Deviations in Representations Induced by Adversarial Attacks

Anonymous submission

Abstract

Deep learning has been a popular topic and has achieved success in many areas. It has drawn the attention of researchers and machine learning practitioners alike, with developed models deployed to a variety of settings. Along with its achievements, research has shown that deep learning models are vulnerable to adversarial attacks. This finding brought about a new direction in research, whereby algorithms were developed to attack and defend vulnerable networks. Our interest is in understanding how these attacks effect change on the intermediate representations of deep learning models. We present a method for measuring and analyzing the deviations in representations induced by adversarial attacks, progressively across a selected set of layers. Experiments are conducted using an assortment of attack algorithms—on the CIFAR-10 dataset—with plots created to visualize the impact of adversarial attacks across different layers in a network.

Code is available at https://anonymized/for/submission.

1 Introduction

It has been shown that carefully crafted adversarial instances can deceive deep learning models (Szegedy et al. 2014). This prompted further research to (1) develop new attack techniques, (2) defend against instances created with such methods, and (3) explore and characterize the properties of adversarial attacks. Here we're interested in the latter task, seeking to understand how adversarial instances effect change on the hidden layers of their target networks.

Our Contribution We present a method for measuring and analyzing the deviations in representations induced by adversarial instances. Experiments are conducted with an assortment of attack algorithms, using two separate distance metrics for measuring deviations in representations. Our plots show the transitional effect on representations induced by adversarial attacks.

2 Preliminaries

There are various ways to generate adversarial instances. A typical approach starts with an input x and synthesizes a small additive perturbation Δx for which $x + \Delta x$ would deceive a neural network—whereas a human would not perceive much difference between x and $x + \Delta x$.

Here we briefly cover the three adversarial attack methods that we incorporate into our experiments. In addition to the x and Δx notation mentioned earlier, y represents a ground truth class label and J is a loss function for a neural network—which has already been trained in this scenario.

Fast Gradient Sign Method (FGSM) (Goodfellow, Shlens, and Szegedy 2015) generates an adversarial perturbation $\Delta x = \epsilon \cdot \operatorname{sign}(\nabla_x J(x, y))$, which is in the approximate direction of the loss function gradient. The sign function transforms each element to -1, 0, or 1, with the result then scaled by ϵ. Thus, the L_∞ norm of Δx is bounded by ϵ.

Basic Iterative Method (BIM) (Kurakin, Goodfellow, and Bengio 2017) applies FGSM iteratively. At each step, $x_t^{adv} = x_{t-1}^{adv} + \alpha \cdot \operatorname{sign}(\nabla_x J(x_{t-1}^{adv}, y))$, with initialization $x_0^{adv} = x$. On each iteration, the L_∞ norm of the latter addition term is bounded by α, and clipping at each step can be used to constrain the final x^{adv} to an ϵ-neighborhood of x.

Carlini & Wagner (CW) (Carlini and Wagner 2017) constructs an adversarial perturbation using gradient descent to solve $\Delta x = \operatorname{argmin}_\delta(\|\delta\|_p + c \cdot f(x+\delta))$. Function f is one for which $f(x+\delta) \leq 0$ if and only if the attack is successful on the target classifier; their f_6 formulation was found most effective. Positive constant c can be determined using binary search. A box constraint on $x + \delta$ is satisfied by clipping or change of variables. p specifies which norm is used.

3 Method

To inspect the deviations in representations induced by adversarial attacks, we start with a dataset of original (non-attacked) instances, and an attacked version of the same dataset. For the neural network that's targeted—or some other network, e.g., if the context is transferability—we select a set of checkpoints, locations in the network for which we'll extract representation vectors.

By passing an original instance x through the network, we extract representation vector ϕ_i at checkpoint i. We repeat this for the adversarial counterpart x', extracting ϕ_i'. Next, we measure the distance $d(\phi_i, \phi_i')$ between the vectors. While we don't specify a particular distance function d, our experiments use Euclidean distance and cosine distance.

The volume occupied by representations can differ at separate layers of a neural network. Therefore, it's important to normalize our distance measurements prior to comparing them across checkpoints. We follow the approach of Mahendran and Vedaldi, Section 4. Our earlier formulation $d(\phi_i, \phi_i')$ becomes $d(\phi_i, \phi_i')/N_i$. Normalization constant N_i

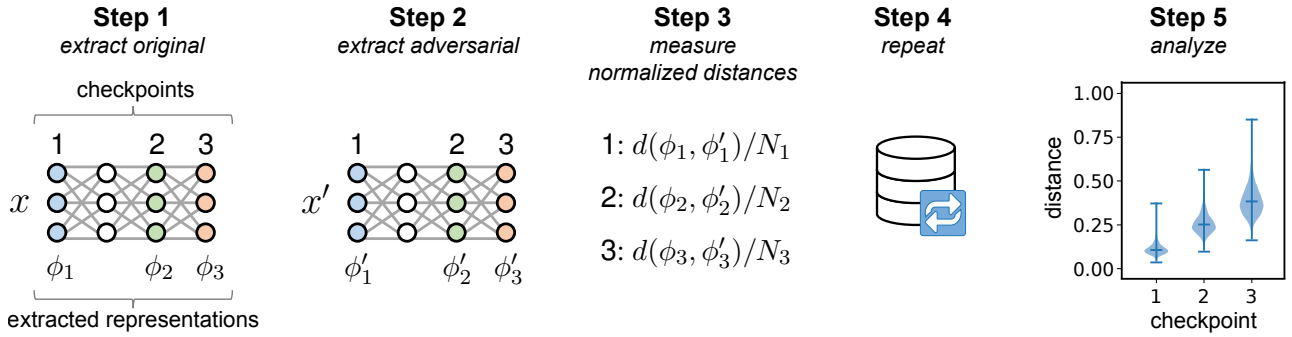

Figure 1: Method illustration. **Step 1**: Input x is passed through a neural network, with representation vectors ϕ_1, ϕ_2, and ϕ_3 extracted from selected checkpoints 1, 2, and 3. **Step 2**: Input x'—an adversarially perturbed version of x—is passed through the network, with representation vectors ϕ'_1, ϕ'_2, and ϕ'_3 extracted from the same checkpoint locations used earlier. **Step 3**: For each checkpoint, distance function d (e.g., Euclidean) is used to measure the distance between corresponding representations extracted for x and x'. The values are scaled using checkpoint-specific normalization constants, discussed further in Section 3. **Step 4**: The earlier steps, one through three, are repeated for a dataset of inputs. **Step 5**: The results are analyzed, showing the distribution of deviations induced by an adversarial attack at the selected checkpoints of the neural network.

(for checkpoint i) is the average pairwise distance between representation vectors—computed using d—across a sample of instances.

We calculate the normalized distances across all network checkpoints for the full dataset of original and attacked instances (limited in our experiments to include only pairings where the attack was successful). This provides a sample of deviations at each checkpoint, for which the distributions can be analyzed. We use violin plots, but other techniques could also provide insight. The method is illustrated in Figure 1.

4 Experiments

4.1 Experimental Settings

Our experiments utilized the CIFAR-10 dataset (Krizhevsky 2009), comprised of 60,000 32×32 RGB images—with a designated split into 50,000 training images and 10,000 test images—across 10 classes. Using the training data, we fit a neural network classifier that follows the kuangliu ResNet-18 architecture[1] with 11,173,962 parameters. With pixel values scaled by $1/255$ to be between zero and one, the network was trained for 100 epochs using Adam (Kingma and Ba 2015) for optimization, with the training data augmented via (1) random horizontal flipping and (2) random crop sampling on images padded with four pixels per edge. The resulting model had accuracy of 92.67% on the test data.

We use 10 checkpoints throughout the model for extracting representations. Figure 2 displays the model architecture, highlighting the locations of the numbered checkpoints. The shape and dimensionality of representations at each checkpoint are shown in Table 1.

Adversarial Attacks We utilized the `cleverhans` library (Papernot et al. 2018) to generate untargeted adversarial attacks for the 9,267 correctly classified test images.

[1]This differs in filter counts and depth from the ResNet-20 architecture that was used for CIFAR-10 in the original ResNet paper (He et al. 2016).

Table 1: Shape and dimensionality of representations at the checkpoints used for our experiments. Shapes are reported channels-first, followed by height then width (CHW format).

Checkpoint(s)	Shape	Dimensionality
1	$3 \times 32 \times 32$	3,072
2, 3	$64 \times 32 \times 32$	65,536
4	$128 \times 16 \times 16$	32,768
5	$256 \times 8 \times 8$	16,384
6	$512 \times 4 \times 4$	8,192
7	512	512
8, 9, 10	10	10

Attacks were conducted with the FGSM, BIM, and CW algorithms. The attacked images were clipped between zero and one for all attacks, and quantized to 256 discrete values—whereby each image could be represented in 24-bit RGB space—for FGSM and BIM. We did not quantize the CW-attacked images, as doing so would essentially revert the attack[2].

For FGSM, ϵ was set to $3/255$ for a maximum perturbation of three intensity values out of 255 for each pixel on the unnormalized data. Model accuracy was 16.80% on the 9,267 attacked images from the test dataset. That is, the attack succeeded at a rate of 83.20%.

Our attacks generated with BIM used 10 iterations with $\alpha = 1/255$ and $\epsilon = 3/255$. This limits the per-step maximum perturbation to one unnormalized intensity value per pixel, ultimately clipped to a maximum perturbation of three intensity values. The resulting model accuracy on the attacked images was 0.29% (i.e., an attack success rate of 99.71%).

We used CW with an L_2 norm distance metric along with

[2]The original CW paper (Carlini and Wagner 2017) addresses the issue with a greedy search process that restores the attack quality one pixel at a time—an approach that we did not utilize.

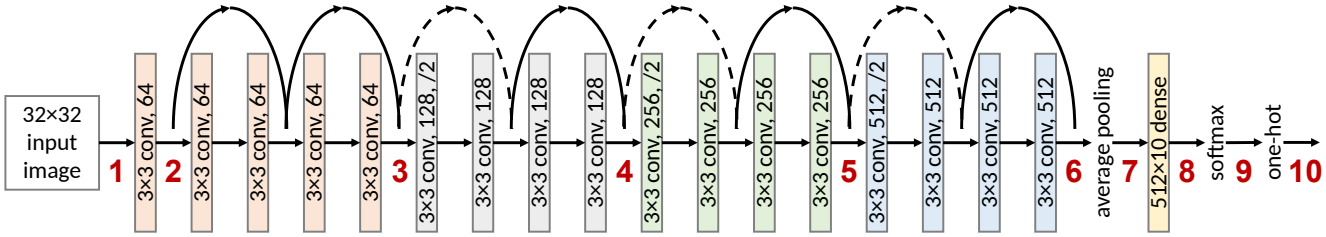

Figure 2: Model architecture and checkpoints. Red labels indicate where numbered checkpoints were placed in the ResNet-18 network. The first checkpoint captures the network's input prior to its subsequent processing. Solid arrow skip connections perform identity mapping, and the dashed arrow skip connections use pointwise convolution and stride to transform inputs to the necessary shapes for addition. The text in the colored rectangles indicates (1) filter shapes, counts, and stride when present (e.g., "/2" for stride of two), for convolutions, and (2) the weight matrix shape for the dense layer. Convolutional layers all include rectified linear unit (ReLU) activation functions. Where applicable, checkpoints are placed after the application of skip connection addition operations.

the package's default parameters—five binary search steps, a learning rate of 0.005, and up to 1,000 iterations. Accuracy after attack was 0.00%, a perfect attack success rate.

Figure 3 shows examples of randomly selected CIFAR-10 images and their adversarially perturbed counterparts.

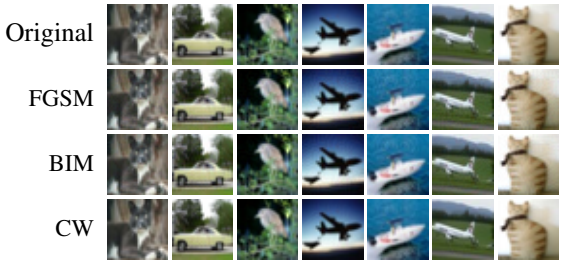

Figure 3: Example CIFAR-10 images before and after adversarial perturbation. The top row has the original images, followed by three rows corresponding to FGSM, BIM, and CW attacks. Images were randomly selected from the 9,267 test images that were correctly classified without perturbation (i.e., the set of images for which attacks were generated). The original image classes from left to right are `cat`, `automobile`, `bird`, `airplane`, `ship`, `airplane`, and `cat`.

Measuring Distances As a next step we extract representations at the 10 selected checkpoints. This is conducted for all 9,267 test images that were correctly classified originally. We then repeat the extraction for the adversarially perturbed counterparts of these same images. With representations extracted for both original and adversarially attacked images, for each image we measure distances between the original representations and corresponding adversarial counterparts. For our experiments we use two metrics for measuring the distance between vectors u and v, Euclidean distance $\|u-v\|$ and cosine distance $1-(u\cdot v)/(\|u\|\|v\|)$.

As we mentioned earlier, it's important to normalize the measured distances. We calculate normalization constants using the 9,267 test images that were correctly classified initially. For each checkpoint we extract the corresponding representations and then compute the average pairwise distance across all $\binom{9{,}267}{2} = 42{,}934{,}011$ pairs, to serve as divisors for scaling the originally calculated distances. This is done separately for Euclidean and cosine metrics. That is, the originally measured Euclidean distances are normalized by scaling constants calculated using Euclidean distance, and the cosine distances are scaled with normalization constants determined using cosine distance.

4.2 Results

We just discussed the details of how we measured the distances between representations of original images and their adversarial counterparts. To analyze the results, violin plots were employed to visualize the distribution of distance measurements so that we could see how adversarial attacks induce deviations progressively throughout the network.

Figure 4 shows the plots, which were generated across the three adversarial attacks and two distance metrics described earlier. For each attack algorithm, our analysis is limited to the subset of images that could be successfully attacked from the 9,267 correctly classified test images—7,710 images for FGSM, 9,240 for BIM, and 9,267 for CW. Because we only consider succesfully attacked images, the deviations at checkpoint 10 are constant—the distance between two non-equal one-hot vectors is always the same.

5 Discussion

Trend Something that stands out is that the average deviation in representations—induced by adversarially perturbing inputs—primarily increases as a function of depth (i.e., normalized distances tend to be larger for higher numbered checkpoints). This matches the general expectation we had prior to running the experiments. Interestingly, the pattern does not hold for checkpoint 8, where the average distance was lower than it was at checkpoint 7. Checkpoint 8 occurs after applying the linear portion of a densely connected layer, whereas the earlier checkpoints are chiefly positioned after the application of a convolution and ReLU activation function (the exceptions are checkpoint 1, which captures the input prior to subsequent network processing, and checkpoint 7, which follows a pooling operation).

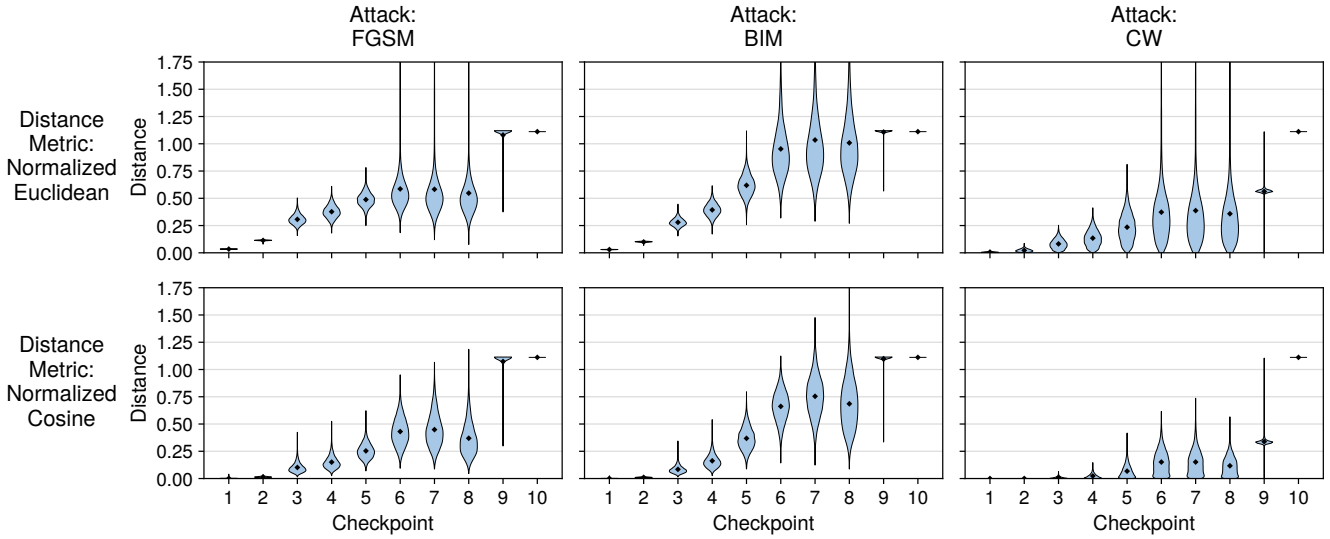

Figure 4: Violin plots showing the distribution of deviations in representations across selected checkpoint layers, induced by adversarial attacks. The figure subplots each correspond to a specific distance metric (indicated by the leftmost labels) and a specific attack algorithm (indicated by the header labels). Distance measurements were normalized. The diamond-shaped markers show sample means. We generated the plots using `matplotlib` (Hunter 2007), with the default settings of 100 points for Gaussian kernel density estimation and Scott's Rule for calculating estimator bandwidth.

Attacks Relative to the other attacks, it appears that BIM induces the smoothest transition in deviations from input to output. For FGSM there is a noticeable jump in normalized distances from checkpoint 8 to 9. A similar condition is present for CW, but occurs between checkpoint 9 and 10. For the configurations of attacks we considered, CW led to the smallest change in intermediate representations for original versus adversarially perturbed images.

Distance Metrics The first row's plots, which use Euclidean distance, are similar to the plots on the second row, which use cosine distance. The average normalized distances are always higher for the Euclidean metric than for cosine, not counting checkpoint 10 where the means are the same. This might be related to the property that two non-equal vectors in the same direction have zero cosine distance between them, but positive Euclidean distance. For vectors of slightly different direction, cosine distance would be small, whereas Euclidean distance could be large.

6 Conclusion

We set out to inspect how adversarial perturbations impact neural network representations. Our proposed technique considers the normalized representation distances at selected checkpoints between original images and their adversarially perturbed counterparts. Experiments conducted on CIFAR-10 data using an assortment of attack algorithms allowed us to visualize the deviations in representations induced by adversarial attacks. We observed different behavior—in the way normalized distances transition from input to output throughout the network—across the configurations of attacks we considered.

References

Carlini, N.; and Wagner, D. 2017. Towards Evaluating the Robustness of Neural Networks. *arXiv:1608.04644 [cs]*.

Goodfellow, I.; Shlens, J.; and Szegedy, C. 2015. Explaining and Harnessing Adversarial Examples. In *International Conference on Learning Representations*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778.

Hunter, J. D. 2007. Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering*, 9(3): 90–95. Conference Name: Computing in Science & Engineering.

Kingma, D.; and Ba, J. 2015. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*.

Krizhevsky, A. 2009. Learning Multiple Layers of Features from Tiny Images. Technical report.

kuangliu. 2017. https://github.com/kuangliu/pytorch-cifar.

Kurakin, A.; Goodfellow, I.; and Bengio, S. 2017. Adversarial examples in the physical world. *arXiv:1607.02533 [cs, stat]*.

Mahendran, A.; and Vedaldi, A. 2015. Understanding deep image representations by inverting them. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 5188–5196.

Papernot, N.; Faghri, F.; Carlini, N.; Goodfellow, I.; Feinman, R.; Kurakin, A.; Xie, C.; Sharma, Y.; Brown, T.; Roy, A.; Matyasko, A.; Behzadan, V.; Hambardzumyan, K.; Zhang, Z.; Juang, Y.-L.; Li, Z.; Sheatsley, R.; Garg, A.; Uesato, J.; Gierke, W.; Dong, Y.; Berthelot, D.; Hendricks, P.; Rauber, J.; Long, R.; and McDaniel, P. 2018. Technical Report on the CleverHans v2.1.0 Adversarial Examples Library. *arXiv:1610.00768 [cs, stat]*.

Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I.; and Fergus, R. 2014. Intriguing properties of neural networks. In *International Conference on Learning Representations*.