COMPETENCY AWARE MACHINE LEARNING USING NULL SPACE AVOIDANCE

By

SRINIVAS SOURABHREDDY SATTIREDDY MEDAPATI

A THESIS PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

2019

ACKNOWLEDGMENTS

Thanks to all the help I have received from Dr. Gader, Dr. Zare and Dr. Wilson in developing this research. Thanks to Ron, Nick, Yuanhang and the entire lab for helping. Lastly I want to thank the editorial office for providing an easy to use template for writing this report

TABLE OF CONTENTS

LIST OF FIGURES

Abstract of Thesis Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Master of Science

COMPETENCY AWARE MACHINE LEARNING USING NULL SPACE AVOIDANCE

By

Srinivas Sourabhreddy Sattireddy Medapati

May 2019

Chair: Paul Gader
Major: Computer Science

Neural networks can be easily fooled using adversarial inputs. These inputs may look different from training data but manage to cause the network to classify it as one of the training classes. Adversarial inputs may be artificially generated or might exist naturally in the test dataset. This makes it harder for these networks to generalize on new data samples that are different from it's training data. In this thesis we propose a modified training procedure for neural networks to prevent some of these problems. We essentially focus on adversarial inputs generated by adding vectors from the null space of the weight matrix to training data. The proposed training method is a modification of the standard backpropagation algorithm adding steps to ensure that the projection of training data onto the null space of weight matrix is minimized. When a new input comes in, it's projection onto the null space is computed and the network can detect if the input doesn't belong to any of it's training classes. Thus, helping the network become aware of when it can be competent with it's predictions.

CHAPTER 1
INTRODUCTION

1.1 Background

Neural networks have performed well in a wide range of classification and recognition problems[1, 2] but are prone to the problems of adversarial inputs[3]. Adversarial inputs can be extremely harmful based on the context in which these neural networks are used. For example, in a self driving car the network being used to detect objects could be fooled to misclassify an object as a stop sign. Hence it is important to develop methods that allow the neural network to defend against these attacks and increase it's competency to perform the task it has been trained on.

These adversarial inputs could fall into two categories, ones that add a slight perturbation to training data and confuse the network to give a different output or ones that look very different from training data but manage to fool the network to produce a desired output. We introduce a novel way to generate the latter type of adversarial inputs exploiting the null space of the weight matrix and propose a method to make the network more robust against these kinds of inputs.

1.2 Statement of Problem

1.2.1 Mathematical Formulation

We consider a linear classifier for stating the problem but the same argument can be extended to non-linear classifiers like neural networks. Let $X \in R^{N \times D}$ be our training data where we have D samples and N features in each sample, $Y^p \in R^{M \times D}$ be our predicted outputs where we have M outputs for each instance. We assume $N >> M$, which is typical in deepnet architectures where there is large downsizing at each layer from input to output. This essentially gives rise to a non-trivial null space that we use to generate adversarial inputs. Let $W \in R^{N \times M}$ be our weight matrix, thus we have

$$Y^p = W^T X$$

Let $N \in R^{N \times N-M}$ be the null space of W^T and $\vec{n} \in N$ be a vector in that null space. We could pick a training sample $\vec{x} \in X$ and construct a new input sample $\vec{x}_{fake}$ such that

$$\vec{x}_{fake} = \vec{x} + \lambda \vec{n} \quad , \quad \lambda \in R$$

where λ and $\vec{n}$ can be chosen based on how different the generated sample should look from the original sample. We note that the predicted output for this generated sample is same as the original sample as,

$$W^T \vec{x}_{fake} = W^T \vec{x}$$

1.2.2 Examples

We present some examples to support the above theory. A neural network with a single hidden layer of 200 neurons was trained to classify 28×28 digit images from the MNIST dataset[4] and produce 10 outputs corresponding to the 10 digit classes. The original class labels were converted to network outputs using one hot encoding. Weight updates were made in batch mode using gradient descent based backpropagation algorithm over 50000 epochs resulting in a test error rate of 0.08. We took the first layer weight matrix $W \in R^{784 \times 200}$ and applied the above method to generate adversarial inputs shown below.

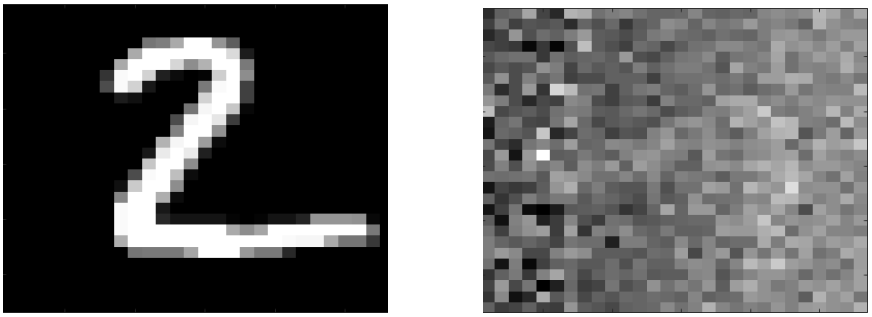

Figure 1-1. Original and adversarial images producing output of 2

1.3 Summary of Investigation

To defend the networks against adversarial inputs, methods have been tried to train the network with generated adversarial inputs[5], but it's not sufficient to guard the network against such attacks because it doesn't take into account the fact that the space of adversarial images is not known beforehand. This stems from the fact that the space of possible images that can generate a given output is infinite and only a fraction of those images look like training data which humans could recognize, for a neural net if the input can produce the right activations then it belongs to that output class. In contrast, our proposed approach focuses more on letting the network develop a "sense" of the training data and then use that to reject adversarial inputs of the kind mentioned above. This helps in making the net more aware on how confident it can be in it's predictions given the data it has been trained on.

CHAPTER 2
LITERATURE REVIEW

2.1 Adversarial Inputs

2.1.1 Types of Adversarial Inputs

Adversarial attacks on neural networks have been researched extensively in existing literature[3, 6]. Adversarial inputs can be targeted or nontargeted based on whether they're designed to just confuse the network to produce wrong outputs or aimed at producing a desired output. These inputs can also be classified as ones that add a small amount of perturbation to inputs so as to be negligible to the human eye vs those that appear as complete noise but manage to produce a desired output. What's more harmful about these inputs is that they are transferrable across different models[7]. Popular methods to produce such inputs include using gradient descent to determine what change in the input produces a higher probability for the desired output class. Another method would be to use genetic algorithms where you start with multiple images and in every generation you keep images that maximize the probability of your output class and keep generating new images by crossing these images at every stage. It has also been shown that these adversarial inputs can fool humans in a limited time setting[8]. and that these images suceed in fooling the network even if captured by a cellular device[9], thus increasing the possible scope of attack.

2.1.2 Black Box Attacks

Black box classification systems are ones where you only have access to the outputs and don't know anything else about the network. It is possible to use one of the available methods to generate adversarial inputs on a different model and then use those images to successfully attack commercial image classification systems[10]. These attacks can also be executed by getting outputs for multiple input images and then computing the gradient of the output with respect to change in input thus allowing the attacker to formulate an adversarial image which would produce a desired output.

2.1.3 Defense against Adversarial Inputs

There have been techniques developed to defend the neural networks against adversarial inputs but this is an ongoing research topic and new methods of producing adversarial inputs and defense techniques against these inputs are being proposed continuously. Methods to defeat these defense techniques are also being discovered subsequently. One such failed defense method is gradient masking where instead of outputting probabilities of classes we just limit the outputs to highest probability class label. This can be defeated by having our own substitute model which would replicate the gradients and then can be used to produce adversarial inputs to trick the black box system[11]. There has been some progress at developing methods that modify the training procedure so as to reduce the impact of adversarial inputs on the network. One such method trains the network with generated adversarial inputs and adds an additional step in the backpropagation procedure to make it robust to adversarial inputs. They use an additional gradient descent step to update weights after the standard backpropagation pass[5]. Since it's not possible to theoretically formulate all the techniques of generating adversarial inputs it's not possible to predict if a given defense will work in all cases.

2.1.4 Handwriting Recognition

The problem of neural networks misclassifying inputs that look similar was observed in handwriting recognition research[12, 13]. It is especially harder for the networks to recognize entire words as it involves segmenting the word into characters and deciding which segmentation would lead to a sensible result. In this case, individual characters might be identified wrong without the context of the surrounding word. An example is shown below,

Highlighted below is a segment that could be identified either way leading to different results

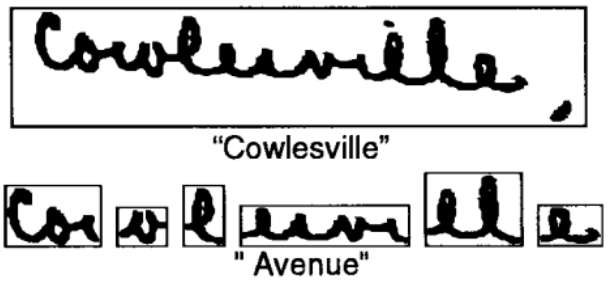

Figure 2-1. Segmentation of handwritten words

Figure 2-2. Classifying segmented images requires context

High accuracy character recognition might not lead to high accuracy word recognition because context and segmentation play an important role. Another example has been shown below to demonstrate the point.

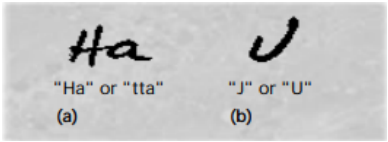

Figure 2-3. Segmentation of handwritten words

Since we would have to pass these confusing segmented images through out network these could serve as natural adversarial inputs. It is a challenge for neural networks to generalize for unseen data that is different from it's training data and the MNIST dataset is a good example to demonstrate this point. The images for threes and fives look very similar but the fours look slightly different. A neural network trained using backpropagation to classify only threes and fives fails to detect fours and classifies those images as threes.

2.2 Matrix Algebra

Let $W \in R^{N \times M}$ be our weight matrix, the null space of W is defined as the subspace containing all vectors $\vec{x}$ such that $W\vec{x} = 0$. If $N >> M$ we have a non-trivial nullspace as multiple vectors from R^N are mapped to 0 in R^M

2.2.1 Fundamental Subspaces

There are four fundamental subspaces for any matrix W. Let $N(W)$ as the null space of W. The column space denoted by $C(W)$ is a subspace which comprises of all linear combinations of column vectors of W. Similarly the row space denoted by $C(W^T)$ contains all the linear combinations of row vectors of W. The left null space which is denoted as $N(W^T)$ defines the null space of the transposed matrix W. Among these subspaces we have the following orthogonality relations

$$C(W) \perp N(W^T)$$

$$C(W^T) \perp N(W)$$

Thus we can take any vector and represent it as it's projected components onto $C(W)$ and $N(W^T)$ as these spaces are orthogonal.

2.2.2 QR Decomposition

QR decomposition is factorizing a matrix into an orthogonal matrix Q and an upper triangular matrix R. QR decomposition is used widely to solve linear least squares problem as computing QR decomposition is stable. QR decompositions can also be used to compute the $N(W^T)$ and $C(W)$ efficiently.

2.2.3 Givens Rotation

A Givens rotation matrix is defined as below

$$G(i,j,\theta) = \begin{bmatrix} 1 & \dots & 0 & \dots & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \dots & c & \dots & -s & \dots & 0 \\ \vdots & & \vdots & \ddots & \vdots & \dots & \vdots \\ 0 & \dots & s & \dots & c & \dots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & \dots & 0 & \dots & 1 \end{bmatrix}$$

where $c = cos(\theta)$, $s = \sin(\theta)$ and $i < j$ we have

$$g_{kk} = 1 \quad for \quad k \neq i,j \tag{2-1}$$

$$g_{ii} = g_{jj} = c \tag{2-2}$$

$$g_{ij} = g_{ji} = -s \tag{2-3}$$

Multiplying a matrix with the Givens rotation matrix rotates the i^{th} and j^{th} column vectors in the plane comprising of those vectors by θ. Givens rotations are used to compute the QR decomposition of a matrix by successively zeroing out the lower diagonal elements by applying Givens rotations.

2.2.4 Computing Null Space

To compute $N(W^T)$ where $N >> M$ we can do a QR decomposition where $Q \in R^{N \times N}$ and $R \in R^{N \times M}$ and partition Q as follows

$$Q = [Q1 \quad Q2] \quad where \quad Q1 \in R^{N \times M}, Q2 \in R^{N \times N-M}$$

$Q1$ forms the basis for $C(W)$ and is orthogonal to $Q2$ which forms the basis for the left null space $N(W^T)$[14]

2.2.5 Projection of Vector Onto Subspace

Let $Q \in R^{D \times D}$ be a basis for our subspace, we define the projection of vector $\vec{x}$ onto Q as

$$\vec{p} = \sum_{d=1}^{D} \vec{q_d}(\vec{q_d}^T \vec{x})$$

2.3 Optimization Methods

There might not exist exact closed form solutions to a given equation in which case we apply optimization methods to arrive at approximate solutions that can be made arbitrarily close to the optimal solution. Machine learning algorithms typically use the gradient descent method of optimizing objective functions.

2.3.1 Gradient Descent

let $f(x)$ be a convex objective function that needs to be minimized, we apply gradient descent by updating x in the opposite direction of it's gradient,

$$x_{t+1} = x_t - \eta \frac{df}{dx}_{|x=x_t}$$

where η is our learning rate that needs to be adjusted so as to prevent skipping over the local minima. Gradient descent might not find the global minima and faces the problem of getting stuck in a local minima.

2.3.2 Givens Coordinate Descent

To optimize objective functions of form $f(U)$ where U is an orthogonal matrix, gradient descent wouldn't serve the purpose entirely because it's expensive the retore the orthogonality after it's broken. In order to resolve this issue Givens coordinate descent was proposed[15]. The authors consider a manifold where every point is an orthgonal matrix and moving between points is equivalent to performing Givens rotations on these matrices. The algorithm introduces an efficient way of optimizing f through a series of Givens rotations of form $G(i, j, \theta)$ applied to U as these rotations preserve the orthogonality.

Algorithm 2.1. ***procedure*** GIVENS COORDINATE DESCENT$(f, U_{initial})$

$U_0 \leftarrow U_{initial}$

while *not converged* ***do***

choose i, j *randomly*

$\theta_{t+1} = \arg\min f(U_t G(i, j, \theta))$

$U_{t+1} = U_t G(i, j, \theta_{t+1})$

end while

return U_{final} ▷ *optimal* U

end procedure

CHAPTER 3
TECHNICAL APPROACH

We aim to modify to the standard backpropagation procedure in a way that the projection of training data onto the null space is minimized. In order to achieve that we aim for maximizing the projection of training data onto the column space.

3.1 Linear Classifier

Let $X \in R^{M \times D}$ be our training data and $W \in R^{N \times M}$ be our weight matrix, let $Q \in R^{N \times M}$ be the basis for column space of W. Thus our modified objective function in the case of linear classifier can be written as

$$f = \frac{\left\|W^T X - Y^a\right\|^2}{D} - P \tag{3-1}$$

where we define P as the average squared projection ratio over the dataset, Q is the column space for W

$$P = \frac{\sum_{d=1}^{D} (\left\|Q^T \vec{x}_d\right\|^2 / (\|\vec{x}_d\|^2)}{D}$$

After training is done, both MSE and projection onto null space are low. When a new input comes in we compute it's projection onto $N(W^T)$ and if it's higher than the threshold decided in the training precedure the network can tell if it's not in one of it's training classes. We apply Givens coordinate descent to minimize the projection term as we need to keep Q orthogonal otherwise it won't be a basis.

3.2 Path Finding in Orthogonal Manifold

We developed a simple application of Givens coordinate descent to verify it's practical applications. Let $O(d)$ be orthogonal space of all $d \times d$ orthogonal matrices and Q, E are 2 random orthogonal matrices in $O(d)$

$$Q, E \in O(d)$$

A path in cartesian space can be thought of as set of moves starting from one point that lead to another. A path in the orthogonal manifold mentioned above would be a series of Givens rotations which when applied to Q would lead to E

$$Q = \begin{bmatrix} q_{11} & q_{12} & q_{13} & \cdots & q_{1n} \\ q_{21} & q_{22} & q_{23} & \cdots & q_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ q_{d1} & q_{d2} & q_{d3} & \cdots & q_{dn} \end{bmatrix}$$

. Our aim here is to find a series of orthogonal transformations which when applied to Q make it look as similar as possible to T , or the goal could be stated as minimizing

$$f = \sum_{r=1}^{d} \sum_{c=1}^{d} (q_{rc} - t_{rc})^2$$

So we apply the given's coordinate descent to this problem, essentially we are trying to find a series of given's rotations which when applied to Q successively will make it look like E.

We need to obtain an equation for $f(QG(i, j, \theta))$ first , in order to then determine what θ will minimize f.

$$Q = \begin{bmatrix} \vec{q_1} & \vec{q_2} & \cdots & \vec{q_d} \end{bmatrix}$$

$$Q^{new} = QG(i, j, \theta)$$

Notice that given's rotation only affects the i'th and j'th columns , so we have

$$\vec{q_k^{new}} = \begin{cases} \vec{q_i} cos(\theta) + \vec{q_j} sin(\theta), & \text{if k == i} \\ \vec{q_j} cos(\theta) - \vec{q_i} sin(\theta), & \text{if k == j} \\ \vec{q_k}, & \text{otherwise} \end{cases}$$

Thus we have an equation for f as

$$f = \sum_{k=1}^{d} [q_{ki} cos(\theta) + q_{kj} sin(\theta) - e_{ki}]^2 + \sum_{k=1}^{d} [q_{kj} cos(\theta) - q_{ki} sin(\theta) - e_{kj}]^2 + C$$

$$\frac{df}{d\theta} = 2\sum_{k=1}^{d}(q_{ki}cos(\theta) + q_{kj}sin(\theta) - e_{ki})(-q_{ki}sin(\theta) + q_{kj}cos(\theta))$$

$$+2\sum_{k=1}^{d}(q_{kj}cos(\theta) - q_{ki}sin(\theta) - e_{kj})(-q_{kj}sin(\theta) - q_{kj}cos(\theta))$$

Equating $\frac{df}{d\theta}$ to 0 doesn't lead to a closed form solution so we use gradient descent to minimize f and keep iterating until convergence. We can go arbitrarily close to E following the above procedure.

3.3 Maximizing Projection of Training Data onto Weight Matrix

Let's isolate the projection term in our objective function, so we can define our objective function f as

$$f = -P$$

and proceed to minimize f using Givens co-ordinate descent since it depends on orthogonal matrix Q, but the problem we encountered in this approach is performing Givens rotation on Q does not affect the projection length of X onto Q, we have included a proof in the next section.

3.3.1 Givens Rotation Doesn't Change Projection Norm

Let's calculate the new projection length after a Givens rotation $G(i, j, \theta)$ is applied to Q, thus we have

$$q_i^{new} = q_i cos(\theta) + q_j sin(\theta)$$

$$q_j^{new} = q_j cos(\theta) - q_i sin(\theta)$$

Thus if we look at one of our training instances $x \in R^{N\times 1}$ and compute it's projection vector coefficients onto Q, we see that only the i^{th} and j^{th} coefficients change. Let the projection vector before givens rotation be $p \in R^{M\times 1}$, we have the k^{th} coefficient as

$$p_k = q_k^T x$$

Let $p^{new} \in R^{M\times 1}$ be the new projection vector, so if we look at the i^{th} and j^{th} coefficients we have

$$p_i^{new} = (q_i^{new})^T x$$
$$p_j^{new} = (q_j^{new})^T x$$

These can be further written as

$$p_i^{new} = \sum_{n=1}^{N} q_{in}^{new} x_n$$
$$p_j^{new} = \sum_{n=1}^{N} q_{jn}^{new} x_n$$

Thus if we compute the new projection length we have

$$\|p_{new}\|^2 = \sum_{m=1}^{M} (p_m^{new})^2$$

but since only 2 coefficients change after givens rotation, we rewrite it as

$$\|p_{new}\|^2 = K + (p_i^{new})^2 + (p_j^{new})^2$$

where $K = \sum_{m=1, m\neq i,j}^{M} (p_m)^2$ which is the part from the original projection norm that remains same. Looking at the part that changed, we have

$$C = (p_i^{new})^2 + (p_j^{new})^2$$

$$C = \sum_{m=1}^{M} ((q_{im} cos(\theta) + q_{jm} sin(\theta)) x_m)^2 + \sum_{m=1}^{M} ((q_{jm} cos(\theta) - q_{im} sin(\theta) x_m))^2$$

$$C = \sum_{m=1}^{M} (q_{im}^2 + q_{jm}^2) x_m^2$$

$$C = p_i^2 + p_j^2$$

Thus we conclude that the overall projection norm of $\vec{x}$ onto Q remains unchanged by a givens rotation and we come up with an alternative solution.

3.3.2 Projection Maximization Using Rotation Matrices

The solution is to use a rotation matrix U such that projection of X onto UQ is maximized. Every orthogonal matrix is a rotation matrix so we choose U to be orthogonal in order to utilize Givens cooridnate descent to optimize the objective function

3.3.3 Mathematical Derivation

We define P^{new} as

$$P^{new} = \frac{\sum_{d=1}^{D}(\left\|Q_{new}^{T}\vec{x}_d\right\|^2 / (\|\vec{x}_d\|^2)}{D}$$

where the modified basis is Q^{new} ,

$$Q^{new} = UQ$$

If we consider a single training sample $\vec{x} \in X$ and observe how it's projection length onto Q changes after Givens rotation we have

$$\vec{p} = Q^T\vec{x}$$

After Givens rotation it would be

$$\vec{p}^{new} = Q_{new}^{T}\vec{x}$$

This can be written as

$$\vec{p}^{new} = Q^T(UG(i,j,\theta))^T\vec{x}$$

This can be further interpreted as

$$\vec{p}^{new} = Q^T\vec{x}^{new}$$

$$\vec{x}^{new} = (UG(i,j,\theta))^T\vec{x}$$

We note that Givens rotation will only change the i^{th} and j^{th} column vectors of U, thus we see that only those 2 terms would change in $\vec{x}_d^{new}$

$$\vec{u}_i^{new} = \vec{u}_i cos(\theta) + \vec{u}_j sin(\theta)$$

$$\vec{u}_j^{new} = \vec{u}_j cos(\theta) - \vec{u}_i sin(\theta)$$

The changed terms in $\vec{x}^{new}$ can be written as

$$\vec{x}_i^{new} = (\vec{u}_i^{new})^T \vec{x}$$

$$\vec{x}_j^{new} = (\vec{u}_j^{new})^T \vec{x}$$

Thus if look at the contribution of $\vec{x}$ we have

$$f_x = - \| p^{new} \|^2$$

Looking at the n^{th} term we have,

$$p_n^{new} = - \sum_{n=1}^{N} ((\vec{q}_n)^T \vec{x}^{new});$$

Thus we can rewrite to observe the terms that change with θ, let K denote the constant terms

$$p_n^{new} = K + (\vec{q}_n)_i (\vec{x}_i^{new})_i + (\vec{q}_n)_j (\vec{x}^{new})_j$$

We have

$$\frac{dp_n^{new}}{d\theta} = (\vec{q}_n)_i \frac{d(\vec{x}^{new})_i}{d\theta} + (\vec{q}_n)_j \frac{d(\vec{x}^{new})_j}{d\theta}$$

The derivatives can be written as

$$\frac{d(\vec{x}^{new})_i}{d\theta} = (\vec{u}_j cos(\theta) - \vec{u}_i sin(\theta))^T \vec{x}$$

$$\frac{d(\vec{x}^{new})_j}{d\theta} = -(\vec{u}_j sin(\theta) + \vec{u}_i cos(\theta))^T \vec{x}$$

Thus we have

$$\frac{df_x}{d\theta} = -2\sum_{n=1}^{N} p_n^{new} \frac{dp_n^{new}}{d\theta}$$

Extending the equations to f as we have

$$f = \frac{(\sum_{x \in X} \frac{f_x^2}{\|x\|^2})}{D}$$

So we can write the derivative as

$$\frac{df}{d\theta} = \frac{(\sum_{x \in X} \frac{\frac{df_x^2}{d\theta}}{\|x\|^2})}{D}$$

3.3.4 Projection Maximization Procedure

We present the formal algorithm that takes in $W \in R^{N \times M}$ and X, returns a modified W such that the projection X onto W is maximized

Algorithm 3.1. ***procedure*** MAXIMIZE-PROJECTION*(W, X)*

$q_{old}, r_{old} \leftarrow qr(W)$ ▷ *qr decomposition of W*

$Q = q(:, 1 : M)$ ▷ *column space of W*

$N = q(:, M + 1 : N)$ ▷ *left null space of W*

while *not converged* ***do***

choose i, j randomly

while *gradient descent not converged* ***do***

$\theta_{t+1} = \theta_t - \eta \frac{df}{d\theta}$ ▷ *f as described above*

end while

$U_{t+1} = U_t G(i, j, \theta_{t+1})$

end while

$q_{new} = [UQ \quad UN]$

$W = q_{new} r_{old}$

return W ▷ *projection of X onto W is maximized*

end procedure

3.4 Standard Backpropagation

We look at the standard backpropation algorithm and see how reducing the MSE affects the projection of training data onto 1st layer weight matrix.

Let $X_{N\times D}$ is our training data with D samples each consisting of N features and $Y^a_{M\times D}$ is our output ground truth with D columns each containing M outputs for a single training instance. Let $Y^p_{M\times D}$ be the predicted output from our neural network. Thus we have our loss function f which is chosen to reflect the MSE (mean square error), another popular choice is the cross-entropy function.

$$f = \frac{0.5 * \|Y^p - Y^a\|^2}{D}$$

Let there be L hidden layers in our network of sizes $H_1, H_2, ...H_L$ respectively and $W_1 \in R^{N\times H1}, W_2 \in R^{H_1\times H_2}, ...W_{L+1} \in R^{H_L\times M}$ be our weight matrices and $B_1, B_2...B_{L+1}$ be our bias matrices for these layers. Let σ be our activation function, thus we have our forward pass as given below

$$Y^p = \sigma(W^T_{L+1}\sigma(W^T_L\sigma(...\sigma(W^T_1X + B_1)... + B_L) + B_{L+1})$$

Let $X_1, X_2, ..X_{L+1}$ be the inputs fed into the respective layers of our network, thus we have

$$\begin{aligned} X_1 &= X \\ X_l &= \sigma(W^T_{l-1}X_{l-1} + B_{l-1}) \quad for \quad l \in [1, L+1] \\ Y^p &= \sigma(W^T_{L+1}X_{L+1} + B_{L+1}) \end{aligned}$$

Let $Z_1, Z_2, ...Z_{L+1}$ be the inputs right before activation is applied at the neurons, thus we have

$$Z_1 = W_1^T X + B_1$$
$$Z_l = W_l^T X_l + B_l \quad for \quad l \in [1, L+1]$$
$$X_l = \sigma(Z_{l-1})$$
$$Y^p = \sigma(Z_{L+1})$$

The backward pass is where we update the W_is and B_is by computing derivatives of f w.r.t weights and biases, we also define our error vectors $\delta_1, \delta_2, ...\delta_{L+1}$ such that

$$\delta_l = \frac{df}{dZ_l} \quad for \quad l \in [1, L+1]$$

So we have the last layer error as given by

$$\delta_{L+1} = \frac{dY^p}{dZ_{L+1}} \circ (Y^p - Y^a)$$

and for inner layers error can be written as

$$\delta_l = \frac{dX_{L+1}}{dZ_l} \circ (W_{l+1}\delta_{l+1}) \quad for \quad l \in [1, L]$$

Thus our backpropagation update equations for the weights and bias are as follows.

$$\frac{df}{dW_l} = X_l \delta_l^T$$
$$\frac{df}{dB_l} = \delta_l$$
$$W_l = W_l - \eta * \frac{df}{dW_l}$$
$$B_l = B_l - \eta * \frac{df}{dB_l}$$

3.5 Backpropagation with Null Space Avoidance(Nusa)

3.5.1 Maximizing Projection by Doing Alternating Optimizations

We modified the standard backpropagation updates described in the previous section by adding an additional step that ensures projection of X_l onto W_l is high, where $W_l \in R^{H_{l-1} \times H_l}, H_{l-1} > H_l$. We do alternating optimizations so the gradient descent step reduces error and the maximize projection procedure increases projection onto column space. At the end of the training procedure both the goals are achieved i.e MSE is low and projection onto weight matrices is higher.

So our modified backpropagation update equations include an additional projection maximization step.

3.5.2 Proposed Training Procedure

We present the formal algorithm for the modified training procedure

Algorithm 3.2. ***procedure*** MODIFIED-BACKPROPAGATION(X, Y^a)

while *not converged* ***do***

$Y^p = \sigma(W_{L+1}^T \sigma(W_L^T \sigma(...\sigma(W_1^T X + B_1)... + B_L) + B_{L+1})$ ▷ *forward pass*

$l = L + 1$

while $l \geq 1$ ***do*** ▷ *backward pass*

$W_l = W_l - \eta * \frac{df}{dW_l}$ ▷ *decreasing loss function*

$W_l = MAXIMIZE - PROJECTION(W_l, X_l)$ ▷ *increasing projection*

$B_l = B_l - \eta * \frac{df}{dB_l}$

$l = l - 1$

end while

end while

return W

end procedure

CHAPTER 4
RESULTS AND CONCLUSIONS

4.1 Path Finding

We took 2 random orthogonal matrices $Q1, Q2 \in R^{100\times100}$ and find a path from $Q1$ to $Q2$ using Givens coordinate descent. We generated the orthogonal matrices using QR decompositions of two random matrices of required dimensions. Givens co-ordinate descent runs very fast in practice. Below is the progress in the normalized loss function f defined as

$$f = \frac{\|Q1 - Q2\|^2}{\|Q2\|^2}$$

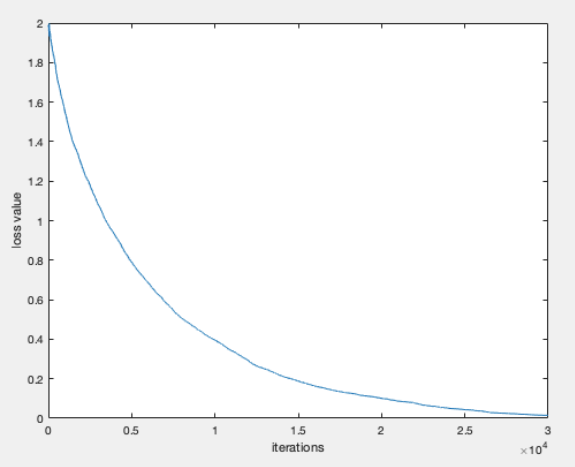

Figure 4-1. loss value vs iterations

4.2 Projection Maximization

We used the computed bar features[16] for digit images in the MNIST dataset belonging to the 3 and 5 class labels. We start with $X \in R^{60\times6000}$ and a random $W \in R^{60\times40}$ and modify W using the algorithm mentioned above. We plot the average squared null space projection ratio n for the training data,

$$n = \frac{\sum_{\vec{x}\in X} \left\|N^T\vec{x}\right\|^2 / \|\vec{x}\|^2}{D}$$

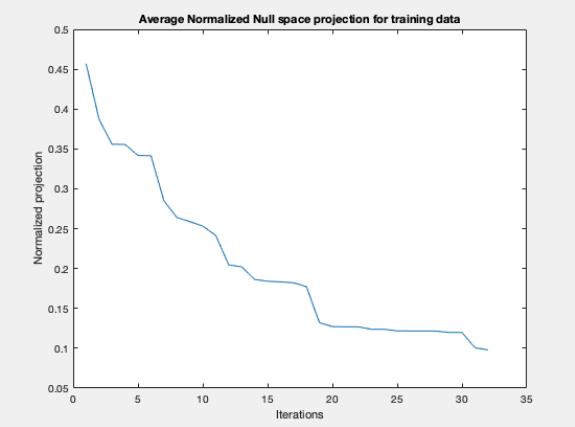

Figure 4-2. n vs iterations

As seen, n decreases gradually and below are the null space projection ratios for test data $X_{test} \in R^{60\times4000}$ before and after the training. The test null space projections before and after are plotted below

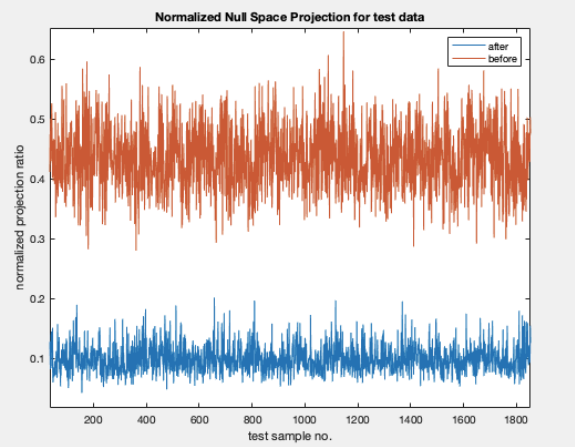

Figure 4-3. test data null space projections before vs after training

The null space projection ratios for test data onto W after the training has decreased.

4.3 Iris Classifier (Standard vs NuSA)

We chose the Iris Verginica and Iris Versicolor flower features from the Iris dataset[17] as these are not possible to seperate linearly while Iris Setosa is known to be linearly seperable from those two classes. We used a backpropagation network with 1 hidden layer that had 3 neurons. The input layer was 4 neurons and there were 2 outputs. Sigmoid

activation function was used at each neuron and weight updates were made using batch mode. The network was trained for 20000 epochs with early stopping if the MSE meets a value of 0.2. The network was trained using both standard and NuSA modes with training set data. For the NuSA mode we only apply projection maximization at the first layer. Below is the training progress for both modes

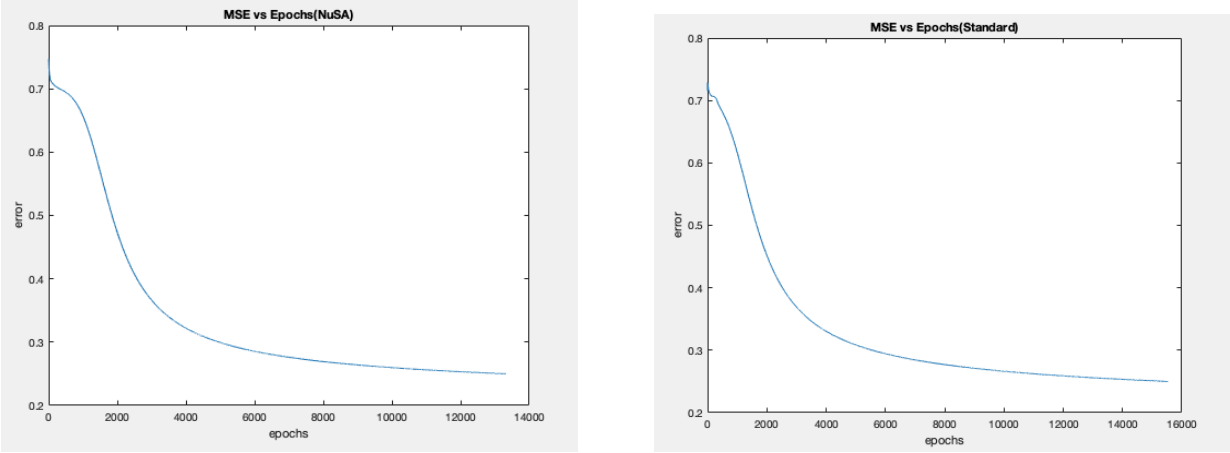

Figure 4-4. Iris classifier training MSE for NuSA vs Standard mode

The training phase projection is also shown below

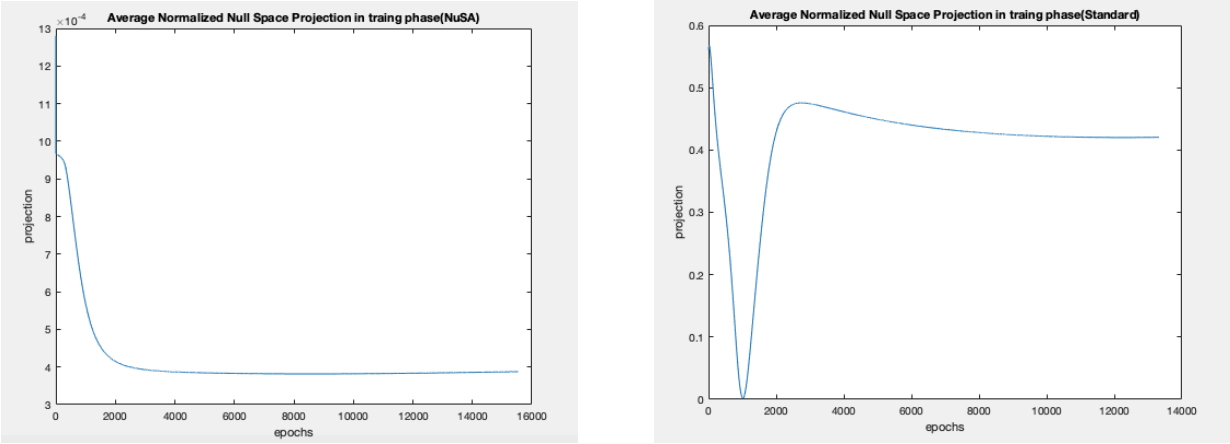

Figure 4-5. training null space projection for NuSA vs Standard mode

The average squared null space projection ratio reduces gradually in NuSA mode, while it swings randomly in the standard mode but ends up at a high value at the end of the training. Below the null space projections for the test dataset are shown

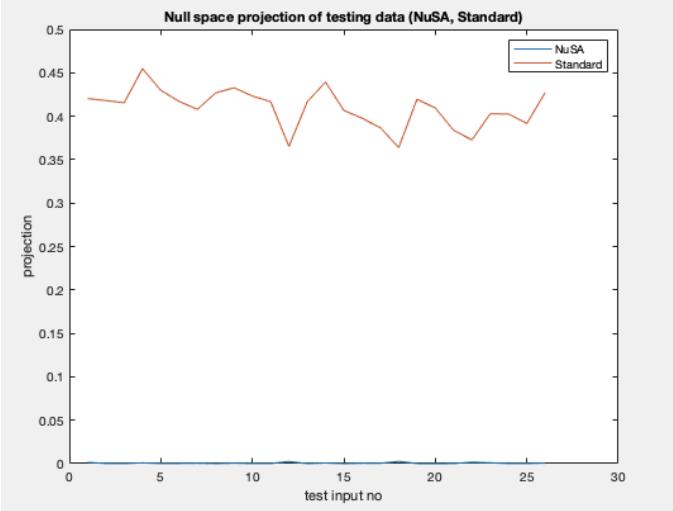

Figure 4-6. Iris classifier test null space projection for NuSA vs Standard mode

As can be seen in the above figure the test data null space projections are smaller in NuSA mode.

4.4 Handwriting Recognition (Standard vs NuSA)

We trained a 1 hidden layer neural network with 15 neurons in it's hidden layer to classify between the bar features for the threes and fives. In NuSA mode we applied projection maximization at the first layer. Sigmoid activation function was used at every neuron. The outputs were hot encoded and the network was trained for 100000 epochs with an early stopping for an MSE of 0.2. Below are the training progress for both modes

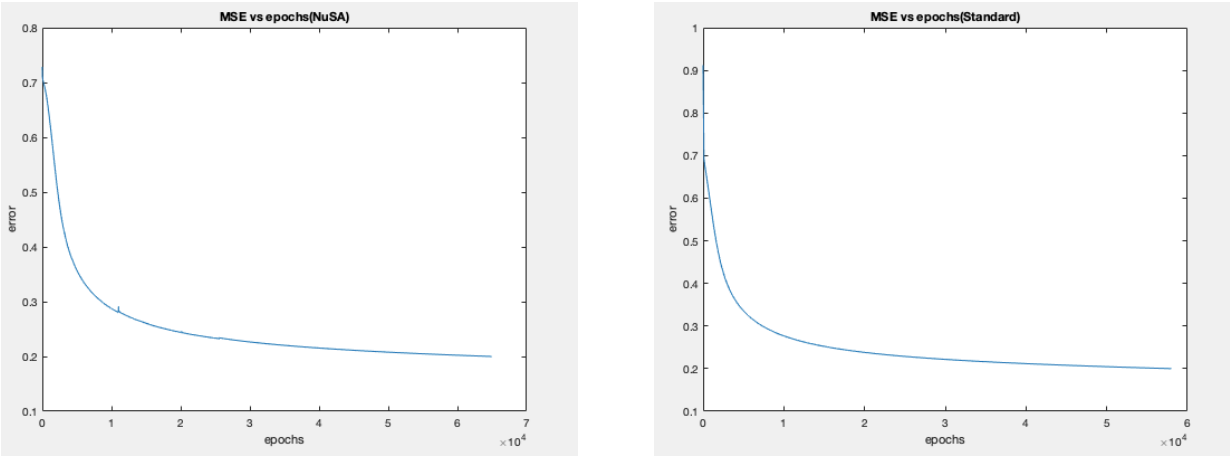

Figure 4-7. Bar feature classifier training MSE for NuSA vs Standard mode

The training average null space projection ratio decreases as training progresses in NuSA mode as shown below

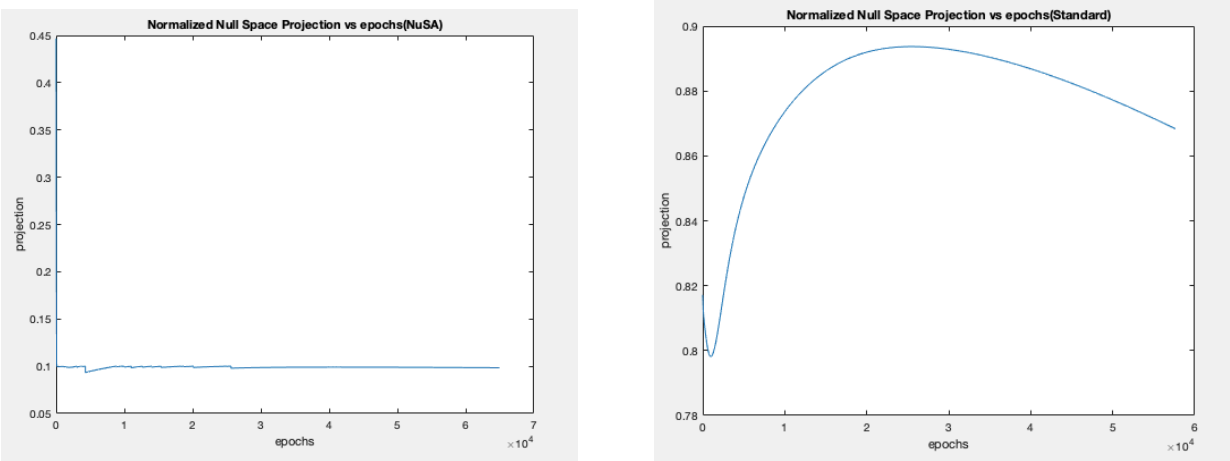

Figure 4-8. Bar feature classifier training MSE for NuSA vs Standard mode

The test error rates for the NuSA and Standard models were 0.069 and 0.074 respectively. The null space projection ratios for test data in both modes is as shown below.

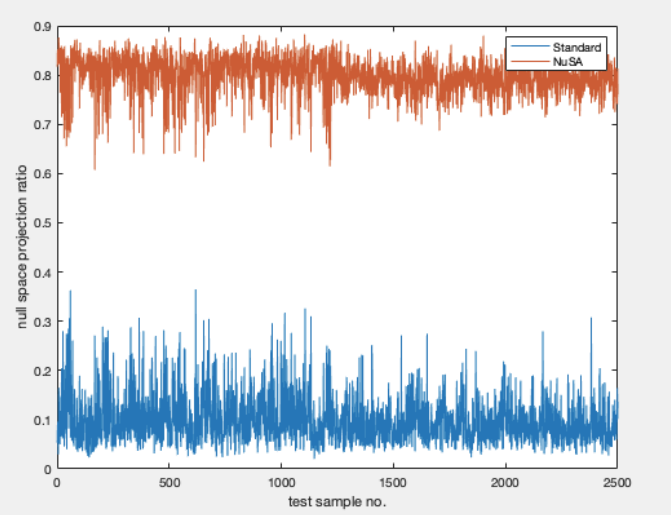

Figure 4-9. Bar feature classifier test null space projection for NuSA vs Standard mode

NuSA mode training helps decrease the null space projection ratios of test data.

We also computed the null space projections of the bar features resulting from digit four images and note that it's easier to distinguish those inputs if fed into our network. Below are the null space projections of fours and test data onto the null space of first layer weight matrix.

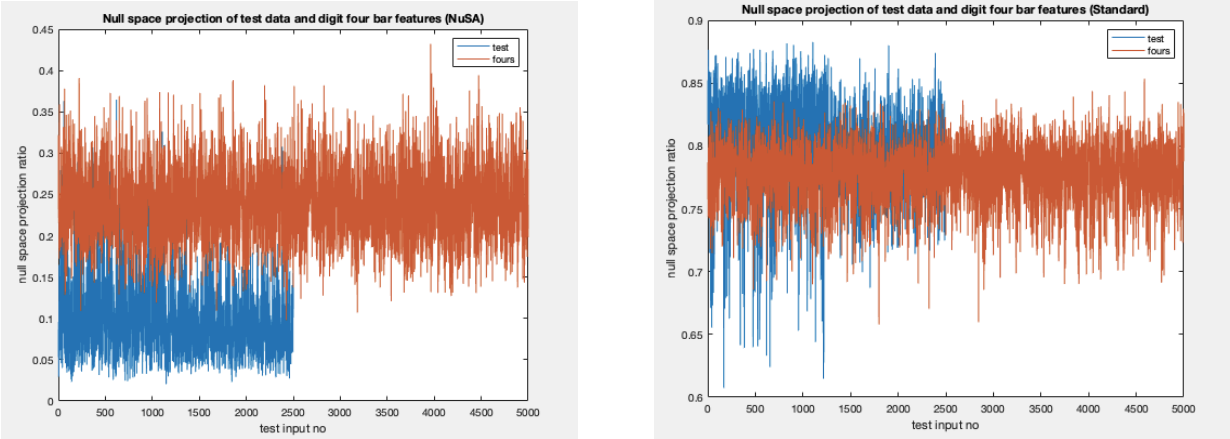

Figure 4-10. Bar feature classifier test vs fours null space projections for NuSA vs Standard mode

CHAPTER 5
FUTURE WORK

In this section we highlight the various aspects of this research that may lead to future work.

5.1 Applications to Convolutional Neural Networks

We have primarily focused on shallow neural networks but the null space problem is a universal one and gets worse in deeper architectures, one could also show that image recognition CNNs could be fooled using the technique decsribed here to generate adversarial images. Modification of the training procedure of CNNs to include a projection maximization step seems to be the natural extension to this research.

5.2 Givens Coordinate Descent for Tensor Optimizations

A lot of optimization problems we encounter in machine learning can be formulated as orthogonal tensor decompositions and it could then be viewed as the tensor equivalent of the objective function optimizations we studied in this research. Extending Givens coordinate descent to optimize functions involving orthogonal tensors would be a useful study.

5.3 Using SVD Instead of QR

We have used QR decompositions to compute null space but SVD could also be used. If we use SVD decomposition we would have 2 orthogonal matrices instead of 1 and the way the function could be optimized is doing alternative Givens coordinate descent on both orthogonal matrices.

5.4 Other Ways of Measuring Projection

The way we define our projection coefficient term in the objective function affects the performance of our method. Other ways of defining projection norms could be explored and their affects on the performance of NuSA training could be studied further.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017. [Online]. Available: http://doi.acm.org/10.1145/3065386

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014. [Online]. Available: http://arxiv.org/abs/1409.1556

[3] A. M. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," *CoRR*, vol. abs/1412.1897, 2014.

[4] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[5] A. Nøkland, "Improving back-propagation by adding an adversarial gradient," *CoRR*, vol. abs/1510.04189, 2015.

[6] I. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," 2015. [Online]. Available: http://arxiv.org/abs/1412.6572

[7] Y. Liu, X. Chen, C. Liu, and D. Song, "Delving into transferable adversarial examples and black-box attacks," *CoRR*, vol. abs/1611.02770, 2016. [Online]. Available: http://arxiv.org/abs/1611.02770

[8] G. F. Elsayed, S. Shankar, B. Cheung, N. Papernot, A. Kurakin, I. J. Goodfellow, and J. Sohl-Dickstein, "Adversarial examples that fool both human and computer vision," *CoRR*, vol. abs/1802.08195, 2018. [Online]. Available: http://arxiv.org/abs/1802.08195

[9] A. Kurakin, I. J. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," *CoRR*, vol. abs/1607.02533, 2016.

[10] Y. Liu, X. Chen, C. Liu, and D. Song, "Delving into transferable adversarial examples and black-box attacks," *CoRR*, vol. abs/1611.02770, 2016. [Online]. Available: http://arxiv.org/abs/1611.02770

[11] A. Athalye, N. Carlini, and D. A. Wagner, "Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples," *CoRR*, vol. abs/1802.00420, 2018.

[12] P. D. Gader, J. M. Keller, R. Krishnapuram, , and M. A. Mohamed, "Neural and fuzzy methods in handwriting recognition," *Computer*, vol. 30, no. 2, pp. 79–86, Feb 1997.

[13] and P. D. Gader, "Hybrid fuzzy-neural systems in handwritten word recognition," *IEEE Transactions on Fuzzy Systems*, vol. 5, no. 4, pp. 497–510, Nov 1997.

[14] G. H. Golub and C. F. Van Loan, *Matrix Computations (3rd Ed.)*. Baltimore, MD, USA: Johns Hopkins University Press, 1996.

[15] U. Shalit and G. Chechik, "Efficient coordinate-descent for orthogonal matrices through givens rotations," *CoRR*, vol. abs/1312.0624, 2013.

[16] P. D. Gader, M. A. Mohamed, and J.-H. Chiang, "Handwritten word recognition with character and inter-character neural networks," *IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society*, vol. 27 1, pp. 158–64, 1997.

[17] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

BIOGRAPHICAL SKETCH

Sourabh graduated from Maulana Azad National Institute of Technology, India with a B.S. in electronics prior to joining the University of Florida to pursue a master's degree in computer science.