Neural Networks and the Search for a Quadratic Residue Detector

Michael Potter[*]
mjp2010@rit.edu

Leon Reznik[*]
lr@cs.rit.edu

Stanisław Radziszowski[*]
spr@cs.rit.edu

[*] *Department of Computer Science*
Rochester Institute of Technology
Rochester, NY 14623

Abstract—**This paper investigates the feasibility of employing artificial neural network techniques for solving fundamental cryptography problems, taking quadratic residue detection as an example. The problem of quadratic residue detection is one which is well known in both number theory and cryptography. While it garners less attention than problems such as factoring or discrete logarithms, it is similar in both difficulty and importance. No polynomial–time algorithm is currently known to the public by which the quadratic residue status of one number modulo another may be determined. This work leverages machine learning algorithms in an attempt to create a detector capable of solving instances of the problem more efficiently. A variety of neural networks, currently at the forefront of machine learning methodologies, were compared to see if any were capable of consistently outperforming random guessing as a mechanism for detection. Surprisingly, neural networks were repeatably able to achieve accuracies well in excess of random guessing on numbers up to 20 bits in length. Unfortunately, this performance was only achieved after a super–polynomial amount of network training, and therefore we do not believe that the system as implemented could scale to cryptographically relevant inputs of 500 to 1000 bits. This nonetheless suggests a new avenue of attack in the search for solutions to the quadratic residues problem, where future work focused on feature set refinement could potentially reveal the components necessary to construct a true closed–form solution.**

1. Introduction

Artificial Neural Networks (ANNs) have proven useful in solving many challenging problems across a wide range of fields. In addition to solving pragmatic problems such as image recognition, ANNs have also been brought to bear against fundamental math and science problems. Unfortunately, progress in these more fundamental spaces has been slow compared to the rapid advancements in the applied domain. Nevertheless, there are many open problems in number theory where practical solutions would have an enormous impact on cryptography and computer security (factoring, discrete logarithms, etc.). No asymptotically efficient algorithms for these problems have yet been found. This paper investigates the possibility of employing ANNs in the search for a solution to the quadratic residues (QR) problem, which is defined as follows:

Suppose that you have two integers a and b. Then a is a quadratic residue of b if and only if the following two conditions hold: [1]

1) The greatest common divisor of a and b is 1, and
2) There exists an integer c such that $c^2 \equiv a \pmod{b}$.

The first condition is satisfied automatically so long as $a \in Z_b^*$. There is, however, no known polynomial–time (in the number of bits of a and b) algorithm for determining whether the second condition is satisfied. The difficulty and significance of this problem will be discussed in more detail in Section 1.3.

1.1. Machine Learning Background

For all that it is generally believed to be both an important and difficult problem, it is not clear that anyone has yet published any attempts to find a solution for quadratic residue detection using modern machine learning (ML) techniques. This project sought to remedy the apparent shortcoming in the contemporary literature by seeking evidence for the potential tractability of quadratic residue detection within an ML context.

ANNs have garnered a great deal of attention in the literature recently for their achievements in a variety of domains. In the field of game AI, ANNs have been able to successfully learn to play Chess [2] and recently even the game Go [3] – a feat which had previously been considered well beyond the capacity of modern machine learning. In such applications it appears that these neural networks are able to collapse what would otherwise be an exponentially large search tree down into a polynomial–time approximation function.

ANNs, and in particular Convolutional Neural Networks (CNNs), have also been extremely successful in pattern recognition. Their stunning success in the image processing domain, in particular on the well known database ImageNet, has inspired a great deal of interest in CNN development. [4] In fact, since its publication in 2012 this early proof of CNN effectiveness has been cited over 5000 times. What is perhaps most surprising, however, is that CNNs have been shown recently to support transfer learning – the ability to train a CNN on one dataset and then use it on another with very minimal retraining required. [5] [6] [7] Such networks are so effective that they can outperform systems which have been developed and tuned for years on a specialized problem set. [7] This power comes from the fact that many of the hidden layers within a CNN serve to perform generic feature extraction – removing the need for humans to generate creative data representations in order to solve a problem. Once this feature extractor has been created, the final layer or two of a CNN is all that needs to be adjusted in order to move between different application areas – a process which requires relatively little time and data. It was hoped that this ability to transfer learning between problem sets would help to create a network able to detect residues for any number n with minimal retraining required. The drawback, however, is that the inputs to a CNN must be meaningfully related to one–another, for example having pixel 'A' be above and to the left of pixel 'B', in order for patterns derived from the relative positions of those inputs to be meaningful. While it is not clear how to satisfy such a constraint when dealing with a number theory problem, the principles and effectiveness of the transfer learning paradigm were still of interest in this study.

The success of ANNs is not without theoretical precedent. The universality theorem, a well known result in machine learning, suggests that neural networks with even a single hidden layer may be used to approximate any continuous function to arbitrary accuracy (given certain constraints on network topology). [8] Based on this premise, if there were a continuous function capable of testing quadratic residues, it ought to be possible to mimic it using a neural network. While such a function, if one exists, might well be discontinuous and therefore beyond the purview of the theorem, there is still hope that a continuous approximation might suffice.

While the theoretical results are interesting, one could argue that such a network would be impossible to train in practice. This was indeed at least a partial motivation behind the deep networks that have become popular in modern literature. There are, however, concrete results suggesting that shallow networks really are capable of learning the same behavior as their deeper counterparts. [9] In their work, Ba and Caurana are able to train shallow networks very effectively by leveraging the outputs produced by a deep network for ground truth scoring as opposed to using the original supervised labeling. While this is not immediately useful since it still requires a trained deep network, it suggests that if such a deep network can be developed it could potentially then be simplified in order to improve performance.

1.2. Mathematics Background

Although there is not yet a known polynomial–time QR detection algorithm, there are methods for efficiently computing partial solutions. For example, for a prime p, the residue status, r, of an integer a is given by $r = a^{\frac{p-1}{2}} \pmod p$. If the result is 1 then a is a quadratic residue of p. If the result is -1 then a is a non–residue, and if the result is 0 then a and p are not relatively prime. [10] This value, also known as the Legendre symbol of a with respect to p, forms the basis of the Jacobi symbol: a generalized approximation of quadratic residues for composite numbers. Suppose $n = \prod_{i=1}^{k} p_i^{e_i}$. Then the Jacobi symbol for an integer a is defined as $\left(\frac{a}{n}\right) = \prod_{i=1}^{k} r_i^{e_i}$, where r_i is the i–th Legendre symbol. While clearly trivial to compute given the factorization of n, this value may also be computed using the Jacobi algorithm which has running time $O(\log(n)^2)$ and does not require knowledge of the factorization of n. [11] The Jacobi algorithm provides the basis for the feature set investigated during this work, so it is given for reference as Algorithm 1.

Algorithm 1 Jacobi Symbol Computation $(\frac{a}{n})$ [11]

```
Jacobi($a$, $n$):
if $a$ == 0 then
    return 0
else if $a$ == 1 then
    return 1
end if
Let $e$, $a_1$ such that $a = 2^e * a_1$, $a_1$ odd
$s$ = 0
if $e$ mod 2 == 0 then
    $s$ = 1
else
    comparator = $n$ mod 8
    if comparator == 1 or comparator == 7 then
        $s$ = 1
    else if comparator == 3 or comparator == 5 then
        $s$ = -1
    end if
end if
if $n$ mod 4 == 3 and $a_1$ mod 4 == 3 then
    $s$ = -$s$
end if
$n_1$ = $n$ mod $a_1$
if $a_1$ == 1 then
    return s
end if
return s*Jacobi($n_1$, $a_1$)
```

Unfortunately, knowing the Jacobi symbol $\left(\frac{a}{n}\right)$ is not sufficient to determine whether a is a quadratic residue of n. Consider the case where $n = p * q$. Then if $\left(\frac{a}{n}\right) = -1$ it holds that a is not a quadratic residue of n. If, however, $\left(\frac{a}{n}\right) = 1$, then it could either be that both r_p and r_q are positive, in which case a is a residue of n, or it could be that both r_p and r_q are negative, in which case a is not a residue of n. Either case is equally likely, meaning that if the Jacobi symbol evaluates to 1 there will be a 50% chance that the number will be a quadratic residue. There are no known polynomial–time algorithms which are capable of improving on this estimate. Thus any system capable of outperforming random guessing in such a case would represent a highly significant result in number theory and cryptography.

1.3. Cryptography Background

While less well known than factoring, the quadratic residue problem also has great significance in cryptography. As seen in Section 1.2, QR detection is clearly not more difficult than factoring. While no polynomial–time reduction from the QR decision problem to factoring is known, it is conjectured to be of similar difficulty. This conjecture has been proven within the context of generic ring algorithms, but not for full Turing machines. [12] This proof, however, is not strong evidence since computation of Jacobi symbols is also provably hard for generic ring algorithms despite the polynomial–time algorithm for integers provided in Section 1.2. It could therefore be the case that QR detection is polynomial–time computable while factoring is not. [12] Despite this possibility, it has become common practice for researchers to assume the difficulty of QR detection when attempting to prove the intractability of other less studied problems in computer science and cryptography. [13] [14]

The closely related problem of actually computing the square roots of a quadratic residue, however, has been proven to be as hard as factoring. To see this, suppose that you had an algorithm capable of computing the square root of a QR in polynomial–time. The number of distinct square roots for a residue of some composite number n is equal to 2^L, where L is the number of distinct primes in the factorization of n. [10] Thus for standard cryptographically relevant composites of the form $n = pq$, there will be 4 roots for every quadratic residue. Pick any number x from 1 to $n-1$ and square it mod n. Use the polynomial–time square root algorithm to find a square root of x^2. Since there are four possible solutions, the algorithm will return a root x' distinct from x with probability $\frac{3}{4}$. Of these distinct values, one will be $-x \pmod{n}$, but two will be other values. So long as $x' \neq x$ and $x' \neq -x$, then it may be used to compute $gcd(x-x', n)$ which will return a non–trivial factor of n. This algorithm therefore succeeds with probability $\frac{1}{2}$ which means that after several iterations one can expect to find a non–trivial factor of n. Since the algorithm has no way of knowing which x value you started with, even an algorithm that returns solutions with unequal probability should succeed if run starting with several different values for x.

RSA is a well known encryption algorithm which relies on the difficulty of factoring numbers of the form $n = pq$. It has not been proven, however, that breaking RSA encryption actually requires the ability to factor n. [15] There are encryption schemes, however, which are provably as secure as the modular square root problem – and hence as secure as factoring. One such protocol is the Rabin Cryptosystem. In this scheme, a private key of two large primes p and q is generated. The public key n is then posted. To encrypt a message m into ciphertext $c = m^2 \pmod{n}$. Since the private key contains knowledge of the factorization of n, it can then be used to compute the four square roots of c, one of which will be m. [15]

While the generic problem of factoring is believed to be hard, its difficulty can depend on the particular values of p and q which form the prime decomposition. In particular, a notion of 'strong' primes has been proposed where p and q are considered strong if $p-1$ and $q-1$ have large prime factors. Such primes are more difficult for certain factoring algorithms to contend with, though the best modern algorithms are not significantly impacted by such conditions. [16] Nonetheless, the use of strong primes is required by ANSI standards in an effort to create a system which is as secure as possible. One way to ensure that the strong prime criterion is met is to pick p and q such that $p = 2r + 1$ and $q = 2t + 1$, where r and t are also prime. This project restricts its analysis to primes of this form since they represent the most difficult class of numbers to handle and therefore give a better indication of the cryptographic significance of any solutions encountered.

1.4. Problem Statement and Goal

The objective of this work was to use neural networks to aid in the search for a solution to the problem of quadratic residue detection. If a neural network can be found which is capable of solving QR, then by selectively editing the network inputs and architecture it might be possible to infer a previously unknown closed–form algorithm. Neural networks are often viewed as black–box solutions, but where humans have not been able to find a solution, such a black–box could provide helpful constraints on the solution space. Even if no progress can be made on that front, the existence of a black–box solution to quadratic residue detection would present a deep challenge to conventional wisdom in cryptography, and open the way for applying neural networks to other open problems in mathematics.

2. Data Generation

There are two significant factors which may influence the difficulty of quadratic residue detection. The first is the composition of the modular base being considered. This base could be prime, in which case a polynomial–time solution is known, or the product of multiple primes, in which case a polynomial–time solution is unknown. The number of prime factors composing the base controls both the number of quadratic residues which are present in the modular ring, and also how many roots each residue will have.

The second factor which was expected to contribute to the difficulty of the problem is the size of the modular base. As the length of this number increases, more elements are added to the modular space. This could make it more difficult for the system to learn any underlying patterns in the data, but also allows for an increased number of potential training instances. Cryptographically relevant numbers contain hundreds of digits, so even if machine learning is capable of discriminating QRs with high accuracy on small numbers it might not pose a threat to modern encryption schemes.

For each base, n, training and testing data are required. For small bases the complete set of residues and non–residues can be easily enumerated and randomly sampled. In order to generate data for the larger bases, random numbers were selected from Z_n. These numbers were squared mod n in order to acquire values which are guaranteed to be quadratic residues. In order to acquire non–residues, random numbers were drawn from Z_n until a non–residue was found. This check is possible in polynomial–time since the prime factorization for each base is known. Non–residues were only included into the dataset if they had Jacobi symbol equal to 1, since said symbol already provides a method to discriminate between residues and non–residues having the alternative sign. The testing datasets were comprised of 50% residues and 50% non–residues, meaning that any system which improve upon 50% accuracy by a notable margin is at least a partial success.

3. Experimental Results

We examined the behavior of ANN architectures, implemented using the TensorFlow library, using data generated as described in Section 2. The networks considered two integers a and n corresponding to the question: "Is a a quadratic residue of n?" There are several important considerations when building a network to attempt to answer this question. First and foremost is what feature representations should be used in order to attempt to find the solution. Different network architectures may also be more or less effective at finding a viable solution. Before taking on the challenging task of detecting quadratic residues relative to a composite modular base, experiments were completed on a prime modular basis. For each case, a 50/50 random guess procedure was run on the testing data 10 times. The resulting accuracy along with standard deviation was recorded so that the neural network's performance can be compared to see whether it is significantly better than what one would expect from chance.

3.1. Primes

The first network architecture investigated was a MultiLayer Perceptron (MLP) with 4 hidden layers, each containing 100 neurons employing hyperbolic tangent activation functions. These units also employ a dropout layer before the softmax output layer in an effort to combat overfitting. A variety of input feature combinations were then explored in the search for potentially useful inputs. As discussed in Section 1.2, the quadratic residue problem for prime bases can be solved explicitly using the Jacobi symbol, so this symbol value was not considered as a potential input. For initial tests, all residues and non–residues from 1 to n were taken for the data set, with 80% being assigned to training data and the remaining 20% held out for testing. A cryptographically relevant system would need to be capable of learning off of a small subset of this data, but that is not necessary for feature viability exploration.

TABLE 1. ANN PERFORMANCE ON $n = 1000667$

Trial	**Iterations to 98% Testing Accuracy**	**Final Training Accuracy**	**Final Testing Accuracy**
1	1,081,000	100.00%	99.93%
2	1,177,000	99.45%	99.71%
3	1,728,000	98.62%	98.02%

For each feature set, three networks were independently initialized and trained in order to study the sensitivity of the final system to variations in starting configuration. An n value of 1000667 was selected since it is a strong prime (see Section 1.3) which is large enough to allow for plentiful training instances, but not so large as to make network training times prohibitively long.

To establish a performance baseline, a network was trained given only the binary representations of a and n. This network was not able to outperform random guessing on the test data set. Switching to a feature set inspired by the Jacobi symbol algorithm gave more promising results. Using variable names as outlined in Algorithm 1, these features included $e \pmod 2$, $n \pmod 8$, $n \pmod 4$, $a_1 \pmod 4$, and $n \pmod{a_1}$ – each normalized to the range [0,1]. For better numerical stability and transferability of learning, the $n \pmod{a_1}$ feature was computed as a boolean corresponding to the question "is $n \pmod{a_1}$ equal to 1". These features were computed at every level of the recursive algorithm, which is feasible since the maximum recursive depth of the algorithm is $\lceil \log_2 n \rceil$. Since five features are provided per iteration, the total feature count is still polynomial in the length of the input, and therefore represents a cryptographically viable feature space. One consideration when implementing this feature set is that normally the recursive algorithm would terminate after a_1 reaches a value of 1 or 0. Since all feature vectors to the neural network must be the same length, once a_1 reaches one of these termination values the feature vector [0,1,2,2,1] is appended until the necessary number of iterations has been completed. These values were chosen since they would leave the output of the true Jacobi algorithm unchanged if they were to appear during one of its iterations. Initial training results are given in Table 1.

While the results in Table 1 make it clear that the neural networks were not a computationally efficient substitute for simply using the plain Jacobi algorithm, the trained networks were able to re-train to other values of n trivially, indicating that they are finding the generalizable rules we know to exist from the Jacobi algorithm. The transfer training results are given in Table 2.

TABLE 2. ANN PERFORMANCE WHEN RETRAINED FOR NEW n

Network ID	n	Initial Testing Accuracy	Training Iterations to 98% Testing Accuracy	Final Testing Accuracy
1	524387	99.97%	0	99.97%
1	786803	99.93%	0	99.82%
1	1048127	66.43%	<1,000	99.92%
2	524387	99.75%	0	99.86%
2	786803	99.66%	0	99.84%
2	1048127	66.33%	<1,000	99.75%
3	524387	98.70%	0	98.49%
3	786803	98.10%	0	98.64%
3	1048127	64.98%	22,000	98.31%

The networks here appear to transfer more readily to smaller values of n than to larger ones. This is likely due to the fact that, when trained on smaller n values, the final features (representing the maximum possible recursion depth) are not necessary and so not tuned properly. Nevertheless the networks appear to be able to quickly integrate them once they become necessary. Also worth noting is that networks which take longer to reach peak performance also tend to have lower peak performance, as well as requiring longer to retrain. Such networks are likely finding locally optimal solutions which are more complex than the true algorithm, suggesting that it is worthwhile to try multiple initializations when searching for more complex solutions in the composite case.

3.2. Composites

3.2.1. Jacobi Algorithm Features. While the Jacobi symbol is an effective mechanism for differentiating quadratic residues from non–residues in a prime modular base, it is insufficient for a composite modular base, providing only a 50% probability of success if $n = pq$ where p and q are prime. This does not mean, however, that a neural network trained with the modular discriminants used by the Jacobi symbol algorithm will necessarily fail. With that in mind, the inputs which led to the results summarized in Table 1 were employed to train new networks for the composite case given the basis $347 * 359 = 124573$. For this experiment the dataset consisted exclusively of residues and non–residues with Jacobi symbol equal to one, meaning that simply mimicking the Jacobi algorithm would not be of any help in solving the problem. These results are summarized in Table 3.

TABLE 3. ANN PERFORMANCE ON $n = 124573$

Trial	Iterations to 60% Testing Accuracy	Training Accuracy	Testing Accuracy
1	116,700	88.00%	65.77%
2	146,000	86.22%	64.58%
3	103,100	88.89%	64.66%

Surprisingly, the accuracy of the neural networks was well in excess of 50% despite that the Jacobi algorithm itself should have been of no use. In an effort to determine whether final accuracy was impacted by network architecture, the experiment was rerun in several different configurations. First, the hyperbolic tangent activation function was exchanged for a rectified linear one (RELU-6), resulting in network performance given in Table 4.

TABLE 4. ANN PERFORMANCE ON $n = 124573$ USING THE RELU–6 ACTIVATION FUNCTION.

Trial	Iterations to 60% Testing Accuracy	Training Accuracy	Testing Accuracy
1	44,100	83.56%	63.88%
2	27,300	84.89%	64.67%
3	140,400	71.56%	61.39%

It appears that despite the recent popularity of the RELU activation function in image processing applications, the hyperbolic tangent function is more suitable for our particular application; providing on average an absolute gain of 2.5% testing accuracy. The RELU networks did, however, train dramatically faster than the hyperbolic tangent networks. The third trial was an exception, taking a very long time to reach 60% accuracy. This appears to be due to the fact that it became stuck in a very unfavorable local minimum, causing 60% to be close to its asymptotic final performance level. If training speed is desired over final performance, initializing a large number of RELU networks and keeping the best one may be the optimal choice.

Another key parameter of network architecture is size, both in the form of hidden layer depth and hidden layer breadth. To explore the impact of the former variable, hidden layer count was increased from 4 to 5. The result was virtually no change in testing accuracy over the baseline. To investigate the latter variable, a network having 125 neurons per layer (rather than 100) was also trained. It too failed to show accuracy improvements any larger what would be expected from random noise, despite being much more computationally intensive to train. This being the case, the smaller system was deemed suitable for the purposes of the investigation.

Although the Jacobi symbol itself should be of no use when testing against a composite base, it was conjectured that networks pretrained on the Jacobi algorithm would provide a better starting point than random initialization for differentiating quadratic residues. To determine whether this was indeed the case, the networks from Table 1 were used as the starting points for training the new quadratic residue detection systems. The results of this experiment are given in Table 5.

TABLE 5. ANN PERFORMANCE ON $n = 124573$ INITIALIZED FROM JACOBI NETWORKS

ID	Iterations to 60% Testing Accuracy	Training Accuracy	Testing Accuracy
1	39,700	93.78%	65.25%
2	40,600	93.33%	65.17%
3	24,200	93.33%	64.84%

Although the pre-initialized networks did not attain a higher testing accuracy than those which were randomly initialized, they did approach this value 3 times faster, requiring approximately 74,000 fewer iterations to reliably exceed 60%. Although the network training started out much faster based on this pre–training, hundreds of thousands of iterations were still required to fully re–train the network. This is weak evidence that whatever data property is being exploited by the system is not directly connected to the decisions made in the Jacobi algorithm, though something about the architecture of networks which have learned that algorithm is clearly valuable.

One obvious problem with these approaches is that the number of training iterations needed to achieve better than random guessing is larger than the value of n being tested. In order to be useful for cryptographic applications, a neural network would need to be trained on one value of n and then be re–tasked to different n values with minimal retraining. To determine whether this is feasible, a network was first trained on $n = 124573$ and then was retrained on an n value of 263*479 = 125977. The outcome is recorded in Table 6.

TABLE 6. ANN PERFORMANCE ON $n = 125977$ TRANSFERED FROM PRE–TRAINED NETWORKS

Trial	Iterations to 60% Testing Accuracy	Training Accuracy	Testing Accuracy
1	14,500	90.22%	64.23%
2	17,300	88.00%	64.07%
3	16,200	93.33%	64.68%

Although the final testing accuracy of the transfer learning system was no better than before, the training accuracy did increase towards this asymptote much more rapidly. Whereas the original training required an average of 108966 iterations to achieve 60% testing accuracy, the retrained system was able to achieve this accuracy after an average of only 16000 iterations: nearly 7 times faster. This processes is still slower than simply factoring n, but if the underlying pattern being exploited by the network can be better understood it may be possible to either extract an explicit algorithm or else design networks which retrain more quickly in the future.

Finally, additional features can also be added on top of the Jacobi algorithm comparators in an effort to boost performance. Since it is not yet known how the neural networks are achieving their improved performances, it is not clear what features should be added. Testing a variety of modular comparators at random, however, reveals that computing $n \pmod 7$ at each stage of recursion has a substantive impact on accuracy. These improvements are summarized in Table 7.

TABLE 7. ANN PERFORMANCE ON $n = 124573$ WITH ADDITIONAL FEATURE: $n \pmod 7$

Trial	Iterations to 60% Testing Accuracy	Training Accuracy	Testing Accuracy
1	54,800	99.56%	70.71%
2	65,800	98.22%	70.10%
3	56,700	99.11%	71.17%

Adding a single feature, $n \pmod 7$, at each level of recursion lead to an average testing accuracy increase of 4.77%, significantly beyond the influence of random noise variations for the system. It also cut the required training iterations to reach 60% accuracy nearly in half. Other features were also investigated, computing n and a against modular bases of 3, 6, 9, 11, 13, and 17. The even/odd parity of a was also investigated as a feature. None of these individually yielded gains as significant as did the introduction of $n \pmod 7$, though in combination these features pushed the testing accuracy up to 75.02%. It is hoped that other as–of–yet unexplored features might be able to further increase testing accuracies.

In preparation for testing on a wider variety of n values, the network architecture was modified so that each hidden layer contains a number of neurons equal to the number of input features. For the values of n previously discussed this means 102 neurons, an insignificant change given that an increase to 125 neurons was already shown to have very little impact. Performance for these various n values is summarized in Table 8.

TABLE 8. ANN PERFORMANCE VARIOUS n

n	Factors	Iterations to 60% Testing Accuracy	Final Testing Accuracy
124,573	347 * 359	54,800	70.71%
125,321	7 * 17903	<100	100.00%
126,109	23 * 5483	49,100	71.72%
603,241	719 * 839	130,500	80.13%
848,329	863 * 983	460,000	79.29%
854,941	839 * 1019	1,530,000	62.81%
995,893	839 * 1187	1,857,000	61.15%
1,076,437	839 * 1283	2,239,000	61.82%
1,307,377	1019 * 1283	>3,262,000	50.67%
1,551,937	1019 * 1523	>4,821,100	57.38%

There are several interesting phenomena captured in Table 8. First, there was a network which achieved 100% performance, and did so extremely quickly. It is believed that this high level of performance was achieved because one of the factors of n was 7, and the data mod 7 is one of the feature values computed along the way. To see whether small factors in general lead to high performance, an n value was tested having 23 as one of its factors. This network behaved virtually identically to the network trained on an n with two factors around 350, beating out the larger–factor network by only 1%. While this might have indicated a weak pattern of decline proportional to factor size, a network with factors of 719 and 839 managed a testing accuracy of over 80%. It's not clear why the performance jumped so high for this value of n, but it does indicate that the system is able to handle at least certain subsets of larger factors. On the other hand, system performance took a precipitous hit on larger numbers, to the point where it could do barely better than random guessing for $n = 1307377$. These results are complicated by the fact that the n values above 603241 were trained using a different implementation of the system which generated data on the fly in order to reduce memory consumption. When that same technique was applied to $n = 1551937$ it also failed; the 57% accuracy instead being achieved by a highly memory intensive implementation. This may indicate that the decrease in performance observed for the largest n values is a byproduct of implementation difficulties rather than representative of any trend inherent to the problem. Larger n values of 32188213, 860755297, and 25840758901 were also tested, but they had to be terminated due to resource constraints before any significant progress was made.

This brings to the forefront a significant drawback of these results: even if the observed ability to achieve greater than 50% accuracy holds for cryptographically sized values of n, it would be completely infeasible to train such networks. Furthermore, experiments providing the neural networks with only a small ($\log n$ or $\sqrt{n}$) subset of the available data for training consistently failed. This could be evidence that, when successful, the networks are only memorizing patterns unique to each value of n rather than learning a generalizable set of rules for detection. On the other hand, if that were the case then pretraining a network on one n value should not have improved the learning rate when transferring to other n values, as it did in Tables 5 and 6. Unfortunately, efforts to demonstrate transferability of learning amongst the larger n values failed. When attempting to re–train networks which had previously been trained for $n = 603241$, $n = 854941$, and $n = 1076437$, all attempts failed to even move beyond a training accuracy of 50%, let alone having any impact on testing performance. This is what one would expect to see if the step size employed on each training iteration was forced to be very small, a potential byproduct of the way in which the TensorFlow library handles reloading of saved networks. Finding a work–around for this problem and further investigating the transferability of systems at large n values could provide useful insights for future work seeking to discover how these networks are able to generate such unexpectedly high accuracies.

3.2.2. Alternative Features. A fundamentally different approach to quadratic residue detection is to compute $a^x \pmod{n}$ for various values of x and use the resulting values for features. Such features are inspired by the fact that for prime n, $a^{\frac{n-1}{2}} \pmod{n}$ perfectly separates quadratic residues from non–residues. It appears, however, that not all possible powers x contain information useful for solving the quadratic residue problem. Consider an n of the form $n = pq$ where $p = 2r + 1$ and $q = 2t + 1$. Based on network performance testing, features of the form $a^x \pmod{n}$ appear to be useful if and only if x is of the form $m * r$ or $m * t$ for $m \in Z^+$. Features of the form $m * r + y$ and $m * t + y$ can also be useful when paired with y. If no features of those forms are provided then the network is held around 50% testing accuracy. No method was found which could reliably acquire useful features without introducing $O(\sqrt{n})$ candidates to the feature space, at which point trial division is more efficient.

Several other input feature schemes were also considered, none of which lead to effective ANNs. First was the use $a \pmod{i}$ and $n \pmod{i}$ for a logarithmic number of small values of $i \in Z^+$. The hope was that the system would find a pattern based on some previously unconsidered implication of the Chinese Remainder Theorem, though the neural networks were unable to make any headway there.

A wide variety of Fibonacci sequence values modulo different bases were also considered, for example $\text{Fib}_{a+1} \pmod{n}$. These were tested since similar features appear in an unproven primality testing heuristic titled the PSW conjecture, and it was hoped that their utility might extend to other number theory problems. Unfortunately the neural networks were unable to make effective use of these features either.

4. Conclusion

This work has shown that neural networks are capable of learning to mimic the Jacobi symbol algorithm in a way which is transferable to modular bases on which the network was not trained. While these networks are less efficient than the standard Jacobi algorithm, and therefore not of any pragmatic utility, this serves as another example of the ability of neural networks to converge on a moderately complicated algorithm. More importantly, this system also provides a launching point for addressing more complex problems.

Using Jacobi algorithm features to drive machine learning, it appears that a significant improvement over random guessing may be achieved in the problem of quadratic residue detection for composite numbers of the form $n = pq$. Such a result challenges the conventional wisdom of the community, which suggests that quadratic residue detection is as unguessable as a coin toss. [13] [14] There are several key considerations, however, which restrain the impact of this apparent result. First and foremost is that cryptographically relevant n values tend to be on the order of 500 to 1000 bits. The numbers tested during this investigation are only on the order of 17 to 21 bits. It is possible, therefore, that any patterns detected on the small scale are not present at the larger scale. This would have been investigated in greater detail if not for a second problem: training the neural networks from scratch appears to require $O(n)$ data instances and iterations. Using only a personal computer, it is not possible to train a network on such a large set of data due to both memory and time constraints. To fully train a network on a cryptographically relevant scale would be intractable even on a supercomputer unless a more efficient convergence mechanism can be constructed. Based on the experiments presented here, using a deeper network with simpler activation functions could be one way to cut down the training iteration requirement. Another possible avenue of attack would be to take a random–forest style approach to the problem, providing numerous neural networks with only a small amount of training and then taking a majority vote to generate the classification. One potential problem with such a method is that, if the networks are learning some underlying algorithm for residue detection, their errors may be interdependent and thus the ensemble system may not see improved performance. Moreover, it may still require $O(n)$ training iterations simply to exceed 50% accuracy, in which case the ensemble method can offer no advantage.

Given the relatively small size of the numbers investigated during this study, and the resource–intensive training requirements of the neural networks, a much faster solution to quadratic residue detection would have been to simply factor n and then directly compute the residue status. It is worth noting, however, that factoring algorithms have been under development for centuries whereas neural networks provide new avenues of possibilities. Just as factoring algorithms have become more and more efficient over time, it is possible that this methodology might be significantly improved upon in order to decrease resource requirements and further improve accuracy. Over the course of a relatively small number of experiments, features were discovered which provided an absolute testing accuracy performance boost of up to 10% on top of what was already provided by the Jacobi symbol features. If a polynomial–time algorithm for QR detection exists, neural networks of this form could allow for an easy way to identify what sort of computations are involved in it. By studying the impact of different features on network performance, it may be possible to discover a previously unknown algorithm. This would in turn make the computational resource requirement of the neural networks a moot point and allow for solutions to even cryptographically sized inputs. Although neural networks are conventionally treated as black–box systems, there is no reason that they could not be used as relevant–feature detectors. Future work should therefore focus on the search for additional features, as well as on performance optimizations so that applicability on larger n may be investigated.

References

[1] K. H. Rosen, *Elementary number theory and its applications*. Boston: Addison-Wesley, 2011.

[2] M. Lai, "Giraffe: Using deep reinforcement learning to play chess," *CoRR*, vol. abs/1509.01549, 2015. [Online]. Available: http://arxiv.org/abs/1509.01549

[3] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016. [Online]. Available: http://dx.doi.org/10.1038/nature16961

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[5] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" *CoRR*, vol. abs/1411.1792, 2014. [Online]. Available: http://arxiv.org/abs/1411.1792

[6] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, "Decaf: A deep convolutional activation feature for generic visual recognition," *CoRR*, vol. abs/1310.1531, 2013. [Online]. Available: http://arxiv.org/abs/1310.1531

[7] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "CNN features off-the-shelf: an astounding baseline for recognition," *CoRR*, vol. abs/1403.6382, 2014. [Online]. Available: http://arxiv.org/abs/1403.6382

[8] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, pp. 303–314, 1989. [Online]. Available: http://dx.doi.org/10.1007/BF02551274

[9] L. J. Ba and R. Caurana, "Do deep nets really need to be deep?" *CoRR*, vol. abs/1312.6184, 2013. [Online]. Available: http://arxiv.org/abs/1312.6184

[10] D. R. Stinson, *Cryptography Theory and Practice*, 3rd ed. Waterloo: CRC Press, 2006.

[11] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*, 5th ed. Waterloo: CRC Press, 2001.

[12] T. Jager and J. Schwenk, "The generic hardness of subset membership problems under the factoring assumption," Cryptology ePrint Archive, Report 2008/482, 2008. [Online]. Available: http://eprint.iacr.org/

[13] R. L. Rivest, *Advances in Cryptology — ASIACRYPT '91: International Conference on the Theory and Application of Cryptology Fujiyosida, Japan, November 1991 Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, ch. Cryptography and machine learning, pp. 427–439. [Online]. Available: http://dx.doi.org/10.1007/3-540-57332-1-36

[14] M. Kearns and L. Valiant, "Cryptographic limitations on learning boolean formulae and finite automata," *J. ACM*, vol. 41, no. 1, pp. 67–95, jan 1994. [Online]. Available: http://doi.acm.org/10.1145/174644.174647

[15] R. A. Mollin, *An Introduction to Cryptography*, 1st ed. New York: CRC Press, 2001.

[16] R. Rivest and R. Silverman, "Are 'strong' primes needed for RSA," 2001.