Functions NN's Can Learn

WXML: Math AI Lab

Evals

- Final train accuracy
- Final val accuracy
- In-distribution test accuracy
- Out-of-distribution test accuracy

Background (things to mention)

–What is a neural network?

–Mention various choices: number of hidden layers, dimension of hidden layers, activation function, choice of stochastic gradient descent algorithm, but most important choice turned out to be encoding of input and output vectors

–Universal Approximation Theorem (approximate arbitrarily well on *bounded* domain (same true for polynomials..)

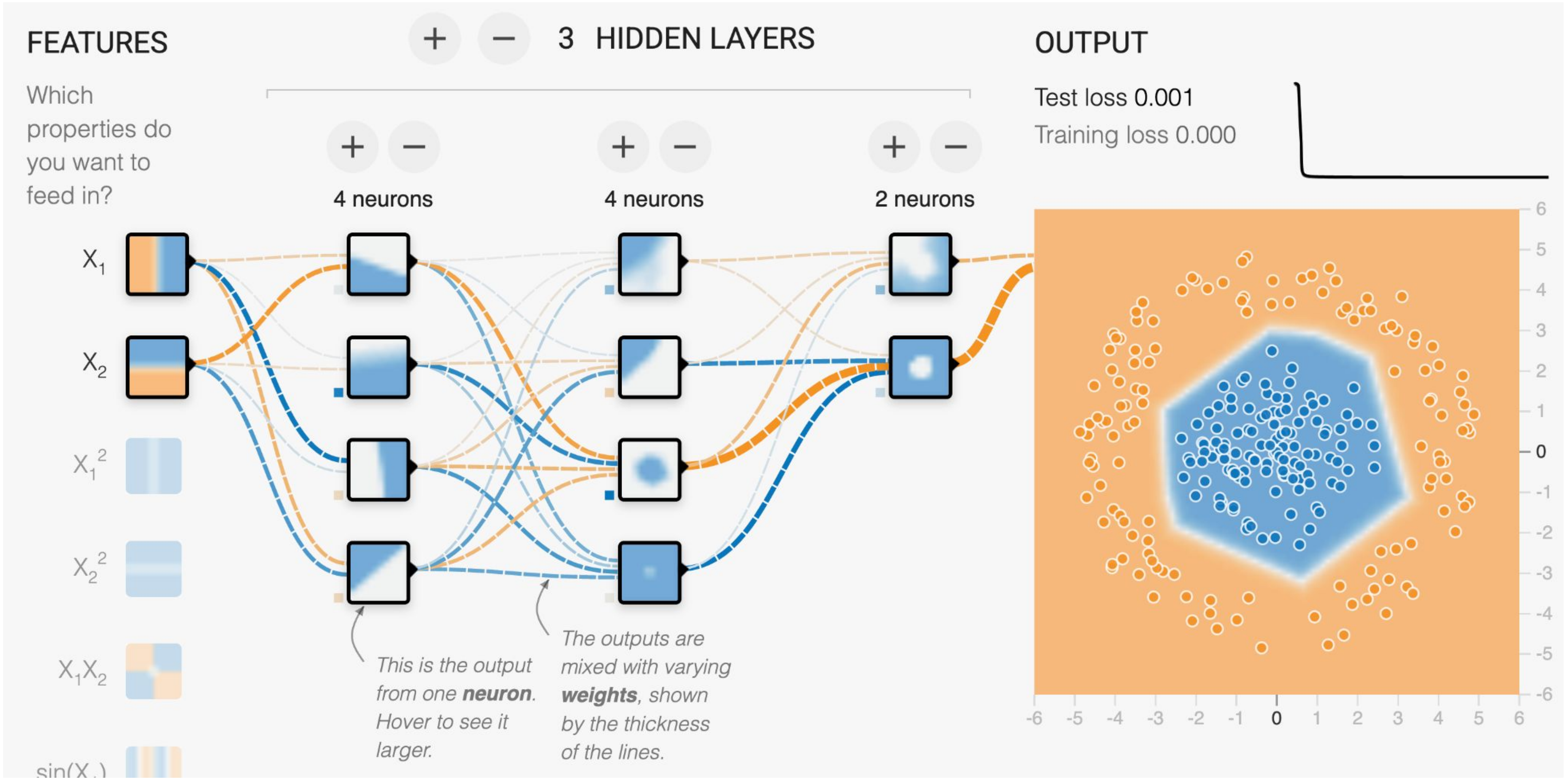
The problem:

How does the *encoding* affect how hard it is for a neural network learn number-theoretic functions?

Model architecture + training details

- Model: 2-layer MLP
- Optimizer: SGD with learning rate 1e-3
- Loss: Binary cross-entropy*
- Hidden Layers: 1
- Input dim: N/A
- Hidden dim: N/A
- Output dim: N/A
- Samples: 8192
- Batch size: 32
- Epochs: 50
- Train-test split: 80% train, 20% test
- Out-of-distribution test samples: 1024

Neural Network

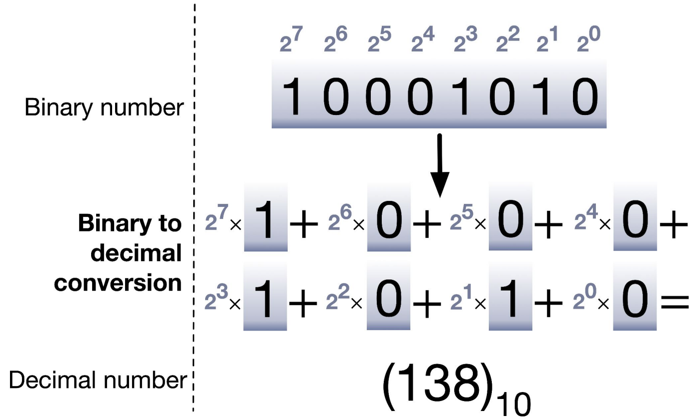


Reference: Tensorflow playground

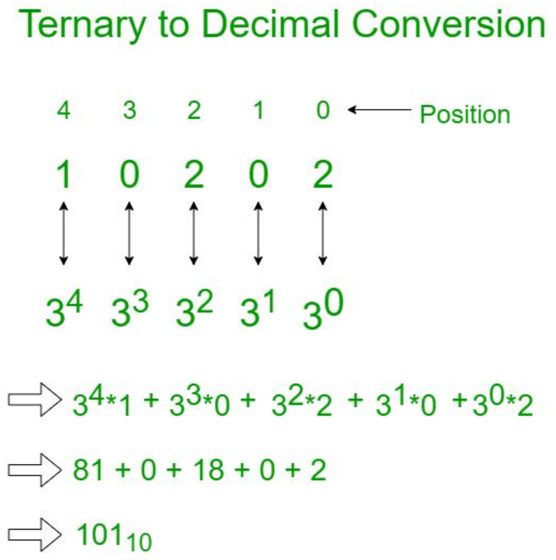
“k-ary” Representations

Examples:

- ➢ Binary representations (k = 2)
- ➢ Ternary representations (k = 3)



Binary number
2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0
1 0 0 0 1 0 1 0
Binary to decimal conversion
$2^7 \times 1 + 2^6 \times 0 + 2^5 \times 0 + 2^4 \times 0 +$
$2^3 \times 1 + 2^2 \times 0 + 2^1 \times 1 + 2^0 \times 0 =$
Decimal number
$(138)_{10}$



References: learningc.org, geeksforgeeks.org

Parity

$$f(x) = x \pmod 2$$

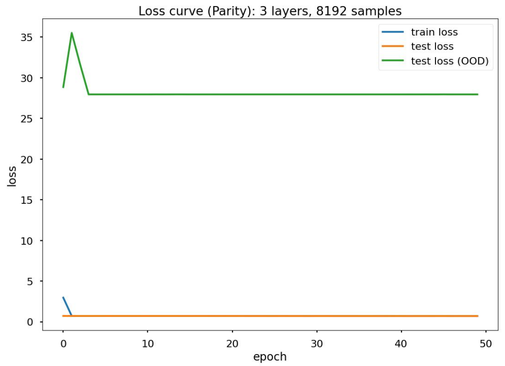
Function Details:

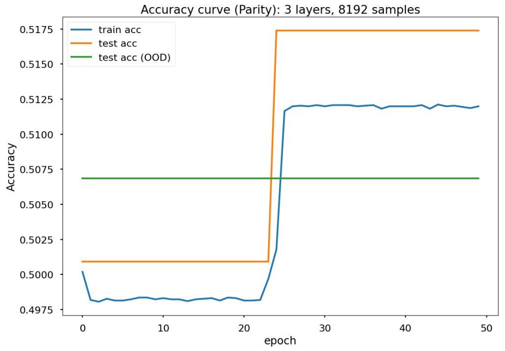
- Input: $x \in \mathbb{Z}$
- Output: $f(x) \in \{0, 1\}$

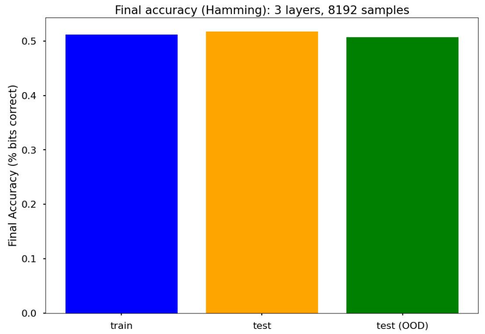
Encoding:

➢ Case 1: The input & output are integers
  - ○ Resorts to random guessing (50% Training Accuracy)

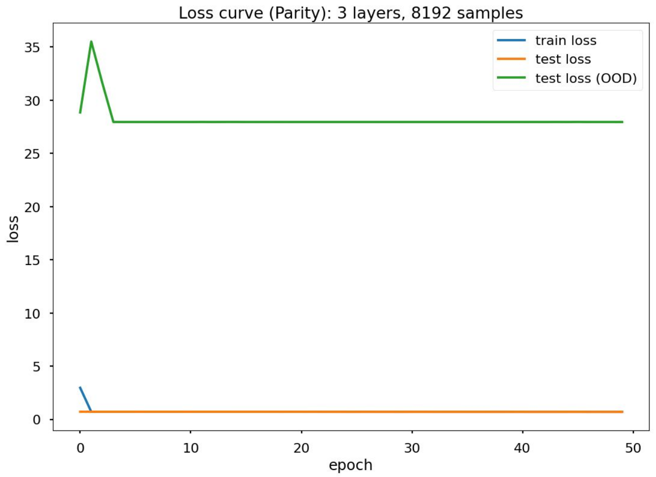
➢ Case 2: The input & output are binary representations
  - ○ 100% Training Accuracy
  - ○ Neural network simply looks at the 0th element of the binary

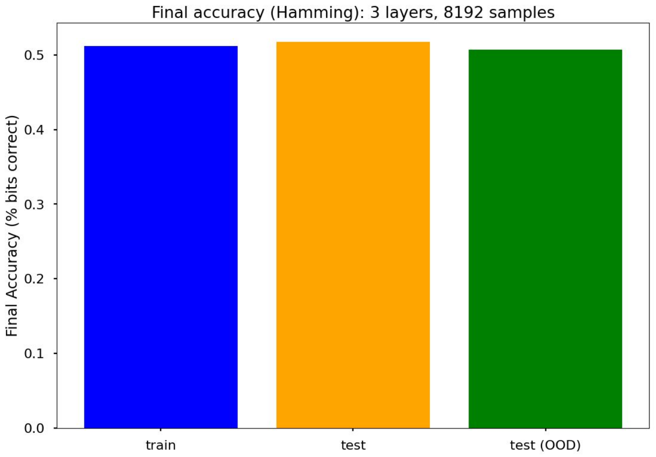


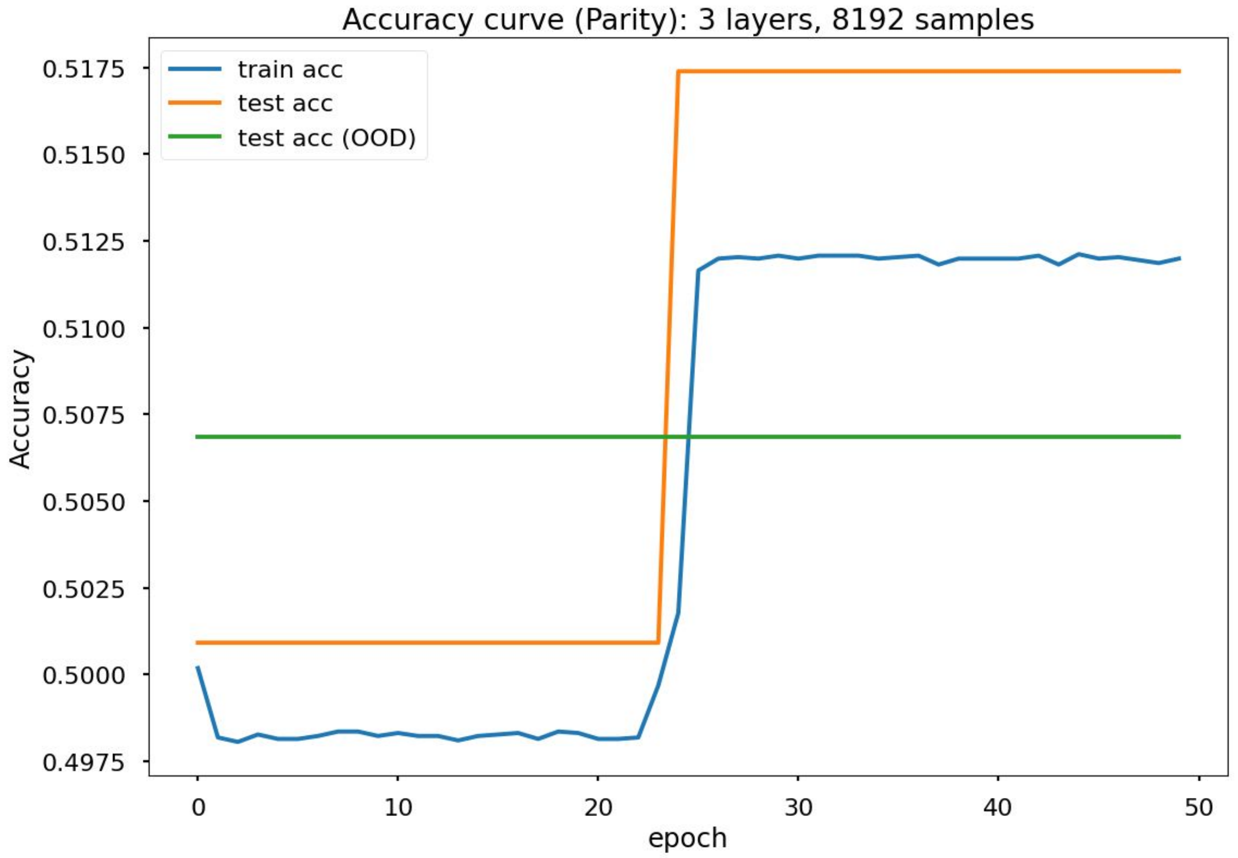




Parity Performance



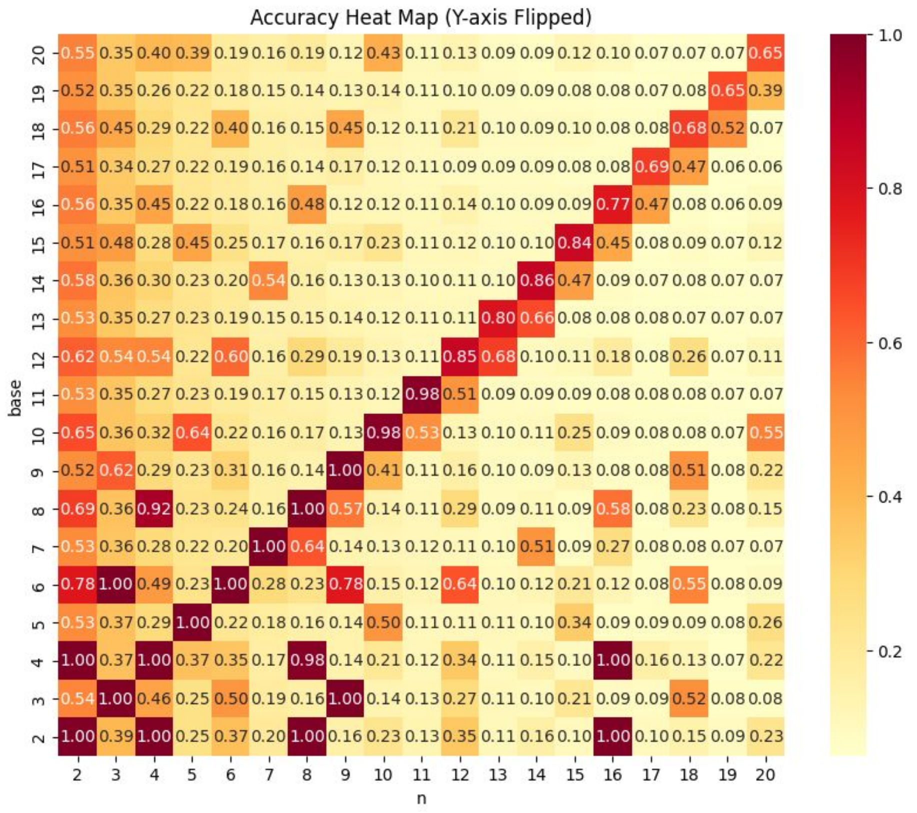




Modular Arithmetic Experimentation

$$f(x) = x \pmod{n}$$

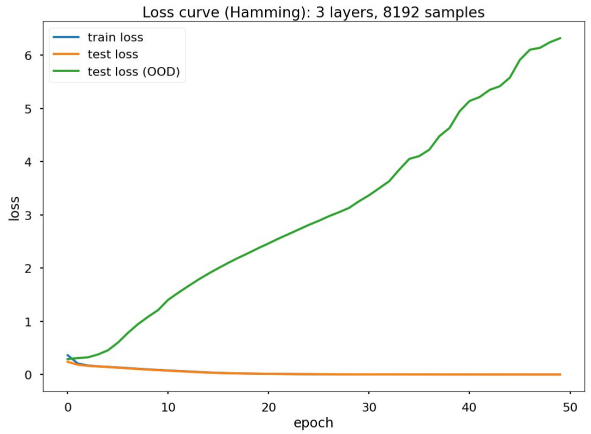
- ➢ We vary parameter k: ‘k-ary’ representation (binary, …, 20-ary)
- ➢ We also vary parameter n: ‘mod n’ (2-20)
- ➢ If k = n, super high accuracy but breaks down around n=k=12
- ➢ If k divides n, super high accuracy!
  - ○ E.g. k = 6, n = 3 vs n = 4
- ➢ If k and n are relatively prime, difficulty
  - ○ E.g. k = 7, n = 6
- ➢ Explanations

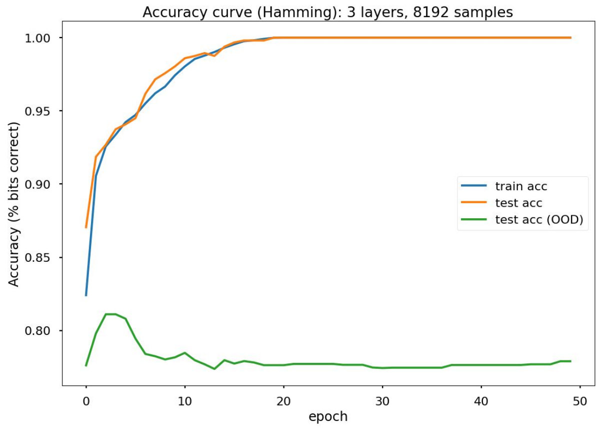


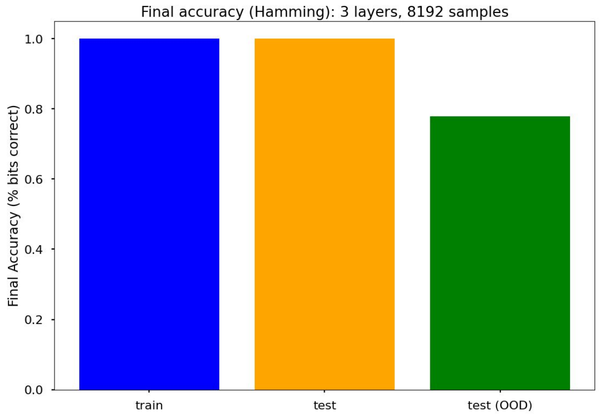
base \ n	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
20	0.55	0.35	0.40	0.39	0.19	0.16	0.19	0.12	0.43	0.11	0.13	0.09	0.09	0.12	0.10	0.07	0.07	0.07	0.65
19	0.52	0.35	0.26	0.22	0.18	0.15	0.14	0.13	0.14	0.11	0.10	0.09	0.09	0.08	0.08	0.07	0.08	0.65	0.39
18	0.56	0.45	0.29	0.22	0.40	0.16	0.15	0.45	0.12	0.11	0.21	0.10	0.09	0.10	0.08	0.08	0.68	0.52	0.07
17	0.51	0.34	0.27	0.22	0.19	0.16	0.14	0.17	0.12	0.11	0.09	0.09	0.09	0.08	0.08	0.69	0.47	0.06	0.06
16	0.56	0.35	0.45	0.22	0.18	0.16	0.48	0.12	0.12	0.11	0.14	0.10	0.09	0.09	0.77	0.47	0.08	0.06	0.09
15	0.51	0.48	0.28	0.45	0.25	0.17	0.16	0.17	0.23	0.11	0.12	0.10	0.10	0.84	0.45	0.08	0.09	0.07	0.12
14	0.58	0.36	0.30	0.23	0.20	0.54	0.16	0.13	0.13	0.10	0.11	0.10	0.86	0.47	0.09	0.07	0.08	0.07	0.07
13	0.53	0.35	0.27	0.23	0.19	0.15	0.15	0.14	0.12	0.11	0.11	0.80	0.66	0.08	0.08	0.08	0.07	0.07	0.07
12	0.62	0.54	0.54	0.22	0.60	0.16	0.29	0.19	0.13	0.11	0.85	0.68	0.10	0.11	0.18	0.08	0.26	0.07	0.11
11	0.53	0.35	0.27	0.23	0.19	0.17	0.15	0.13	0.12	0.98	0.51	0.09	0.09	0.09	0.08	0.08	0.08	0.07	0.07
10	0.65	0.36	0.32	0.64	0.22	0.16	0.17	0.13	0.98	0.53	0.13	0.10	0.11	0.25	0.09	0.08	0.08	0.07	0.55
9	0.52	0.62	0.29	0.23	0.31	0.16	0.14	1.00	0.41	0.11	0.16	0.10	0.09	0.13	0.08	0.08	0.51	0.08	0.22
8	0.69	0.36	0.92	0.23	0.24	0.16	1.00	0.57	0.14	0.11	0.29	0.09	0.11	0.09	0.58	0.08	0.23	0.08	0.15
7	0.53	0.36	0.28	0.22	0.20	1.00	0.64	0.14	0.13	0.12	0.11	0.10	0.51	0.09	0.27	0.08	0.08	0.07	0.07
6	0.78	1.00	0.49	0.23	1.00	0.28	0.23	0.78	0.15	0.12	0.64	0.10	0.12	0.21	0.12	0.08	0.55	0.08	0.09
5	0.53	0.37	0.29	1.00	0.22	0.18	0.16	0.14	0.50	0.11	0.11	0.11	0.10	0.34	0.09	0.09	0.09	0.08	0.26
4	1.00	0.37	1.00	0.37	0.35	0.17	0.98	0.14	0.21	0.12	0.34	0.11	0.15	0.10	1.00	0.16	0.13	0.07	0.22
3	0.54	1.00	0.46	0.25	0.50	0.19	0.16	1.00	0.14	0.13	0.27	0.11	0.10	0.21	0.09	0.09	0.52	0.08	0.08
2	1.00	0.39	1.00	0.25	0.37	0.20	1.00	0.16	0.23	0.13	0.35	0.11	0.16	0.10	1.00	0.10	0.15	0.09	0.23

Generalizing Hamming Weight

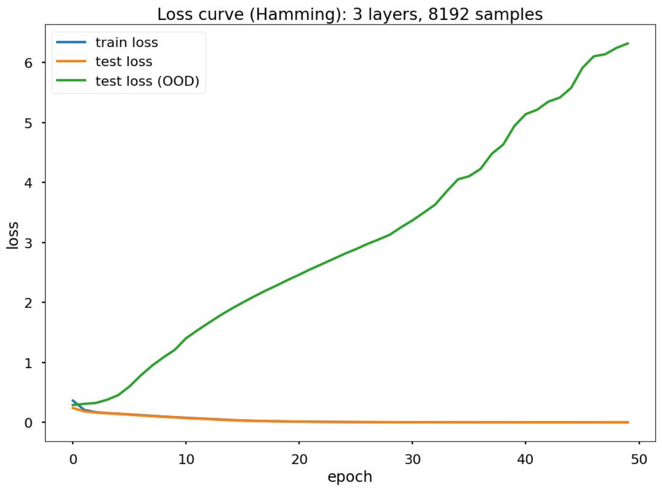
- The *Hamming Weight* of a bitstring *s* is the number of 1's in the bitstring.
  - For example, Wt(00110) = 2, Wt(11011) = 4, Wt(10) = 1
- Two output encodings: either one real output, and another with two outputs measuring probabilities that it is 0 and 1

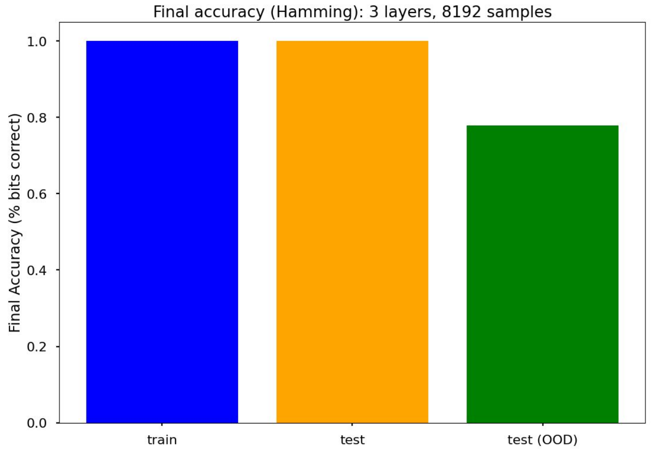


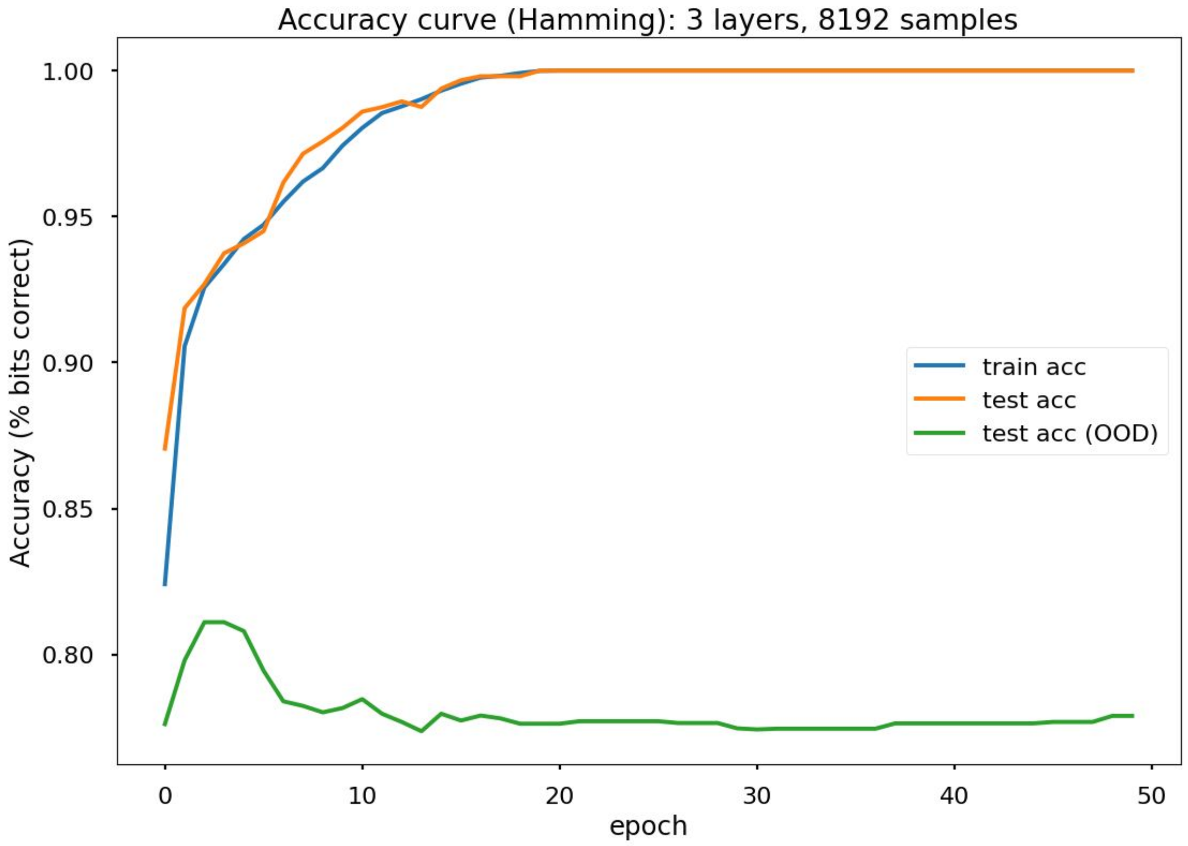




Hamming Weight Performance

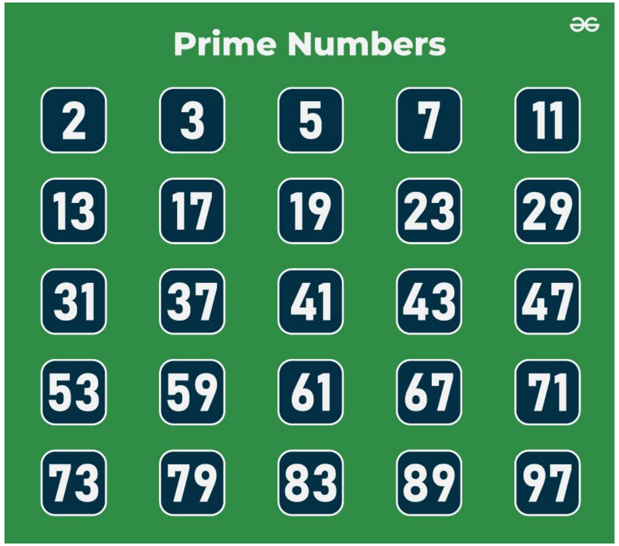






Determine Prime

A prime number is a natural number greater than 1 that is not a product of two smaller natural numbers.



Prime Numbers

2	3	5	7	11
13	17	19	23	29
31	37	41	43	47
53	59	61	67	71
73	79	83	89	97

Image Source: https://www.geeksforgeeks.org/prime-numbers/

The Mobius Function

$$\mu(n) = \begin{cases} 1 & \text{if } n = 1 \\ (-1)^k & \text{if } n \text{ is the product of } k \text{ distinct primes} \\ 0 & \text{if } n \text{ is divisible by a square} > 1 \end{cases}$$

- If n is not square free, Mobius outputs 0
- If n is square free, Mobius tells us the whether n has a odd or even number of prime divisors
  - 1 if even
  - -1 if odd

n	1	2	3	4	5	6	7	8	9	10
$\mu(n)$	1	−1	−1	0	−1	1	−1	0	0	1

Image Source: https://en.wikipedia.org/wiki/M%C3%B6bius_function

Determine Prime

Encoding

Integer: Normalized between 0 and 1

Binary: 14-bit representation

Modular: an array of remainders when divided by primes up to N=100

Loss Curves

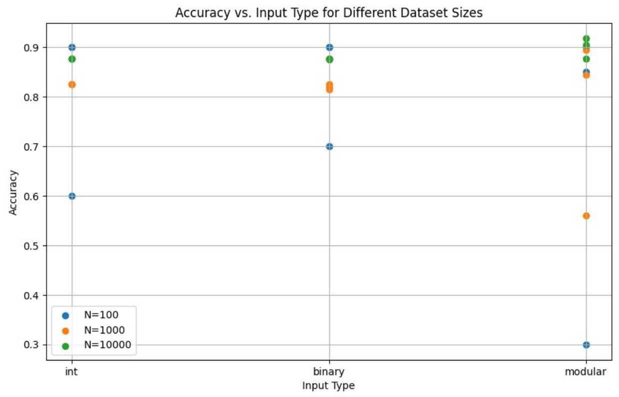
Integer encoding has a nearly constant loss curve, suggesting the model struggles.
Binary and Modular embeddings show a steep drop in loss, indicating faster learning.

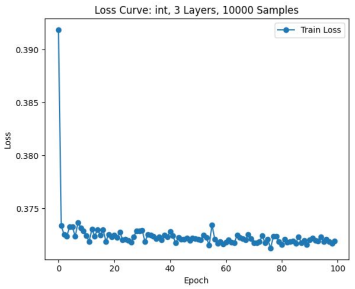
Accuracy Trends

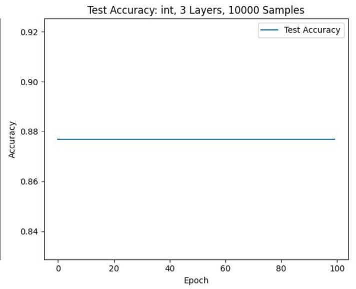
Integer encoding accuracy remains constant (~88%), suggesting a bias toward a simple rule.
Binary encoding achieves stable accuracy but fluctuates.
Modular encoding performs similarly but with more variance.

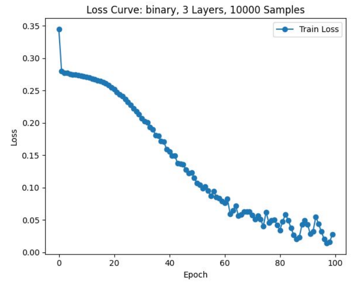
Something Interesting

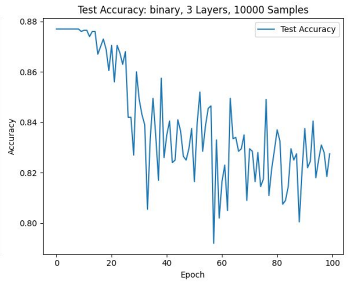
- Increasing the sample size doesn’t seem to help performance
- Modular slightly outperforms the others
- Overall, Prime is learnable

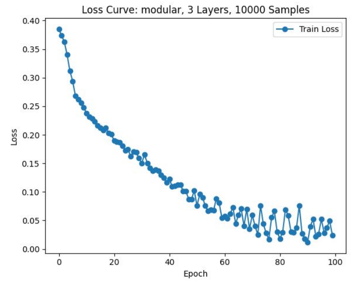


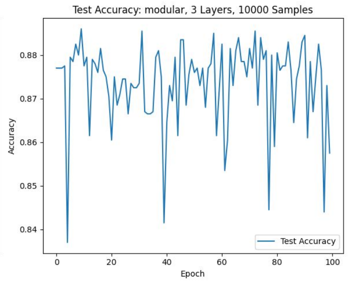




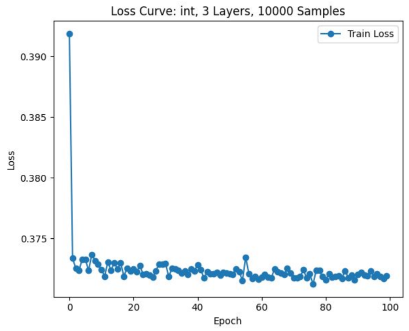


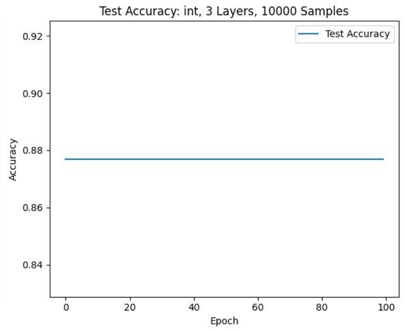


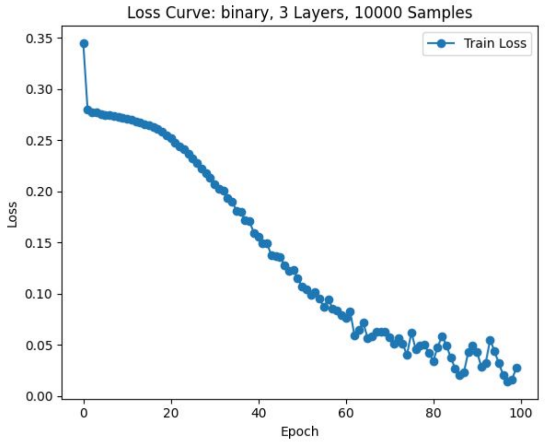


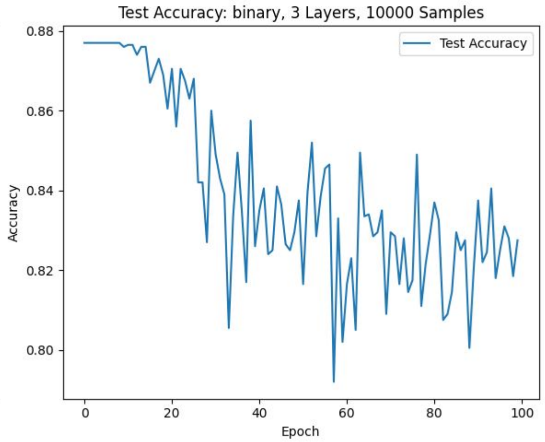


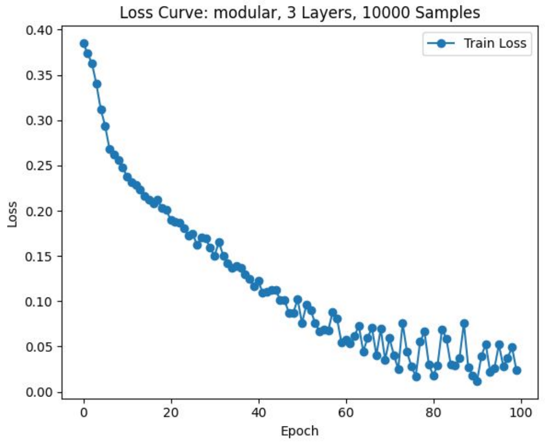
Prime Determination Performance

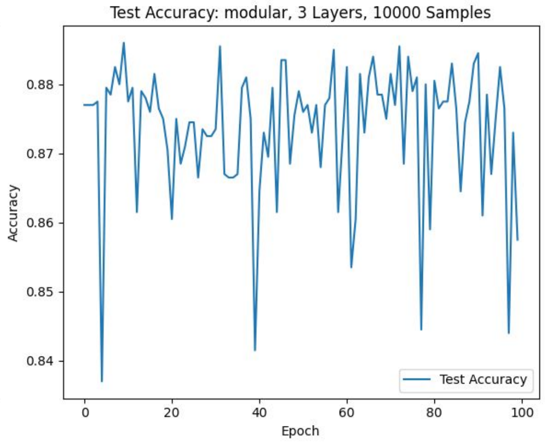












Left to right, top to bottom: Integer, binary, and modular embeddings

The Mobius Function

Integer

- Loss decreases slightly but fluctuates, showing minimal learning progress.
- Accuracy remains constant (~38%), suggesting the model struggles to generalize.

Binary

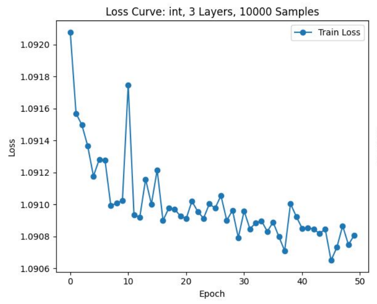
- Loss decreases consistently but at a slower rate compared to modular.
- Accuracy fluctuates but reaches ~57%, making it an effective encoding for learning the Möbius function.

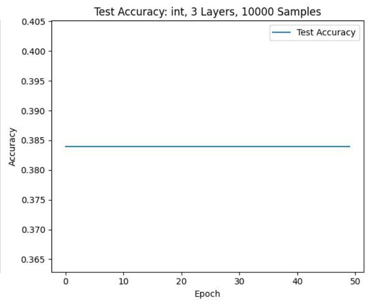
Modular

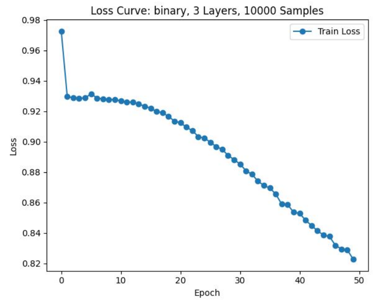
- Loss decreases steadily, indicating effective learning.
- Accuracy starts low (~40%) but gradually improves to ~48%, showing that modular encoding is informative but harder to learn.

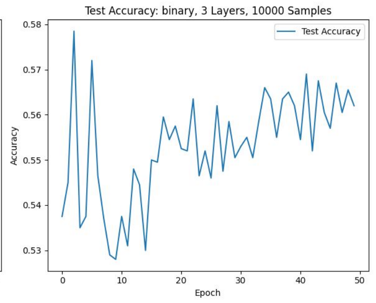
Conclusions

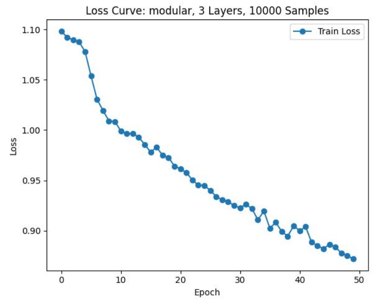
- Integer encoding is the least effective, likely due to its smooth structure, which lacks rich features for learning Möbius.
- Binary encoding performs good, suggesting that the bitwise structure provides useful patterns.
- Modular encoding is intermediate but has higher variance, possibly requiring a deeper network.

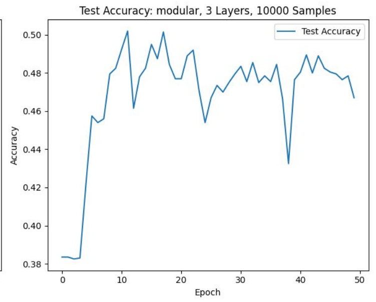




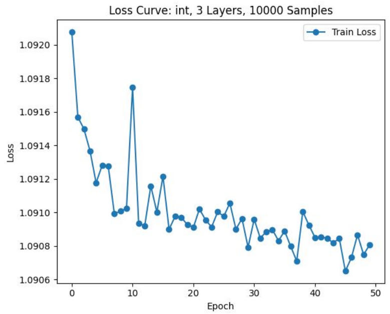


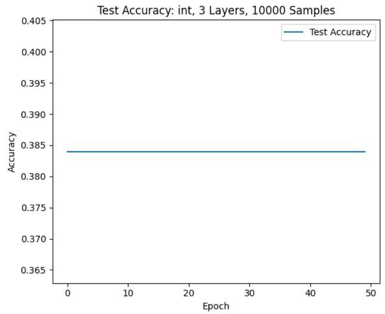


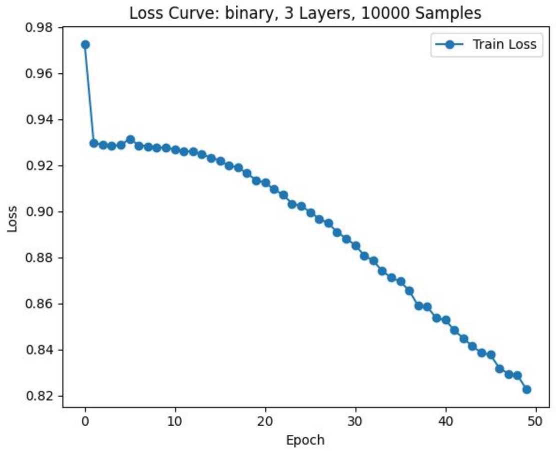


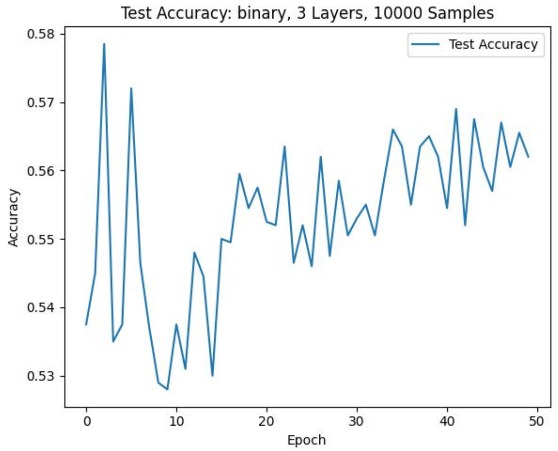


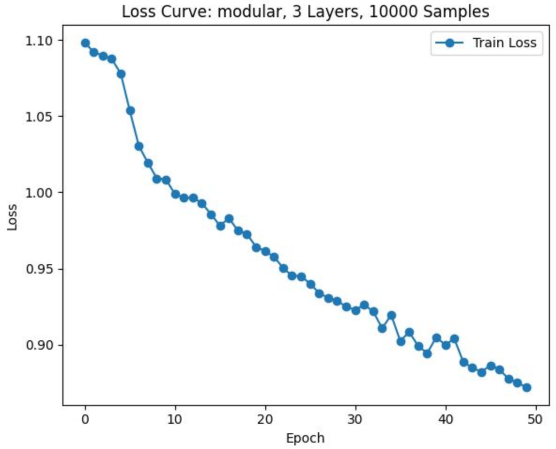
Mobius Function Performance

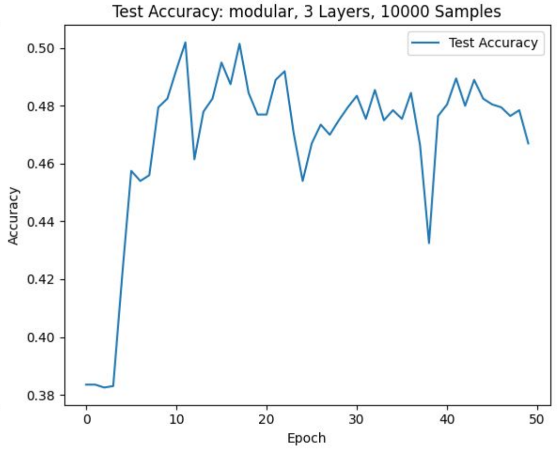










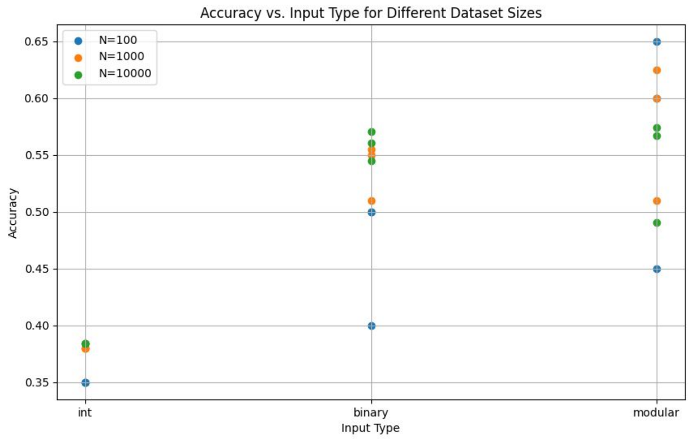


Left to right, top to bottom: Integer, binary, and modular embeddings

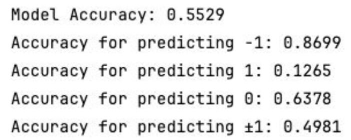
Some More Experiments

Experiments on predicting the Mobius Function varying encoding, sample size, and number of hidden layers.

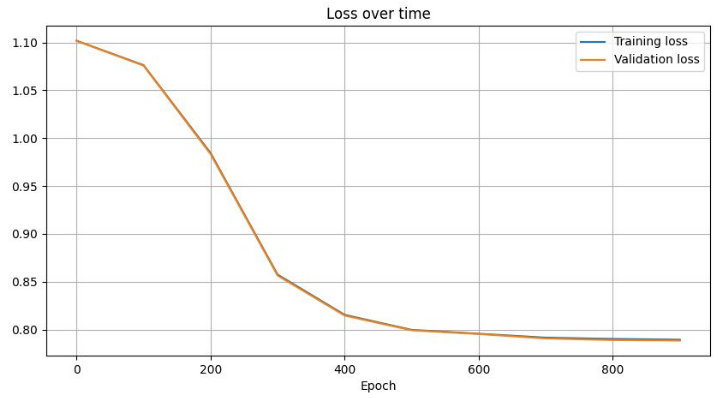
- Modular outperform the others when number of hidden layers increase.
- Sample size and number of hidden layers doesn't affect the performance of binary much.
- Integer struggled.



Inspired by a workshop at Working Mathematician Seminar, which encodes numbers as binary and learns the probability of each possible output.

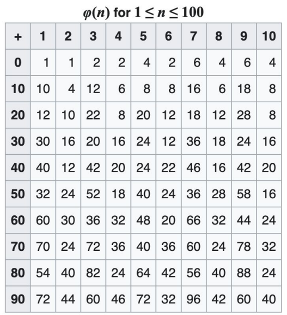


```
Model Accuracy: 0.5529
Accuracy for predicting -1: 0.8699
Accuracy for predicting 1: 0.1265
Accuracy for predicting 0: 0.6378
Accuracy for predicting ±1: 0.4981
```



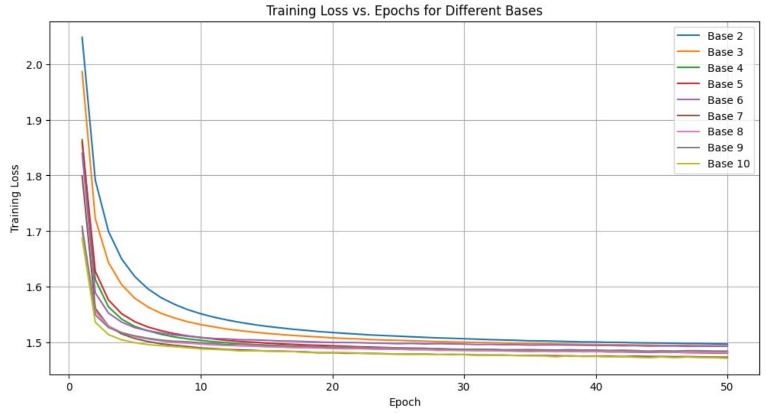
Totient Function

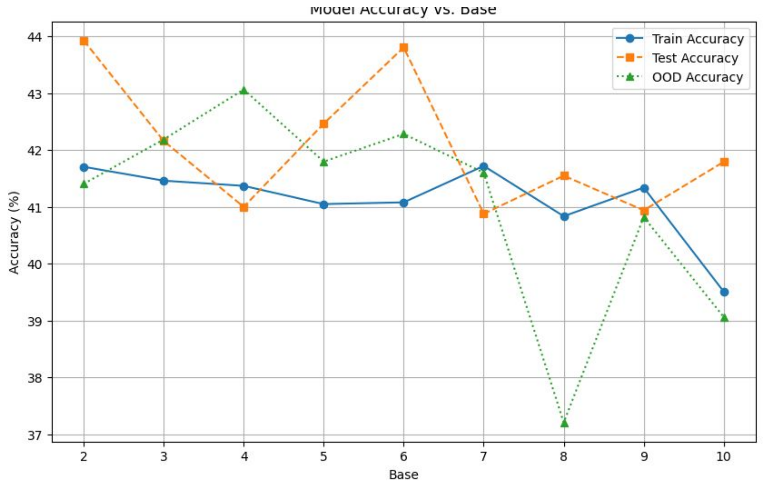
- Definition $$\varphi(n) = n \prod_{p \mid n} \left(1 - \frac{1}{p}\right),$$
- counts the positive integers up to a given integer n that are relatively prime to n
- Encoding: n n in m-ary representation (e.g., binary, base 3, etc.).
- Label: φ(n) mod 10
- For example: Given integer 1234, the label will be φ(1234) % 10, which is 6



$\varphi(n)$ for $1 \le n \le 100$

+	1	2	3	4	5	6	7	8	9	10
0	1	1	2	2	4	2	6	4	6	4
10	10	4	12	6	8	8	16	6	18	8
20	12	10	22	8	20	12	18	12	28	8
30	30	16	20	16	24	12	36	18	24	16
40	40	12	42	20	24	22	46	16	42	20
50	32	24	52	18	40	24	36	28	58	16
60	60	30	36	32	48	20	66	32	44	24
70	70	24	72	36	40	36	60	24	78	32
80	54	40	82	24	64	42	56	40	88	24
90	72	44	60	46	72	32	96	42	60	40





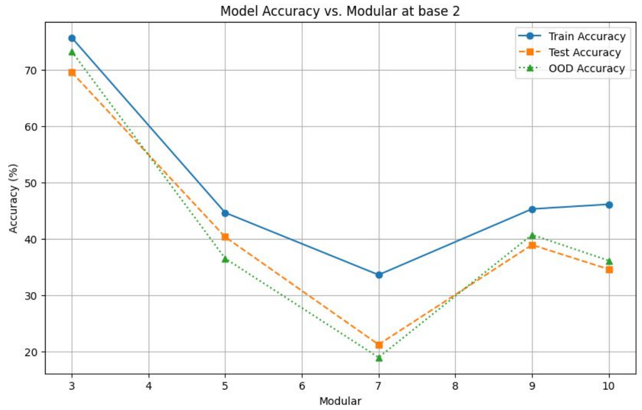
Findings:

1. Training Behavior
   Training loss converges across different bases, showing that the model learns some structure.
2. Accuracy is low (~40%), meaning the MLP struggles to generalize.
3. Test and OOD accuracy fluctuate across bases, with no clear trend of improvement.
4. The model likely captures some structure but fails to generalize well.

Totient Function - more experiments

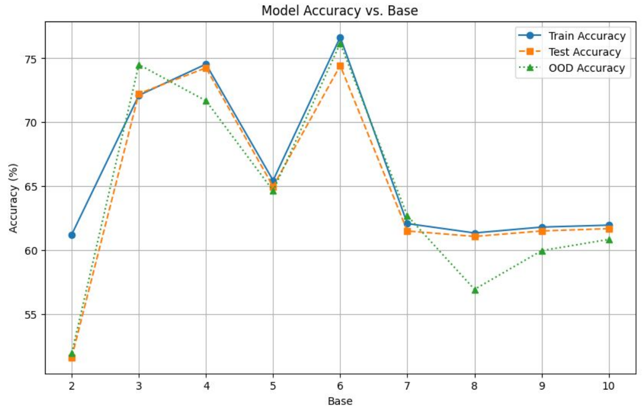
Investigate the impact of changing the modular value when computing φ(n):

- Changing modular of Euler's Totient Function
- Encoding: binary
- Label: φ(n) % modular
  For example: Given integer 1234, the label will be φ(1234) % modular, which is 6 when modular is 10
- Reach the highest accuracy at φ(x) % 3



Whether two given numbers are coprime:

- Due to the definition of Euler's Totient Function, I also examined on simple MLP's ability to learn if two given numbers are coprime.
- The accuracy is around 60 to 70%, which reach the highest accuracy at base 6.
- The simple MLP performs better on coprimality classification than on learning the totient function.



Python Code Synthesis