What Can Activation Patching Tell Us About Adversarial Examples?

Neil Rathi
Stanford University
rathi@stanford.edu

Katherine Yu
Stanford University
kyu2024@stanford.edu

Abstract

Although they achieve state of the art performance on vision tasks, deep neural networks have been shown to be vulnerable to adversarial examples, inputs that appear to be almost identical to ordinary inputs but cause the model to produce incorrect predictions. In order to interpret the behavior of vision models on adversarial examples, we use activation patching, a technique in which we patch in activations from a model given an ordinary input into another run of the model given an adversarial input. This allows us to determine which neurons in a model have the greatest causal importance in producing correct predictions. The task we examine is object classification on the ImageNet dataset, and we specifically analyze the neurons within the GoogLeNet architecture. We find that patching individual neurons or circuits can recover the original class predictions on samples, but identifying which activations to patch depends on both the true label and target label of the attack. Despite this limitation, we argue that patching can still help mitigate the risks from adversarial attacks, as it allows us to more deeply understand how models respond to these inputs.

1. Introduction

Good deep learning models for vision tasks are not robust to **adversarial examples**: small perturbations on clean inputs that lead to highly confident misclassification. These examples are often functionally invisible to humans (see Figure 1, but cause otherwise good and generalizable models to perform extremely poorly [20]. Szegedy et al. [20] generate these examples via straightforward gradient-based optimization, computing and applying a noise mask to an input image; however, other work has also shown that it is possible to construct physical adversarial examples—i.e. objects misclassified from images at a variety of angles [1]. Such examples can be extremely dangerous for vision models applied in the real world.

Since vision models are so complex, interpreting how the model acts on these adversarial inputs is very challenging. While past work on vision model interpretability has focused on explaining classifier predictions (e.g. SHAP, [12]) and understanding model internals as visual filters (e.g. DeepDream, [15]), little work has attempted to understand adversarial examples from the perspective of interpretability.

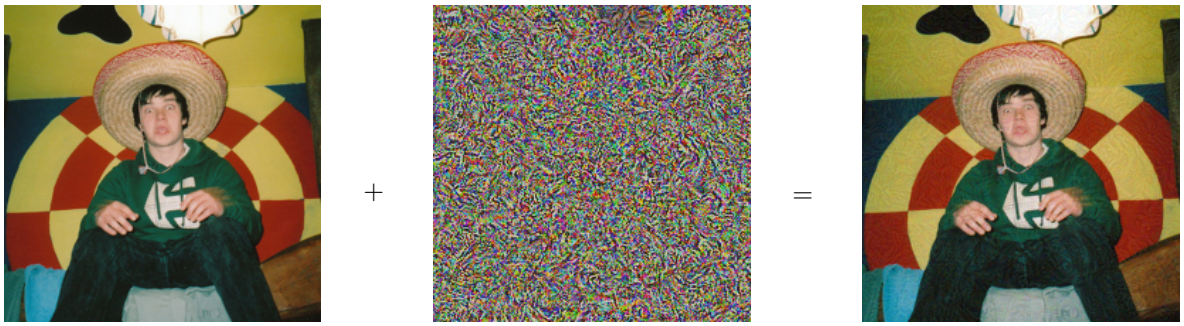
Here, we use techniques from the mechanistic interpretability of *language models* to better understand how vision models process adversarial inputs. In particular, we utilize **activation patching**, or **causal tracing**, a method used for component localization. At a high level, activation patching allows us to identify which model components (neurons, layers, attention heads) are used when processing certain inputs and for making certain predictions. Activation patching has recently been used to discover neural circuits in LLMs [5, 23, 24], and can be used for editing factual associations [14]. The technique makes use of 'clean' and 'corrupt' inputs to causally intervene on activations and determine which neurons are decisive in a model's predictions; in this sense, adversarial examples are a natural testing bed for activation patching.

Here, we investigate how a small pre-trained model (GoogLeNet, [19]) behaves on adversarial examples, using activation patching to identify components responsible for misclassification. The inputs we use are images in an existing dataset (ImageNet) along with adversarial examples we generate from them.

Broadly speaking, we are primarily interested in two questions.

(1) Can activation patching help localize components that respond to adversarial inputs?
(2) Can activation patching help us find a *universal* component that responds to adversarial inputs?

The difference between these questions is somewhat subtle. The goal of much mechanistic interpretability work is to argue that models *can* be completely decomposed into universal component parts—that is, these components should not be dependent on the input. We are interested in both questions: whether adversarial inputs trigger any localization (as it is very possible that they do not) and whether they



"sombrero"
90.69% confidence

PGD noise
(exaggerated 10x)

"magnetic compass"
100.00% confidence

Figure 1: Constructing adversarial examples for GoogLeNet using projected gradient descent [6, 10]. We add imperceptibly small noise (exaggerated tenfold here for visualization) computed by gradient ascent subject to minimizing the ℓ_∞ norm, which causes GoogLeNet to confidently misclassify the new example.

trigger a single 'adversarial input component.'[1]

In what follows, we first detail past work on adversarial examples their interpretability. We then detail the general methodology of the experiments, including both activation patching and generation of adversarial examples. In the following section, we describe the methods and results of three experiments: patching neurons, circuit discovery, and image reconstruction. Section 5 concludes.

2. Related Work

Deep neural networks are not robust to adversarial examples, and various attack methods have been formulated which can be categorized according to certain aspects of the attack. An untargeted attack aims only to produce an incorrect classification by the model while a targeted attack aims to have the model predict a specific label. Regarding knowledge of the model, white-box attacks assume complete access to the model, including its parameters, and black-box attacks may only have access to the training data, the logits outputted by the model, or the final predictions made by the model [7].

In white-box attacks, the task of finding adversarial examples can be formulated as an optimization problem. Szegedy et al. use a weighted sum of the magnitude of the noise and the loss of the model on the resulting image and target label as the objective, and L-BFGS as the optimization algorithm [20]. Improving on this technique, Goodfellow et al. use the Fast Gradient Signed Method (FGSM), in which they compute an untargeted adversarial example by adding the sign of the gradient of the model loss at the input and label scaled by a small factor to the original input. This can efficiently be computed by backpropagation, and is less expensive than L-BFGS. Additionally, the authors hypothesize that the linear nature of neural networks explains their vulnerability to adversarial perturbation, as well as being a possible cause for the generalization of adversarial examples across different architectures and training sets [6]. The transferability of adversarial examples across models is a property used in black-box attacks, in which the examples are generated for a substitute model and then tested on the target model. Since black-box attacks require much less information than white-box attacks, they are more practical and further establish that adversarial examples are not just a theoretical concern.

In general, approaches to the issue of adversarial examples have involved how we can design or train more robust models. A NIPS 2017 competition organized by Google Brain found that common defenses included gradient masking, which is a technique that reduces the effectiveness of attacks that use the gradient, such as by changing the model to make it non-differentiable or have zero gradient in most places. Other defense types include detecting adversarial inputs, although in many cases the detector can be attacked as well, and adversarial training, although the model tends to overfit to the specific attack used in training or learn gradient masking [11]. Madry et al. approach adversarial robustness by framing it as a min-max optimization problem, in which they seek a model that minimizes the maximum possible loss that any attack can induce. The method they use is adversarial training, in which all training examples are replaced with a perturbed counterpart. They claim that projected gradient descent, introduced by Kurakin et al. [10], is a universal "first-order adversary", in that it achieves a greater loss than other adversaries using only first order information about the model. The models that they train with this attack as the adversary are indeed robust to other attacks as well, maintaining a high accuracy on the MNIST dataset.

[1] By component here, we mean a single neuron or a 'circuit' of neurons, which refers to a sparse connected subgraph of the neural network that performs the same function as the network on a certain type of input.

Another observation is that larger models are more robust after adversarial training, suggesting that a more complex decision boundary is needed to classify adversarial examples correctly [13].

Focusing more on interpretability, Tao et al. propose an approach to detecting adversarial examples for face recognition models, which involves extracting a set of neurons, denoted attribute witnesses, that are critical for identifying specific human face attributes, e.g. eyes and nose. These neurons are found by determining which attributes and neurons have bi-directional relations, which means that changes in attributes lead to changes in neurons, and vice versa. A new model is constructed by enhancing the values of attribute witnesses and weakening the values of other neurons. Inconsistent predictions between the new model and original model indicate an adversarial input [21].

We also seek to interpret the behavior of vision models on adversarial examples, and we use the technique of activation patching, introduced by Meng et al. for the purpose of editing factual associations in GPT. The process of calculating a state's contribution to a correct factual prediction involves three steps: there is a clean run, in which a prompt is given to the model and the correct prediction is outputted, a corrupted run, in which the embedding of the subject in the prompt is corrupted with noise so that the model outputs an incorrect prediction, and a corrupted-with-restoration run, in which the model is run on the noisy embedding but an activation from the clean run is patched in. For the last run, if the correct prediction is outputted after patching, we can infer that the activations patched are causally important in producing the correct prediction [14].

Wang et al. extend this technique to the task of indirect object identification in GPT-2, where for the clean run the model is given the original prompt and in the corrupted run the prompt no longer has a clear indirect object completion but has the same sentence structure as the original prompt. Regarding the model as a set of circuits, they determine which circuits are involved in the task by tracing back the information flow through the model, beginning with the logits [23]. For our task, the inputs to the clean and corrupted runs of the activation patching naturally correspond to original and adversarial images, and we follow a similar process to trace information flow in vision models.

In order to understand best practices for activation patching, Zhang and Nanda examine the methodological choices of other works and compare the results of changing a single method in the process. Specifically, they compare corruption methods, evaluation metrics used for patching, and sliding window patching to single layer patching. For evaluation metrics, they consider logit difference and probability difference from patching. When probability is used as the metric and the corrupted input results in a negligible probability of the correct output, they note that negative model components, which hurt performance, are easily overlooked. This is due to probability values having to be non-negative, so logit difference does not suffer from the same issue. We take this result into consideration in our methodology. Another comparison involves sliding window patching, which means patching multiple adjacent layers in the model at once. Compared to single layer patching, this method results in more pronounced patch effect [24].

3. Methods

Following early work on the interpretability of vision models [17], we focus our experiments on GoogLeNet [19], a relatively small (22-layer) pre-trained ConvNet. We implement the model using PyTorch and TorchVision [18, 22]. Our goal with activation patching is to determine which neurons, or circuits of neurons, are responsible for misclassification of adversarial examples.

Mechanistic interpretability techniques can be applied at multiple levels of analysis; here, we define a neuron as a single filter in a convolutional layer. So, for example, in [19], GoogLeNet's first convolutional layer has exactly 64 neurons. We choose this level of analysis because it allows us to have relatively granular control (compared to say, entire layers) without being over-specific: it's unlikely that individual pixels would have a meaningful effect on output [24].

3.1. Activation Patching

Activation patching is a technique that allows us to determine causal relationships between neurons and model behavior. In essence, we run the model on a clean input, storing the activations at each layer. We then run the model on an input that differs in a single dimension, and at a single component, 'patch over' the activation(s) from the clean run (see Figure 2). If the output of the model is now equivalent to the output on the clean run, we can conclude that the dimension of difference causally depended on the patched component.

Formally, we consider tuples (x_{cl}, x_*) of a 'clean' image and a 'corrupted' image (i.e. an adversarial example). We first run a forward pass of the model on x_{cl}, caching the activations at each neuron i. We then run a forward pass of the model on x_*, storing outputs. Then, for each i, we run a forward pass of the model on x_* until neuron i, at which point we replace the activation at i with the cached value from the forward pass on x_{cl}. We then resume the forward pass on x_*, storing the outputs.

Our measure of interest is the **patch effect** defined by [24]. Following their notation, let cl, $*$, and pt be the clean, corrupted, and patched runs, respectively. Define the logit

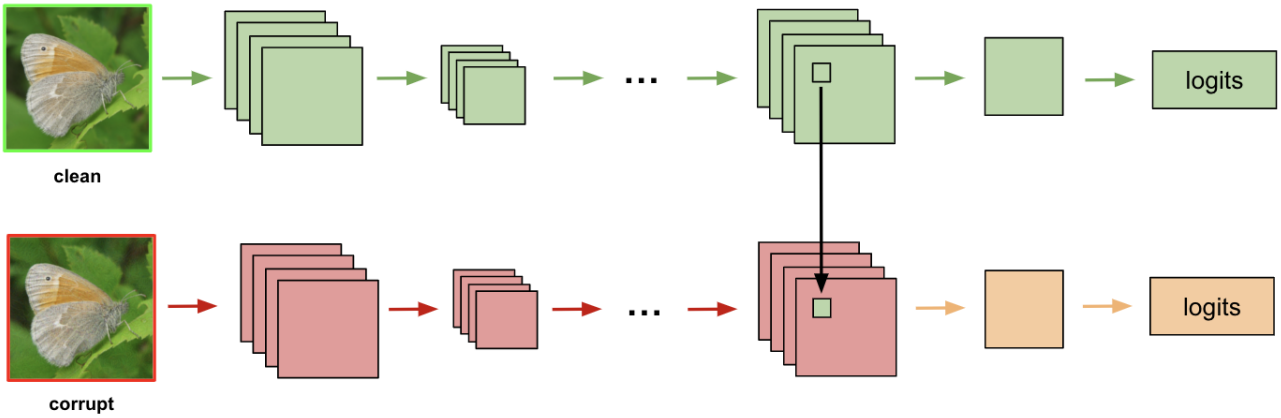


Figure 2: Activation patching. In the patched run, we intervene on the activation of a single component with its cached value from the clean run. We repeat this procedure for every component at our chosen level of analysis.

difference[2] between inputs x, y on a model run r as

$$\text{ld}_r(x, y) = \text{logit}_r(x) - \text{logit}_r(y). \quad (1)$$

We define the patch effect as

$$\text{patch\_effect}_r(x_{\text{cl}}, x_*) = \frac{\text{ld}_{\text{pt}}(x_{\text{cl}}, x_*) - \text{ld}_*(x_{\text{cl}}, x_*)}{\text{ld}_{\text{cl}}(x_{\text{cl}}, x_*) - \text{ld}_*(x_{\text{cl}}, x_*)}. \quad (2)$$

We normalize as in [23], which means that the patch effect is typically in the interval $[0, 1]$, where an effect size of 1 indicates that patching has completely recovered the clean run, and effect size of 0 indicates that the patching has no effect.

To locate relevant neurons, for each image, we sweep through all neurons in convolutional layers, patching in activations from the clean run. We do *not* consider non-convolutional layers as these are less theoretically interesting: they do not

3.2. Dataset

We evaluate the model on a random 256 image[3] subset of the ImageNet validation set [3].[4] As a benchmark, GoogLeNet achieves 74.22% accuracy on this dataset off-the-shelf.

To generate adversarial examples, we use a **projected gradient descent** (PGD) attack [10]. The PGD algorithm maximizes a loss function subject to a *constraint*; here, our constraint is the ℓ_∞ norm. In effect, our goal is to generate an input x_* that is minimally different from x_{cl} with regards to our constraint, such that our model predicts $y_* \neq y_{\text{cl}}$ with maximal confidence. The PGD update rule is given by

$$x_{t+1} = \pi_{x_t,\epsilon}(x_t + \alpha \cdot \text{sgn}(\nabla_x J(\theta, x_t, y_{\text{cl}}))), \quad (3)$$

where $\pi_{x_t,\epsilon}(\cdot)$ is a projection function that clips each element of the input to within a distance of ϵ, a hyperparameter specifying the size of the perturbation, from x_t. In our implementation, we use the `torchattacks` library [8] with a simplex projection and $\alpha = 0.001$ over ten steps. We select corrupted labels y_* randomly.

4. Experiments

In what follows, we detail three experiments on activation patching with GoogLeNet. In Experiment 1, we examine activation patching on individual neurons. In Experiment 2, we explore how activation patching might help us identify relevant circuits of neurons across layers. In Experiment 3, we study how activation patching can be used to re-construct 'clean' inputs from corrupted ones.

4.1. Experiment 1: Patching Neurons

Here, we patch the activations of single neurons in GoogLeNet. Recall that we are particularly interested in two questions:

(1) How does patching activations affect model performance on adversarial inputs?
(2) Is there a single universal 'adversarial input' neuron or circuit of neurons?

If the answer to (2) is *no*, then we would additionally like to know *what* influences which neurons are relevant when parsing adversarial inputs—is it a function of the clean input or the corrupt label?

[2] The patch effect can also be defined in terms of other difference metrics, including output probability and KL divergence [9]. However, logit difference is the standard metric for work on activation patching. See [24] for more discussion.

[3] We use only 256 images due to compute costs—activation patching is computationally intensive, as each image requires a run of the model for *each* neuron.

[4] We choose the validation set rather than the test set in order to verify accuracy, as ImageNet test labels are not made available.

4.1.1 Methods

For each input x_{cl}, we run the PGD adversarial attack described in Section 3.2 to generate a corrupted input x_*. We then follow the procedure outlined Section 3.1: we run the model on x_{cl} and x_*; then, for each neuron i we run the model on x_*, patching the activation at neuron i from the run on x_{cl}.

We consider three sets of inputs. The first is simply our testing set, to which we apply a PGD attack targeting *random* labels y_*. For the second, we use the same clean input for all runs of the model, and generate corrupted inputs by targeting 256 *different* labels; similarly, for the third, we use 256 different clean inputs with the same corrupted label.

This allows us to probe how the clean and corrupted input modulate which neurons are relevant: if the neuron is modulated by the clean label, we expect to see the same relevant neuron across all examples in the second set; if the neuron is modulated by the corrupted label, we expect to see the same relevant neuron across all examples in the second set.

GoogLeNet achieves 0% accuracy on all of our corrupted datasets off-the-shelf.

4.1.2 Results

We find that patching single neurons *does* indeed impact model performance; however, we do *not* find evidence for a 'universal' neuron which responds to adversarial inputs.

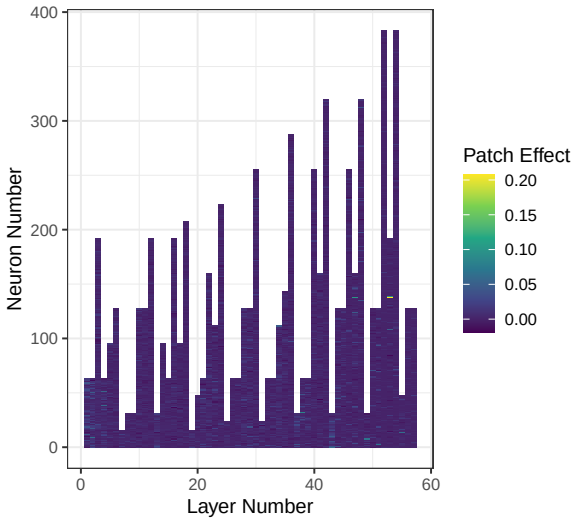


Figure 3: Example of patching neuron activations on a single input. The y-axis labels the neurons for each convolutional layer (N.B. layers vary in number of neurons); fill corresponds to patch effect. We see a single spike in patch effect at layer 53, `inception5b.branch2.0.conv`, in neuron 138.

Figure 3 details an example of patching neuron activations for one input image. We see that exactly one neuron, `inception5b.branch2.0.conv.138`, induces a much larger effect when patched: the mean patch effect across neurons for this example is 0.002 ± 0.0001 (95% CI), while the patch effect for neuron 138 is 0.208.

We notice similar patterns for all input images. We run a one-sample t-test on the maximum patch effect (over neurons) across all examples (against $\mu_0 = 0$) and find that for each example, a single neuron does indeed cause a significant increase in patch effect ($p < 0.001$). On average, this neuron induces a 0.200 ± 0.01 (95% CI) increase in patch effect from the mean.

However, these neurons are unique to the input: for our 256 images, there are 184 unique neurons which, when patched, induce a major increase in patch effect. We also find that holding the clean image or target label constant does not totally control the effect: we find 54 and 87 unique neurons in these conditions, respectively (with average increases in patch effect 0.317 ± 0.049 and 0.179 ± 0.008). Preliminary analysis does not reveal an immediate correlation between inputs or labels which are dependent on the same neuron. So, it is clear that the specific neuron that fires is not a straightforward function of the clean input or corrupt label, but rather likely a function of features from both.

On the other hand, nearly *all* of these neurons occur very late in the model. Of the 256 neurons, 254 are in layer 32 or after, and 161 of these are in the final 15 layers. Intuitively, this means that GoogLeNet parses clean and corrupted inputs similarly in its early layers, and only begins to parse differences later. Or, in other words, PGD attacks exploit neurons in later layers. This resembles prior mechanistic interpretability work on early vision circuits in GoogLeNet—early layers act as filters that parse basic features [16], e.g. as curve detectors or Gabor filters [4]. Since these basic features are common to both the clean and corrupted inputs, patching early activations does not significantly affect the model.

4.2. Experiment 2: Patching Circuits

We next examine **circuits**, or sets of connected neurons across layers, using methods from [23] and [2]. In particular, we use a version of the **Automatic Circuit Discovery** (ACDC) algorithm [2].

4.2.1 Methods

To identify a circuit of neurons, ACDC considers a network to be a computational graph G, where our goal is to isolate a maximally sparse subgraph H such that H and G have approximately the same output for some type of input. For each input to G, we begin by setting $H \leftarrow G$ and perform-

(a) Clean. "wooden spoon," 99.98% conf.

(b) Corrupt. "ox," 99.94% conf.

(c) Patched. "wooden spoon," 99.83% conf.

Figure 4: Corrupted input generated via PGD. The model correctly classifies the clean image, but misclassifies the corrupted image. The model correctly classifies the corrupted image after patching an entire circuit, with high confidence. Note that the patched image is *not* different from the corrupt image; instead, the model activations for each neuron in the circuit are patched from the clean run.

ing a clean run. Then for each node $v \in H$, we iteratively patch each parent of v from the clean run, and measure the patch effect. If this effect is below a certain threshold τ, we remove the node from H.

We run ACDC with $\tau = 0.1$ on the same dataset as Experiment 1, with the same random target labels.

4.2.2 Results

Similarly to Experiment 1, we find that patching circuits *does* impact model performance, but fail to find a single 'universal' circuit that is causally responsible for misclassification.

For each input example, we find a single, highly sparse circuit which has a causal effect on adversarial input classification. The mean logit difference after patching is 0.261 ± 0.019, which is higher than the average when patching a single neuron; running a t-test against $\mu_0 = 0$ we reach significance, $p < 0.001$.

The circuits are sparse: the largest consists of just 11 nodes, and most circuits consist of just 2-3 nodes; additionally, they almost all consist of neurons in the final 20 layers. However, as with single neuron patching, these circuits are unique to the input.

4.3. Experiment 3: Reconstructing Images

Finally, we consider whether activation patching on single neurons can be used to 're-construct' clean images from corrupt ones. We do so using projected gradient descent on patched activations.

Note that this method requires the original activations—as such, it is not a feasible counter to adversarial attacks. Rather, we're primarily concerned with whether a single clean activation alone is enough to reverse-engineer a clean image from a corrupted one.

4.3.1 Methods

To reconstruct clean images, we utilize PGD. Our goal is to generate an image that is 'as close as possible' to the corrupt image, but also matches the patched activations from x_{cl} at the target neuron i. For each image, we take the target neuron as the causal neuron found in Experiment 1.

More formally, recall that in PGD, we are optimizing for a loss function subject to a constraint on distance, such as the ℓ_∞ norm. Here, we consider the loss function to be the **mean squared error** between the model activations at layer L and the patched activations at neuron i. We run PGD on corrupt inputs x_* to generate reconstructed inputs x_{rc} such that at neuron i, the activations for x_{rc} are maximally close to the activations for x_{cl}, while x_{rc} and x_* are minimally different (note that Equation 3 describes gradient *ascent*, whereas here, we perform gradient *descent*).

4.3.2 Results

We find that reconstruction via PGD on activations significantly improves classification of corrupted images. Figure 5 shows an example of reconstruction with patched activations.

Recall that GoogLeNet achieves 74.22% accuracy on our clean inputs, and 0% accuracy on the corrupted inputs. After reconstruction, GoogLeNet correctly classifies 21.86% of inputs. In particular, 22.66% of images are classified with the same label in both the clean and reconstructed

(a) "zebra," 99.44% confidence (b) "tiger," 99.93% confidence (c) "zebra," 98.08% confidence

Figure 5: Reconstructing 'clean' images from adversarial examples, using PGD on patched activations. Image (a) shows the original clean input, (b) shows the corrupted adversarial example, and (c) shows the reconstructed input, with GoogLeNet classifications and confidence.

cases, whereas only 7.03% of corrupt images are classified the same after reconstruction. This means that modulating for only a single activation is enough to re-construct an image that is tagged with the correct label.

5. Conclusion

We investigated how GoogLeNet processes adversarial examples, through the framework of mechanistic interpretability. In particular, we isolated localized components that are responsible for the misclassification of adversarial inputs, using the causal mechanism of activation patching. Overall, we find that mechanistic interpretability provides a useful framework within which to study adversarial attacks, since it allows us to explicitly test causal hypotheses about localization.

We find that indeed, when GoogLeNet parses adversarial inputs, the 'adversarial effect' is localized to a single neuron or a small, sparse circuit. However, we do not find evidence suggesting that there is a 'universal' component that is responsible for the misclassification of adversarial examples. This makes intuitive sense: these inputs are unique and depend on both the original 'clean' input *and* the new 'corrupted' label, so there is no one component that parses them. This cuts against one of the central goals and claims of mechanistic interpretability work, that neural networks can be completely linearly decomposed into components independent of input.

On the other hand, it is possible (and likely) that there is another explainable underlying mechanism behind *which* neurons and circuits are important given a clean and corrupt input pair. Future work would benefit from examining potential sources of this mechanism, such as specific visual features of the input (e.g. after applying filters). In addition, due to compute limitations, we were restricted to just 256 images, so future work would also benefit from performing these interventions on a greater number of examples, as well as a greater variety of data. Further, smaller models are easier to decompose, so running similar experiments on a smaller model with a simpler dataset (e.g. MNIST) could help inform this work.

Contributions

Neil led the work on activation patching, and the methods and experiments sections of the writeup. Katherine led the work on constructing adversarial examples, as well as the literature review section of the writeup.

References

[1] A. Athalye, L. Engstrom, A. Ilyas, and K. Kwok. Synthesizing robust adversarial examples, 2018.

[2] A. Conmy, A. Mavor-Parker, A. Lynch, S. Heimersheim, and A. Garriga-Alonso. Towards automated circuit discovery for mechanistic interpretability. *Advances in Neural Information Processing Systems*, 36:16318–16352, 2023.

[3] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[4] D. Gabor. Theory of communication. part 1: The analysis of information. *Journal of the Institution of Electrical Engineers-part III: radio and communication engineering*, 93(26):429–441, 1946.

[5] A. Geiger, H. Lu, T. Icard, and C. Potts. Causal abstractions of neural networks. *Advances in Neural Information Processing Systems*, 34:9574–9586, 2021.

[6] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples, 2015.

[7] S. Han, C. Lin, C. Shen, Q. Wang, and X. Guan. Interpreting adversarial examples in deep learning: A review. *ACM Comput. Surv.*, 55(14s), jul 2023.

[8] H. Kim. Torchattacks: A pytorch repository for adversarial attacks, 2021.

[9] S. Kullback and R. A. Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[10] A. Kurakin, I. Goodfellow, and S. Bengio. Adversarial machine learning at scale, 2017.

[11] A. Kurakin, I. Goodfellow, S. Bengio, Y. Dong, F. Liao, M. Liang, T. Pang, J. Zhu, X. Hu, C. Xie, J. Wang, Z. Zhang, Z. Ren, A. Yuille, S. Huang, Y. Zhao, Y. Zhao, Z. Han, J. Long, Y. Berdibekov, T. Akiba, S. Tokui, and M. Abe. Adversarial attacks and defences competition, 2018.

[12] S. Lundberg and S.-I. Lee. A unified approach to interpreting model predictions, 2017.

[13] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks, 2019.

[14] K. Meng, D. Bau, A. Andonian, and Y. Belinkov. Locating and editing factual associations in gpt, 2023.

[15] A. Mordvintsev, C. Olah, and M. Tyka. Inceptionism: Going deeper into neural networks. `https://research.google/blog/inceptionism-going-deeper-into-neural-networks/`, 2015.

[16] C. Olah, N. Cammarata, L. Schubert, G. Goh, M. Petrov, and S. Carter. An overview of early vision in inceptionv1. *Distill*, 5(4):e00024–002, 2020.

[17] C. Olah, N. Cammarata, L. Schubert, G. Goh, M. Petrov, and S. Carter. Zoom in: An introduction to circuits. *Distill*, 5(3):e00024–001, 2020.

[18] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[19] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[20] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks, 2014.

[21] G. Tao, S. Ma, Y. Liu, and X. Zhang. Attacks meet interpretability: Attribute-steered detection of adversarial samples, 2018.

[22] TorchVision maintainers and contributors. Torchvision: Pytorch's computer vision library. `https://github.com/pytorch/vision`, 2016.

[23] K. Wang, A. Variengien, A. Conmy, B. Shlegeris, and J. Steinhardt. Interpretability in the wild: a circuit for indirect object identification in gpt-2 small. *arXiv preprint arXiv:2211.00593*, 2022.

[24] F. Zhang and N. Nanda. Towards best practices of activation patching in language models: Metrics and methods, 2024.