DISCLAIMER: This is a DRAFT edition. Unfortunately, my deep learning rig crashed & I lost the code related to the experiments. I have not had a chance to reproduce the experiments - you can be assured that I will relentlessly back it up this time. Though, the mathematics is independent of the code & it will be a true scientific endeavor to reproduce the results (this will happen *very soon)*. Also, I am still researching this topic so if you have any recommendations (papers or mathematics) please get in touch.

An Analysis of Deep Network Preimages

On the Domains, Ranges, Preimages, Invertibility and Vulnerabilities of Neural Networks

Nathaniel I. Rojas
rojasinate@gmail.com

Abstract

In Deep Learning, understanding the inner works of neural networks is key to interpreting their behavior as well as their vulnerabilities. Finding the pre-image for any given value in the range of a neural network is of significant importance. This work contributes an exact parametric solution for defining the preimage of particular class of deep networks.

In this work, steps are made toward a general solution of the pre-image problem. The key is to analyze the mathematical structure under certain assumptions that lead to key insights into the form of the general solution. A general solution to the pre-image problem for a large class of multi-layer perceptions is presented.

With these insights, the nature and source of adversarial attacks on neural networks is explored and initial results are developed for an optimization free "single shot" generation method for adversarial examples.

Keywords: Deep Learning, Neural Network, Preimage, Adversarial Examples.

1 Introduction

In many regards neural networks are black boxes. Gaining a better understanding of how they operate is beneficial in many regards. A very important yet not fully understood feature of deep networks is the problem of invertibility and the problem of finding the preimages for classification problems. Having a solid understanding of this specific part of deep networks will shed light on why some networks generalize while others do not. Secondly, finding an exact definition of deep network preimages will elucidate a deep network's vulnerabilities (eg unsafe classification in self driving vehicles and adversarial attacks).

In this work a large class of networks is analyzed. Specifically, the family of deep fully connected networks using Leaky Relu[1] activation are analyzed. This family of networks admits useful facts for defining a general solution to the problem of invertibility and finding preimages. Using this family of models as grounding, methods for analyzing the inverse as well as preimage are elaborated and made rigorous. These, methods can be extended for use on more general sets of network architectures.

Since the problem is complex, Section 2 will provide some initial definitions, observations and useful facts/lemmas (proofs are in the appendix). Furthermore, simplifying assumptions will be defined and motivated.

Following the initial considerations will be a rigorous problem statement. After this problem statement the heart of the paper will begin. After this, will be an analysis of the mathematical result. Next, some initial experiments are explained which justify some of the assumptions and demonstrate the possibility of an exact method for generating adversarial examples.

Finally, the paper concludes by explaining limitations and enumerating avenues for further research.

2 Related Work

There is a great amount of literature that provides insights into the functioning of neural networks. The most similar work to this one analyzes a similar family of deep networks with the primary difference being that they analyze the preimage problem with networks using standard ReLU activation [1]. Furthermore, the

[1] Leaky Relu is only a familiar activation. All that is necessary is that the activation be piecewise continuous, defined for all input values and be a non-linear *permutation* of the input space.

work in [1] is not as mathematically exact as the contributions herein. Thus, this work provides more rigorous footing hence complementing [1].

Similar works analyze the ability of neural networks to optimally classify input data which satisfies certain conditions [2]. Another, rigorously handles the problem of expressing bounds on the types of functions that neural networks can approximate [3]. Others, analyze the preimage problem by using strategic methods of approximation [4].

This work adds to the literature by providing an exact mathematical recurrence formula for defining the preimage of a certain class of deep networks (much like the backward recurrence for back propagation). This is a solid contribution because it does not rely on approximation methods - it is *exact*. Finally, the work presented here can be used to extend the analysis in [1] to define an exact preimage recurrence for a more general class of network architectures.

Next, it can be seen that this work is of theoretical importance because it provides an exact solution to the preimage problem for an entire class of deep networks. Finally, this work presents another perspective on adversarial examples.

3 Initial Considerations

Notation

Layer Number
L - denotes the number of hidden layers. The zero layer is considered the input.

Matrix Spaces
$C(A)$ - column space of a matrix
$R(A) = C(A^T)$ - row space of a matrix
$N(A)$ - null space of a matrix

Parameter Matrices
Θ - denotes an arbitrary parameter matrix.
Θ^+ - denotes the Moore-Penrose pseudo-inverse of an arbitrary parameter matrix.

Null Space
$\mathbf{n}^{[L]}$ - denotes a parameterized basis for $N(\Theta^{[L]})$. That is, it is a linear sum of any null-space basis with *variable coefficients*.

Indexing Layers
$\mathbf{a}^{[l]}, \Theta^{[l]}$ - bracketed superscripts are to index by layer.

Forward Function
$\mathscr{F}^{[l]} = f(\Theta^{[l]}\mathbf{a}^{[l-1]} + \mathbf{b}^{[l]})$ - denotes the forward pass for one layer

$\mathscr{F} = \mathscr{F}^{[L]} \circ \cdots \circ \mathscr{F}^{[1]}$ - denotes the the entire network forward pass.

Pseudo Inverse (backward)
$\mathscr{B}^{[l]}(\mathbf{y}) = (\Theta^{[l]})^+(f^{-1}(\mathbf{y}) - \mathbf{b}^{[l]})$

$\mathscr{B} = \mathscr{B}^{[1]} \circ \ldots \circ \mathscr{B}^{[L]}$

Pseudo Domain
X^+ - the set of all inputs that can be recovered by the pseudo-inverse. There will be more on this shortly.

Pseudo Range
$Y^+ = C(\Theta^{[L]})$
Since the true range (softmax output) is entirely dependent on the final column space of the final parameter matrix this work will only consider this column space.

Assumptions

Full Rank
All $\Theta^{[l]}$ are full row rank. This assumption is key to grounding the analysis. Without it, many key intuitions would be obscured by the details.

Fan in criterium
For all layers it will be assumed that the matrix is either square or has more columns than rows. Formally:

$$\forall l : \Theta^{[l]} \in \mathbb{R}^{m \times n}, m \leq n$$

On Invertibility

Strictly speaking a network does not in general have an inverse. However, parallel to the relationship between a matrix and its pseudo-inverse a network also has a pseudo-inverse. To carry this analogy further we define the domain of invertibility as the pseudo-domain. That is, any input coming from the pseudo-domain will map to an output (in the pseudo-range) and from that output can be reconstructed from the pseudo-inverse. That is, we will see that:

$\forall \mathbf{x}^+ \in X^+ : \mathbf{x}^+ = \mathscr{B} \circ \mathscr{F}(\mathbf{x}^+)$

Useful Facts & Results

Recall that between the row space and column space of a matrix there is a one-to-one correspondence. Formally:

Fact (1): $\Theta : R(\Theta) \leftrightarrows C(\Theta) : \Theta^+$

Next, under the full rank assumption and fan in criterium a useful lemma can be proven:

Lemma (1): $R(\Theta^{[l+1]}) \subseteq C(\Theta^{[l]})$

That is, the row space of parameter matrix is a subset of the column space of the previous layer's parameter matrix.

This lemma coupled with one more fact are key to all subsequent derivations. Recall that a matrix multiplying a vector on the left returns a vector that is in the column space of that matrix. Formally:

Fact (2): $\Theta\mathbf{x} \in C(\Theta)$

Lemma (1) and Fact (1) present another important fact. Namely, that

Theorem (1): If $\mathbf{x} \in C(\Theta^{[l+1]})$ then $(\Theta^{[l+1]})^+\mathbf{x} \in R(\Theta^{[l+1]})$ but this implies $\mathbf{x} \in C(\Theta^{[l]})$

That is, if we take a vector in the column space of $\Theta^{[l+1]}$ then by fact (1) we can use the pseudo-inverse to obtain a vector in the row space of $\Theta^{[l+1]}$ but by lemma (1) this is a subspace of $C(\Theta^{[l]})$ and thus the vector is contained in the column space of the previous parameter matrix.

Fact (2): Under the current assumptions we can ignore the bias and activation when finding pseudo-domain and pseudo range.

Proof sketch: Adding the bias and applying LReLU doesn't add or remove vectors from $C(\Theta^{[l]})$.

What this says, is that the space of vectors coming in is the same as the space that comes out.

Theorem (2): $\prod_{l=1}^{L} (\Theta^{[l]})^+ : C(\Theta^{[L]}) \rightarrow X^+$

That is the the product of all the pseudo inverses acting on the the row space of the final parameter matrix returns the pseudo-domain.

Note: This is not the inverse but just the definition of the pseudo-domain.

What theorem (2) shows is that there exists a mapping from the pseudo-range to the pseudo-domain. Furthermore, it means that if we can define some subset of the pseudo-range we can then define the pseudo-inverse for that subset. Finally, we observe that it is possible to define the pre-image for parameterized subsets of the pseudo-range using some facts from linear algebra (what this means will be explained in depth).

4 Problem Statement

In this work, the class of neural networks analyzed are deep and fully-connected with invertible activations that are bijective (eg Leaky Relu) from domain to range in the hidden layers followed by a final softmax output for a classification task with k classes.

Using this architecture the general equations for the pseudo-domain, range and pre-image are presented.

Formal Problem Statement

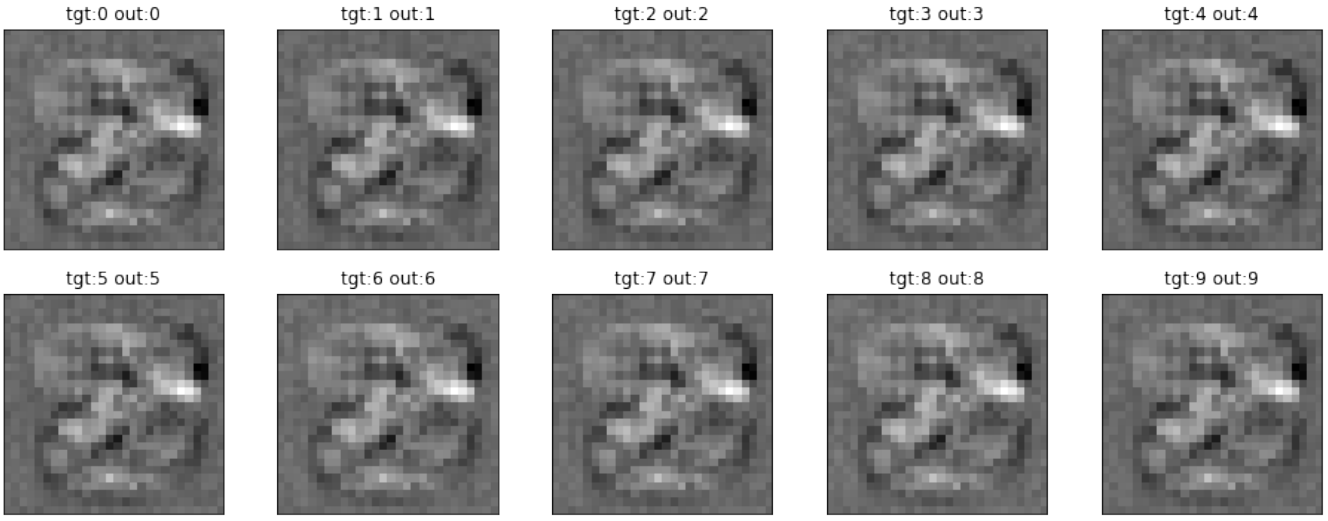
Let a parameterized subset of the pseudo-range be some subset which can be expressed *explicitly*. Of particular interest are subsets of the pseudo-range which get classified as a specific class. Denote these class subsets as:

$Y_c^+ \subseteq Y^+$

The problem is then to find a function/algorithm Pre such that:

$\mathscr{F} \circ Pre(Y_c^+) = Y_c^+$

That is, to find the process which returns the pre-image of the parameterized class range. It turns out that is problem has an explicit solution and presenting this solution is the heart of the matter.



SGD,p=0 , batch size 5, 5 epochs linear, z = np.random.rand(1)/100,delta = np.random.rand(10,1)/100. SGD weights produce more instability than Adam weights… just check the min and max of the pre to see if its possible.

5 Heart of the Matter

Initialization

Consider an arbitrary class c amongst k classes and denote its probability by p_c then:

Condition (1): $\text{argmax}(\hat{\mathbf{y}}) = c \Rightarrow p_c > \frac{1}{k}$

That is, the argmax of our output probability vector will be the target class c only if the probability of that class is greater than the "uniform threshold."

However, condition (1) is implied by another condition that will lay the foundation for the explicit definition of the class pre-image.

Condition (2): $\text{argmax}(\mathbf{z}^{[L]}) = c$

However, all vectors which satisfy condition (2) are precisely these vectors we defined to be Y_c^+. Thus:

$$Y_c^+ = \{\mathbf{z}_i \,|\, \text{argmax}(\mathbf{z}_i) = c\}$$

Furthermore we see that

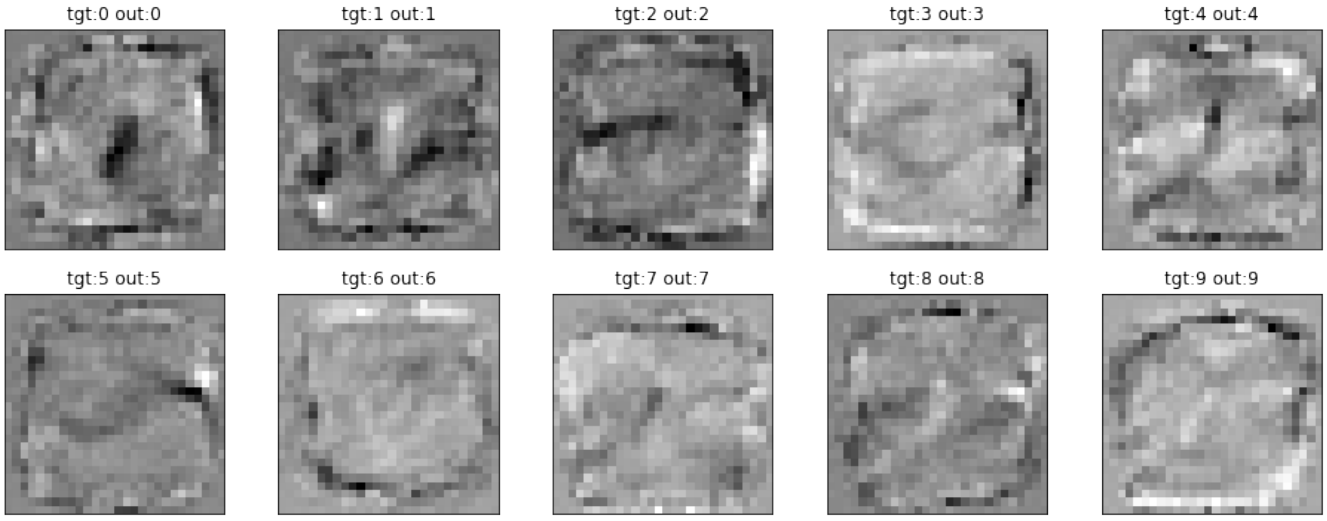
$$\text{softmax}(Y_c^+) = Y_c$$

The convenience of condition (2) is that it presents an explicit definition that is easy to work with. That is, a vector $\mathbf{z}_c^{[L]} \in Y_c^+$ is of the form:

Dim	Net1	Net2	Net3	Net4	Net5
Input	784	784	784	784	784
Layer1	10	128	128	128	512
Layer2		10	64	64	256
Layer3			10	32	128
Layer4				10	64
Layer 5					32
Layer 6					10
Adam	92.73	97.78	97.46	97.08	97.38
SGD	92.20	97.24	97.46	97.19	97.12
Rank	**Full**	**Full**	**Full**	**Full**	**Full**

Figure X: The number of hidden units for each hidden layer, the rank of *all* trained parameter matrices and the test accuracy for each network and optimizer used.

$$\mathbf{z}_c^{[L]} = \mathbf{z}_c - \delta = \begin{bmatrix} z_c - \delta_1 \\ \vdots \\ z_c \\ \vdots \\ z_c - \delta_k \end{bmatrix}$$



Adam, linear, p = 1000,z=0,d=0, no null, question: is p=1000 even possible given the restriction the input domain? It is interesting to note that these preimages are not as crisp for p < 5 but keep their relative crispness as p grows to values even up to 100,000,000. Unlike SGD linear which becomes noisy for large values.

Where

$\delta_i > 0$

And

$\delta_i, z_c \in \mathbb{R}$ (ie real variables).

That is, the delta vector ensures that the argmax is c. This delta vector and the z_c constant present the first set of parameters of the pre-image function.

We can then recover the family of previous activations that result in our family of z-vectors using:

$$\mathbf{a}_c^{[L]} = (\Theta^{[L]})^+(\mathbf{z}_c^{[L]} - \delta - \mathbf{b}^{[L]}) + \mathbf{n}^{[L]}$$

The Central Result

With the starting case defined, it is possible to extract the general backwards recurrence which goes from activation to activation and terminates at $X_c = \mathbf{a}_c^{[0]}$.

Backward Recurrence:

$$\mathbf{z}_c^{[l]} = f^{-1}(\mathbf{a}_c^{[l+1]})$$

$$\mathbf{a}_c^{[l-1]} = (\Theta^{[l]})^+(\mathbf{z}_c^{[l]} - \mathbf{b}^{[l]}) + \mathbf{n}^{[l]}$$

With special cases

$$\mathbf{a}_c^{[L]} = (\Theta^{[L]})^+(\mathbf{z}_c^{[L]} - \delta - \mathbf{b}^{[L]}) + \mathbf{n}^{[L]}$$

$$\mathbf{a}_c^{[0]} = X_c$$

6 Analysis

What has just been demonstrated is that it is possible to define any class pre-image in terms of the the parameter matrices and a set of vectors and parameters that are derived from the parameters themselves. Formally:

$$Pre(Y_c^+) = F(c, z_c, \delta, \{\Theta^{[l]}\}, N)$$

That is, the pre-image of any class is a function of the class number, the delta vector, the parameter matrices and the coefficients of the null space vectors.

Note: Within the pre-image function we must assume that a null-space bases have already been found. An algorithmic implementation would need to extract these.

7 Initial Experiments

It seems that leaky relu as the network grows deeper has a "higher" dimensional pseudo inverse than 10 dimensions. It seems that relu does not if don't include null parts. Furthermore it seems that relu might be

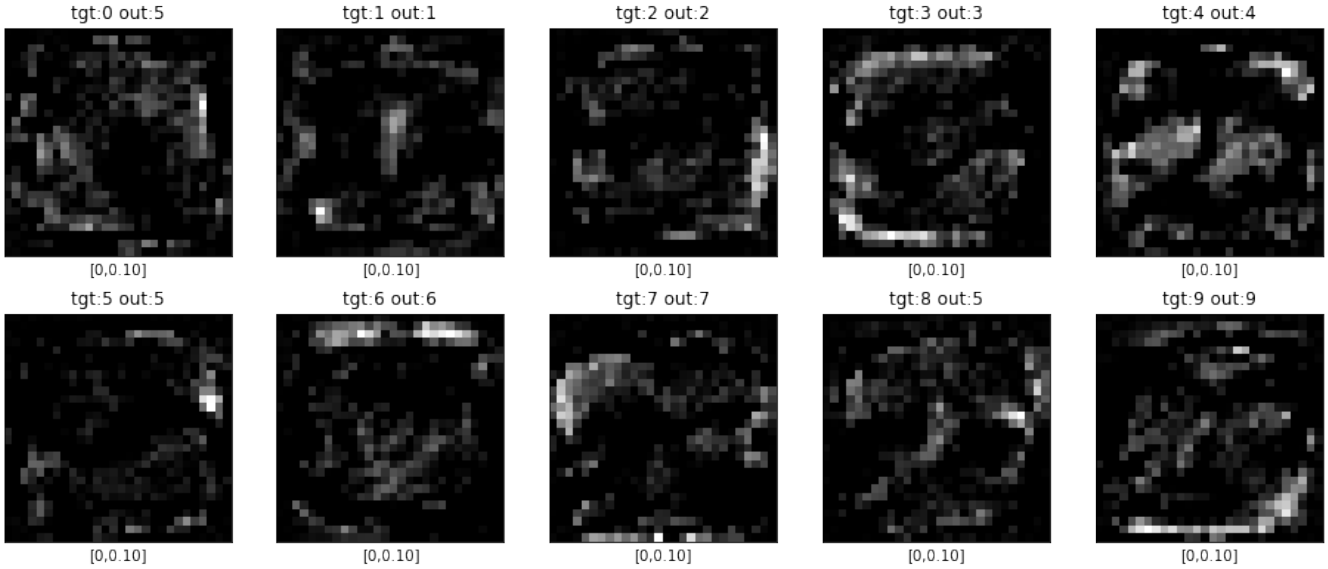


Figure 3: Even with a naive "forcing" method to make the preimage "natural" and constraining the pixel values to be between 0 and 0.10 the Adam preimages for the linear still produce 8 of the 10 the target outputs. This means that for these 8 classes the 774 null vectors can be used to generate arbitrary looking images and have this added on top without much loss in fidelity. This points to the fact that the robust nature of neural networks is a double edged sword - even something so arbitrary as zeroing all negative values and standardizing to a small range. Interestingly, SGD weights are not susceptible to this low fidelity attack.

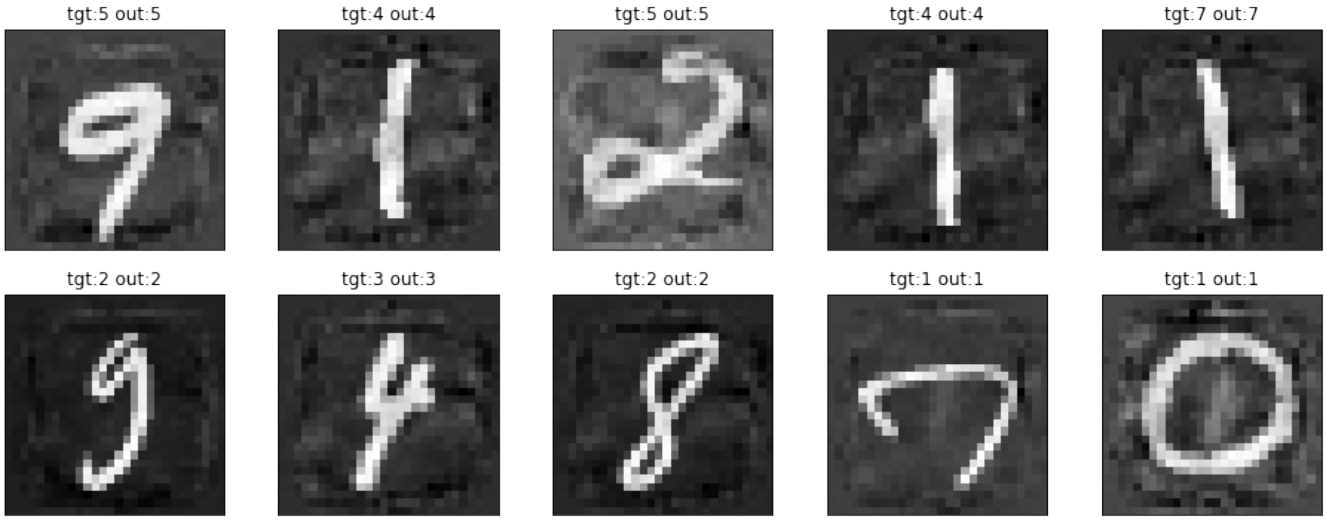


Figure 4: Adam linear adversarial Examples occur because of the robust freedom allowed by the null space vectors. If one removes the pseudo part from any example and adds the low foot print noise they can attack the network with these examples. Interestingly these examples where generated *exactly* no optimization needed. z = 0 p=1000 delta = np.zeros((10,1) noise_scale = 10 exclude = [0,8]

even easier to attack than leaky relu if we use the simple linear pseudo inverse of relu.

Data

Pytorch MNIST data was used for all experiments (60,000 training set and 10,000 test set). For simplicity, the data was used raw (ie no augmentation or centering).

Networks Architectures Used

The final non-linear mapping uses softmax. Other than this final layer, each layer uses a linear operation with

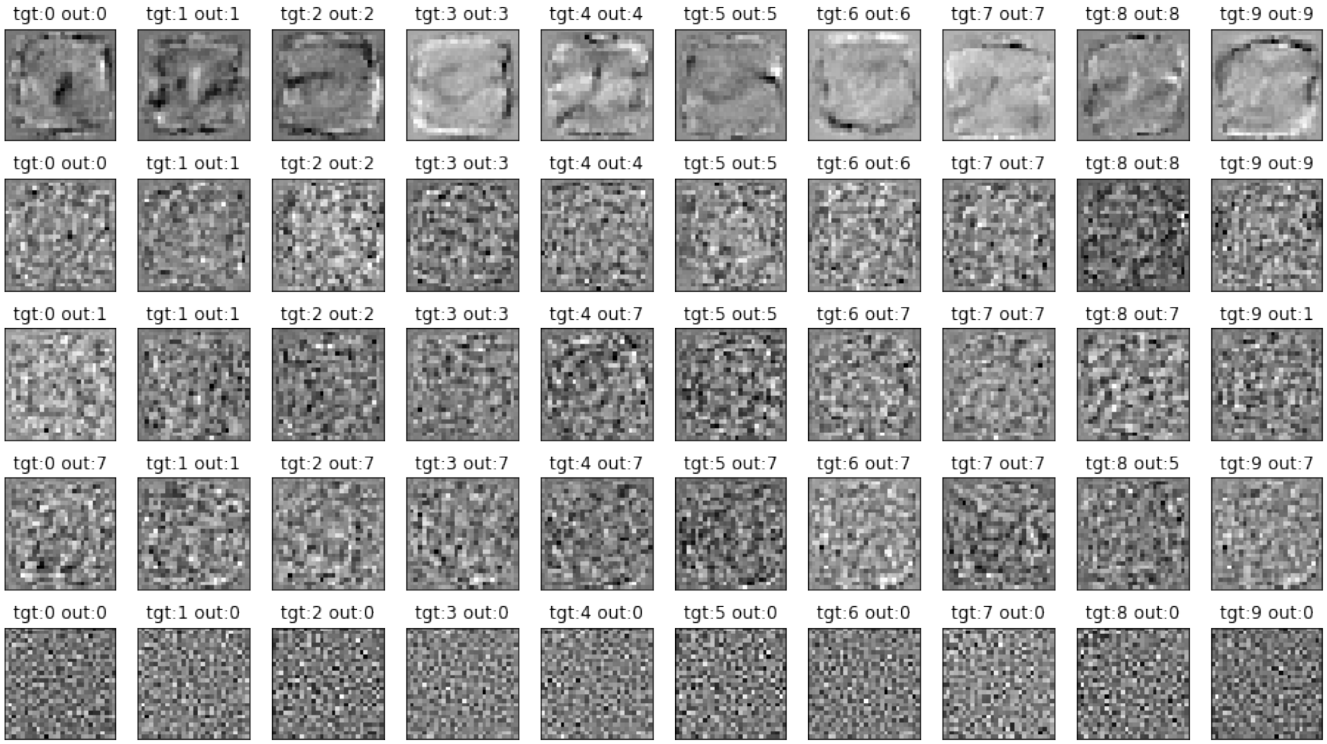
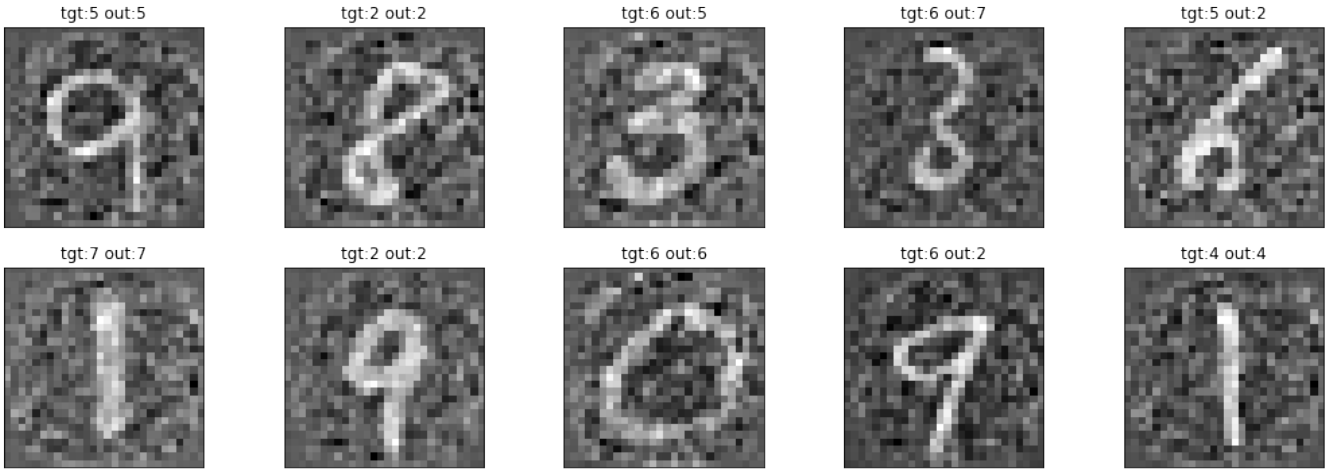


Figure 5: Adam preimages for each model tested with no null vectors. As the network gets deeper the numerical error begins to make the preimage calculation highly inaccurate. Furthermore, the results become more noisy. This could mean that the null vectors need to be chosen more strategically or point to a computational limit on numerical preimage calculation. The pixel values begin to grow large as the network gets deeper. For example, the first row has pixel ranges like [-0.2,0.2] but the final row has [-2000,2000].



Adversarial Examples for Adam-net1. Here tgt denotes the class we want the network to output.

a bias and leaky relu. Each layer uses the same negative slope for the leaky relu nonlinearity. Thus, all that must be specified to list the networks is the dimension of each feature mapping.

Training

Since this was an experiment involving the preimages of each trained network, there was no tuning. For comparison, two optimizers were used: Adam and SGD with momentum = 0.9. The learning rate for both

networks is 0.001. All models where trained for five epochs. The criterion was cross entropy loss.

Preimage Calculation

Numerical issues, padding p, difference between Adam & SGD

Plausibility of Preimage Based Attacks

Null Preimage Poisoning
Low Fidelity Attack
By removing the pseudo part of an input image one can "inject" adversarial data.

$$\tilde{\mathbf{x}} = \mathbf{x} - \mathbf{x}^{+} + \mathbf{a}$$

8 Limitations

The analysis herein serves as solid ground for understanding and extending to a general solution. However, much of this analysis was made possible by convenient simplifying assumptions. Furthermore, there are other more complex architectures that will require more granular analysis.

6 Refining the Result

It should be pointed out that the current result does not take into consideration constraints presented by natural data. For example, normalized pixel data restricts the preimage to vectors having components in the range [0,1]. The current recurrence can be refined further to incorporate these data restrictions.

Forward Constraining

To incorporate an approximate solution to the "natural" constraint on the z-vector all that needs to be done is forward propagate min-max constraints for each component. Using a simple lemma we can define the forward constraint:

Lemma:
$\forall \mathbf{x} \in [a,b]^n : \mathbf{argmax}_{\mathbf{x}}(\Theta\mathbf{x})_i = b\,||[\mathbf{row}_i(\Theta)^T]_+||_1 + a\,||[\mathbf{row}_i(\Theta)^T]_-||_1$

That is, the i-th component of a matrix multiplied by a vector with components constrained to the range [a,b] is maximized when the vector uses its minimum a on the negative components of the row vector and its maximum on the positive values. That is, the inner product tries to make each term in the sum as positive as possible.

This lemma puts bounds on what we can sample. That is, we cannot sample a z-vector with any part outside the bounds. However, it doesn't provide an exact way to find valid z-vectors. The problem is that once a component is fixed the bounds on the others might decrease and acquire "holes." Thus what is needed is a way to cascade from component to component. However, delineating such constraints is the subject of further research.

8 Further Research

Generalizing

To make the analysis complete, the case of variably ranked parameter matrices must be investigated. Furthermore, it should be investigated whether max rank is good assumption. That is, if neural networks tend to learn max rank parameter matrices.

In defining the inverse recurrence the invertibility of the activation was assumed. Surjective functions like Relu should be investigated.

Subsequent works or follow up research should consider generalizing the analysis to arbitrary activations and architectures (eg CNN, RNN, AutoEncoders).

Adversarial Examples

A discussion on the pre-image of neural network classes is incomplete without at least mentioning adversarial examples. It should be noted that the existence of adversarial examples means that there exist "encroachments" between the pre-images of different classes. That is, these messy overlaps contain vectors which satisfy a certain condition.

Attack Condition: A vector $\tilde{\mathbf{x}} \in X_a$ satisfying $||\tilde{\mathbf{x}} - \mathbf{x}_b|| < \epsilon$ for some $\mathbf{x}_b \in X_b$ and small value ϵ is an adversarial example.

The attack condition states that an attack vector must be in the pre-image of a class a but be *quantitatively* close to some vector from class b where the example $\mathbf{x}_b$ looks "natural." That is, it must look like it belongs to another class. This is nothing new, however, further

analysis of the pre-image recurrence could yield insights. Fleshing out the theory of adversarial attacks is critical for AI safety.

An interesting thought is that there might be rigorously definable attacks overtime on RNNs (eg feints). Finally, attempts should be made to define the universal properties of attacks on neural networks and general classes of "counter" measures should be devised. An interesting though: what if we added adversarial noise for all classes to "drown" out an attack in "competing noise" thus allowing the greater information present in the image as a whole to "dominate" - predict the raw image, predict the "doped" image and if predictions defer flag the image.

More rigorous analysis of adversarial attacks and adversarial example generation. For example: finding the minimum amount of adversarial noise to move across decision boundaries. Develop higher resuolution attacks using the preimage method.

SGD-Linear does not admit such a "naive" method.

Natural Constraints

Real data is constrained (eg normalized pixel values). A quick run down of the math and difficulty involved:

A good research endeavor would be to survey all experiments and open sourced code to verify that adversarial examples and high confidence noise generated do not go "out of bounds." Thus making these examples "invalid" in a "natural" sense.

Implementation

Closely tied to the natural constraint is the implementation.
Having theoretical understanding is not enough. Algorithms for finding pre-images and discovering vulnerabilities as well as "firewalls" should be implemented and stress tested.

Numerical Stability

Are there subsets of z-vectors which are inherently unstable to network pre-imaging? Can we find the stable parts? Is there a more effective way than using the pseudo inverse directly?

9 Conclusion

This work serves as the initial starting point for a series of publications to come. Where the points alluded to in the previous section will be explored and developed. It is hoped that this work serves to help ground our understanding of deep networks on a more rigorous and theoretical level.

10 Acknowledgements

10 Bibliography

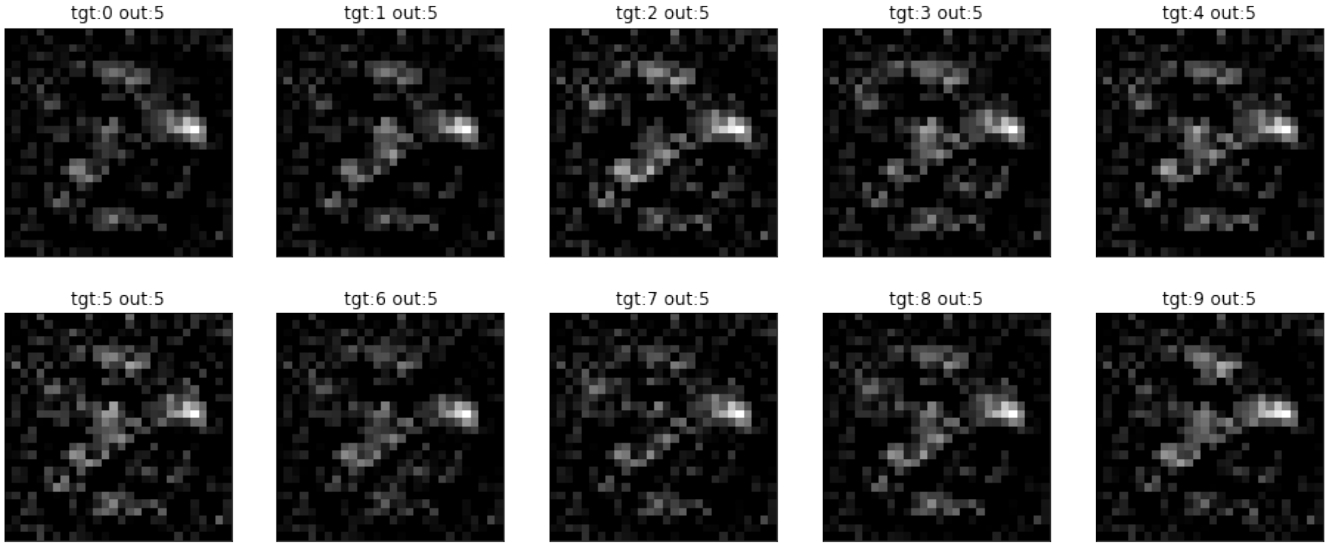
Montufar, G., Pascanu, R., Cho, K., & Bengio, Y. (n.d.) (2014). On the Number of Linear Regions of Deep Neural Networks. *ArXiv 402.1869v2*.

Basri, R., & Jacobs, D. (2016). Efficient Representation of Low-Dimensional Manifolds using Deep Networks. *ArXiv:1602.04723v1*.

Carlsson, S., Azizpour, H., & Razavian, A. (2016, November 4). doi: https://openreview.net/pdf?id=HJcLcw9xg

Mahendran, A., & Vedaldi, A. (2014). Understanding Deep Image Representations by Inverting Them. *ArXiv:1412.0035v1*.

Ochoa, B. (2015, January 16). The Null Space of a Matrix. Retrieved January 28, 2020, from https://



cseweb.ucsd.edu/classes/wi15/cse252B-a/nullspace.pdf

11 Appendices

The Admissibility of All Null Space Vectors

A concern one might have with using the null space vectors in the backward recurrence is whether or not *all* the null space vectors are admissible in the pre-image. That is, can we prove that no null space vector will shift the output to be in a different class?

At face value it is obvious that all null space vectors as admissible because they cannot affect the output. Thus they can only "broaden" the preimage. However, A direct proof is possible by substituting the definition of the pre-image into the network forward function:

First a new notation is necessary to for the proof:

$$\mathscr{F}^{[i:j]} = \mathscr{F}^{[j]} \circ \ldots \circ \mathscr{F}^{[i]}$$

That is, it is a sub path of the network forward function if we view the full network as a path.

Evaluating the network on the class pre-image we get:

$$\mathscr{F}(X_c) = \mathscr{F}^{[2:L]} \circ f(\Theta^{[1]}(\Theta^{[1]})^{+}(\mathbf{z}_c^{[1]} - \mathbf{b}^{[1]}) + \Theta^{[1]}\mathbf{n}^{[1]} + \mathbf{b}^{[1]})$$

This simplifies to

$$\mathscr{F}(X_c) = \mathscr{F}^{[2:L]} \circ f(\mathbf{z}_c^{[1]})$$

because $\mathbf{z}_c^{[1]} - \mathbf{b}^{[1]} \in C(\Theta^{[1]})$ and $\Theta^{[1]}\mathbf{n}^{[1]} = \mathbf{0}$
But we can again simplify this to

$$\mathscr{F}(X_c) = \mathscr{F}^{[2:L]}(\mathbf{a}_c^{[2]})$$

That is, we see that

$$\mathscr{F}^{[i:L]}(\mathbf{a}_c^{[i]}) = \mathscr{F}^{[i+1:L]}(\mathbf{a}_c^{[i+1]})$$

We can carry this all the way to the last layer

$$\mathscr{F}^{[L]}(\mathbf{a}_c^{[L-1]}) = \Theta^{[L]}\mathbf{a}_c^{[L-1]} + \mathbf{b}^{[L]}$$

But we can substitute the definition of $\mathbf{a}_c^{[L-1]}$ into the equation to obtain

$$\mathscr{F}^{[L]}(\mathbf{a}_c^{[L-1]}) = \Theta^{[L]}((\Theta^{[L]})^{+}(\mathbf{z}_c^{[L]} - \mathbf{b}) + \mathbf{n}^{[l]}) + \mathbf{b}^{[L]}$$

Which reduces to

$$\mathscr{F}^{[L]}(\mathbf{a}_c^{[L-1]}) = \mathbf{z}_c^{[L]} \quad \blacksquare$$

What this demonstrates is that the backward recurrence is admissible for all null space vectors. Additionally, there are no more vectors that could be in the pre-image because the row space is orthogonal to the null space and hence the null space basis combined with the row space basis are a full basis.