Using deep learning for heuristic detection of elliptic curves having high rank

Matija Kazalicki[1], Domagoj Vlah[2]

[1]University of Zagreb, Faculty of Science, Department of Mathematics
[2]University of Zagreb, Faculty of Electrical Engineering and Computing, Department of Applied Mathematics

This research was supported by Croatian Science Foundation (HRZZ) Grant PZS-2019-02-3055 from "Research Cooperability" program funded by the European Social Fund.

Elliptic curve

What is a rank of an elliptic curve?

- elliptic curve E over $\mathbb{Q}$: $y^2 = x^3 + ax + b, \quad a, b \in \mathbb{Q}$
- E has conductor N
- $E(\mathbb{Q})$ is group of rational points on E
- Mordell's Theorem: $E(\mathbb{Q})$ is a finitely generated abelian group isomorphic to $E(\mathbb{Q})_{tors} \times \mathbb{Z}^r$
  - $E(\mathbb{Q})_{tors}$ is the torsion subgroup of $E(\mathbb{Q})$ (15 possibilities)
  - r is the rank of $E(\mathbb{Q})$
- rank r of $E(\mathbb{Q})$ is called the rank of E

- Rank is expensive and hard to compute. High rank curves are very rare.
- It is not known what values can rank attain. (Unbounded or not?)
- It is of importance to find curves of high rank. Current record is 28 (Elkies, 2006)

L-function attached to $E/\mathbb{Q}$

- E can be written as a minimal equation:

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \qquad a_1, a_2, a_3, a_4, a_6 \in \mathbb{Z}$$

- reducing the coefficients modulo prime p defines an elliptic curve $E(\mathbb{F}_p)$ over the finite field $\mathbb{F}_p$ (if $p \nmid N$)
- $\#E(\mathbb{F}_p)$ is the number of points on $E(\mathbb{F}_p)$
- If $p \nmid N$ (good reduction) we define $a_p(E) = p + 1 - \#E(\mathbb{F}_p)$
- If $p|N$, we set $a_p(E) = 0, -1$, or 1 if, respectively, E has additive, split multiplicative or non-split multiplicative reduction at p
- L-function attached to $E/\mathbb{Q}$:

$$L_E(s) = \prod_{p|N} \left(1 - \frac{a_p(E)}{p^s}\right)^{-1} \prod_{p\nmid N} \left(1 - \frac{a_p(E)}{p^s} + \frac{p}{p^{2s}}\right)^{-1},$$

which converges absolutely for $\Re(s) > 3/2$ and extends to an entire function by the Modularity theorem (Wiles, 1995; Breuil et al., 2001)

Mestre-Nagao-like sums

Birch and Swinnerton-Dyer (BSD) conjecture

The order of vanishing of $L_E(s)$ at $s = 1$ (analytic rank) is equal to the rank of $E(\mathbb{Q})$.

- BSD conjecture motivated construction of certain Mestre-Nagao-like sums, e.g.:

$$S_0(B) = \frac{1}{\log B} \sum_{\substack{p<B, \\ \text{good reduction}}} \frac{a_p(E)\log p}{p},$$

$$S_3(B) = \sum_{\substack{p<B, \\ \text{good reduction}}} \frac{-a_p(E)+2}{p+1-a_p(E)} \log p,$$

$$S_5(B) = \sum_{\substack{p<B, \\ \text{good reduction}}} \log\left(\frac{p+1-a_p(E)}{p}\right) + \sum_{\substack{p<B, \\ \text{split mult. reduction}}} \log\left(3/2 \cdot \frac{p-1}{p}\right)$$

- $\lim_{B\to\infty} S_i(B)$ is connected to rank of E (e.g. $\lim_{B\to\infty} S_0(B) = -r + 1/2$)

Rank heuristics

How to heuristicaly determine rank of curve E?

- $\lim_{B\to\infty} S_i(B)$ is numerically approximated by a finite sum for some large B and from a finite sequence of computed $a_p(E)$-s.
- It is much faster to compute $a_p(E)$-s than rank directly!
- Problem: How to determine sum thresholds for different ranks?
- Notice that all sums are in general of the form

$$S_i(B) = \sum_{p<B} \mathscr{F}_i(a_p(E), p, N_E),$$

where $\mathscr{F}_i$ is some nonlinear function.

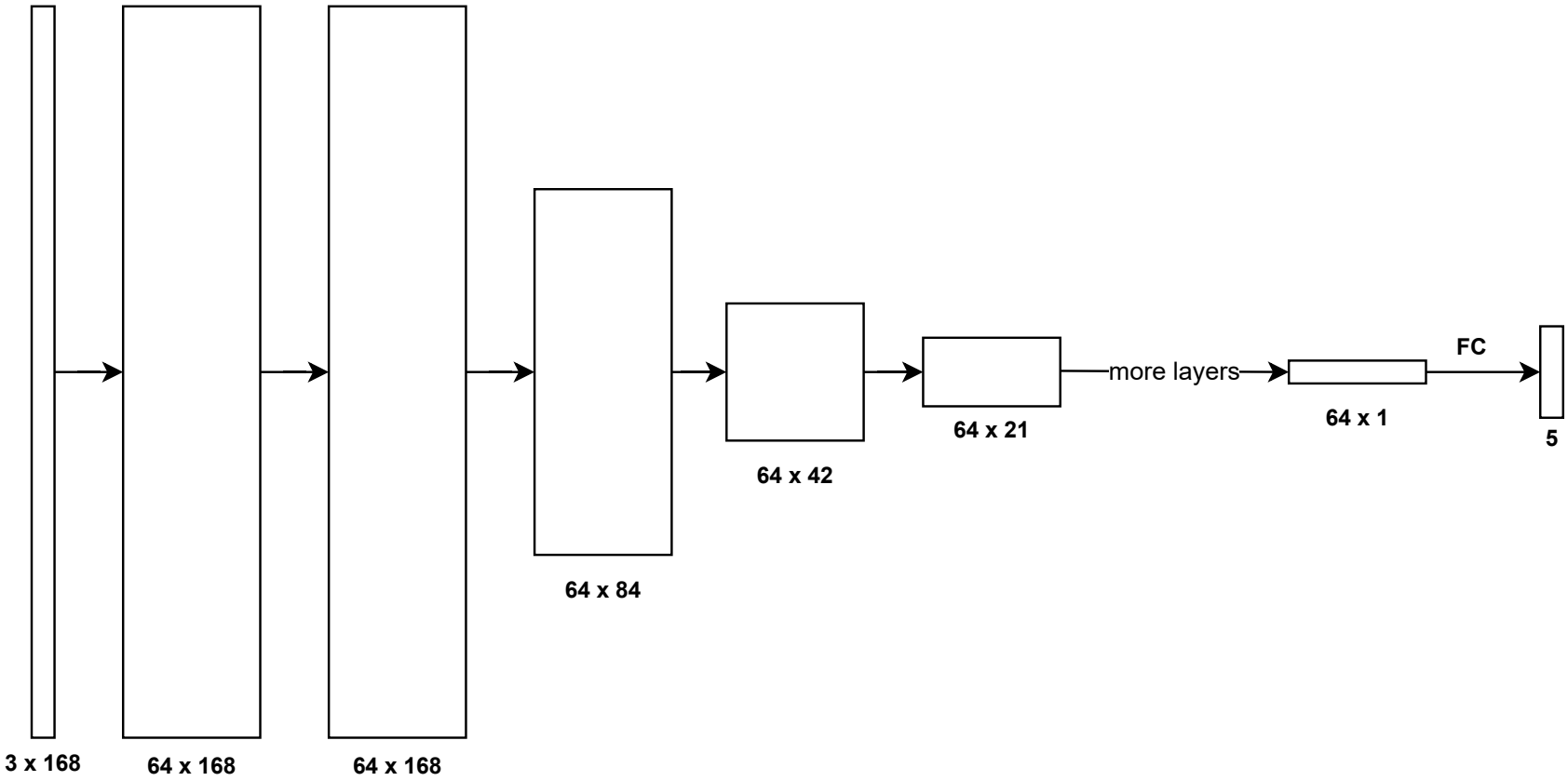
- What if we could instead of manual crafting $\mathscr{F}$, find optimal $\mathscr{F}$ in some large space of possible nonlinear functions?
- Even better, for a fixed B, we want to find rank classifier function directly, depending on the sequence of $a_p(E)$-s and N_E - use deep learning!

Modeling rank classifier using deep neural network

- architecture: a sequence of convolutional neural network layers + fully connected classification layer in the end
- activation function: ReLU - pointwise $f(x) := \max(x, 0)$
- loss function:
  - weighted cross entropy loss
  - weights reflect relative size of classes - different ranks
- gradient descent optimizer: Adam (a variant of SGD)
- autograd using Pytorch library
- dataset - for a number of elliptic curves we computed pairs of:
  - input: a sequence of a_p-s and conductor
  - target: exact rank computed using PARI/GP function *ellrank*
- input normalization
- train / validation / test set split
- quality of classification was decided using Matthews correlation coefficient (or phi coefficient)

Architecture of convolutional neural network

Example: $B = 1000$ - first 168 primes, kernel size 33, cca. $773,000$ parameters



Loss function

- weighted cross entropy loss
- a combination of *LogSoftmax* function and negative log likelihood loss
- given by:

$$l(x,y) = \frac{1}{N}\sum_{n=1}^{N}\left(-\sum_{c=1}^{C} w_c \log \frac{\exp(x_{n,c})}{\sum_{i=1}^{C}\exp(x_{n,i})} y_{n,c}\right)$$

  - x - input (scores)
  - y - target (exact classes: 0 or 1)
  - w - weight
  - C - number of classes
  - N - minibatch size
- averaged over minibatch
- weights are computed to reflect the relative size of different classes:

$$w_c = \frac{\text{number of all dataset elements (curves)}}{\text{number of elements (curves) of class (rank) c}}$$

Datasets

- we constructed 12 different datasets by varying:
  a) used curves:
     - LMFDB curves: cca. 3 milion curves, conductors $< 300,000,000$, ranks $<= 4$
     - custom curves: cca. 2 milion curves, conductor $< 10^{30}$, ranks $<= 10$
  b) range of a_p-s considered: primes $p < 10^3, 10^4$, and 10^5
  c) selection of test curves:
     - uniformly selected (20% from the dataset)
     - all curves in top conductor range (which is $[10^8, 10^9]$ for LMFDB and $[10^{29}, 10^{30}]$ for custom dataset)
- for each curve E in each dataset we computed:
  - conductor N_E
  - sequence of $a_p(E)$-s for a given range of primes
  - exact rank - using PARI/GP function *ellrank*
- for each dataset we trained:
  - CNN classifier
  - for each of 7 considered Mestre-Nagao-like sums $S_0 \ldots S_6$, a simple baseline FCNN classifier (4 hidden layers of 128 neurons each, input is S_i and N_E)
  - simple FCNN classifier Ω involving all 7 sums and N_E

Matthews correlation coefficient

- balanced measure of classification quality
  - even for classes of very different sizes
- e.g. for binary classification, MCC is computed using:

$$\mathrm{MCC} = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP+FP)\cdot(TP+FN)\cdot(TN+EP)\cdot(TN+FN)}}$$

  - TP - number of true positives
  - FN - number of false negatives
  - TN - number of true negatives
  - FP - number of false positives
- MCC takes into account both false positives and false negatives
- MCC generalizes to more than 2 classes
- MCC lies in the segment $[-1,1]$
- $\mathrm{MCC} = 1$ only in the case of perfect classification

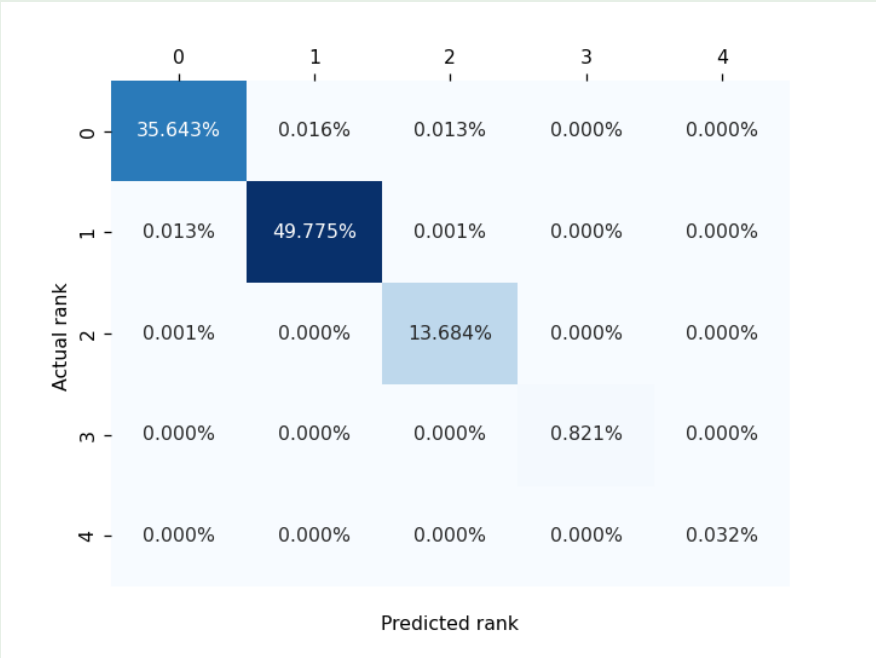
Results

Comparison of different classifiers for LMFDB dataset with uniform test set

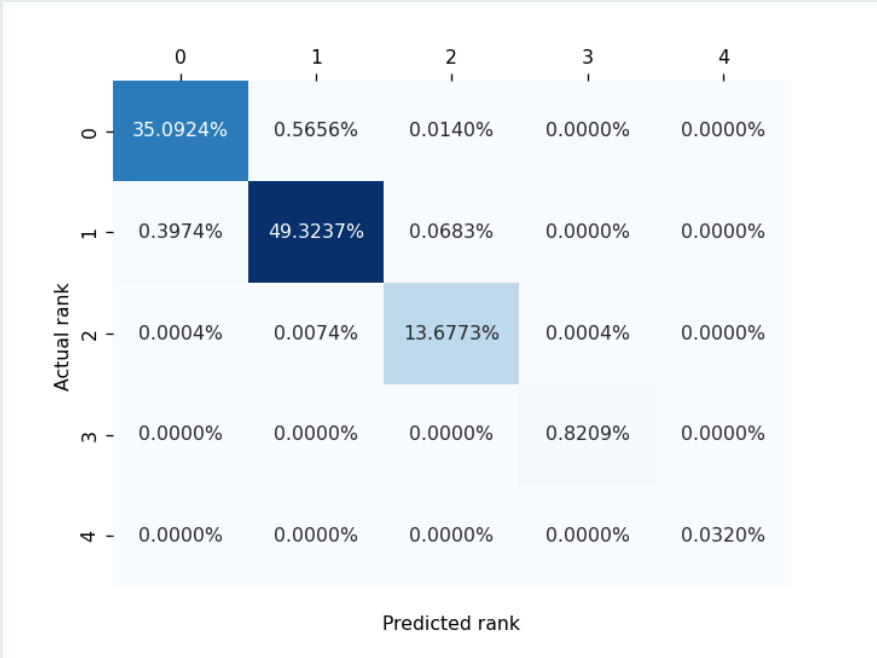
Type of classifier	Number of a_p-s used		
	$p < 10^3$	$p < 10^4$	$p < 10^5$
CNN	**0.9507**	**0.9958**	**0.9992**
S_0	0.6823	0.8435	0.9068
S_1	0.6848	0.8507	0.9301
S_2	0.7277	**0.8697**	0.9359
S_3	0.6933	0.8499	0.9142
S_4	0.2678	0.3015	0.1525
S_5	0.6132	0.7774	0.8463
S_6	**0.6969**	0.8647	**0.9381**
Ω	**0.8685**	**0.9602**	**0.9826**

Results

Confusion matrices of CNN and Ω for $p < 10^5$ and uniform test dataset



CNN MCC $= 0.9992$



Ω MCC $= 0.9826$

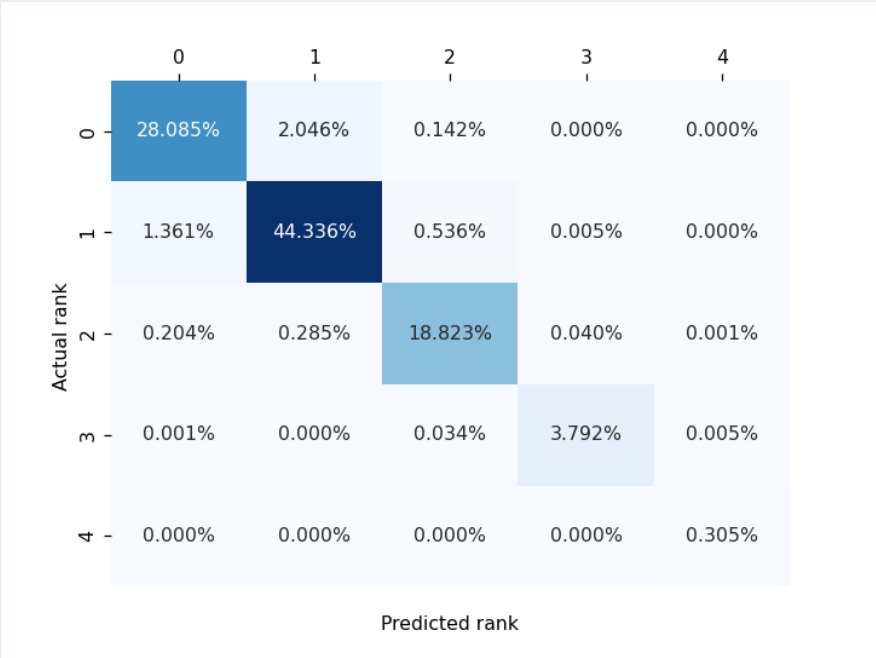
Results

Comparison of different classifiers for LMFDB dataset with top conductor range test set

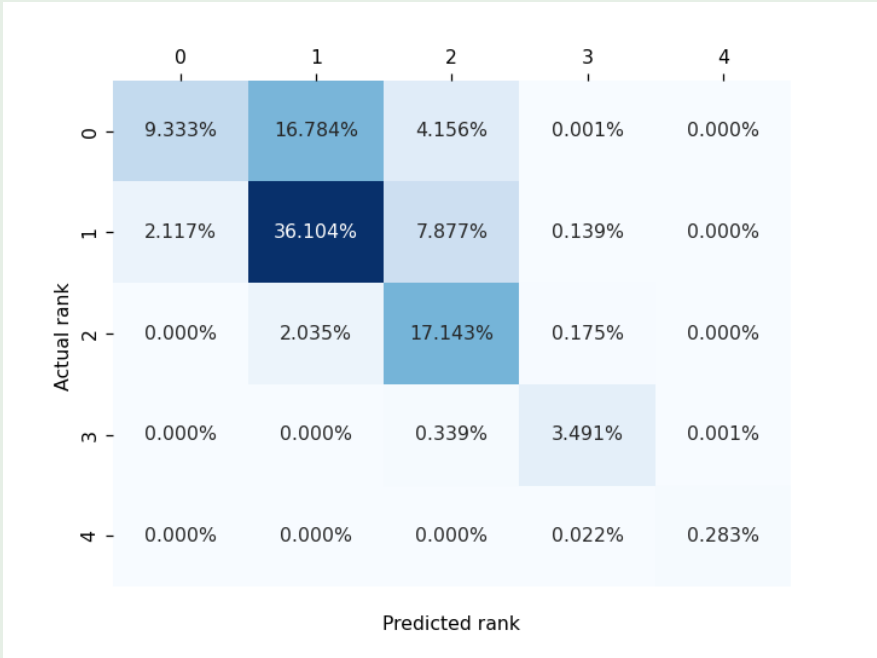
Type of classifier	Number of a_p-s used		
	$p < 10^3$	$p < 10^4$	$p < 10^5$
CNN	**0.5669**	**0.9289**	**0.9846**
S_0	0.2880	**0.5057**	0.6545
S_1	0.2791	0.4883	0.6658
S_2	0.2790	0.4968	**0.6730**
S_3	0.2897	0.5030	0.6574
S_4	0.1352	0.1424	0.1850
S_5	**0.2960**	0.3913	0.5261
S_6	0.2632	0.4542	0.6416
Ω	**0.4433**	**0.7013**	**0.8530**

Results

Confusion matrices of CNN and S_0 for $p < 10^4$ and top conductor range test dataset



CNN MCC $= 0.9289$



S_0 MCC $= 0.5057$

Results

Comparison of different classifiers for custom dataset with uniform test set

Type of classifier	Number of a_p-s used		
	$p < 10^3$	$p < 10^4$	$p < 10^5$
CNN	**0.6129**	**0.7218**	**0.7958**
S_0	0.5738	0.6782	0.7462
S_1	**0.5780**	**0.6890**	**0.7592**
S_2	0.5649	0.6761	0.7521
S_3	0.5551	0.6616	0.7361
S_4	0.2893	0.2472	0.2251
S_5	0.4987	0.5990	0.6696
S_6	0.5230	0.6509	0.7361
Ω	**0.5999**	**0.7069**	**0.7807**

Results

Comparison of different classifiers for custom dataset with top conductor range test set

Type of classifier	Number of a_p-s used		
	$p < 10^3$	$p < 10^4$	$p < 10^5$
CNN	0.2147	0.3019	0.3655
S_0	0.2533	**0.3233**	**0.3719**
S_1	**0.2573**	**0.3291**	**0.3834**
S_2	0.2340	0.3118	0.3688
S_3	**0.2556**	0.3189	0.3645
S_4	0.1234	0.1228	0.1024
S_5	0.2081	0.2858	0.3380
S_6	0.1803	0.2757	0.3527
Ω	**0.2622**	**0.3246**	**0.3905**

Future work

- M. Kazalicki, D. Vlah, *Ranks of elliptic curves and deep neural networks*, arXiv:2207.06699
- Find optimal CNN model architecture hyperparameters and optimal optimizer hyperparameters
- Try some more advanced neural architectures:
  - recently developed NLP models: Transformer, Perceiver, ...
  - multitask learning: autoencoder + classifier head
  - recent developments in variational auto-encoders: IWAE - smaller dataset - bigger models
- Train on more datasets with different conductor ranges and different number of a_p-s used
- Use deep neural network based classifiers to find record breaking elliptic curves of high rank
- Congruent number curves

Application of deep learning

- Well established ML technique in: computer vision, natural language processing, speech recognition, machine translation, board game programs, ...
- Find previously unexploited or underexploited areas in science (mathematics)
- Generalize known techniques and models to these areas
- Problem meta-modeling - reducing the need for expert knowledge
- I previously did or currently doing applications in:
  - Statistics (extreme value analysis): MIWAE
  - Bioinformatics (protein classification): NLP models - Transformers
  - Bilevel optimization (improved numerical efficiency): deep convolutions
  - Number theory (eliptic curves over $\mathbb{Q}$) - THIS WORK: deep convolutions
  - Numerical methods for PDE-s (PINN): ???
  - ...

Which problems in mathematics are good for DL?

- For objects that can have meaningful numerical representations (real or complex vectors, matrices or tensors)
- Finding rare objects where exact search is too expensive
- Classification problems (even discovering unknown classes)
- Finding counterexamples to conjectures in extremal combinatorics and graph theory
- Finding better heuristics (meta-heuristics)
- Inverse problems
- Numerical approximation problems - natural application