Exploring patterns in number theory with deep learning

a case study with $\mu(n)$ and $\mu^2(n)$

David Lowry-Duda

October 2024

CMSA Program in ML and Mathematics

The Möbius function $\mu(n)$ is defined by

$$\mu(n) = \begin{cases} 0 & \text{if } p^2 \mid n \text{ for some } p \\ (-1)^d & \text{if } n \text{ is a product of } d \text{ distinct primes.} \end{cases}$$

For example, $\mu(2) = \mu(3) = \mu(5) = -1$, and $\mu(6) = \mu(10) = \mu(15) = 1$, and $\mu(4) = \mu(9) = \mu(12) = 0$. The Möbius function is fundamental in number theory. It appears in many places, but one is that the reciprocal of the Riemann zeta function $\zeta(s) = \sum_{n \geq 1} \frac{1}{n^s}$ is

$$\frac{1}{\zeta(s)} = \sum_{n \geq 1} \frac{\mu(n)}{n^s}.$$

It encodes data about prime numbers and prime factorization.

Despite being fundamental, understanding $\mu(n)$ is extraordinarily hard. The fact that

$$\sum_{n \leq X} \mu(n) = o(X)$$

is equivalent to the prime number theorem (counting the number of primes up to X).

More generally, it is conjectured that $\mu(n)$ is orthogonal to any function f that "isn't too complex", or rather that

$$\left| \sum_{n \leq X} \mu(n) f(n) \right| = o\left(\left| \sum_{n \leq X} f(n) \right| \right).$$

Observe that $\mu^2(n)$ is 0 if n is divisible by a (nontrivial) square, and otherwise is 1. It's the squarefree indicator function.

Though it feels simple, in practice it's also *extremely difficult* to compute exactly.

What should we expect?

About 60% of integers are squarefree, and squarefree integers have approximately equal probabilities of having $\mu(n) = 1$ or $\mu(n) = -1$. We're looking for improvements over guessing $\mu(n) = 0$ (true 40% of the time) or $\mu^2(n) = 0$ (true 60% of the time).

It's easy to write down a silly algorithm: determine if 4 or 9 or 25 (and so on) divides n. If so, output 0. Otherwise guess ± 1 randomly. (This already classifies $\mu^2(n)$ for 90% of integers n).

A generic neural network on input-output pairs $(n, \mu^2(n))$ slowly learns to predict the silly algorithm above. To paraphrase Jordan Ellenberg (who gave a talk that touched on this earlier this program), this leads to an algorithm with nearly 100% accuracy and nearly 0% understanding.

Guiding Question

Can we learn something about $\mu(n)$ and $\mu^2(n)$ beyond testing divisibility by squares of small prime numbers?

Guiding Question

Can we learn something about $\mu(n)$ and $\mu^2(n)$ beyond testing divisibility by squares of small prime numbers?

And to do this, I chose to use François Charton's Int2Int small transformer model (`https://github.com/f-charton/Int2Int`).

(I've also run each of the experiments I describe below on neural networks. In practice, fully trained models had similar performance as Int2Int models, but took longer to train).

Experimental Settings

More precisely, we think of this as a supervised translation task. Given an integer n, we choose a representation of n as a sequence of integer tokens, and train a sequence-to-sequence transformer model to minimize the cross-entropy with model predictions.

The transformers have 4 layers, 512 dimensions, and 8 attention heads. We used Adam as our optimizer.

The inputs for all initial experiments came from 10^7 integers n sampled uniformly (without repetition) from $(1, 10^{13})$.

(All credit here goes to Charton, who made Int2Int to be extremely easy to use; and to Edgar Costa, who helped make it even easier).

Initial Experiments

With the settings above, we need to choose how to encode n. Simply encoding n directly (which in Int2Int means representing it as a base B integer) leads to estimating the is-divisible-by-square-of-small-prime function.

After trying several possibilities, I found that using a Chinese Remainder Theorem type representation led to different behavior.

I initially trained four models:

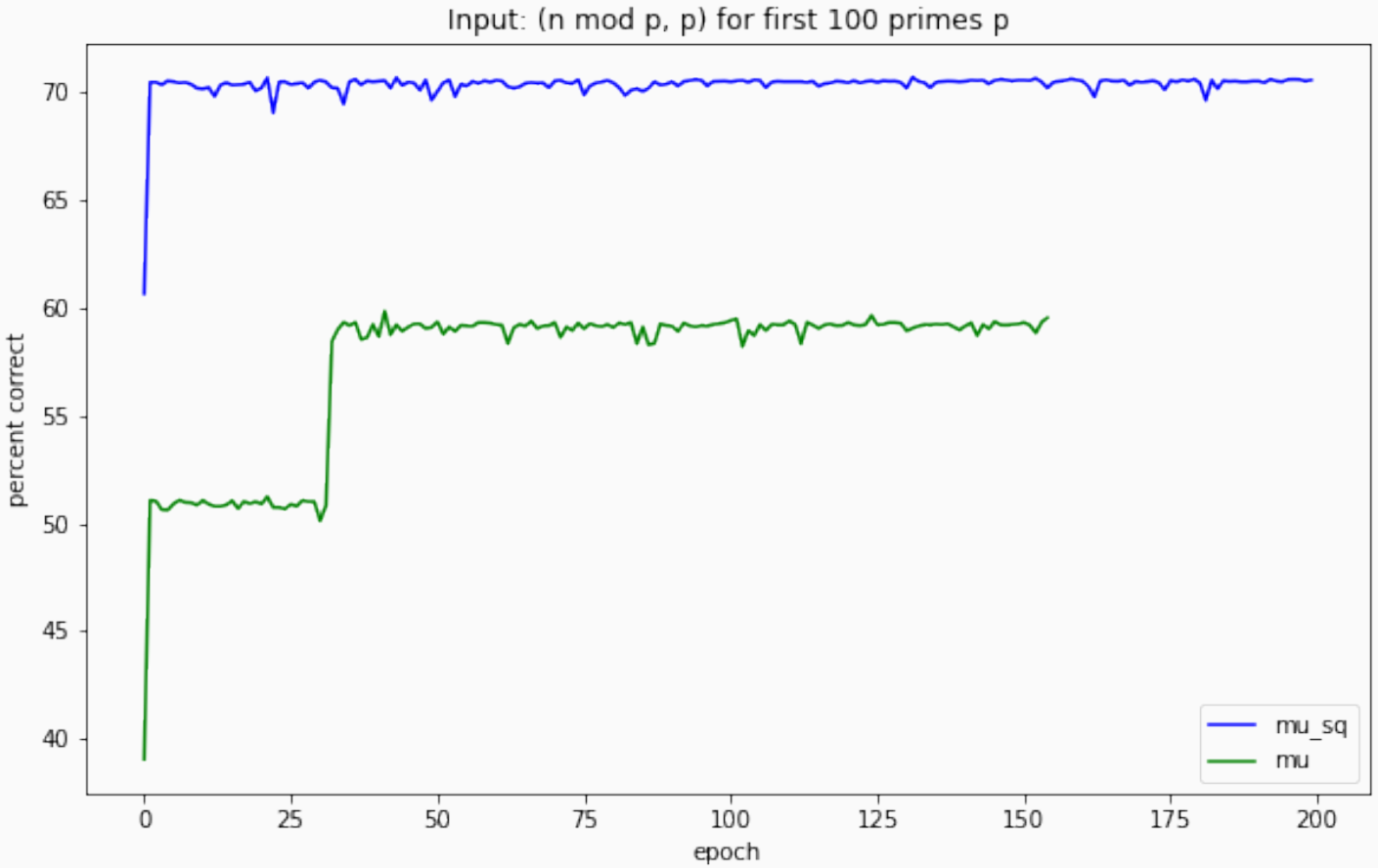
Initial Experiments

With the settings above, we need to choose how to encode n. Simply encoding n directly (which in Int2Int means representing it as a base B integer) leads to estimating the is-divisible-by-square-of-small-prime function.

After trying several possibilities, I found that using a Chinese Remainder Theorem type representation led to different behavior.

I initially trained four models:

Input	Output
Pairs $(n \bmod p, p)$ for the first 100 primes	$\mu(n)$
Pairs $(n \bmod p, p)$ for the first 100 primes	$\mu^2(n)$
Triples $(n \bmod p, \chi_p(n), p)$ for the first 100 primes	$\mu(n)$
Triples $(n \bmod p, \chi_p(n), p)$ for the first 100 primes	$\mu^2(n)$



1. Vastly better than chance.
2. I don't show it, but including $\chi_p(n)$ gives similar behavior.

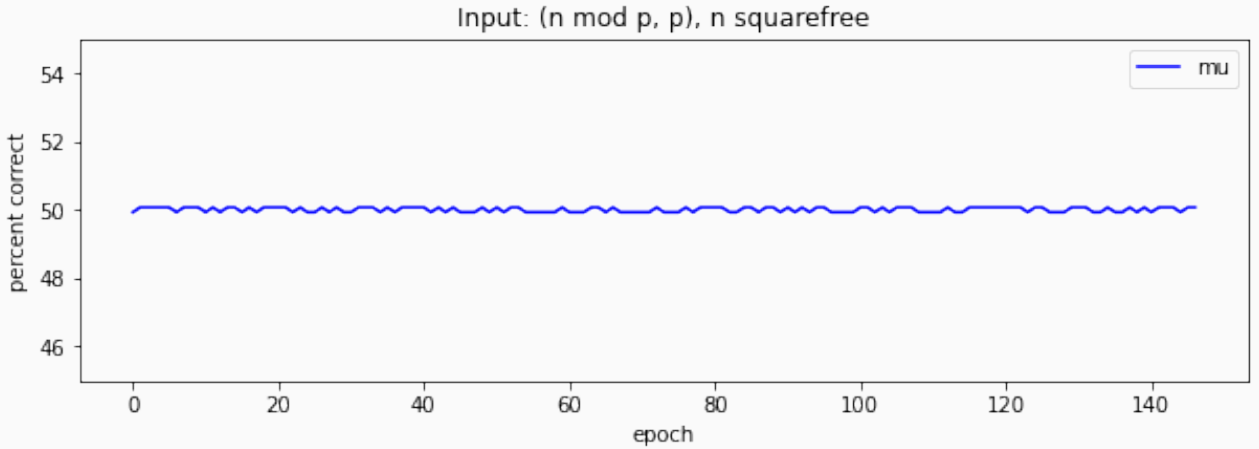
Let's look at one of the successful $\mu(n)$ prediction tables.

Prediction	# Correct	# in Eval Set	% Correctly Predicted
0	3646	3924	92.92
1	1193	3026	39.42
-1	1116	3050	36.59

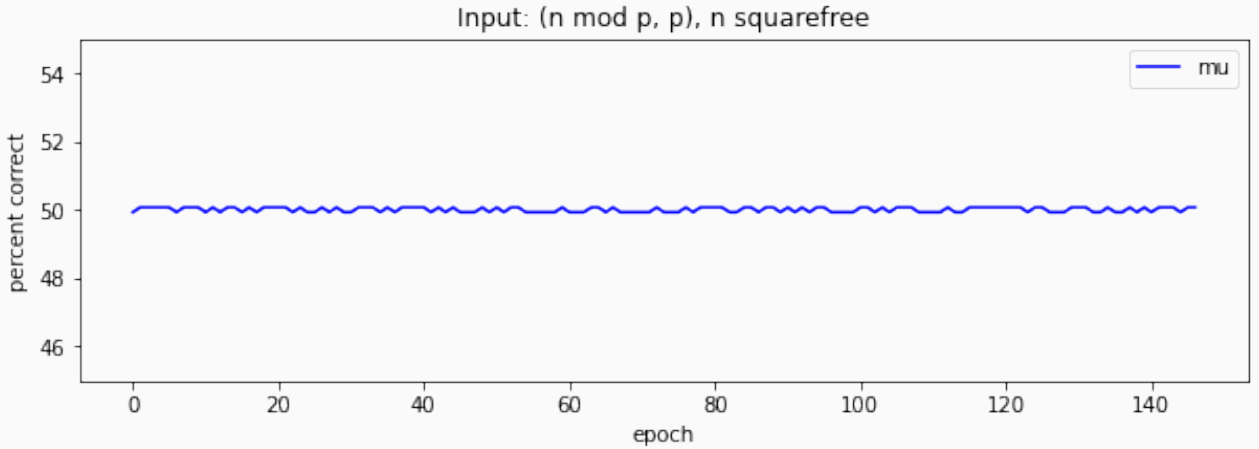
Examining its predictions on the evaluation set closer, one sees that the model was pretty good at determining when $\mu(n) = 0$. Otherwise it guessed randomly between 1 and -1. (Sometimes it would pick $\mu(n) = -1$ always, or $+1$ always.)

This broad form appeared with each of the input formats. One way to force attention here is to restrict inputs to squarefree n. Training a model with $(n \bmod p, \chi_p(n), p)$ inputs and restricting training and evaluation data to squarefree n gives

This broad form appeared with each of the input formats. One way to force attention here is to restrict inputs to squarefree n. Training a model with $(n \bmod p, \chi_p(n), p)$ inputs and restricting training and evaluation data to squarefree n gives



This broad form appeared with each of the input formats. One way to force attention here is to restrict inputs to squarefree n. Training a model with $(n \bmod p, \chi_p(n), p)$ inputs and restricting training and evaluation data to squarefree n gives



Möbius Challenge

Train a model with inputs that are "easy to compute" (say, computable in time $\ll \log^A(n)$) and that distinguishes between cases $\mu(n) = 1$ and $\mu(n) = -1$ with probability greater than 50%.

Understanding model predictions

Can we figure out how the models compute $\mu^2(n)$ so well?

How would *I* compute $\mu^2(n)$ given n mod p for lots of primes? I would use the Chinese Remainder Theorem to recover n and then do something like trial division on n.

Understanding model predictions

Can we figure out how the models compute $\mu^2(n)$ so well?

How would *I* compute $\mu^2(n)$ given n mod p for lots of primes? I would use the Chinese Remainder Theorem to recover n and then do something like trial division on n.

Of course, knowing n mod p *isn't enough* to compute $\mu(n)$ actually. This specifies what n is modulo $\prod p$. We also need to know that (for example) $n, \prod p$.

Is the model learning the Chinese Remainder Theorem

Is the transformer model doing the same thing that I would do?

To figure this out, I set up a series of experiments to try to test this.

- First, let's test the Chinese Remainder Theorem directly. Feed in inputs ($n \bmod p$) for the first 10 primes; output n (guaranteed to be less than $\prod p$). Does this work?

Is the model learning the Chinese Remainder Theorem

Is the transformer model doing the same thing that I would do?

To figure this out, I set up a series of experiments to try to test this.

- First, let's test the Chinese Remainder Theorem directly. Feed in inputs ($n \bmod p$) for the first 10 primes; output n (guaranteed to be less than $\prod p$). Does this work? *No. But we shouldn't expect it to.*

Is the model learning the Chinese Remainder Theorem

Is the transformer model doing the same thing that I would do?

To figure this out, I set up a series of experiments to try to test this.

- First, let's test the Chinese Remainder Theorem directly. Feed in inputs ($n \bmod p$) for the first 10 primes; output n (guaranteed to be less than $\prod p$). Does this work? *No. But we shouldn't expect it to.*
- Maybe it's learning some aspects of the Chinese Remainder Theorem. So let's keep our inputs: triples $(n \bmod p, \chi_p(n), p)$. But now output the indicator function for the interval $[1, 5 \cdot 10^{12}]$. Does this work?

Is the model learning the Chinese Remainder Theorem

Is the transformer model doing the same thing that I would do?

To figure this out, I set up a series of experiments to try to test this.

- First, let's test the Chinese Remainder Theorem directly. Feed in inputs (n mod p) for the first 10 primes; output n (guaranteed to be less than $\prod p$). Does this work? *No. But we shouldn't expect it to.*
- Maybe it's learning some aspects of the Chinese Remainder Theorem. So let's keep our inputs: triples (n mod $p, \chi_p(n), p$). But now output the indicator function for the interval $[1, 5 \cdot 10^{12}]$. Does this work? *No!*
- Maybe it's learning p-adic data. Given (n mod $p, \chi_p(n), p$) triples, output n mod 4. Does this work?

Is the model learning the Chinese Remainder Theorem

Is the transformer model doing the same thing that I would do?

To figure this out, I set up a series of experiments to try to test this.

- First, let's test the Chinese Remainder Theorem directly. Feed in inputs (n mod p) for the first 10 primes; output n (guaranteed to be less than $\prod p$). Does this work? *No. But we shouldn't expect it to.*
- Maybe it's learning some aspects of the Chinese Remainder Theorem. So let's keep our inputs: triples (n mod $p, \chi_p(n), p$). But now output the indicator function for the interval $[1, 5 \cdot 10^{12}]$. Does this work? *No!*
- Maybe it's learning p-adic data. Given (n mod $p, \chi_p(n), p$) triples, output n mod 4. Does this work? *Also no!*
- Similarly, given (n mod $p, \chi_p(n), p$) for the first 100 primes *but skipping* 3, can it recover n mod 3? *No!*

Is the model learning the Chinese Remainder Theorem

Is the transformer model doing the same thing that I would do?

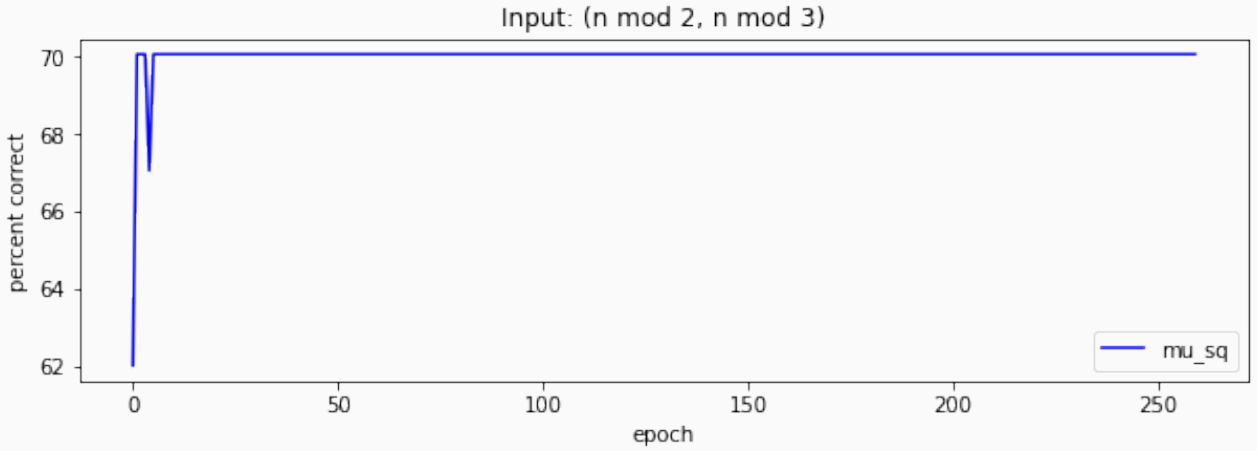
To figure this out, I set up a series of experiments to try to test this.

- First, let's test the Chinese Remainder Theorem directly. Feed in inputs (n mod p) for the first 10 primes; output n (guaranteed to be less than $\prod p$). Does this work? *No. But we shouldn't expect it to.*
- Maybe it's learning some aspects of the Chinese Remainder Theorem. So let's keep our inputs: triples (n mod $p, \chi_p(n), p$). But now output the indicator function for the interval $[1, 5 \cdot 10^{12}]$. Does this work? *No!*
- Maybe it's learning p-adic data. Given (n mod $p, \chi_p(n), p$) triples, output n mod 4. Does this work? *Also no!*
- Similarly, given (n mod $p, \chi_p(n), p$) for the first 100 primes *but skipping* 3, can it recover n mod 3? *No!*
- Given (n mod $p, \chi_p(n), p$) for the *second* 100 primes, can it still compute $\mu^2(n)$ with high probability? *No!*

The last experiment was the biggest clue. What inputs are actually being used?

The last experiment was the biggest clue. What inputs are actually being used?

Restricting to the first 2 primes:



(If you could look *very* closely, you would see that this doesn't do quite as well as the first 100 primes. . . but it's pretty close.)

This gives a concrete, purely mathematical claim. Knowing n mod 6 is enough to guess $\mu^2(n)$ with probability about 70%. (In fact, it was that this was a purely correlative result that threw me off!)

Explaining model predictions

Let's look at the slightly simpler case of just the prime 2. Let $P(\mathrm{sf})$ denote the probability that a random integer is squarefree; let $P(2)$ and $P(\widehat{2})$ denote the probability that a random integer is divisible by 2 or not divisible by 2, respectively.

Explaining model predictions

Let's look at the slightly simpler case of just the prime 2. Let $P(\mathrm{sf})$ denote the probability that a random integer is squarefree; let $P(2)$ and $P(\widehat{2})$ denote the probability that a random integer is divisible by 2 or not divisible by 2, respectively.

Basic probability shows that $P(\mathrm{sf}|2) = P(\mathrm{sf}, 2)/P(2)$. What is $P(\mathrm{sf}, 2)$? Let's look at each prime: an even squarefree number must be 2 mod 4, which happens with probability $1/4$. For each other prime p, we must have $n \not\equiv 0 \bmod p^2$, which happens with probability $(1 - p^{-2})$. Hence

$$\begin{aligned} P(\mathrm{sf}|2) = 2P(sf, 2) &= \frac{2}{4} \prod_{p \neq 2} \left(1 - \frac{1}{p^2}\right) = \frac{1}{2}\left(1 - \frac{1}{4}\right)^{-1} \prod_{p} \left(1 - \frac{1}{p^2}\right) \\ &= \frac{2}{3}\zeta(2)^{-1} \approx 0.4052\ldots \end{aligned}$$

Similarly $P(\mathrm{sf}|\widehat{2}) = \frac{4}{3}\zeta(2)^{-1} \approx 0.8105$.

How would this theoretical computation work? Given even n, predict $\mu^2(n) = 0$ (which is correct with probability 0.5948). Otherwise, say $\mu^2(n) = 1$ (correct with probability 0.8105). Each scenario occurs with probability 0.5. In total, this should be correct 70.2% of the time.

A similar calculation using only the prime 3 would be correct 63.7% of the time. Using both 2 and 3 is slightly higher. Using the first 25 primes would be correct 70.34% of the time.

How would this theoretical computation work? Given even n, predict $\mu^2(n) = 0$ (which is correct with probability 0.5948). Otherwise, say $\mu^2(n) = 1$ (correct with probability 0.8105). Each scenario occurs with probability 0.5. In total, this should be correct 70.2% of the time.

A similar calculation using only the prime 3 would be correct 63.7% of the time. Using both 2 and 3 is slightly higher. Using the first 25 primes would be correct 70.34% of the time. (As an aside: I don't know what the limiting behavior should be).

How would this theoretical computation work? Given even n, predict $\mu^2(n) = 0$ (which is correct with probability 0.5948). Otherwise, say $\mu^2(n) = 1$ (correct with probability 0.8105). Each scenario occurs with probability 0.5. In total, this should be correct 70.2% of the time.

A similar calculation using only the prime 3 would be correct 63.7% of the time. Using both 2 and 3 is slightly higher. Using the first 25 primes would be correct 70.34% of the time. (As an aside: I don't know what the limiting behavior should be).

More generally,

$$P(\mathrm{sf}|p_1, \ldots, p_N, \widehat{q_1}, \ldots, \widehat{q_D}) = \prod_{p_i} \left(\frac{p_i}{1 + p_i} \right) \prod_{q_j} \left(\frac{q_j^2}{q_j^2 - 1} \right) \frac{6}{\pi^2}.$$

The cross correlation tables aren't uniform. Of course we should expect ML to pick up on correlation.

How would this theoretical computation work? Given even n, predict $\mu^2(n) = 0$ (which is correct with probability 0.5948). Otherwise, say $\mu^2(n) = 1$ (correct with probability 0.8105). Each scenario occurs with probability 0.5. In total, this should be correct 70.2% of the time.

A similar calculation using only the prime 3 would be correct 63.7% of the time. Using both 2 and 3 is slightly higher. Using the first 25 primes would be correct 70.34% of the time. (As an aside: I don't know what the limiting behavior should be).

More generally,

$$P(\mathrm{sf}|p_1, \ldots, p_N, \widehat{q_1}, \ldots, \widehat{q_D}) = \prod_{p_i} \left(\frac{p_i}{1+p_i}\right) \prod_{q_j} \left(\frac{q_j^2}{q_j^2 - 1}\right) \frac{6}{\pi^2}.$$

The cross correlation tables aren't uniform. Of course we should expect ML to pick up on correlation. We can further test this by inputting $\mathrm{is\_divis}_p(n)$ instead of n mod p: the results for $\mu^2(n)$ are essentially indistinguishable.

In hindsight, we should have expected this. It's important to remember that the model is trying to predict $\mu^2(n)$, not trying to compute $\mu^2(n)$.

When predicting $\mu(n)$, as the most common prediction class is now squarefull numbers (40%, instead of 30% for each $\mu(n) = \pm 1$), basic correlation tables tend to favor overpredicting $\mu(n) = 0$. This is one explanation for why the $\mu(n)$ prediction data correctly identified over 90 percent of squarefull numbers: it was predicting squarefull by default.

Conclusion

From afar, this looks similar to initial neural network experiments. Neural networks trained on $(n; \mu^2(n))$ seem to learn the is-divisible-by-a-small-prime-square function.

Int2Int trained on $\Big(\{n \bmod p, \chi_p(n)\}; \mu^2(n)\Big)$ seems to learn the is-divisible-by-a-small-prime function.

But that's an unkind characterization. It's also true that this set of experiments led to *me* learning something about the distribution of squarefree numbers.

(It's not particularly deep, but it's also not nothing. And I talked about this for a couple of days with some people and we were all a bit stumped. I should note that Noam Elkies immediately realized what was going on.)

The fact that Int2Int is small and rapidly trainable means that I could use it as a one-sided oracle. (That is, I could feed it data, and if it successfully makes lots of predictions than I should expect the inputs to have lots of explanatory power. By restricting inputs, it's sometimes possible to study which parts have explanatory power. Of course it's an "oracle" because it gives no explanations for its premonitions. And it's "one-sided" because often it simply fails to recognize the relationship between the input and output, and this isn't indicative).

Thank you very much.

Please note that these slides are
(or will soon be)
available on my website davidlowryduda.com.