Linear Classification on Noisy Hardware

A Project Report

submitted by

VENKATA SAI CHELAGAMSETTY

in partial fulfilment of the requirements
for the award of the degree of

BACHELOR OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, MADRAS.
May 2019

THESIS CERTIFICATE

This is to certify that the thesis entitled **Linear Classification on Noisy Hardware**, submitted by **Venkata Sai Chelagamsetty** (**EE18B042**), to the Indian Institute of Technology, Madras, for the award of the degree of **Bachelors of Technology**, is a bona fide record of the research work carried out by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr.Avhishek Chatterjee
Research Guide
Assistant Professor
Dept. of Electrical Engineering
IIT-Madras, 600 036

Place: Chennai

Date:

ACKNOWLEDGEMENTS

I would like to express my gratitude to my guide, Dr. Avhishek Chatterjee for presenting us with such an interesting problem statement on noisy learning in hardware, which combines machine learning with communication. I would also like to thank my partner, Prakrithi Pradeep, a Computer Science student at Carnegie Mellon University for working with me throughout the project.

ABSTRACT

KEYWORDS: Linear Classifier, Noisy Computation

We investigate the impact of hardware noise and quantization errors on the accuracy of inference using linear classifiers, motivated by the growing interest in machine learning on nanoscale edge devices. Our tests utilising well-accepted models for hardware noise and defects on synthetic and actual data sets reveal that they have a considerable impact on accuracy. An easily implementable technique for reducing those impacts is presented by integrating concepts from linear classification, convex analysis, and concentration of measure. This basic method greatly improves performance in both synthetic and actual data sets, according to tests.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

DDF	DIVIDE, DECIDE and FUSE
CMOS	Complementary Metal Oxide Semiconductor
SNR	Signal-to-Noise Ratio

NOTATION

r	Radius
a	Angle of thesis in degrees
Z	Random Variable
x	Data points
w	Weight Vector
Y	Bernoulli Random variable
$\underline{\mathbf{1}}$	All 1 vector

CHAPTER 1

INTRODUCTION

1.1 Motivation and Problem Statement

Machine learning is projected to infect every edge device connected to the next high-speed 5G networks. Many of these devices will be used for life-saving applications such as health monitoring, elderly and child care, and industrial sensing. As a result, precise execution of machine learning algorithms on these edge devices is critical. Edge devices, on the other hand, must be inexpensive, quick, energy efficient, and tiny in order to meet the needs of the intended applications. As a result, nanoscale CMOS and beyond-CMOS technologies were a perfect fit for these devices. However, they are frequently boisterous. Quantization errors also affect the computations on these devices, as their memories are severely constrained to ensure tiny form factors. This raises the question: Can reliable learning and inference be done on the noisy and error prone hardware of edge devices? This forms the basis for our motivation.

One might ask the question why we are interested in linear classifiers. The main reason behind asking this question is that linear classifiers are the simplest and the most prevalent non-neural classifiers. Hence, they are the most suitable candidates for deployment on edge devices. Moreover, the last layer of many high performing neural classifiers, which is also one of the most sensitive layers, is essentially a linear unit. Hence, exploring the linear classifiers only of practical interest, it is also likely to boost our understanding of the aforementioned broader question.

CHAPTER 2

Approach

2.1 Noise Models

A unit vector $w \in \mathbb{R}^d$ and $b \in \mathbb{R}$ define a binary linear classifier for d-dimensional data. Using $sign(w^T x - b)$, this classifier determines the label of a data point $x \in \mathbb{R}^d$. The dot product is the most important computational step in a linear inference.

There are numerous implementations of dot products in upcoming nanoscale technologies. However, the hardware noise that those dot product circuits experience can be divided into two types: additive noise and random gate failures

Additive noise: The output of the dot product circuit in this case is $w^T x + Z$, where Z is a zero mean random variable. The variance of Z, or the energy of the noise, is proportional to the dot product dimension. The thermal noise of the dot product circuit is known to be well-modeled by Gaussian distributed Z.

Random gate failures: Gate failures are typical in low-energy nanoscale technology. One or more components of the dot product may not be captured in the final output due to short circuits and other faults, or the signs of those components may be flipped. This means that instead of $w^T x$, the dot product circuit's final output would be $\sum_{i=1}^{d} w_i \, x_i \, Y_i$, with $\{Y_i\}$ being either $\{0, 1\}$ or $\{\pm 1\}$ valued i.i.d. Bernoulli random variables.

2.2 Intuition

In this part, we use ideas from linear classifier characteristics, convex optimization, and probability theory to develop an understandable error mitigation technique.

2.2.1 Affine transformation of the data space

Consider the case where, in the absence of hardware noise, perfect linear classification is attainable, i.e. the dataset is linearly separable . Two hypotheses with convex and non-overlapping supports are used to characterise the underlying generative model for this data set . In this situation, the hyper-plane that is equidistant from the supports is the best linear classifier (i.e., the one with the highest margin). This hyper-plane is also perpendicular to the line connecting the two nearest points of the corresponding supports. The data space can be affinely modified (in particular, orthonormal transformation and translation) in such a way that the best classifier is $sign(\bar{u}^T\tilde{x})$, here, $\bar{u} = \frac{1}{\sqrt{d}}\mathbf{1}$, where $\mathbf{1}$ all 1 vector, and $\tilde{x}$ is a modified data point. Furthermore, at the origin, the line connecting the two closest points of the corresponding supports (in converted space) intersects the classifier. So, from now on, we'll concentrate on the case where the best classifier is $sign(\bar{u}^Tx)$, data x originates from one of the two convex support sets distributions, and those two support sets are closest to each other near the origin.

2.2.2 Identifying vulnerable data points

It is obvious from the previous observations that in the presence of hardware noise, the data points closest to the origin are the ones most likely to be categorised incorrectly. Furthermore, because $\bar{u}$ is symmetric, there are no components with large absolute values near the origin. As a result, the components that contribute favourably (or negatively) to the dot product of a data point near the origin are likely to be well spread out and have similar contributions in terms of magnitude in the (noiseless) dot product. Let us assume that a data point x near zero has total positive and negative contributions in the (noiseless) dot product of S_+^x and S_-^x, respectively, and that $S_+^x > S_-^x$. Consider dividing the data point's d components into k groups of size $\frac{d}{k}$ each at random, provided k divides d. Because of the symmetry of the splitting and the fact that the components contribute roughly equally to the dot product, each group should have more positive (S_+^x/k) than negative (S_-^x/k) contributions to the noiseless dot product. This implies that when $S_+^x > S_-^x$, the sign of the (partial) dot product in each group is more likely to be positive. Similarly, one could argue that the sign of the dot product in each group is more likely to be negative when $S_+^x < S_-^x$.

2.2.3 Group-wise noisy dot products and their signs

Consider the noisy dot product, which captures hardware noise and quantization mistakes under additive noise. The sign of a group's (noisy) dot product can differ from that of the noiseless case in the presence of noise. Because when $S_+^x > S_-^x$, the noiseless dot product of a group is more likely to be positive, let that probability be $\frac{1}{2} + \alpha$ for some $\alpha \in (0, \frac{1}{2})$. Assume that the probability of noise flipping the sign of a group's dot product is δ. When $S_+^x > S_-^x$, the likelihood of a group's noisy dot product being positive is given by

$$(\frac{1}{2} + \alpha)(1 - \delta) + (\frac{1}{2} - \alpha)\delta = \frac{1}{2} + \alpha(1 - 2\delta)$$

As a result, for $\delta < \frac{1}{2}$, $\alpha > 0$ and sufficiently big k, when $S_{+}^{x} > S_{-}^{x}$, the likelihood of a group's (noisy) dot product being positive is no less than $\frac{1}{2} + \epsilon_{+}$, where $\frac{1}{2} + \epsilon_{+}$ grows with $\frac{1}{2} + \epsilon_{+}$. Similarly, for $\delta < \frac{1}{2}$, when $S_{+}^{x} < S_{-}^{x}$, the chance of a group's (noisy) dot product being negative is no less than $\frac{1}{2} + \epsilon_{-}$ for some $\epsilon_{-} > 0$. It may be claimed that $\delta < \frac{1}{2}$ is the sole instance of interest for error mitigation. Consider the situation where a group's noiseless dot product has the same sign as the overall noiseless dot product. In this scenario, the probability that the sign of a group-wise dot product is flipped by noise is strictly smaller than $\frac{1}{2}$ for any symmetric noise distribution such as Gaussian and uniform. Because a group's noiseless dot product is more valuable,On average, the probability of flip, i.e.,δ, is strictly smaller than $\frac{1}{2}$ and has the same sign as the overall noiseless dot product. The case where group-wise dot products are sent across a completely random binary symmetric channel is referred to as $\delta = \frac{1}{2}$ in information theory. Theoretically, error mitigation for the route is impossible.

2.2.4 Fusing the group-wise signs

When $|S_{+}^{x} - S_{-}^{x}|$ is constrained away from 0 owing to measure concentration, the difference between the number of positive and negative group-wise (noisy) dot products is $\geq \epsilon_{+}k - c\sqrt{k}$ ($\leq -\epsilon_{-}k + c\sqrt{k}$), with high probability, for some $c > 0$. As a result, the final label for x can be determined simply by counting the difference between the number of positive and negative dot products in each group. In practise, however, ϵ_{+} (ϵ_{-}) are not known before hand. Consider the worst-case scenario: when the likelihood of a group's (noisy) dot product being positive or negative is $\frac{1}{2}$, i.e., $\epsilon_{+} = \epsilon_{-} = 0$. Even in this instance, the difference in the number of groups with positive and negative (noisy) dot products by concentration of measure lies between $-c\sqrt{k}$ and $c\sqrt{k}$ with high probability, for some $c > 0$. In reality, where ϵ_+ (ϵ_-) are unknown before hand, a decision can be taken with high confidence whenever the absolute value of the difference between the number of groups with positive and negative (noisy) dot products is $\gg \sqrt{k}$. This leads to a natural error mitigation strategy which we refer to as DIVIDE, DECIDE AND FUSE strategy, and in short, DDF.

2.3 Error Mitigation Strategy

DIVIDE, DECIDE, AND FUSE: Using a random number generator, divide the d components of the classifier vector w into k groups of size d k$\frac{d}{k}$ each. (If k does not divide d, use zero padding to increase dimension.) Let's call those parts $w^{(1)}, w^{(2)}, \ldots, w^{(k)}$. Let $x^{(1)}, x^{(2)}, \ldots, x^{(k)}$ be the vectors corresponding to the components for any data point x. For some $C > 0$, declare the label for x to be positive if $\sum_{j=1}^{k} sign\left(w^{(j)^T} x^{(j)}\right) > C\sqrt{k}$ and negative if $\sum_{j=1}^{k} sign\left(w^{(j)^T} x^{(j)}\right) < -C\sqrt{k}$. In other circumstances, choose at random. This method was created with additive noise in mind, such as thermal noise and quantization mistakes. However, because random gate failures are symmetric across groups, the impact on each group's dot product is expected to be zero. DIVIDE, DECIDE, AND FUSE, on the surface, appears to alleviate mistakes caused by random gate failures. The following code snippet shows our implementation of the strategy with normal noise.

```
def test_perceptron_noise(X, Y, w, std,first,second)

i = 0

acc_error = 0

acc_total_sum = 0
```

```
total =  0
val = 0
for row in X:
    total_val=0
    for j in range(first):
        for k in range(second):
            total_val += (w[0][second*j+k]*row[(second*j)+k])
        noise = np.random.normal(0,std/first,1)
        total_val += noise[0]
        if (total_val >= 0):
            accum+= 1
        if (total_val < 0):
            accum-=1
        total_val = 0
    if (accum > int((first**0.5)+1)):
        pred = 1
    elif (accum < -int((first**(0.5)+1))):
        pred = -1
    else:
        result = np.random.binomial(1,0.5)
        if result == 1:
            pred = 1
        else:
            pred = -1
    accum = 0
    if (pred != Y[i]):
        acc_error += 1

    acc_total_sum += 1

    i+= 1

return (acc_total_sum-acc_error)/acc_total_sum
```

CHAPTER 3

Data Generation

We use three synthetic data sets, namely, symmetric separable data set, rotated separable data set, and asymmetric separable data set. The first data set is generated by sampling from two distributions: uniformly random from two nonoverlapping spheres centered respectively at $+\mathbf{1}$ and $-\mathbf{1}$, where $\mathbf{1}$ is the d-dimensional all 1 vector.

3.1 Skew

The distribution is skewed if one tail is longer than the other. Because they lack symmetry, these distributions are sometimes referred to as asymmetric or asymmetrical. Symmetry describes how one half of a distribution mirrors the other half. The normal distribution, for example, is a symmetric distribution with no skew. The tails are interchangeable. The skewed data set is generated by picking points from two skewed d dimensional distributions with non-overlapping supports and their modes are at $\pm\mathbf{1}$.

3.2 Rotation

For the simplest case of rotation in a two-dimensional plane, the rotated vector is related to the initial vector by

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos a & -\sin a \\ \sin a & \cos a \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

for a right-hand rotation through the angle a. The generator of this rotation is represented by the matrix $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$. The square of this matrix is the negative of the identity matrix, its cube is its own negative, and the full exponentiation is

$$\exp\left[a\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}\right] = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + a\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} - \frac{a^2}{2!}\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} - \frac{a^3}{3!}\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} + \cdots$$

$$= \cos a\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + \sin a\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$$

$$\exp\left[a\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}\right] = \begin{pmatrix} \cos a & -\sin a \\ \sin a & \cos a \end{pmatrix}$$

leading to the rotation matrix given above. The generator of the rotation can be written as an outer product of the two unit vectors along the x-axis and the y-axis,

$$\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}\begin{pmatrix} 1 & 0 \end{pmatrix} - \begin{pmatrix} 1 \\ 0 \end{pmatrix}\begin{pmatrix} 0 & 1 \end{pmatrix} = \hat{\mathbf{y}}\hat{\mathbf{x}}^{\mathrm{T}} - \hat{\mathbf{x}}\hat{\mathbf{y}}^{\mathrm{T}}$$

where the transposed vector on the right of each term is multiplied leftward onto each component of the vector to produce matrices that are added together. Because the previous definitions of generators in terms of outer products are written in vector notation, they may be used to explain rotation in any hyperplane formed by two n-dimensional vectors right away. Given two orthogonal unit vectors n1 and n2, which means

$$\mathbf{n}_1^{\mathrm{T}}\mathbf{n}_1 = 1; \quad \mathbf{n}_1^{\mathrm{T}}\mathbf{n}_2 = \mathbf{n}_2^{\mathrm{T}}\mathbf{n}_1 = 0 \quad \mathbf{n}_2^{\mathrm{T}}\mathbf{n}_2 = 1$$

the generator of rotations in the hyperplane spanned by the two vectors is

$$L_{\mathbf{n}_1\mathbf{n}_2} = \mathbf{n}_2\mathbf{n}_1^T - \mathbf{n}_1\mathbf{n}_2^T$$

Forming powers of thin generator,

$$\begin{aligned} L^2_{\mathrm{n}_1\mathrm{n}_2} &= -\left(\mathrm{n}_1\mathrm{n}_1^{\mathrm{T}} + \mathrm{n}_2\mathrm{n}_2^{\mathrm{T}}\right) \\ L^3_{\mathrm{n}_1\mathrm{n}_3} &= -\left(\mathrm{n}_2\mathrm{n}_1^{\mathrm{T}} - \mathrm{n}_1\mathrm{n}_2^{T}\right) \end{aligned}$$

the n dimensional rotation matrix is simply

$$\exp\left\{aL_{n_1n_2}\right\} = I + \left(\mathbf{n}_2\mathbf{n}_1^T - \mathbf{n}_1\mathbf{n}_2^{\mathrm{T}}\right)\sin a + \left(\mathbf{n}_1\mathbf{n}_1^{\mathbf{T}} + \mathbf{n}_2\mathbf{n}_2^{\mathbf{T}}\right)(\cos a - 1)$$

The extra negative sign from the neighbouring outer product accounts for the difference in sign in the last term when compared to Rodrigues' rotation formula. This nxn matrix can now be used to obtain the final value of an n-dimensional vector after rotation in the hyperplane given.

Here is a code snippet that generates skewed data after randomly sampling from Muller data.

```
import random

import numpy as np
import math


def skew_norm_pdf(x,e=0,w=1,a=0):
    t = (x-e) / w
    return 2.0 * w * stats.norm.pdf(t) * stats.norm.cdf(a*t)


def func(radius,dim):
    train_data = []
    train_labels = []
    test_data = []
    test_labels = []
    temp_labels = []
    temp_data = []
    num_points = 1000
    for i in range (num_points):
        #Muller's method: randomly sampling from d-ball
        location = 0.0
        scale = 1.0
        temp_arr = np.random.rand(dim)
        data_arr = skew_norm_pdf(temp_arr,location,scale,-3)
        norm = np.sum(data_arr**2)**(0.5)
        r = random.random()**(1.0/dim)
        data_point = r*data_arr/norm
        data_elem = []
        #positive or negative point with prob 0.5
        prob = 0.5
        rand_label = np.random.binomial(1,prob)

        #centre for positive ball is (1,1,..)
        #centre for negative ball is (-1,-1,..)
        for x in data_point:
            if (rand_label == 1):
                x = (x*radius) + 1
            else:
                x = (x*radius) - 1
                rand_label = -1
            data_elem.append(x)
        temp_labels.append(rand_label)
        temp_data.append(data_elem)
    train_num_points = 700

    #Training data
    for k in range(train_num_points):
        train_data.append(temp_data[k])
        train_labels.append(temp_labels[k])

    #Testing data
        for j in range(k,num_points,1):
        test_data.append(temp_data[j])
        test_labels.append(temp_labels[j])
    return (train_data,train_labels,test_data,test_labels)
```

CHAPTER 4

Evaluation

This section assesses the effectiveness of the suggested DIVIDE, DECIDE, AND FUSE strategy (DDF). We investigate the accuracy of final classification for several noise models at various strengths. The accuracy of a perceptron supplemented with DDF is demonstrated in Figures 6.1 for data dimensions 30 and 100, respectively. The tuples in the picture legends relate to different groupings of data vector components. The tuple's first number relates to k, whereas the second number corresponds to $\frac{d}{k}$. When compared to plain perceptron inference (referred to as "raw" in the figure), there is a considerable boost in test accuracy. Because uniform noise is the best model for quantization errors, we repeat the preceding experiments with uniform noise and present the comparisions in Fig 6.2. We also show how the raw perceptron works on the various datasets as in Fig 6.2. The major finding is the same as before: the DDF technique improves accuracy significantly. We introduced DDF by emulating the maximum margin classifier for linearly separable data sets. So it's only natural to wonder if DDF can help noisy linear classifiers perform better on data sets that aren't linearly separable. We employ DDF for naive Bayes' classifier in the face of hardware noise to find a solution to this query. We compare the performance of naive Bayes when reinforced with DDF to plain (raw) naive Bayes in Fig 6.3. The figures imply that DDF can help linear classifiers perform better on data sets that aren't linearly separable. We also present the tables which include the corresponding data for the dimensions other than the ones mentioned above.Table 6.5 shows Noise vs Radius (Variance) results on rotated data of dimension 40. Table 6.6 shows Noise vs Radius (Variance) results on rotated data of dimension 40 split into 4 and 10. Table 6.7 shows the same but the split is now 8 and 5. The following tables after the previous ones give the data for 50 dimension and their corresponding splits. The next set of tables are generated on the asymmetric dataset.

CHAPTER 5

Future Work

The primary lessons from this research are twofold: hardware noise has a large negative impact on classification, and simple implementable solutions based on the classifier's structure and data distribution can reduce this. Though the earliest findings appear promising, the investigation is far from finished. Our research presents several new questions that need to be investigated further. To begin, we ignored the affine transformation and randomly split the classifier vector w into k equal sized groups for energy, storage, and computing restrictions of edge devices. Prior to the change, the classifier vector w is unlikely to be symmetric. As a result, a weighted split of w, which accounts for component asymmetry, is expected to perform better, particularly at high SNR. Identification of the best asymmetric split method, selection of the best k, and comprehensive testing of various techniques on real data sets are all critical. Second, designing a provably accurate yet simple to implement error mitigation technique is a mathematical quest of practical interest. Even if the argument is only valid for a limited set of data distributions and noise models, it is still a good exercise because the research is likely to reveal more information about the problem. It's also interesting seeing if the structural ideas from the noisy classification problem can be merged with the coding schemes from to come up with a basic but deterministic system. Finally, because many state-of-the-art classifiers are based on deep neural networks, efficient error mitigation for noisy nanoscale neural networks is important.

CHAPTER 6

Graphs and tables

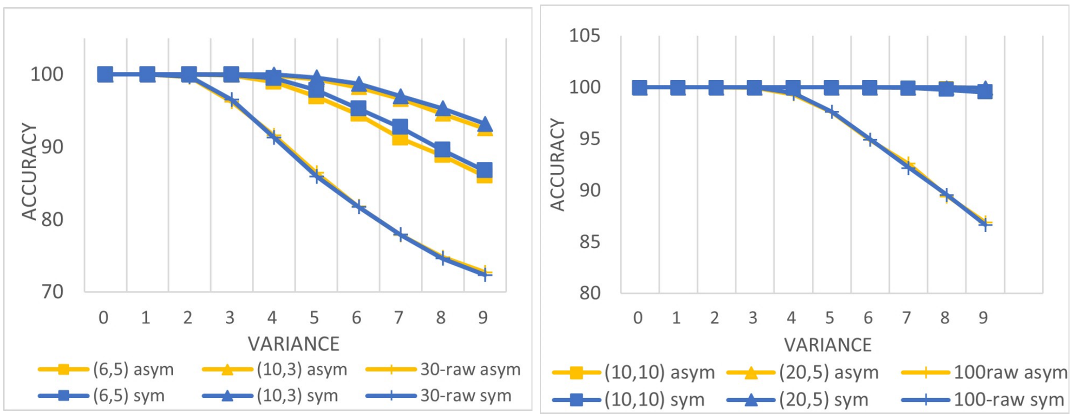


Figure 6.1: Perceptron (with DDF) with Gaussian Noise on symmetric and asymmetric data sets.

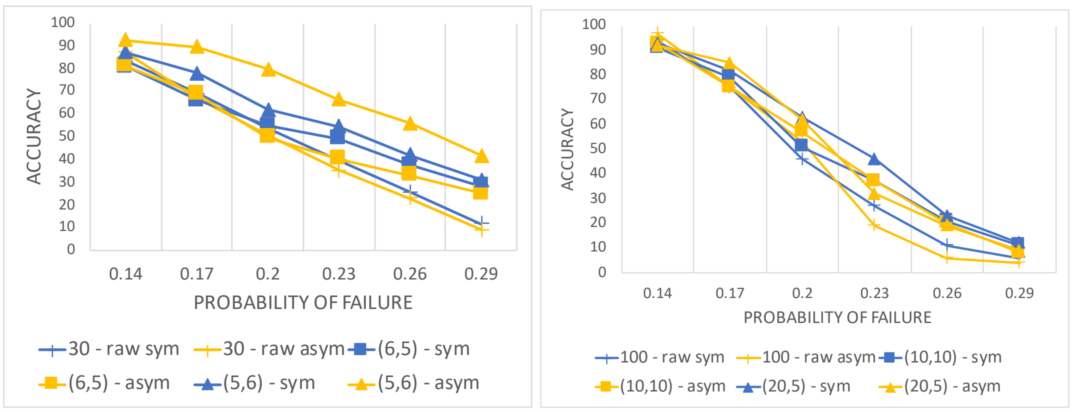


Figure 6.2: Perceptron (with DDF) with Binary Noise on symmetric and asymmetric data sets.

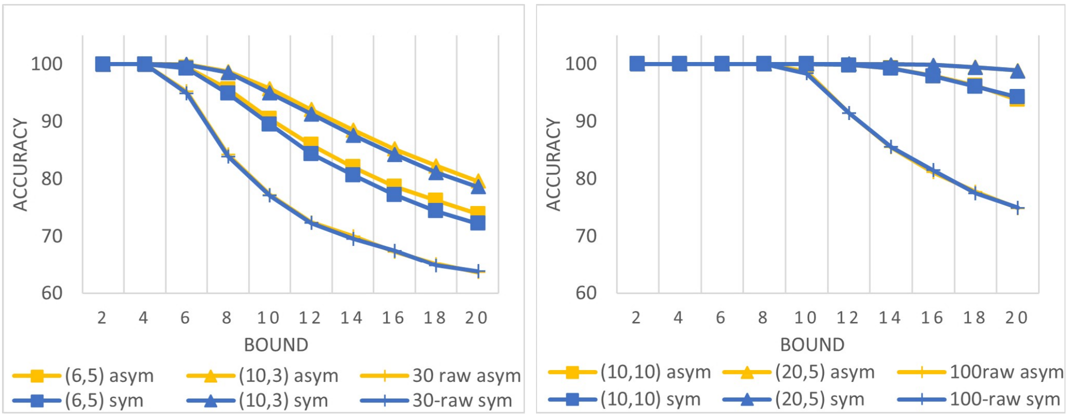


Figure 6.3: Perceptron (with DDF) with Binary noise on symmetric and asymmetric data sets.

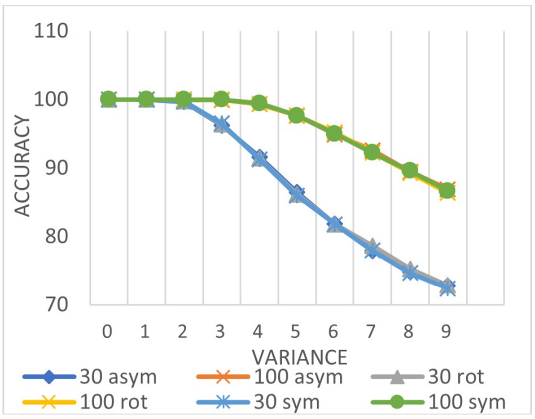
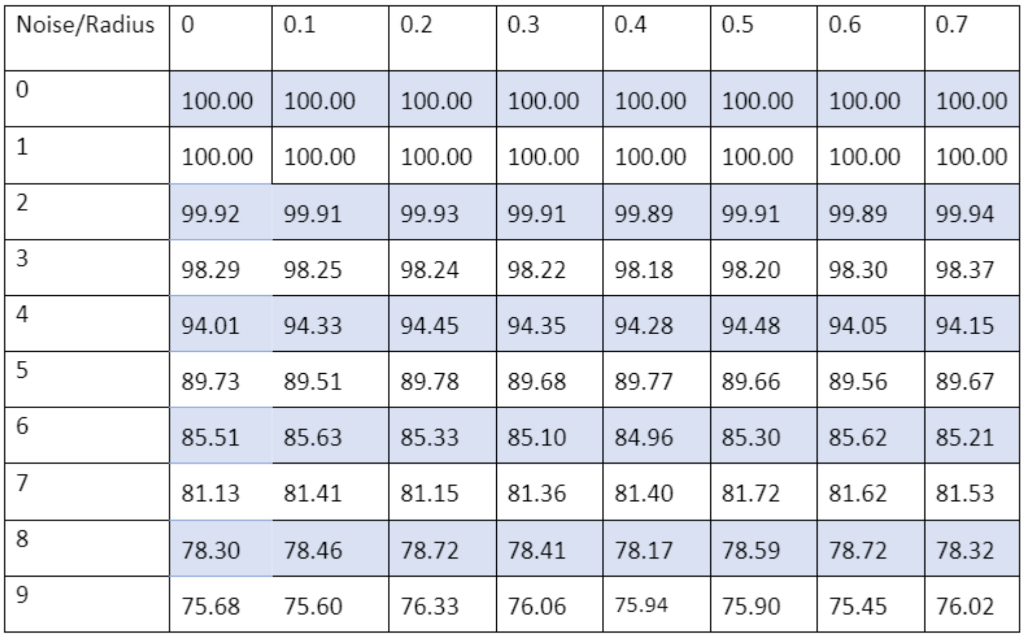
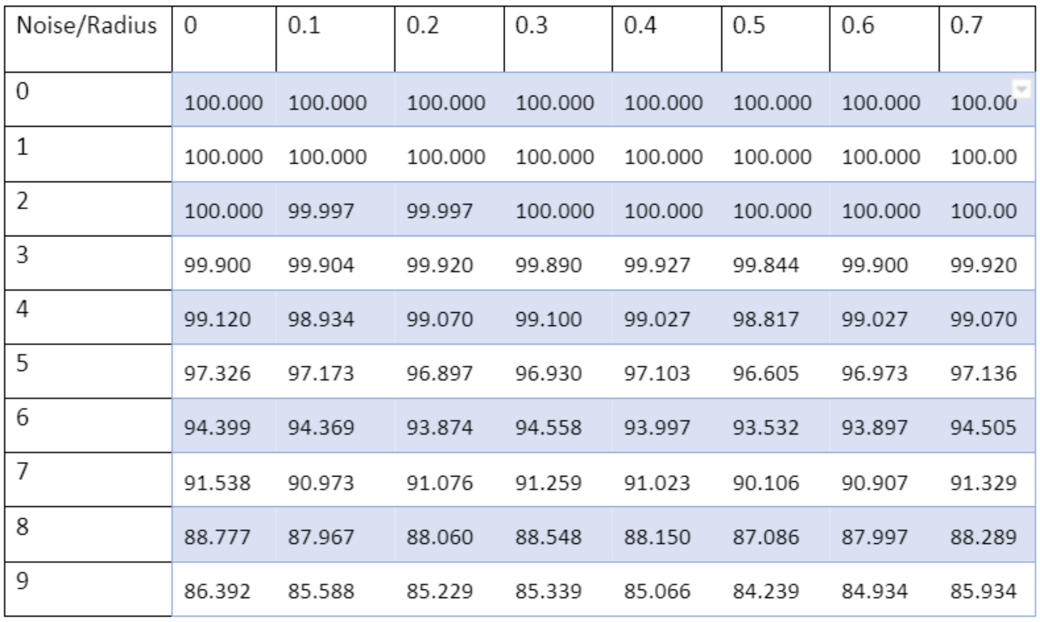


Figure 6.4: Perceptron on symmetric, asymmetric, rotated datasets with Gaussian noise.



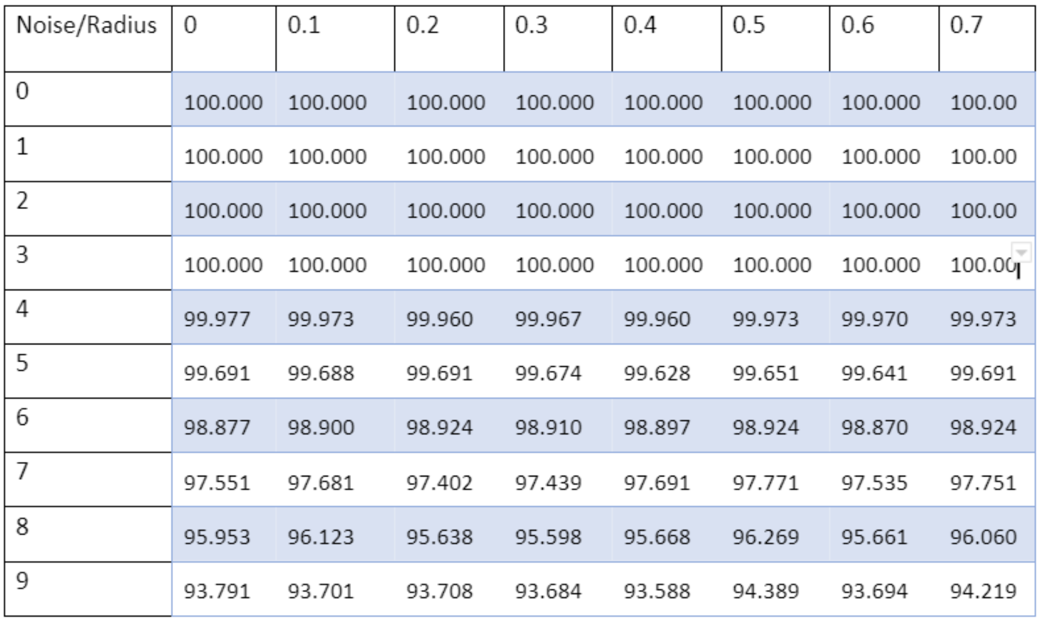
Noise/Radius	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7
0	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
1	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
2	99.92	99.91	99.93	99.91	99.89	99.91	99.89	99.94
3	98.29	98.25	98.24	98.22	98.18	98.20	98.30	98.37
4	94.01	94.33	94.45	94.35	94.28	94.48	94.05	94.15
5	89.73	89.51	89.78	89.68	89.77	89.66	89.56	89.67
6	85.51	85.63	85.33	85.10	84.96	85.30	85.62	85.21
7	81.13	81.41	81.15	81.36	81.40	81.72	81.62	81.53
8	78.30	78.46	78.72	78.41	78.17	78.59	78.72	78.32
9	75.68	75.60	76.33	76.06	75.94	75.90	75.45	76.02

Figure 6.5: Table showing Noise vs Radius (Variance) results on rotated data of dimension 40.



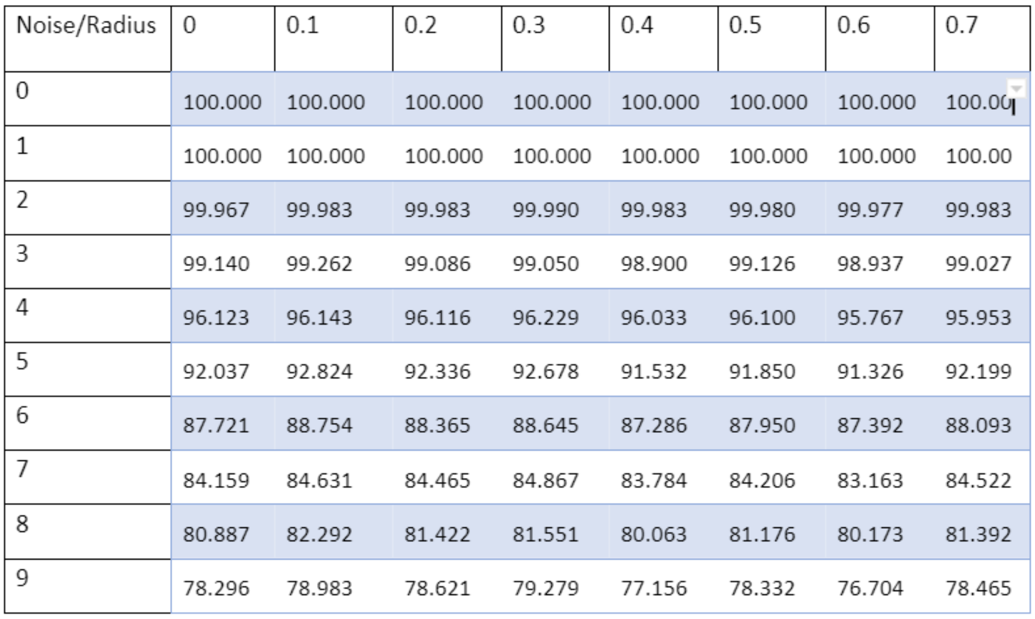
Noise/Radius	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7
0	100.000	100.000	100.000	100.000	100.000	100.000	100.000	100.00
1	100.000	100.000	100.000	100.000	100.000	100.000	100.000	100.00
2	100.000	99.997	99.997	100.000	100.000	100.000	100.000	100.00
3	99.900	99.904	99.920	99.890	99.927	99.844	99.900	99.920
4	99.120	98.934	99.070	99.100	99.027	98.817	99.027	99.070
5	97.326	97.173	96.897	96.930	97.103	96.605	96.973	97.136
6	94.399	94.369	93.874	94.558	93.997	93.532	93.897	94.505
7	91.538	90.973	91.076	91.259	91.023	90.106	90.907	91.329
8	88.777	87.967	88.060	88.548	88.150	87.086	87.997	88.289
9	86.392	85.588	85.229	85.339	85.066	84.239	84.934	85.934

Figure 6.6: Table showing Noise vs Radius (Variance) results on rotated data of dimension 40 split into 4 and 10.



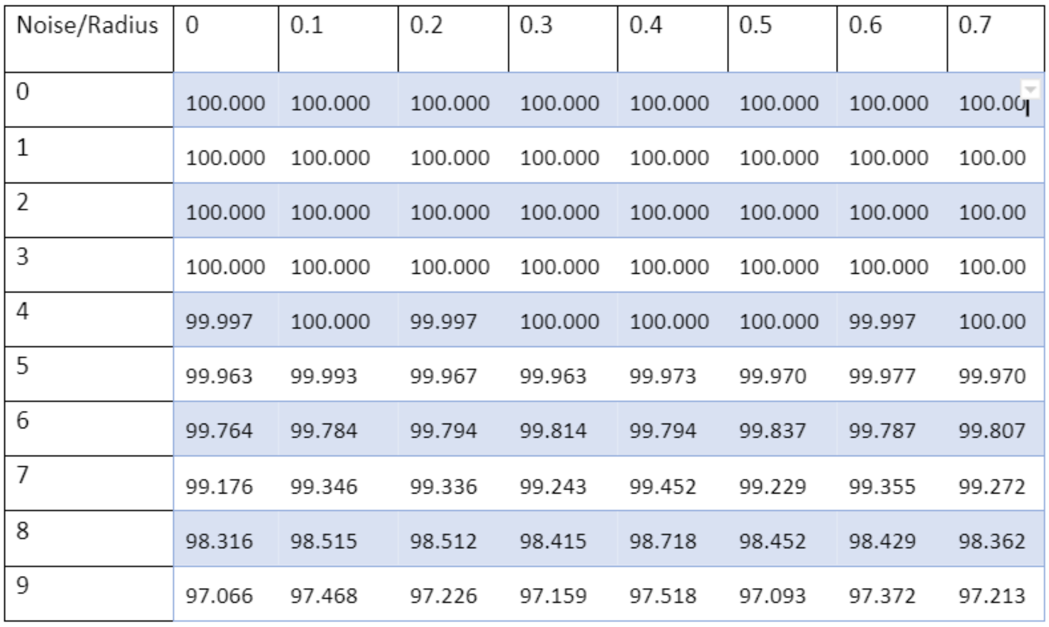
Noise/Radius	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7
0	100.000	100.000	100.000	100.000	100.000	100.000	100.000	100.00
1	100.000	100.000	100.000	100.000	100.000	100.000	100.000	100.00
2	100.000	100.000	100.000	100.000	100.000	100.000	100.000	100.00
3	100.000	100.000	100.000	100.000	100.000	100.000	100.000	100.00
4	99.977	99.973	99.960	99.967	99.960	99.973	99.970	99.973
5	99.691	99.688	99.691	99.674	99.628	99.651	99.641	99.691
6	98.877	98.900	98.924	98.910	98.897	98.924	98.870	98.924
7	97.551	97.681	97.402	97.439	97.691	97.771	97.535	97.751
8	95.953	96.123	95.638	95.598	95.668	96.269	95.661	96.060
9	93.791	93.701	93.708	93.684	93.588	94.389	93.694	94.219

Figure 6.7: Table showing Noise vs Radius (Variance) results on rotated data of dimension 40 split into 8 and 5.



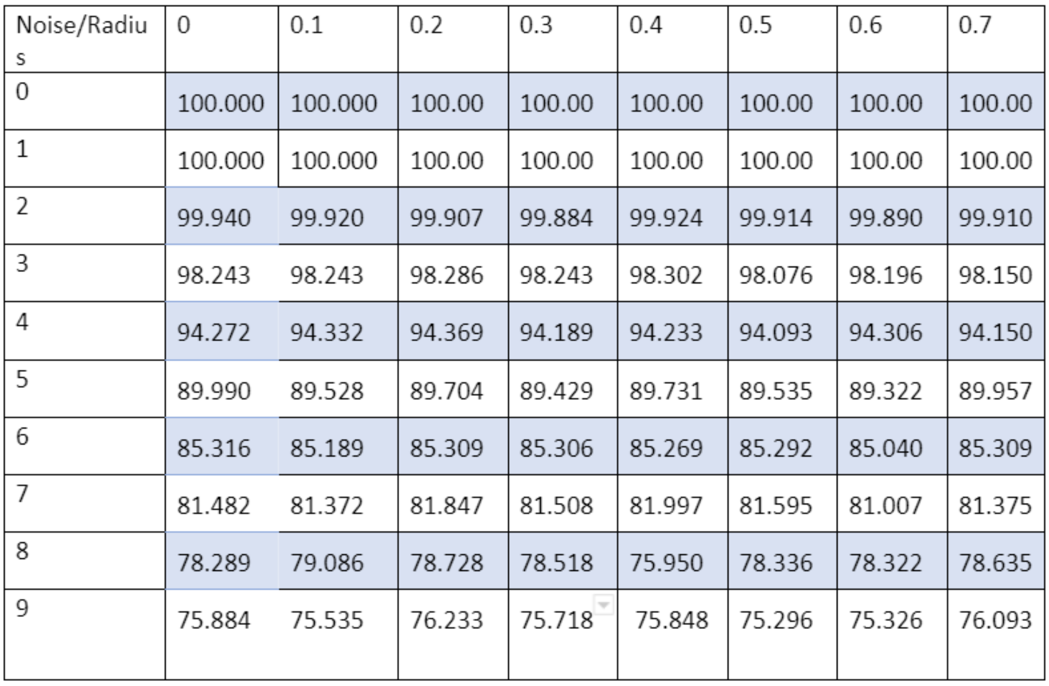
Noise/Radius	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7
0	100.000	100.000	100.000	100.000	100.000	100.000	100.000	100.00
1	100.000	100.000	100.000	100.000	100.000	100.000	100.000	100.00
2	99.967	99.983	99.983	99.990	99.983	99.980	99.977	99.983
3	99.140	99.262	99.086	99.050	98.900	99.126	98.937	99.027
4	96.123	96.143	96.116	96.229	96.033	96.100	95.767	95.953
5	92.037	92.824	92.336	92.678	91.532	91.850	91.326	92.199
6	87.721	88.754	88.365	88.645	87.286	87.950	87.392	88.093
7	84.159	84.631	84.465	84.867	83.784	84.206	83.163	84.522
8	80.887	82.292	81.422	81.551	80.063	81.176	80.173	81.392
9	78.296	78.983	78.621	79.279	77.156	78.332	76.704	78.465

Figure 6.8: Table showing Noise vs Radius (Variance) results on rotated data of dimension 50 split into 2 and 25.



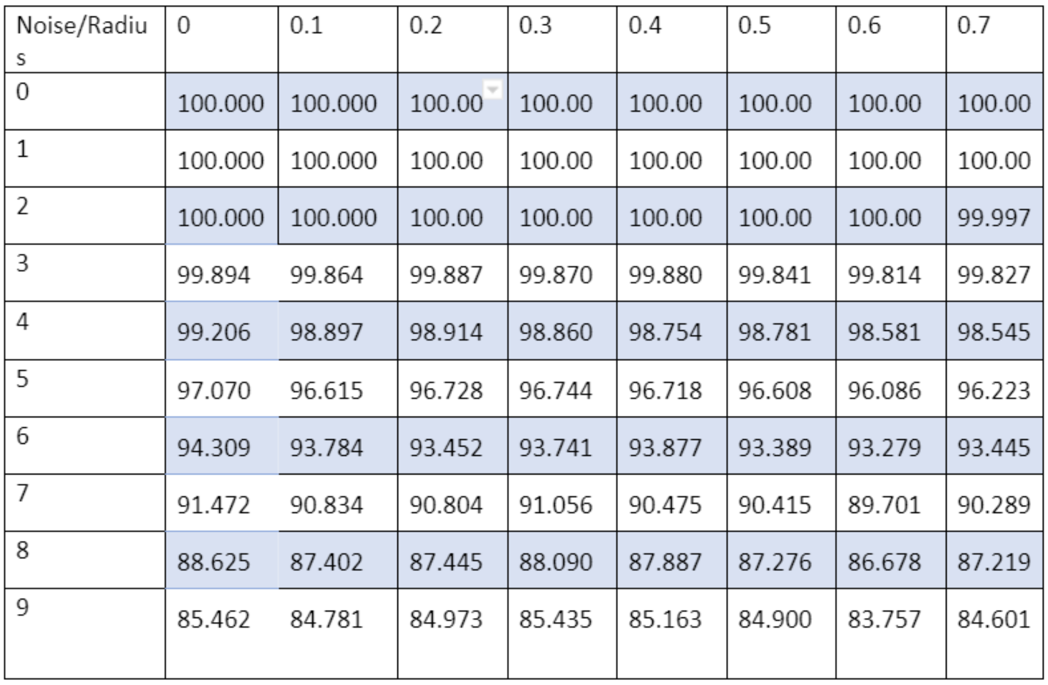
Noise/Radius	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7
0	100.000	100.000	100.000	100.000	100.000	100.000	100.000	100.00
1	100.000	100.000	100.000	100.000	100.000	100.000	100.000	100.00
2	100.000	100.000	100.000	100.000	100.000	100.000	100.000	100.00
3	100.000	100.000	100.000	100.000	100.000	100.000	100.000	100.00
4	99.997	100.000	99.997	100.000	100.000	100.000	99.997	100.00
5	99.963	99.993	99.967	99.963	99.973	99.970	99.977	99.970
6	99.764	99.784	99.794	99.814	99.794	99.837	99.787	99.807
7	99.176	99.346	99.336	99.243	99.452	99.229	99.355	99.272
8	98.316	98.515	98.512	98.415	98.718	98.452	98.429	98.362
9	97.066	97.468	97.226	97.159	97.518	97.093	97.372	97.213

Figure 6.9: Table showing Noise vs Radius (Variance) results on rotated data of dimension 50 split into 10 and 5.



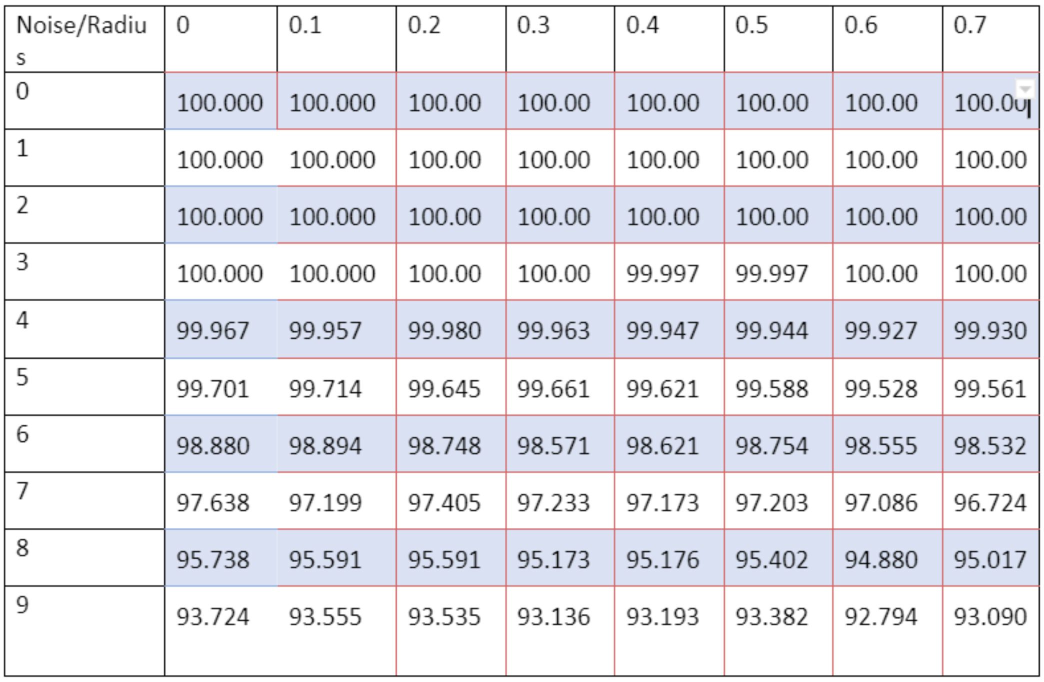
Noise/Radius	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7
0	100.000	100.000	100.00	100.00	100.00	100.00	100.00	100.00
1	100.000	100.000	100.00	100.00	100.00	100.00	100.00	100.00
2	99.940	99.920	99.907	99.884	99.924	99.914	99.890	99.910
3	98.243	98.243	98.286	98.243	98.302	98.076	98.196	98.150
4	94.272	94.332	94.369	94.189	94.233	94.093	94.306	94.150
5	89.990	89.528	89.704	89.429	89.731	89.535	89.322	89.957
6	85.316	85.189	85.309	85.306	85.269	85.292	85.040	85.309
7	81.482	81.372	81.847	81.508	81.997	81.595	81.007	81.375
8	78.289	79.086	78.728	78.518	75.950	78.336	78.322	78.635
9	75.884	75.535	76.233	75.718	75.848	75.296	75.326	76.093

Figure 6.10: Table showing Noise vs Radius (Variance) results on asymmetrical data of dimension 40.



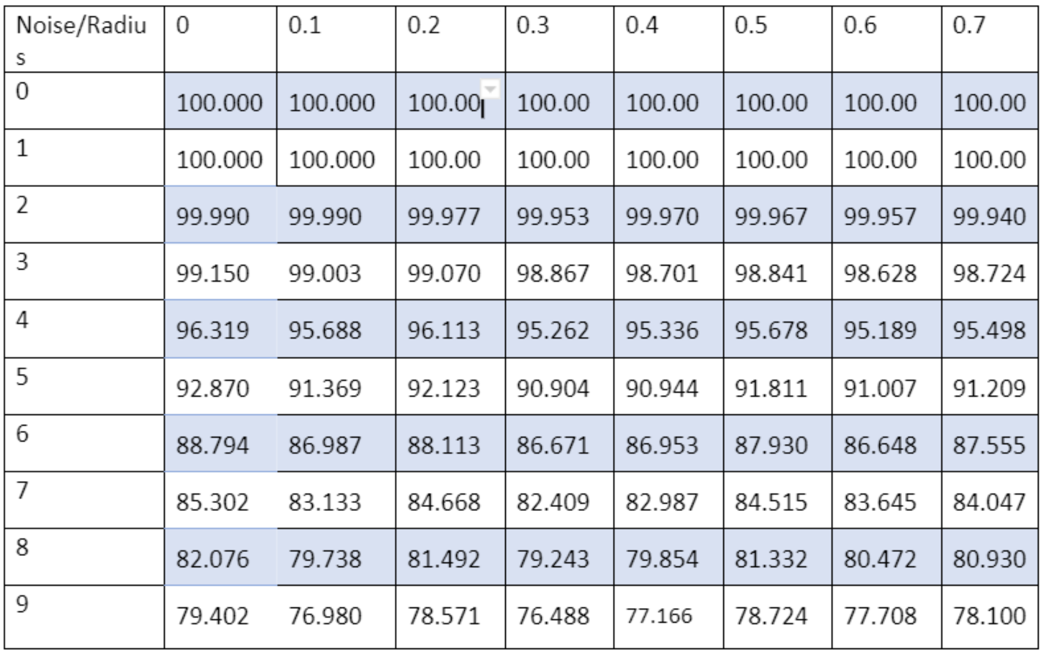
Noise/Radius	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7
0	100.000	100.000	100.00	100.00	100.00	100.00	100.00	100.00
1	100.000	100.000	100.00	100.00	100.00	100.00	100.00	100.00
2	100.000	100.000	100.00	100.00	100.00	100.00	100.00	99.997
3	99.894	99.864	99.887	99.870	99.880	99.841	99.814	99.827
4	99.206	98.897	98.914	98.860	98.754	98.781	98.581	98.545
5	97.070	96.615	96.728	96.744	96.718	96.608	96.086	96.223
6	94.309	93.784	93.452	93.741	93.877	93.389	93.279	93.445
7	91.472	90.834	90.804	91.056	90.475	90.415	89.701	90.289
8	88.625	87.402	87.445	88.090	87.887	87.276	86.678	87.219
9	85.462	84.781	84.973	85.435	85.163	84.900	83.757	84.601

Figure 6.11: Table showing Noise vs Radius (Variance) results on asymmetrical data of dimension 40 split into 4 and 10.



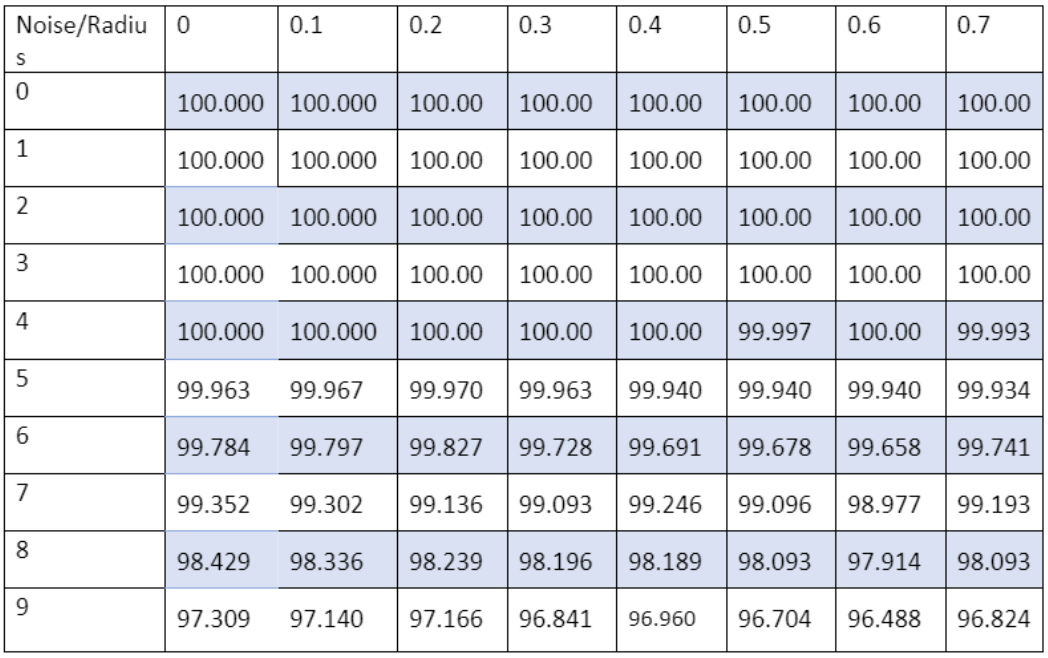
Noise/Radius	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7
0	100.000	100.000	100.00	100.00	100.00	100.00	100.00	100.00
1	100.000	100.000	100.00	100.00	100.00	100.00	100.00	100.00
2	100.000	100.000	100.00	100.00	100.00	100.00	100.00	100.00
3	100.000	100.000	100.00	100.00	99.997	99.997	100.00	100.00
4	99.967	99.957	99.980	99.963	99.947	99.944	99.927	99.930
5	99.701	99.714	99.645	99.661	99.621	99.588	99.528	99.561
6	98.880	98.894	98.748	98.571	98.621	98.754	98.555	98.532
7	97.638	97.199	97.405	97.233	97.173	97.203	97.086	96.724
8	95.738	95.591	95.591	95.173	95.176	95.402	94.880	95.017
9	93.724	93.555	93.535	93.136	93.193	93.382	92.794	93.090

Figure 6.12: Table showing Noise vs Radius (Variance) results on asymmetrical data of dimension 40 split into 8 and 5.



Noise/Radius	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7
0	100.000	100.000	100.00	100.00	100.00	100.00	100.00	100.00
1	100.000	100.000	100.00	100.00	100.00	100.00	100.00	100.00
2	99.990	99.990	99.977	99.953	99.970	99.967	99.957	99.940
3	99.150	99.003	99.070	98.867	98.701	98.841	98.628	98.724
4	96.319	95.688	96.113	95.262	95.336	95.678	95.189	95.498
5	92.870	91.369	92.123	90.904	90.944	91.811	91.007	91.209
6	88.794	86.987	88.113	86.671	86.953	87.930	86.648	87.555
7	85.302	83.133	84.668	82.409	82.987	84.515	83.645	84.047
8	82.076	79.738	81.492	79.243	79.854	81.332	80.472	80.930
9	79.402	76.980	78.571	76.488	77.166	78.724	77.708	78.100

Figure 6.13: Table showing Noise vs Radius (Variance) results on asymmetrical data of dimension 50 split into 2 and 25.



Noise/Radius	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7
0	100.000	100.000	100.00	100.00	100.00	100.00	100.00	100.00
1	100.000	100.000	100.00	100.00	100.00	100.00	100.00	100.00
2	100.000	100.000	100.00	100.00	100.00	100.00	100.00	100.00
3	100.000	100.000	100.00	100.00	100.00	100.00	100.00	100.00
4	100.000	100.000	100.00	100.00	100.00	99.997	100.00	99.993
5	99.963	99.967	99.970	99.963	99.940	99.940	99.940	99.934
6	99.784	99.797	99.827	99.728	99.691	99.678	99.658	99.741
7	99.352	99.302	99.136	99.093	99.246	99.096	98.977	99.193
8	98.429	98.336	98.239	98.196	98.189	98.093	97.914	98.093
9	97.309	97.140	97.166	96.841	96.960	96.704	96.488	96.824

Figure 6.14: Table showing Noise vs Radius (Variance) results on asymmetrical data of dimension 50 split into 10 and 5.

REFERENCES

[1] Chen, J. and Ran, X., 2019. Deep learning with edge computing: A review. Proceedings of the IEEE, 107(8), pp.1655-1674.

[2] Wang, N.C., Gonugondla, S.K., Nahlus, I., Shanbhag, N.R. and Pop, E., 2016, June. GDOT: A graphene-based nanofunction for dot-product computation. In 2016 IEEE Symposium on VLSI Technology (pp. 1-2). IEEE.

[3] Kumar, A., Goyal, S. and Varma, M., 2017, July. Resource-efficient machine learning in 2 KB RAM for the internet of things. In International Conference on Machine Learning (pp. 1935-1944). PMLR.

[4] Ezzeldin, Y.H., Fragouli, C. and Diggavi, S., 2019, July. Quantizing signals for linear classification. In 2019 IEEE International Symposium on Information Theory (ISIT) (pp. 912-916). IEEE.

[5] Li, X. and Li, P., 2019. Generalization error analysis of quantized compressive learning. Advances in Neural Information Processing Systems, 32.

[6] Nahlus, I., Kim, E. P., Shanbhag, N. R., Blaauw, D. (2014, August). Energy-efficient dot product computation using a switched analog circuit architecture. In 2014 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED) (pp. 315-318). IEEE.

[7] Shanbhag, N.R., Verma, N., Kim, Y., Patil, A.D. and Varshney, L.R., 2018. Shannon-inspired statistical computing for the nanoscale era. Proceedings of the IEEE, 107(1), pp.90-107.

[8] Liu, J., Zhang, G., Wang, B., Li, W. and Wang, J., 2019. Gate failure physics of SiC MOSFETs under short-circuit stress. IEEE Electron Device Letters, 41(1), pp.103-106.

[9] Sakr, C., Kim, Y. and Shanbhag, N., 2017, July. Analytical guarantees on numerical precision of deep neural networks. In International Conference on Machine Learning (pp. 3007-3016). PMLR.

[10] Hu, M., Strachan, J.P., Li, Z., Grafals, E.M., Davila, N., Graves, C., Lam, S., Ge, N., Yang, J.J. and Williams, R.S., 2016, June. Dot-product engine for neuromorphic computing: Programming 1T1M crossbar to accelerate matrix-vector multiplication. In 2016 53nd acm/edac/ieee design automation conference (dac) (pp. 1-6). IEEE.

[11] Roth, R.M., 2018. Fault-tolerant dot-product engines. IEEE Transactions on Information Theory, 65(4), pp.2046-2057.

[12] Lenz, A., Bitar, R., Wachter-Zeh, A. and Yaakobi, E., 2021, July. Function-correcting codes. In 2021 IEEE International Symposium on Information Theory (ISIT) (pp. 1290-1295). IEEE.

[13] Huang, K., Siegel, P.H. and Jiang, A.A., 2020, June. Functional Error Correction for Reliable Neural Networks. In 2020 IEEE International Symposium on Information Theory (ISIT) (pp. 2694-2699). IEEE.

[14] Shalev-Schwartz, S. and Ben-David, S., 2014, May. Understanding Machine Learning: From Theory to Algorithms. Cambridge University Press.

[15] Boyd, S. and Vandenberghe, L., 2004. Convex optimization. Cambridge university press.