Primality Testing

With Artificial Feedforward Neural Networks

Tashfeen, Ahmad

CS 5033, Computer Science, University of Oklahoma

Bonus Project Presentation, Fall 2022

Overview

1. Introduction: We'll introduce and motivate some key ideas.
2. Data: We need to be careful.
3. Results: We'll look at some figures.
4. Conclusion & References

Introduction: Prime Numbers

Introduction: Prime Numbers

Prime Numbers

A number $1 \neq p \in \mathbb{N}$ is prime if for all non-trivial $(a, b) \in \mathbb{N}^2$, $p \neq ab$.

Introduction: Prime Numbers

Prime Numbers

A number $1 \neq p \in \mathbb{N}$ is prime if for all non-trivial $(a, b) \in \mathbb{N}^2$, $p \neq ab$.

Non-trivial Neither a or b are 1 or p.

Introduction: Prime Numbers

Prime Numbers

A number $1 \neq p \in \mathbb{N}$ is prime if for all non-trivial $(a, b) \in \mathbb{N}^2$, $p \neq ab$.

Non-trivial Neither a or b are 1 or p.

Divisibility All $1 \neq p \in \mathbb{N}$ which are divisible by 1 or themselves.

Prime Numbers

A number $1 \neq p \in \mathbb{N}$ is prime if for all non-trivial $(a, b) \in \mathbb{N}^2$, $p \neq ab$.

Non-trivial Neither a or b are 1 or p.

Divisibility All $1 \neq p \in \mathbb{N}$ which are divisible by 1 or themselves.

Primality Test Given $x \in \mathbb{N}$, Is x a prime number?

Introduction: Prime Numbers

Prime Numbers

A number $1 \neq p \in \mathbb{N}$ is prime if for all non-trivial $(a, b) \in \mathbb{N}^2$, $p \neq ab$.

Non-trivial: Neither a or b are 1 or p.

Divisibility: All $1 \neq p \in \mathbb{N}$ which are divisible by 1 or themselves.

Primality Test: Given $x \in \mathbb{N}$, Is x a prime number?

Prime: 3?

Introduction: Prime Numbers

Prime Numbers

A number $1 \neq p \in \mathbb{N}$ is prime if for all non-trivial $(a, b) \in \mathbb{N}^2$, $p \neq ab$.

Non-trivial Neither a or b are 1 or p.

Divisibility All $1 \neq p \in \mathbb{N}$ which are divisible by 1 or themselves.

Primality Test Given $x \in \mathbb{N}$, Is x a prime number?

Prime 3?

Prime 11?

Prime Numbers

A number $1 \neq p \in \mathbb{N}$ is prime if for all non-trivial $(a, b) \in \mathbb{N}^2$, $p \neq ab$.

Non-trivial Neither a or b are 1 or p.

Divisibility All $1 \neq p \in \mathbb{N}$ which are divisible by 1 or themselves.

Primality Test Given $x \in \mathbb{N}$, Is x a prime number?

Prime 3?

Prime 11?

Prime 27?

Introduction: Prime Numbers

Prime Numbers

A number $1 \neq p \in \mathbb{N}$ is prime if for all non-trivial $(a, b) \in \mathbb{N}^2$, $p \neq ab$.

Non-trivial Neither a or b are 1 or p.

Divisibility All $1 \neq p \in \mathbb{N}$ which are divisible by 1 or themselves.

Primality Test Given $x \in \mathbb{N}$, Is x a prime number?

Prime 3?

Prime 11?

Prime 27?

Prime 32733906078961418700131896968275991522166420460430
64789483291368096133796404674554883270092325904157
150886684127560071009217256545885393053328527589431?

Introduction: Prime Numbers

Prime Numbers

A number $1 \neq p \in \mathbb{N}$ is prime if for all non-trivial $(a, b) \in \mathbb{N}^2$, $p \neq ab$.

Non-trivial Neither a or b are 1 or p.

Divisibility All $1 \neq p \in \mathbb{N}$ which are divisible by 1 or themselves.

Primality Test Given $x \in \mathbb{N}$, Is x a prime number?

- **Prime** 3?
- **Prime** 11?
- **Prime** 27?
- **Prime** 32733906078961418700131896968275991522166420460430
64789483291368096133796404674554883270092325904157
15088668412756007100921725654588539305332852758943l?

All prime numbers are *odd*, and 2, is the *oddest* of them all.

Introduction: Primality Test Complexity

Introduction: Primality Test Complexity

Naïve Approach

Check all $2 \leq i \leq \sqrt{x}$ whether $i \equiv 0 \mod x$.

Introduction: Primality Test Complexity

Naïve Approach

Check all $2 \leq i \leq \sqrt{x}$ whether $i \equiv 0 \mod x$.

Linear Time So the complexity is $\mathcal{O}(\sqrt{x})$, what's the fuss?

Introduction: Primality Test Complexity

Naïve Approach

Check all $2 \leq i \leq \sqrt{x}$ whether $i \equiv 0 \mod x$.

Linear Time So the complexity is $\mathcal{O}(\sqrt{x})$, what's the fuss?

Bitwise Complexity For a b-bit x, division takes $\mathcal{O}(b^2)$ machine steps.

$$\begin{array}{ccccccccc} 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & \ldots \\ 2^2-2 & 2^2-1 & 2^2 & 2^3-3 & 2^3-2 & 2^3-1 & 2^3 & 2^4-7 & \ldots \\ 4 & 4 & 4 & 9 & 9 & 9 & 9 & 16 & \ldots \end{array}$$

Introduction: Primality Test Complexity

Naïve Approach

Check all $2 \leq i \leq \sqrt{x}$ whether $i \equiv 0 \mod x$.

Linear Time So the complexity is $\mathcal{O}(\sqrt{x})$, what's the fuss?

Bitwise Complexity For a b-bit x, division takes $\mathcal{O}(b^2)$ machine steps.

$$\begin{array}{ccccccccc} 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & \ldots \\ 2^2-2 & 2^2-1 & 2^2 & 2^3-3 & 2^3-2 & 2^3-1 & 2^3 & 2^4-7 & \ldots \\ 4 & 4 & 4 & 9 & 9 & 9 & 9 & 16 & \ldots \end{array}$$

Exponential Time $\sum_{i=1}^{\sqrt{n}} \mathcal{O}(b^2) = \mathcal{O}(\lg^2(n)\sqrt{n}) = \mathcal{O}(n\sqrt{n}) = \mathcal{O}(2^{1.5b})$.

Introduction: Primality Test Complexity

Naïve Approach

Check all $2 \leq i \leq \sqrt{x}$ whether $i \equiv 0 \mod x$.

Linear Time So the complexity is $\mathcal{O}(\sqrt{x})$, what's the fuss?

Bitwise Complexity For a b-bit x, division takes $\mathcal{O}(b^2)$ machine steps.

$$\begin{array}{ccccccccc} 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & \ldots \\ 2^2-2 & 2^2-1 & 2^2 & 2^3-3 & 2^3-2 & 2^3-1 & 2^3 & 2^4-7 & \ldots \\ 4 & 4 & 4 & 9 & 9 & 9 & 9 & 16 & \ldots \end{array}$$

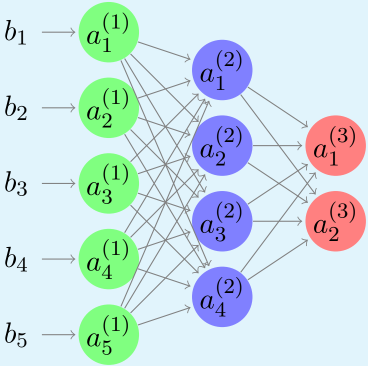
Exponential Time $\sum_{i=1}^{\sqrt{n}} \mathcal{O}(b^2) = \mathcal{O}(\lg^2(n)\sqrt{n}) = \mathcal{O}(n\sqrt{n}) = \mathcal{O}(2^{1.5b})$.

Polynomial Time Algorithm exists, but no proof of optimality.

Introduction: Binary Classification

Universal Approximation Theorem (UAT)

There exists an $n \in \mathbb{N}$ such that a feed-forward neural network (FFNN) with only one hidden layer of n units can approximate any continuos function between euclidean spaces as closely as possible. [2]

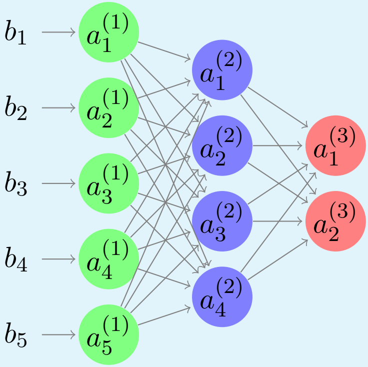


Introduction: Binary Classification

Universal Approximation Theorem (UAT)

There exists an $n \in \mathbb{N}$ such that a feed-forward neural network (FFNN) with only one hidden layer of n units can approximate any continuos function between euclidean spaces as closely as possible. [2]

1. Given a b-bit number, classify it as prime or composite.

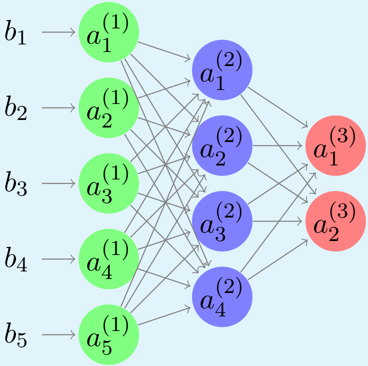


Introduction: Binary Classification

Universal Approximation Theorem (UAT)

There exists an $n \in \mathbb{N}$ such that a feed-forward neural network (FFNN) with only one hidden layer of n units can approximate any continuos function between euclidean spaces as closely as possible. [2]

1. Given a b-bit number, classify it as prime or composite.
2. Caveat, can primality testing be approximated by some continuos function we hope to learn due to UAT?

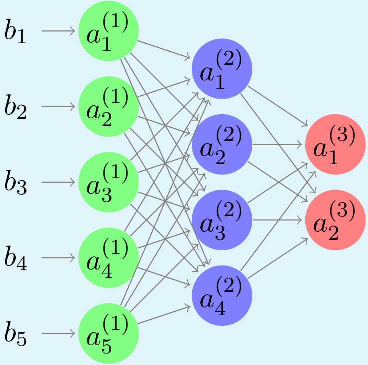


Introduction: Binary Classification

Universal Approximation Theorem (UAT)

There exists an $n \in \mathbb{N}$ such that a feed-forward neural network (FFNN) with only one hidden layer of n units can approximate any continuos function between euclidean spaces as closely as possible. [2]

1. Given a b-bit number, classify it as prime or composite.
2. Caveat, can primality testing be approximated by some continuos function we hope to learn due to UAT?
3. How does n change as b increases?



Approach: Base Plan

1. Let $n \to 1, b \to 7, \varepsilon > 0.5$,
2. If $b > N$, end.
3. Train a FFNN with n hidden units on b bit numbers till log-loss does not improve.
4. If accuracy $< \varepsilon$, $n \to n + 1$. Goto step 3.
5. If accuracy $> \varepsilon$, $n \to 1, b \to b + 1$. Note (b, n) Goto step 2.

Approach: Base Plan Pitfalls

Approach: Base Plan Pitfalls

1. Primes are very sparse, per prime counting function,

$$\pi(2^b) > \frac{2^b}{\ln(2^b)} > \frac{2^b}{\lg(2^b)} = \frac{2^b}{b} = \frac{2^b}{2^{\lg(b)}} = 2^{b-\lg(b)}$$

Then, $1 - (2^{b-\lg(b)}/2^b) = 1 - 2^{-\lg(b)} = \frac{b-1}{b}$ and for 10 bits, we can achieve 90% accuracy by just classifying as composites.

Approach: Base Plan Pitfalls

1. Primes are very sparse, per prime counting function,

$$\pi(2^b) > \frac{2^b}{\ln(2^b)} > \frac{2^b}{\lg(2^b)} = \frac{2^b}{b} = \frac{2^b}{2^{\lg(b)}} = 2^{b-\lg(b)}$$

   Then, $1 - (2^{b-\lg(b)}/2^b) = 1 - 2^{-\lg(b)} = \frac{b-1}{b}$ and for 10 bits, we can achieve 90% accuracy by just classifying as composites.

2. Let's pick the same number of composites to train on. One easy way if for $1 \leq i, j \leq \pi(2^b)$, we compute $p_i \times p_j$.

Approach: Base Plan Pitfalls

1. Primes are very sparse, per prime counting function,

$$\pi(2^b) > \frac{2^b}{\ln(2^b)} > \frac{2^b}{\lg(2^b)} = \frac{2^b}{b} = \frac{2^b}{2^{\lg(b)}} = 2^{b-\lg(b)}$$

   Then, $1 - (2^{b-\lg(b)}/2^b) = 1 - 2^{-\lg(b)} = \frac{b-1}{b}$ and for 10 bits, we can achieve 90% accuracy by just classifying as composites.

2. Let's pick the same number of composites to train on. One easy way if for $1 \leq i, j \leq \pi(2^b)$, we compute $p_i \times p_j$.
3. The composites generated this way will be consistently larger than prime numbers and the network just learns a threshold and calls everything under it prime for a decent accuracy.

Approach: Base Plan Pitfalls

1. Primes are very sparse, per prime counting function,

$$\pi(2^b) > \frac{2^b}{\ln(2^b)} > \frac{2^b}{\lg(2^b)} = \frac{2^b}{b} = \frac{2^b}{2^{\lg(b)}} = 2^{b-\lg(b)}$$

   Then, $1 - (2^{b-\lg(b)}/2^b) = 1 - 2^{-\lg(b)} = \frac{b-1}{b}$ and for 10 bits, we can achieve 90% accuracy by just classifying as composites.
2. Let's pick the same number of composites to train on. One easy way if for $1 \le i, j \le \pi(2^b)$, we compute $p_i \times p_j$.
3. The composites generated this way will be consistently larger than prime numbers and the network just learns a threshold and calls everything under it prime for a decent accuracy.
4. Why can't we just add 1 to each prime in the data? Even numbers.

Approach: Base Plan Pitfalls

1. Primes are very sparse, per prime counting function,

$$\pi(2^b) > \frac{2^b}{\ln(2^b)} > \frac{2^b}{\lg(2^b)} = \frac{2^b}{b} = \frac{2^b}{2^{\lg(b)}} = 2^{b-\lg(b)}$$

   Then, $1 - (2^{b-\lg(b)}/2^b) = 1 - 2^{-\lg(b)} = \frac{b-1}{b}$ and for 10 bits, we can achieve 90% accuracy by just classifying as composites.

2. Let's pick the same number of composites to train on. One easy way if for $1 \leq i, j \leq \pi(2^b)$, we compute $p_i \times p_j$.
3. The composites generated this way will be consistently larger than prime numbers and the network just learns a threshold and calls everything under it prime for a decent accuracy.
4. Why can't we just add 1 to each prime in the data? Even numbers.
5. Why can't we just add 2 to each prime in the data? Twin numbers: $11 + 2 = 13$.

Approach: Base Plan Pitfalls

1. Primes are very sparse, per prime counting function,

$$\pi(2^b) > \frac{2^b}{\ln(2^b)} > \frac{2^b}{\lg(2^b)} = \frac{2^b}{b} = \frac{2^b}{2^{\lg(b)}} = 2^{b-\lg(b)}$$

   Then, $1 - (2^{b-\lg(b)}/2^b) = 1 - 2^{-\lg(b)} = \frac{b-1}{b}$ and for 10 bits, we can achieve 90% accuracy by just classifying as composites.

2. Let's pick the same number of composites to train on. One easy way if for $1 \leq i, j \leq \pi(2^b)$, we compute $p_i \times p_j$.
3. The composites generated this way will be consistently larger than prime numbers and the network just learns a threshold and calls everything under it prime for a decent accuracy.
4. Why can't we just add 1 to each prime in the data? Even numbers.
5. Why can't we just add 2 to each prime in the data? Twin numbers: $11 + 2 = 13$.
6. Loop and generate once then store and load.

Approach: Base Plan Pitfalls

Approach: Base Plan Pitfalls

1. Memory needed to store all 64 bit primes (by storing half gaps),

$$\frac{2^{64-6} \times 16 \text{ bits}}{2^{50} \text{ petabits}} = 2^{12} \text{ petabits} \Rightarrow \frac{2^{12} \text{ petabits}}{8 \text{ bits}} = 512 \text{ petabytes.}$$

Approach: Base Plan Pitfalls

1. Memory needed to store all 64 bit primes (by storing half gaps),

$$\frac{2^{64-6} \times 16 \text{ bits}}{2^{50} \text{ petabits}} = 2^{12} \text{ petabits} \Rightarrow \frac{2^{12} \text{ petabits}}{8 \text{ bits}} = 512 \text{ petabytes.}$$

2. We work with only 32 bit primes. This still takes days of training time and somewhere around 120 gigbytes of memory when the primes and composites' stored as a bit matrix for training.

Results: How it feels.

Figure: At this point, I just ran the experiment on a super computer.

Results: Number of Neurons versus Bits.

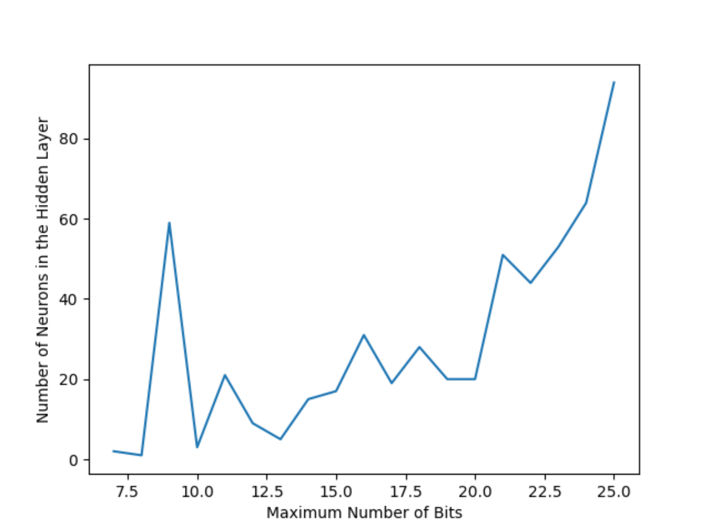


Figure: n vs. b.

Results: Accuracy vs Bits and Neurons.

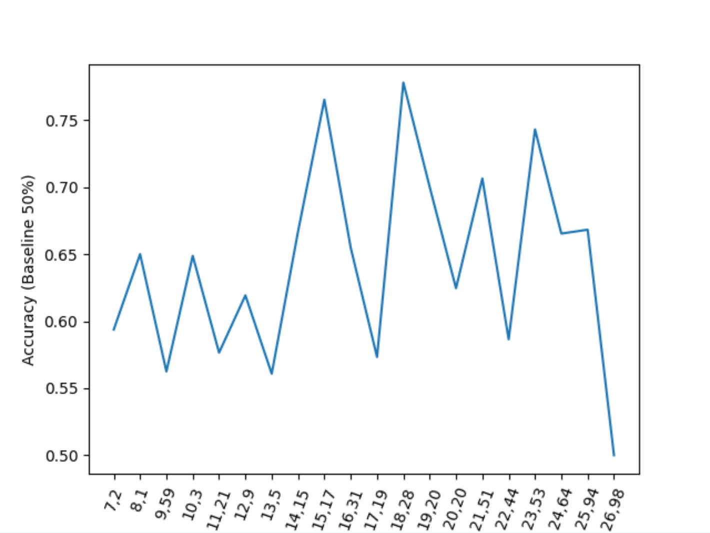


Figure: Accuracy vs (b, n).

Conclusion

1. We can't blindely apply machine learning on number theory problems.
2. Fawzi et al. improved matrix multiplication bound using neural networks [1].

References

Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J R Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, et al.
Discovering faster matrix multiplication algorithms with reinforcement learning.
Nature, 610(7930):47–53, 2022.

Michael A Nielsen.
Neural networks and deep learning, volume 25.
Determination press San Francisco, CA, USA, 2015.

Thank You!
Questions?