuracy and error measures
- Different models/neural nets

Lecture 2: details

- Direct learning of CY metrics
  (details and demo)
- Advanced methods
  Goals and realizations

Tutorial

- Implementations & experiments

Main references
Anderson et al 2012.04656, Douglas et al 2012.04797, Larfors et al 2205.13408, Gerdes et al 2211.12520

Numerical approximations of CY metrics

Traditional methods: *cf James' & Lara's lectures*

- Donaldson's algorithm (iterative fixed point scheme)
- Headrick-Nassar functional minimization

→ Kähler potential using spectral basis

Machine learning methods:

- ML-assisted traditional methods → Kähler potential
- Direct ML → Kähler potential or CY metric

Accuracy checks same for all methods

Benefits of ML approach

- Significant improvement of speed, performance, and scope
  - More accurate approximation for given compute
  - Better scaling
  - More advanced CYs (eg CICY; toric ambient spaces)
  - Learn moduli dependence (realized for 1-2 cpl str moduli on quintic CY)
- Generalize to SU(3) structures (realized for Strominger-Hull ansatz on quintic CY)
- PyTorch, TensorFlow, JAX: ML libraries for auto-differentiation
  → efficiently compute derivatives and optimize loss functions

- Drawback: in contrast to Donaldson, lack $k \to \infty$ proof

Setting up the problem:

Problem: find Ricci flat CY metric $g_{CY} \iff$ find J_{CY} that solves the MA eq.

$$J_{CY} \wedge J_{CY} \wedge J_{CY} = \kappa\, \Omega \wedge \bar{\Omega} = \kappa \text{ d Vol}_{CY}$$

where κ is some complex constant.

While J_{CY} is unknown we know Ω and J_{FS}

- Find $J_{CY} = J_{FS} + \partial\bar{\partial}\phi$ that solves MA eq.
- We will train a neural network to predict $g_{CY} \sim J_{CY}$

Setting up the problem:

Problem: find Ricci flat CY metric $g_{CY} \iff$ find J_{CY} that solves the MA eq.

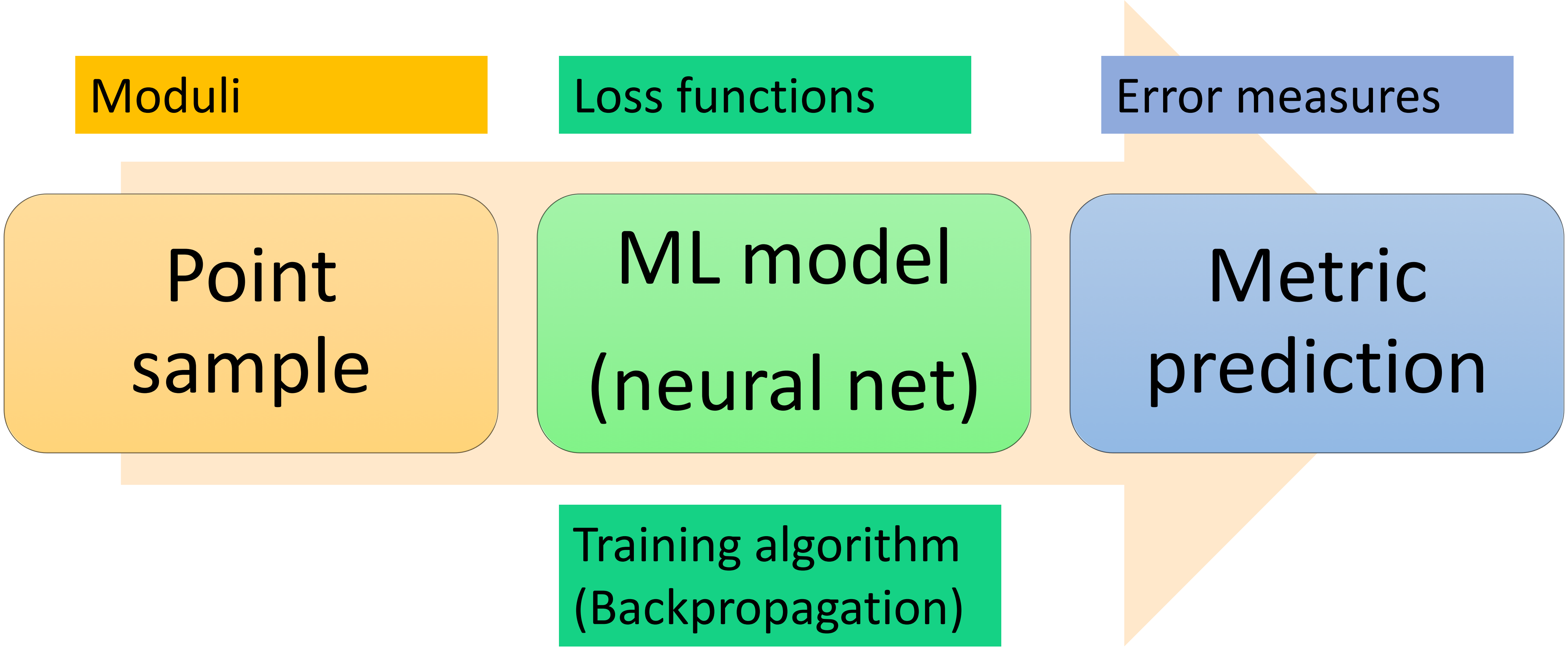
$$J_{CY} \wedge J_{CY} \wedge J_{CY} = \kappa\, \Omega \wedge \bar{\Omega} = \kappa \text{ d Vol}_{CY}$$

where κ is some complex constant.

Let's solve this on a quintic CY, $X \subset \mathbb{P}^4$, defined as zero set $p = 0$
In affine coordinates $\{z_a\}$, can compute

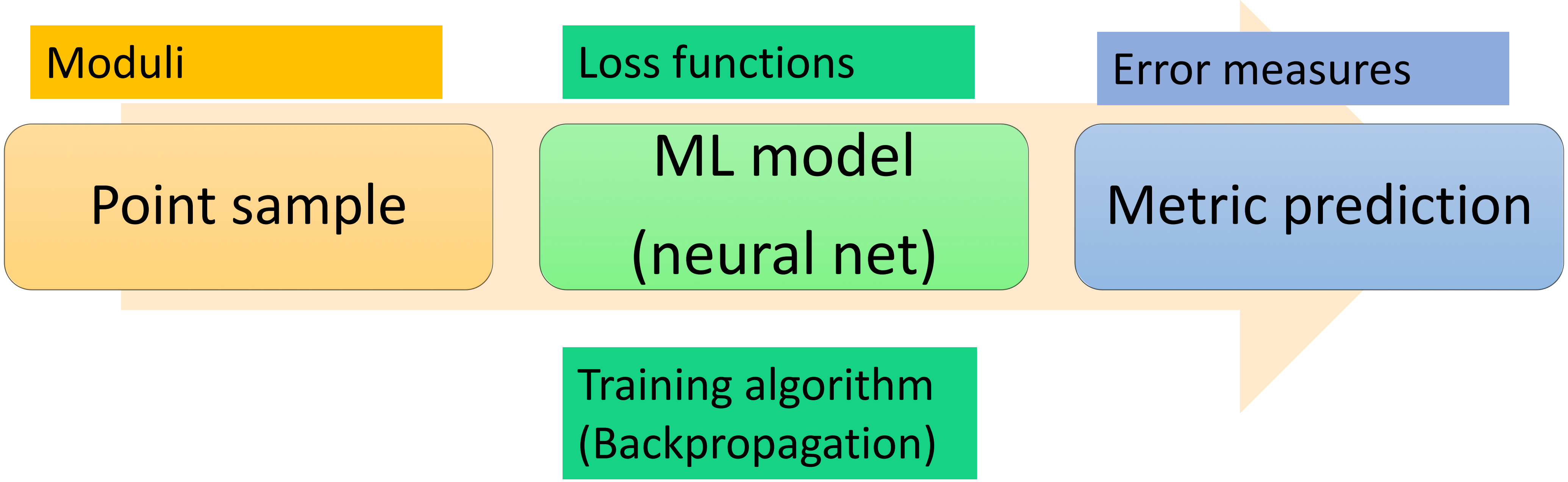
- $\Omega = \frac{dz_1 \wedge dz_2 \wedge dz_3}{\partial_{z_4} p}|_{p=0}$ $\qquad J_{FS} = \frac{i}{2\pi} \partial\bar{\partial} \sum_1^4 \ln(z_i \overline{z_i})|_{p=0}$
- Find (global) $J_{CY} = J_{FS} + \partial\bar{\partial}\phi$ that solves MA eq.

Machine Learning implementation template



Machine Learning implementation template

- A CY metric package provides implementation of template
- While structure is similar, architecture choices abound



CY metric ML packages on Github

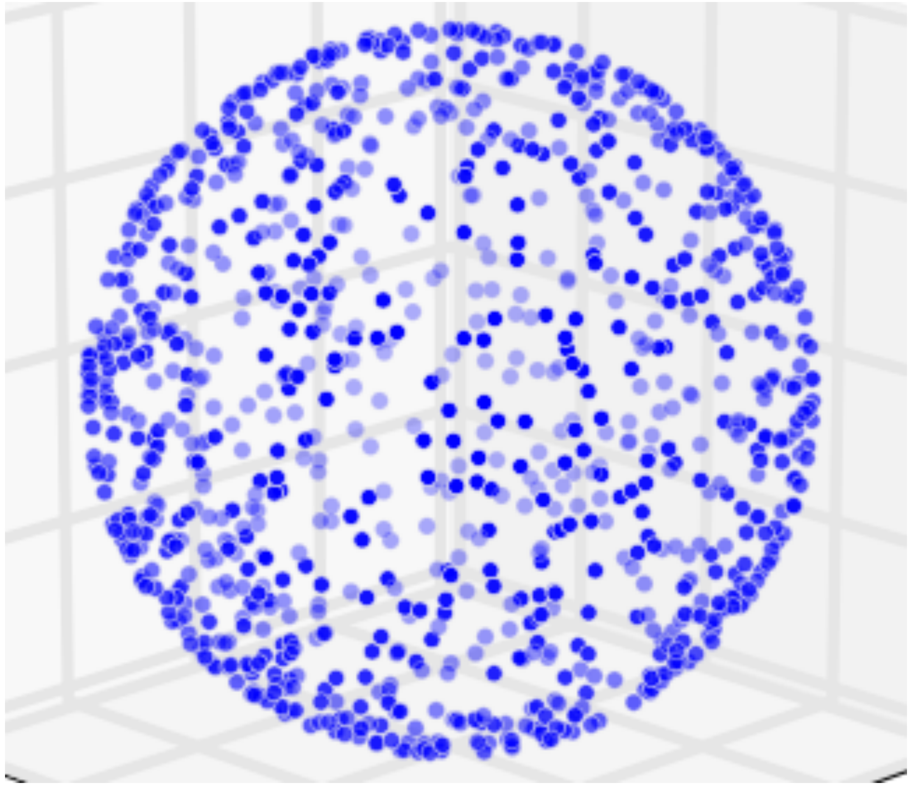
- Holomorphic and bihomogeneous networks — Douglas & Qi
  ML using spectral ansatz, CY hypersurface in $\mathbb{P}^n$
  python/TensorFlow https://github.com/yidiq7/MLGeometry

- cymetric — Ruehle & Schneider
  direct ML methods, works on CICYs and Kreuzer-Skarke CY
  python/TensorFlow & Mathematica
  https://github.com/pythoncymetric/cymetric

- Cyjax — Gerdes & Krippendorf
  ML Donaldson's algebraic ansatz of Kähler potential, CY hypersurface in $\mathbb{P}^n$
  python/JAX https://github.com/ml4physics/cyjax

  Open source packages, can be freely used for projects (& contributions welcome)

Point sample



Weights and weights

- In the following slides we will reuse the term “weights” for discrete integration measures
- We also use the term “weights” for some parameters of neural networks
- Hopefully this will not cause too much confusion…

Generating a random point sample

Goal: Random set of points on CY, sampled w.r.t. known measure dA

Why?

- We need to compute integrals (e.g to check accuracy)

$$\int_X \mathrm{dVol}_{\mathrm{CY}}\, f = \int_X dA\, \frac{\mathrm{dVol}_{\mathrm{CY}}}{dA}\, f$$

- Numerically, evaluate integral as weighted sum

$$\frac{\kappa}{6N} \sum_{i=1}^{N} w_i f(p_i)$$

where

$$w_i = \left.\frac{\mathrm{dVol}_\Omega}{dA}\right|_{p_i} \qquad \mathrm{dVol}_\Omega = \Omega \wedge \bar{\Omega} \qquad \Rightarrow \qquad \mathrm{dVol}_{\mathrm{CY}} = \frac{\kappa}{3!}\mathrm{dVol}_\Omega$$

Generating a random point sample

Let $X: p = 0 \subset \mathbb{P}^4$ be the quintic CY

- Pick random point on $\mathbb{P}^4$, reject all points off X.
- Pick some ambient coordinates, solve for the rest using $p = 0$
- Markov Chain Monte Carlo method
- Algorithm using theorem by Shiffman-Zelditch

Douglas et. al: 06

Generating a random point sample

Algorithm applied to quintic $X: p = 0 \subset \mathbb{P}^4$ Douglas et. al: 06

- Sample uniformly distributed points on S^9, then mod out phase $\rightsquigarrow$ random points on $\mathbb{P}^4$, distributed w.r.t. FS measure on $\mathbb{P}^4$
- 2 such points $q_{1,2} \rightsquigarrow$ line in $\mathbb{P}^4$, intersects X in 5 points
  Solve $p(q_1 + tq_2)$ = 0 → 5 solutions t^*
- Repeating this process M times $\rightsquigarrow 5M$ random points on X
- Shiffman-Zelditch: these points are distributed w.r.t. FS measure on X

Generalizations beyond quintic → tomorrow.

Coordinates, patches and weights

Algorithm gives point sample $\{p_K\}$ on quintic $X: p = 0 \ \subset \mathbb{P}^4$

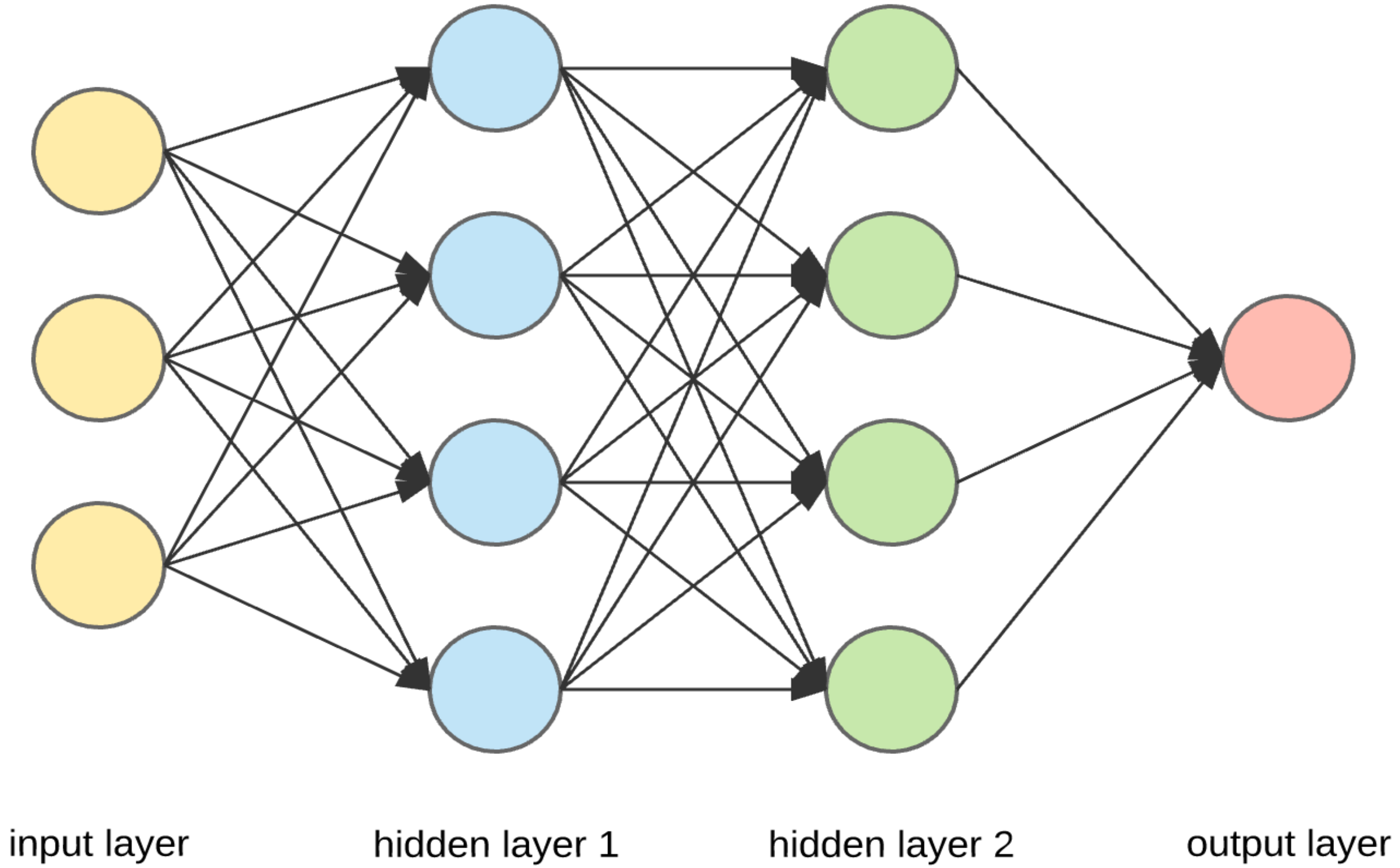
- Homogeneous coordinates $\{p_K\}$ = $\{x_0, x_1, x_2, x_3, x_4\}$; $x_i \sim \alpha x_i$
- Select patch: pick any* non-zero x_i. Say this is x_0

  → affine coordinates $\{p_K\}$ = $\{z_1, z_2, z_3, z_4\} = \left\{\frac{x_1}{x_0}, \frac{x_2}{x_0}, \frac{x_3}{x_0}, \frac{x_4}{x_0}\right\}$

3 lin. indep. coordinates, since $p(x_0, x_1, x_2, x_3, x_4) = 0$

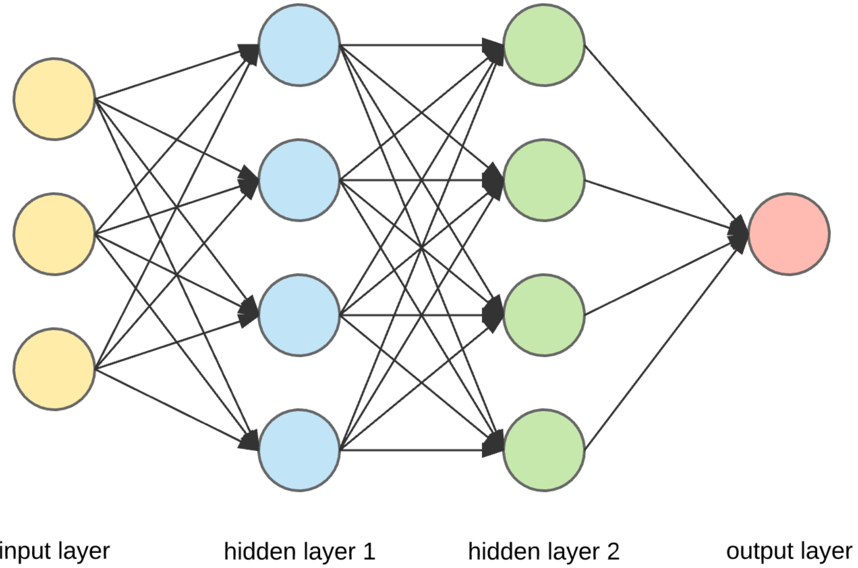
- Compute Ω , J_{FS} at point → weights w_i for numerical integrals

*Be clever: e.g. pick $\mathrm{x_i}$ of largest norm → numerical stability



ML models: Set-up & train

Neural nets for CY metrics: generalities



$$z_k = \sigma_k(W_k z_{k-1} + b_k)$$

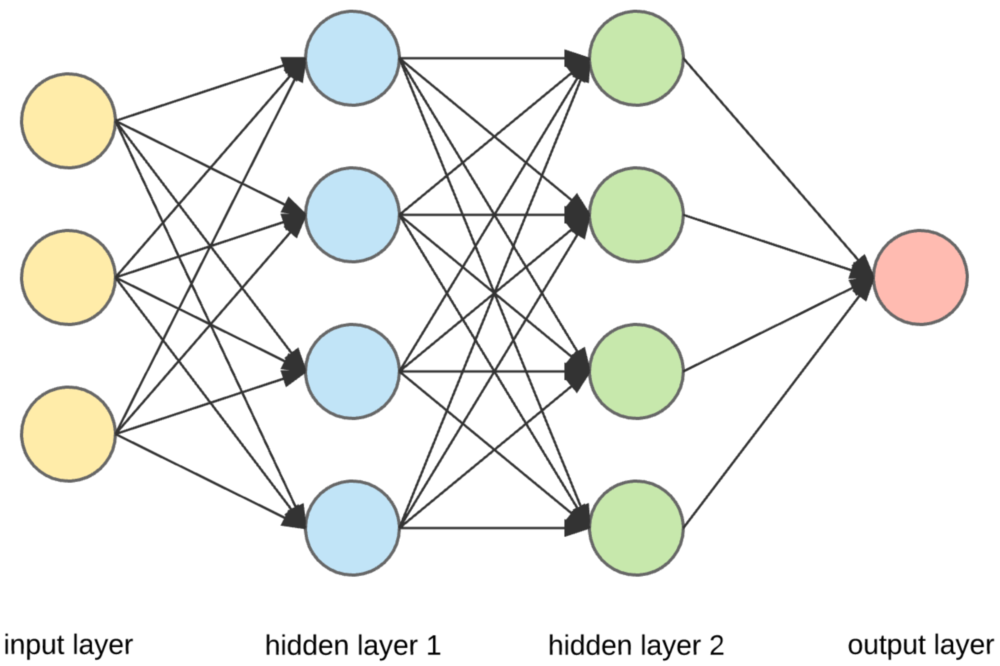
Act. fcn

weight

bias

- Input and output layers
- Hidden layers;
  trainable parameters $\theta_k = (W_k, b_k)$
- Fully connected, feed-forward
- (Semi)supervised learning:
  Minimize (custom) loss functions
- After training:
  NN → approximate CY metric

Neural nets: generalities



$$z_k = \sigma_k(W_k z_{k-1} + b_k)$$

Architectural choices

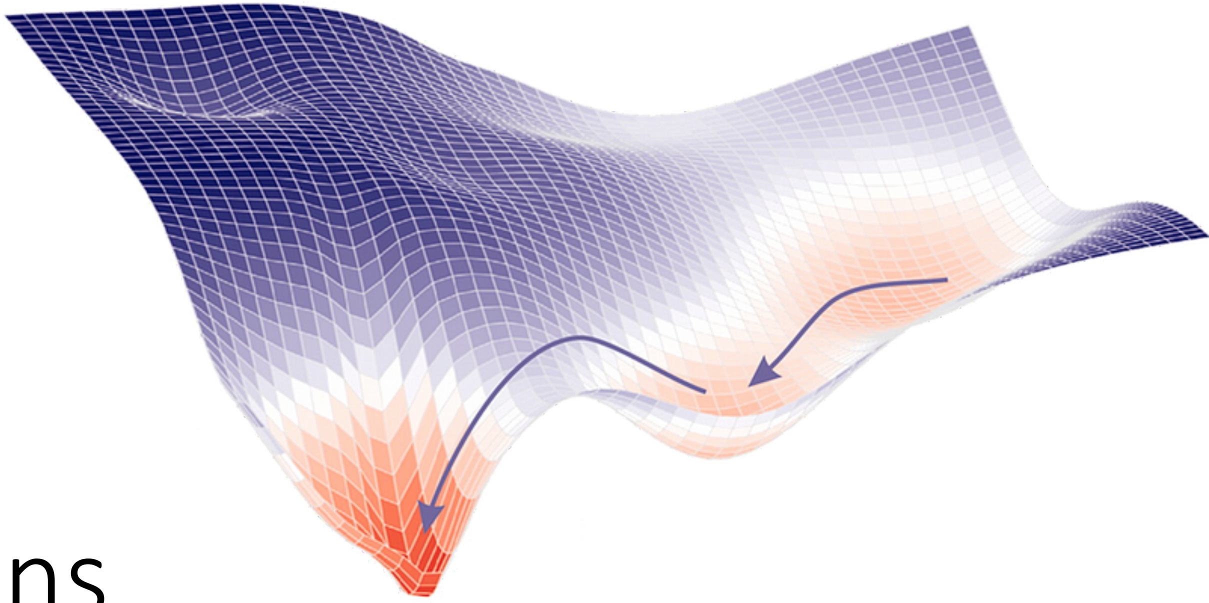
- What to predict
  metric, Kähler pot, H-matrix?
- Encode constraints in NN or loss?
  (global, complex, Kähler…)

Then train

- Minimize loss functions

And check performance

- Error measures



Training and Loss functions

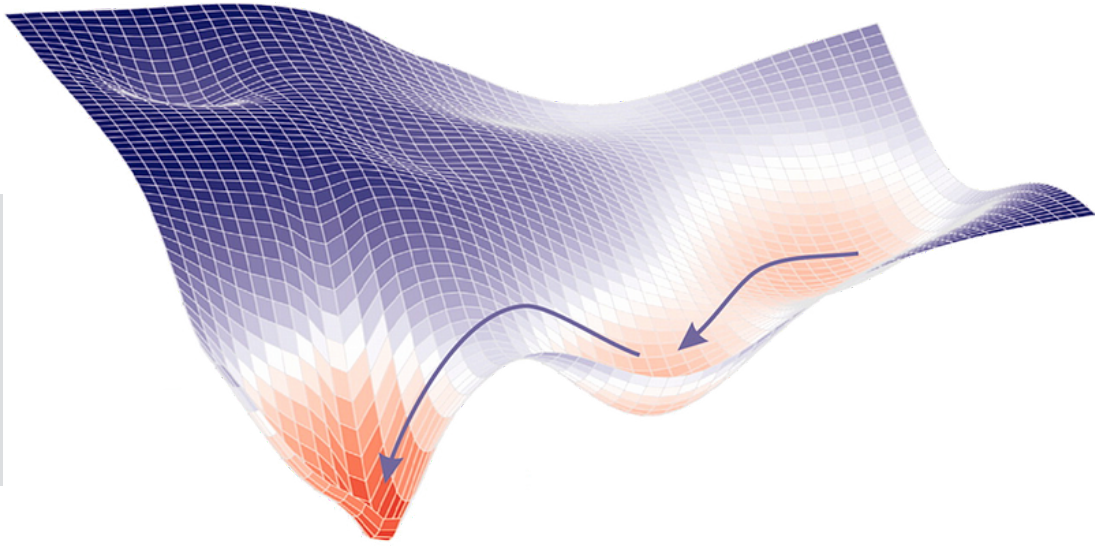
$$\theta^{(i)} \to \theta^{(i)} - \alpha \frac{\partial L}{\partial \theta^{(i)}}$$

Cf. Fabian Ruehle's lecture

Training and Loss functions

- Recall:
  PyTorch, TensorFlow, JAX: ML libraries for auto-differentiation
  → efficiently compute derivatives and optimize loss functions

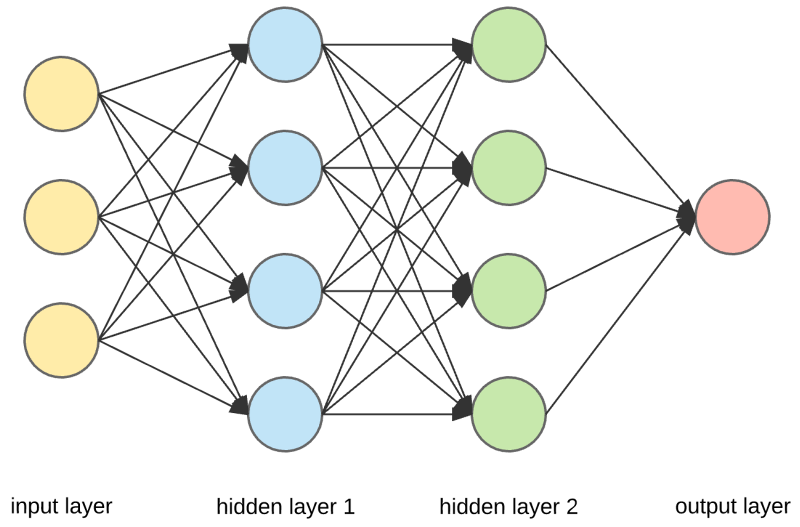
$$\theta^{(i)} \to \theta^{(i)} - \alpha \frac{\partial L}{\partial \theta^{(i)}}$$



Training the network

- NN with layers

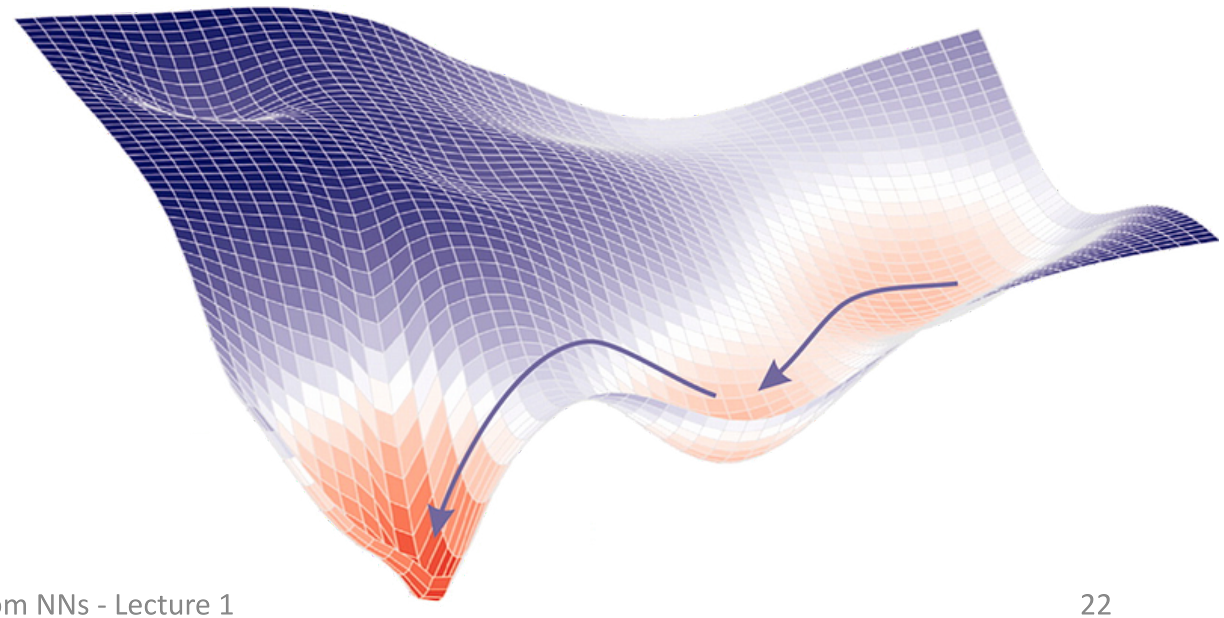
$$z_k = \sigma_k(W_k z_{k-1} + b_k)$$



- Input → prediction → loss

Training by gradient descent

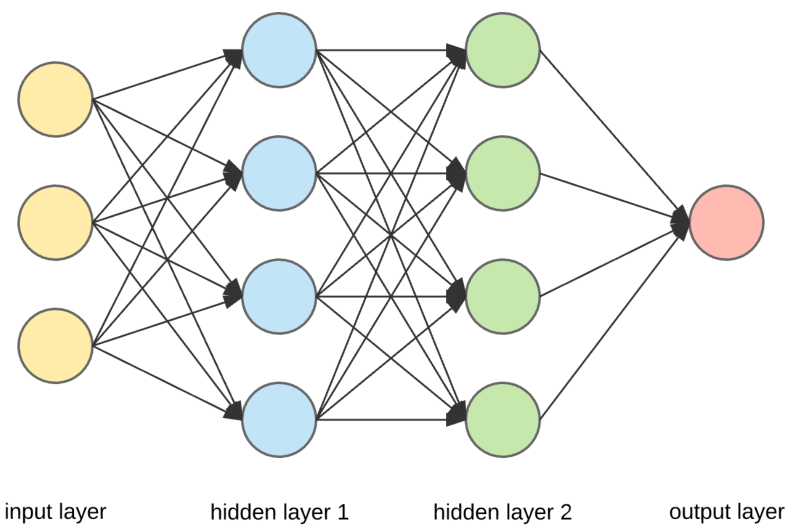
- Compute loss gradients at points
- Move towards smaller loss
- Repeat for many points



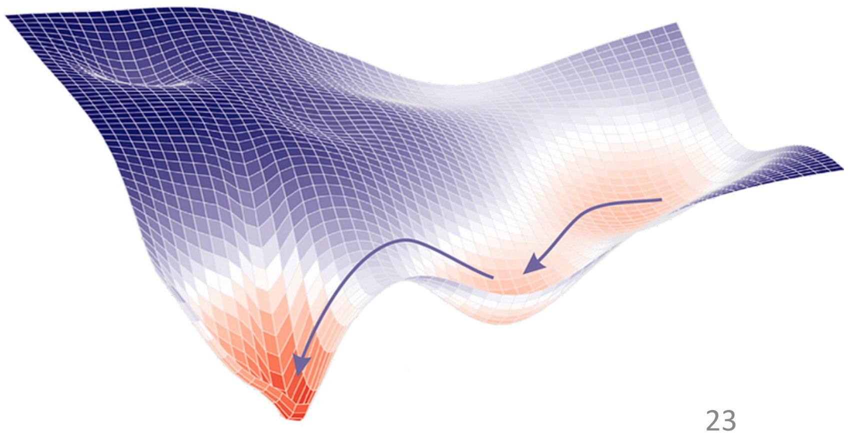
Training the network

Loss function depends on weights

- Backpropagation
  Layer by layer from end to start
  Loss gradients by chain rule
  Update weights to minimize loss
- Stochastic gradient descent
  Mini-batches and epochs
  avoid getting stuck in local min
- ML libraries (TensorFlow etc) have built in algorithms for this



$$\theta^{(i)} \to \theta^{(i)} - \alpha \frac{\partial L}{\partial \theta^{(i)}}$$



So what loss functions should we use?

Loss functions encode math constraints

- We train the network to get Ricci-flat metric (in given Kähler class)
- We don't know metric --- supervised learning not good*
- Resolution: semi-supervised learning
  1. Encode mathematical constraints as (scalar) loss functions
  2. Train network (adapt layer weights) to minimize loss functions
- E.g. satisfy Monge-Ampere eq $\rightsquigarrow$ minimize Monge-Ampere loss

$$\mathcal{L}_{\text{MA}} = \left|\left| 1 - \frac{1}{\kappa} \frac{\det g_{\text{pr}}}{\Omega \wedge \bar{\Omega}} \right|\right|_n$$

Loss functions encode math constraints

- We train the network to get Ricci-flat metric (in given Kähler class)
- Satisfy Monge-Ampere eq $\rightsquigarrow$ minimize MA loss

$$\mathcal{L}_{\text{MA}} = \left|\left| 1 - \frac{1}{\kappa} \frac{\det g_{\text{pr}}}{\Omega \wedge \bar{\Omega}} \right|\right|_n$$

- Set Ricci tensor to zero $\rightsquigarrow$ minimize Ricci loss

$$\mathcal{L}_{\text{Ricci}} = ||R||_n = \left|\left| \partial\bar{\partial} \ln \det g_{\text{pr}} \right|\right|_n$$

- Derivatives: compute by tweaking ML auto-differentiation methods

More loss functions

Also might need to check

- manifold-ness: match metrics on patch overlaps

$$\mathcal{L}_{\text{transition}} = \frac{1}{d}\sum_{(s,t)} \left|\left| g_{\text{pr}}^{(t)} - T_{(s,t)} \cdot g_{\text{pr}}^{(s)} \cdot T_{(s,t)}^{\dagger} \right|\right|_n \quad , \quad T_{(s,t)} \text{ transition matrix}$$

- Kähler-ity: check $d\,J_{pr} = 0$

$$\mathcal{L}_{\text{dJ}} = \sum_{ijk} ||\Re c_{ijk}||_n + ||\Im c_{ijk}||_n \,, \quad \text{with } c_{ijk} = g_{i\bar{j},k} - g_{k\bar{j},i} \quad \text{and } g_{i\bar{j},k} = \partial_k g_{i\bar{j}}$$

- Same Kähler class $J_{pr} \sim J_{FS}$ (not needed on quintic)

Architectural choices will determine which loss functions we need
→Importance of loss functions should be tunable (on/off)

Accuracy and performance

Check performance

- After the network is trained, want to check performance
- Separate test/validation sets

Input and performance

Input: N points p_i, randomly distributed w.r.t to known measure $\mathrm{d}A$ on CY.

After training, measure performance:

does the MA equation hold? is the metric Ricci flat?

Input and performance

Input: N points p_i, randomly distributed w.r.t to known measure dA on CY.

After training, measure performance:

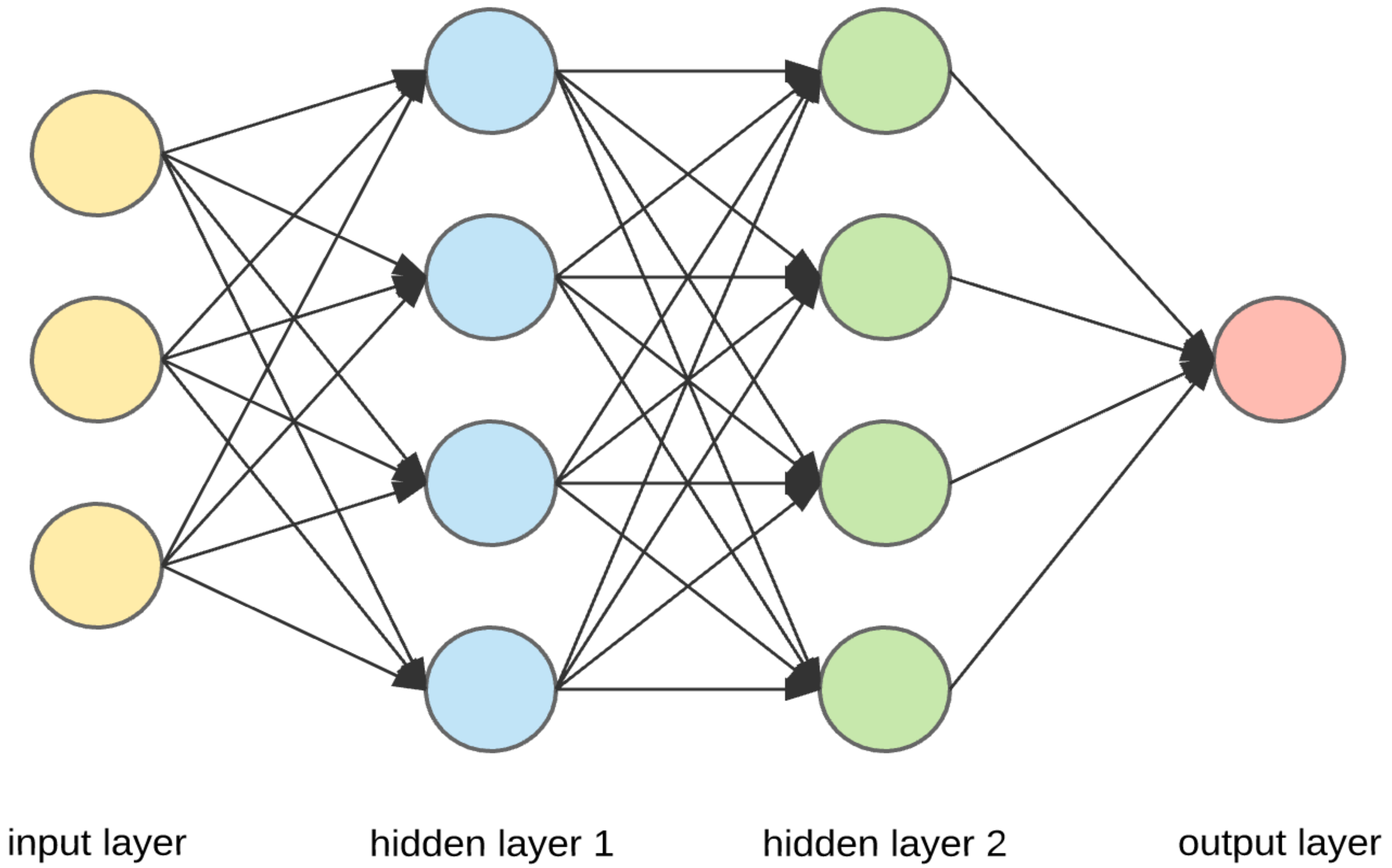
does the MA equation hold? is the metric Ricci flat?

Check via established benchmarks:

$$\sigma = \frac{1}{\mathrm{Vol}_{\mathrm{CY}}} \int_X \left| 1 - \kappa \, \frac{\Omega \wedge \overline{\Omega}}{(J_{\mathrm{pr}})^3} \right| \; , \; \mathcal{R} = \frac{1}{\mathrm{Vol}_{\mathrm{CY}}} \int_X |R_{\mathrm{pr}}| \; .$$

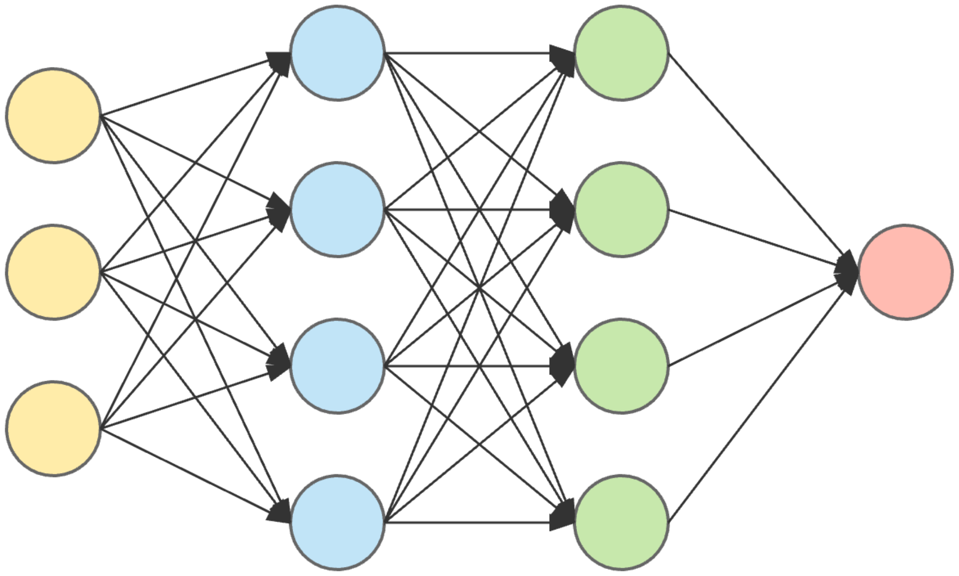
using Monte Carlo integration for any function f

$$\int_X \mathrm{dVol}_{\mathrm{CY}} f = \int_X \frac{\mathrm{dVol}_{\mathrm{CY}}}{\mathrm{d}A} \mathrm{d}A \, f = \frac{1}{N} \sum_i w_i f|_{p_i} \quad \text{with} \quad w_i = \frac{\mathrm{dVol}_{\mathrm{CY}}}{\mathrm{d}A}|_{p_i}$$



Different ML implementations

Neural nets: generalities

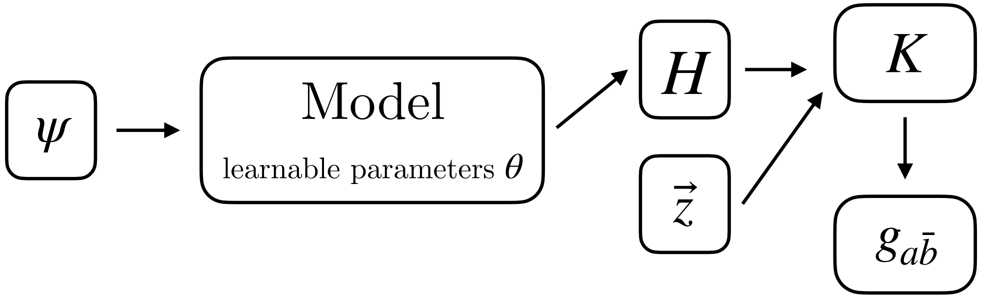


- Input and output layers
- Hidden layers; trainable weights
- (Semi)supervised learning
- Minimize (custom) loss functions
- After training:
  NN → approximate CY metric

ML implementations

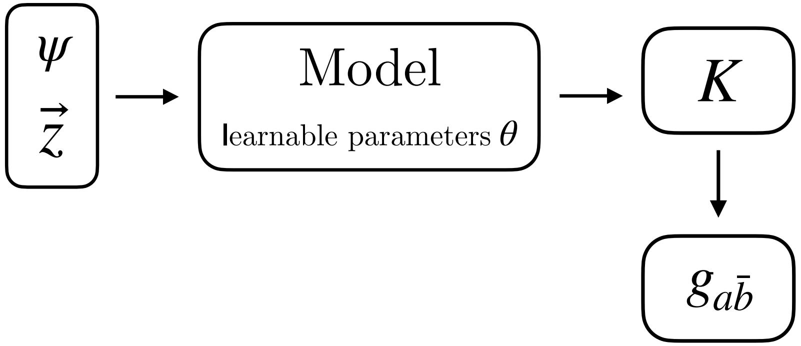
1. Learn Donaldson's H matrix

   cyjax



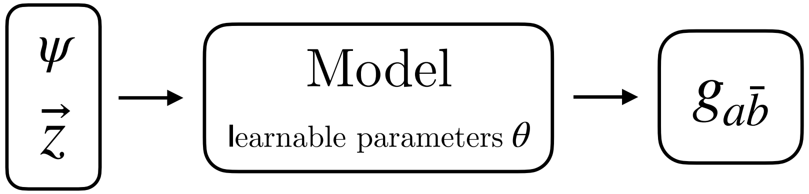
2. Learn Kähler potential

   Hol/Bihom network



3. Learn metric

   cymetric



Pro/con

Learning H or K

Pro

- Kähler
- Globally defined
- Donaldson's alg: convergence as $k \to \infty$

Con

- Scaling (of spectral basis)
- No generalization beyond Kähler

Learning metric

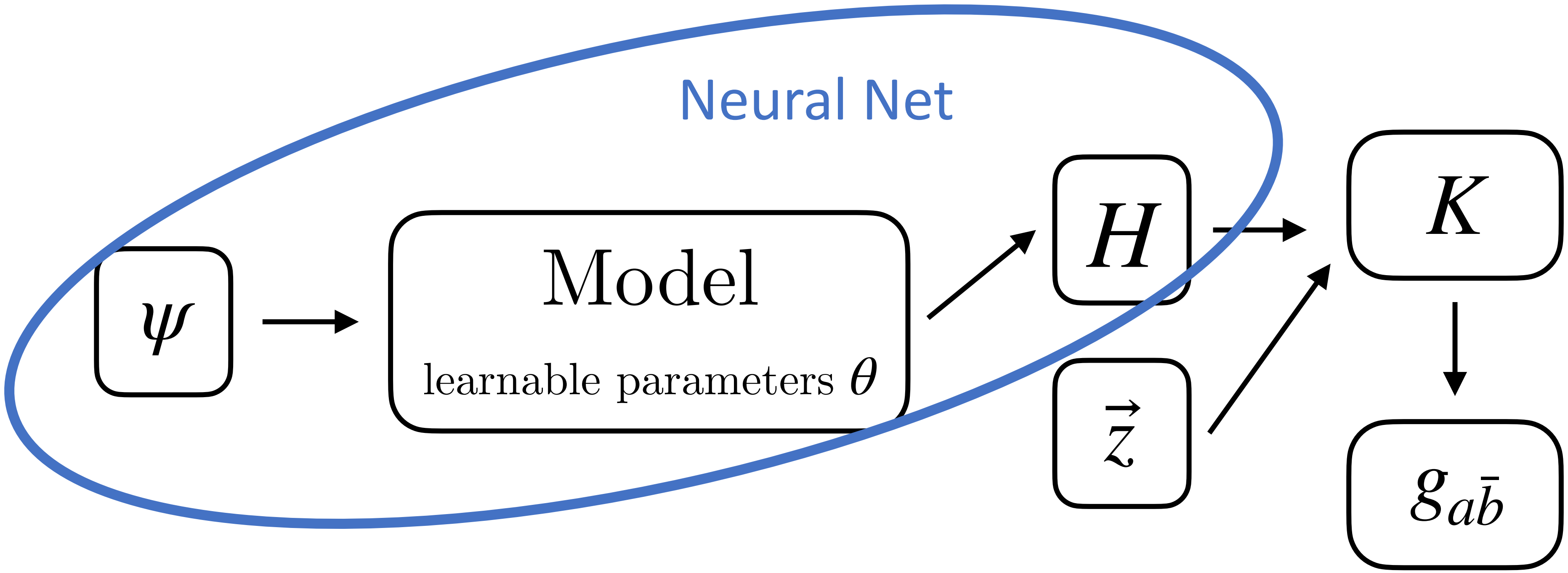
Pro

- Always learn 9 comps of 3*3 Hermitian metric
- Generalizes (e.g. non-Kähler SH metric)

Con

- Not Kähler
- Not globally defined

1. Learn Donaldson’s H matrix



1. Learn Donaldson's H matrix

Donaldson's algorithm:

Iterate T-operator until get balanced H

Compute Kähler potential

$$K = \frac{1}{2\pi k} \ln\Big(s_\alpha H_{\alpha\bar{\beta}} s_{\bar{\beta}}\Big)$$

- s_α monomials of order k (sections of holomorphic line bundle)
- H: $N_k \times N_k$ Hermitian matrix, "balanced metric"
- Larger k gives larger set of s_α → more accurate K
- Problem: Curse of dimensionality, need to use discrete symmetries

1. Learn Donaldson's H matrix

Donaldson's algorithm: algebraic K from H

$$K = \frac{1}{2\pi k} \ln\left(s_\alpha H_{\alpha\bar{\beta}} s_{\bar{\beta}}\right)$$

NN that predicts H

- Input layer: complex structure moduli
- Output layer: H matrix
- Predicted H + s_α at points → K in spectral basis → algebraic metric
- Either supervised learning
- or semi-supervised learning with MA/Ricci loss function

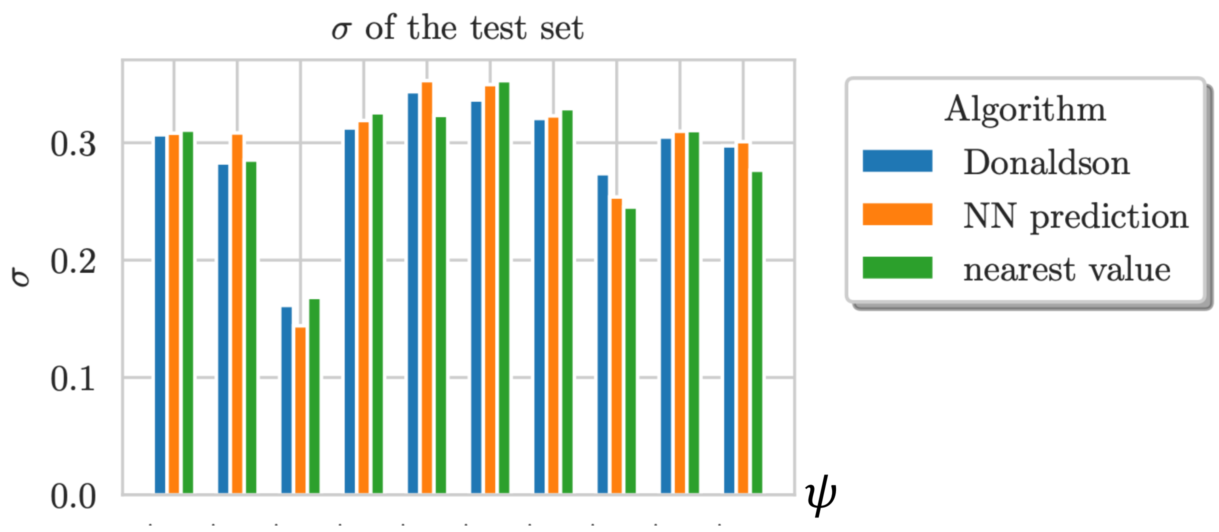
Anderson et al 2012.04656, Gerdes et al 2211.12520, cyjax

Example: supervised learning of H

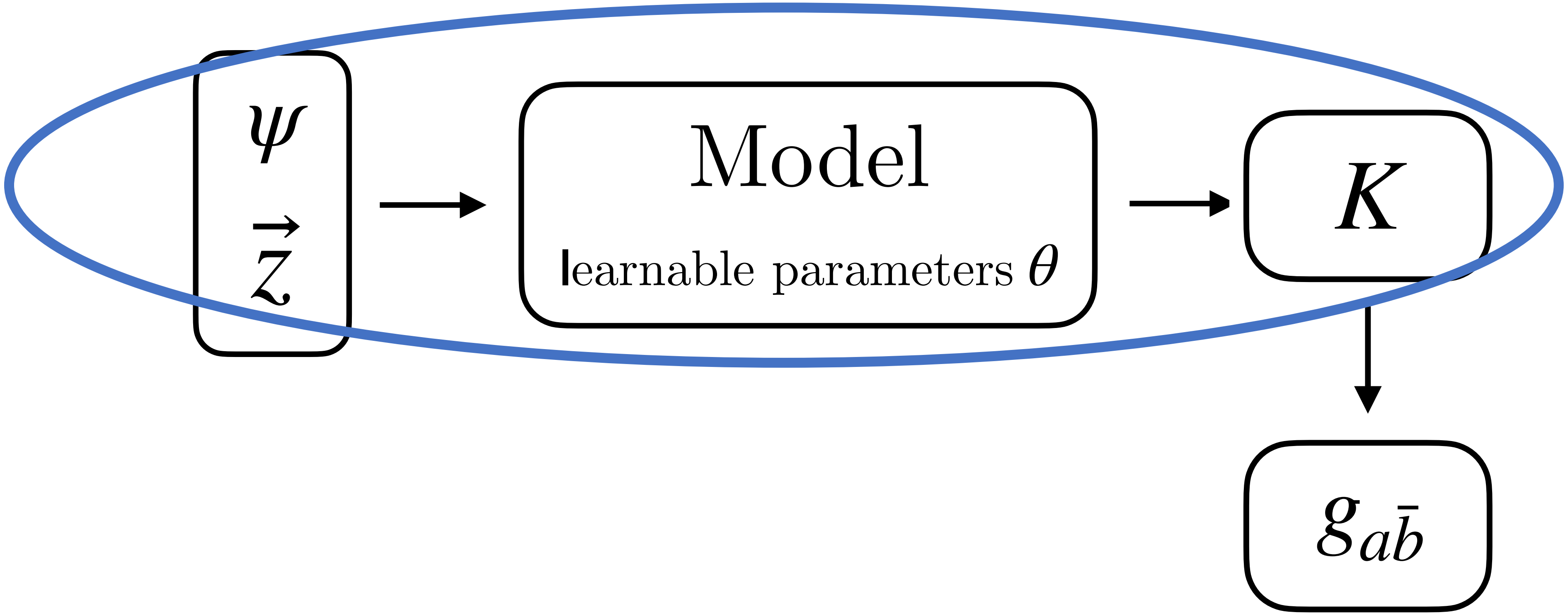
Anderson et al 2012.04656

- $p = x_0^5 + x_1^5 + x_2^5 + x_3^5 + x_4^5 - \psi x_0 x_1 x_2 x_3 x_4$
- $k = 3 \;\rightarrow$ 35-dim basis of sections s_α
- Input Re ψ, Im ψ, Abs ψ
- Output Re, Im of H components; compare with Donaldson
- FF NN, LSE loss function, ADAM opt.

Layer	Number of Nodes	Activation	Number of Parameters
input	3	–	–
hidden 1	100	leaky ReLU	400
hidden 2	1000	leaky ReLU	101 000
hidden 3	1000	leaky ReLU	1 001 000
output	N_k^2	identity	$1000 \times N_k^2 + N_k^2$



2. Learn Kähler potential directly



2. Learn Kähler potential directly

Douglas et al 2012.04797, holomorphic and bihomogeneous NN

- Input: points on CY
- Output: prediction for K

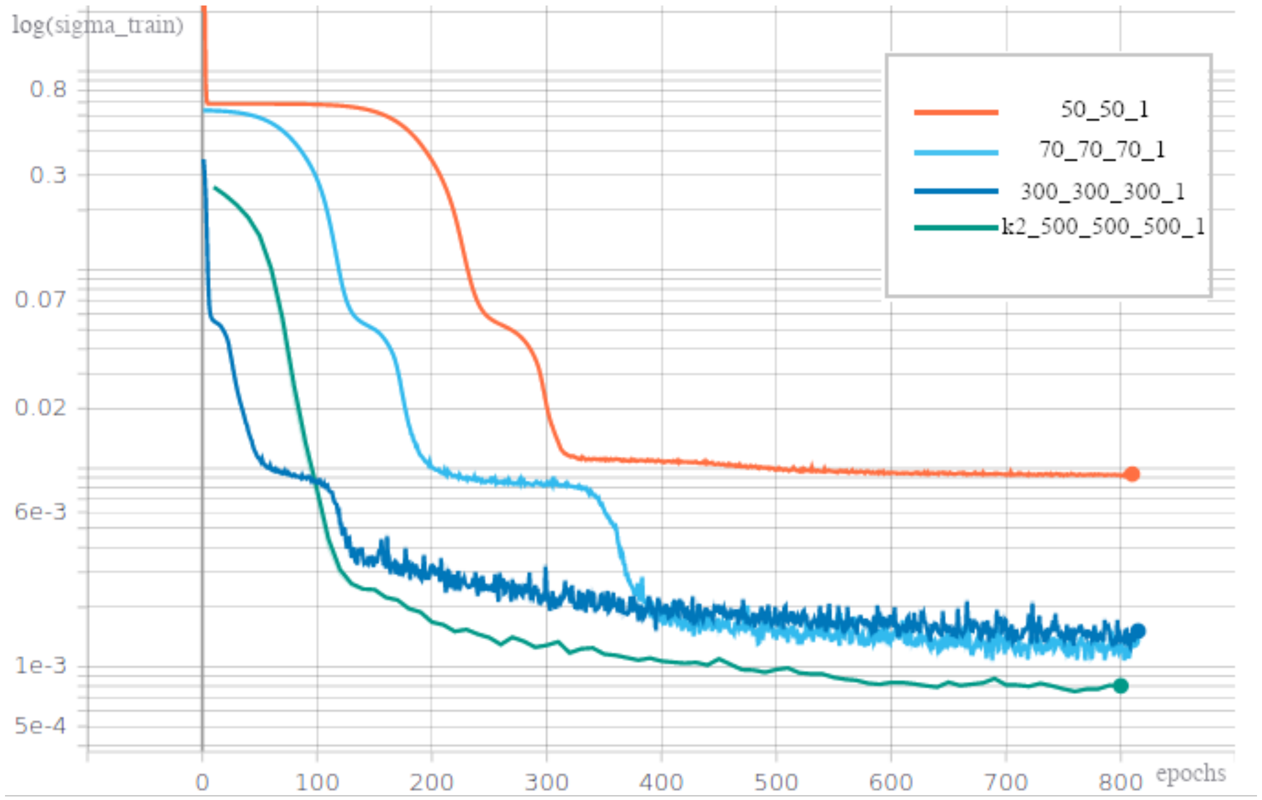
- Must ensure K is globally defined
  Guaranteed if expand in section basis (Donaldson, Headrick-Nassar)
  Or construct embedding NN (holomorphic or bihomogeneous)
- Bihomogeneous NN:
  Input $x_a \to x_a \overline{x_b} \to Re, Im$; Act. fcn: $\sigma: x \to x^2$
- $K = \log W^d \circ \sigma \circ \cdots \circ \sigma \circ W^1(x_a \overline{x_b})$

Example: semisupervised learning of K

- Semi-supervised learning
- MAPE version of MA loss
- After training:
  NN → K → approximate CY metric

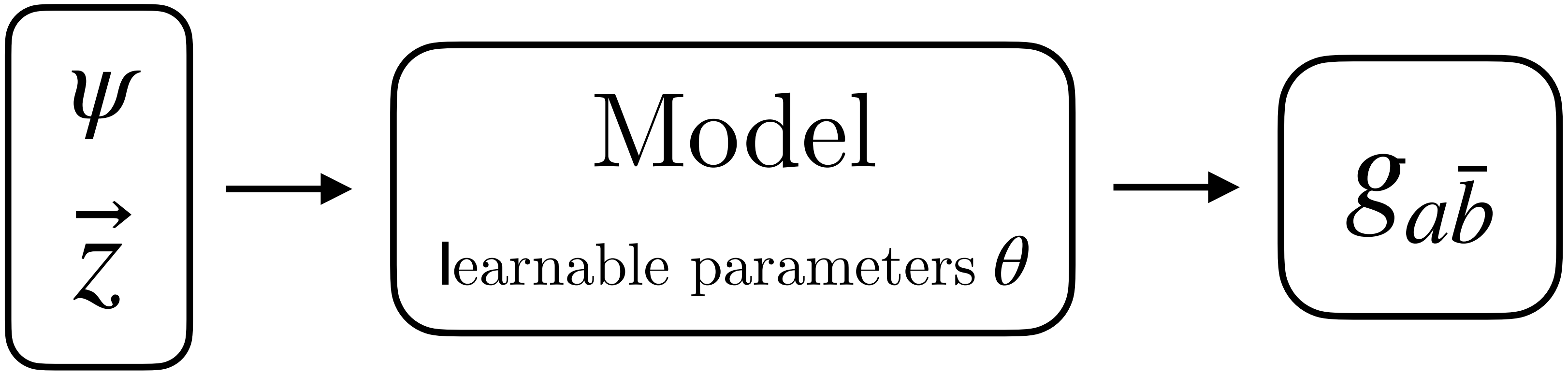
- Also non-symmetric quintics
- Gradient blow-ups/deep NN

Douglas et al 2012.04797



: The training curves for Equation (3) with $\psi = 0.5$, trained with Adam optimizer and MAPE loss. The data for k2_500_500_500_1 was recorded every 10 epochs.

3. Direct ML of metric



3. Direct ML of metric: neural network

- Input: point on CY
  Quintic: input layer has 10 nodes = $Re(x_I), Im(x_I)$
- Output: metric prediction - different Ansatze possible
  9 (or 1) node – no scaling
- Semi-supervised learning using custom loss function
- After training:
  NN → approximate CY metric

Anderson et al 2012.04656, Larfors et al 2205.13408, cymetric

3. Direct ML of metric: neural network

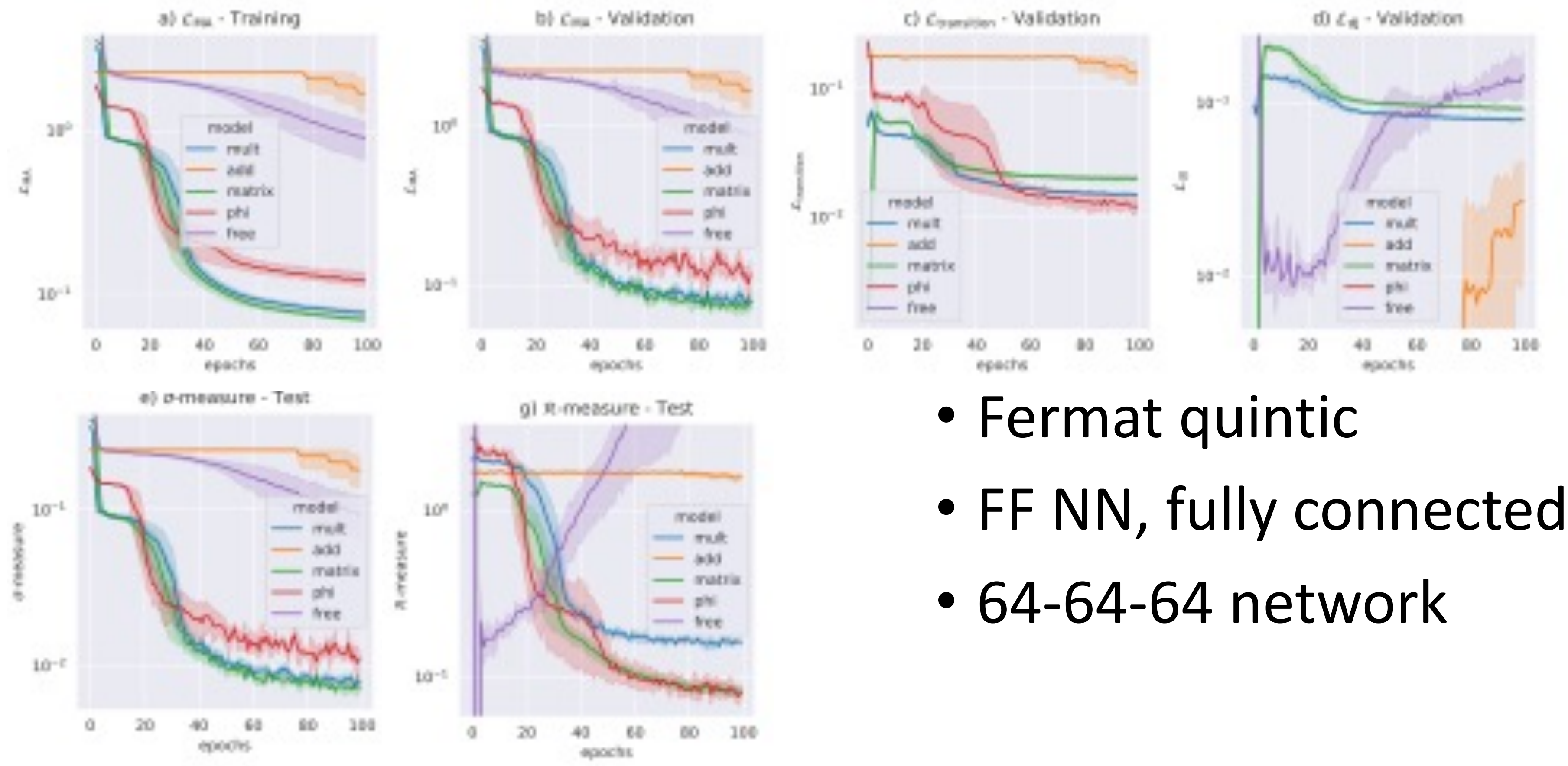
- Different Ansatze possible for metric prediction g_{pr}
  Encode more/less of math knowledge
- In the cymetric package, can choose between

Name	Ansatz
Free	$g_{\text{pr}} = g_{\text{NN}}$
Additive	$g_{\text{pr}} = g_{\text{FS}} + g_{\text{NN}}$
Multiplicative, element-wise	$g_{\text{pr}} = g_{\text{FS}} + g_{\text{FS}} \odot g_{\text{NN}}$
Multiplicative, matrix	$g_{\text{pr}} = g_{\text{FS}} + g_{\text{FS}} \cdot g_{\text{NN}}$
ϕ-model	$g_{\text{pr}} = g_{\text{FS}} + \partial\bar{\partial}\phi$

On quintic, same as learning K

Example: direct learning of g

[Larfors et al 2205.13408]



- Fermat quintic
- FF NN, fully connected, GELU
- 64-64-64 network

Summary of this lecture

- ML improves speed, performance, and scope for CY metric approx's
- Architecture determined by what you want to learn
- Loss functions encode math/physics constraints
- PyTorch, TensorFlow, JAX: ML libraries, efficient auto-differentiation
- Open-source packages on GitHub