Proceedings of Machine Learning Research vol 107:1–26, 2020

Deep Learning Interpretation: Flip Points and Homotopy Methods

Roozbeh Yousefzadeh ROOZBEH.YOUSEFZADEH@YALE.EDU
Departments of Computer Science and Genetics
Yale University, New Haven, CT

Dianne P. O'Leary OLEARY@CS.UMD.EDU
Department of Computer Science and Institute for Advanced Computer Studies
University of Maryland, College Park, MD

Abstract

Deep learning models are complicated mathematical functions, and their interpretation remains a challenging research question. We formulate and solve optimization problems to answer questions about the models and their outputs. Specifically, we develop methods to study the decision boundaries of classification models using *flip points*. A flip point is any point that lies on the boundary between two output classes: e.g. for a neural network with a binary yes/no output, a flip point is any input that generates equal scores for "yes" and "no". The flip point closest to a given input is of particular importance, and this point is the solution to a well-posed optimization problem. To compute the closest flip point, we develop a homotopy algorithm to overcome the issues of vanishing and exploding gradients and to find a feasible solution for our optimization problem. We show that computing closest flip points allows us to systematically investigate the model, identify decision boundaries, interpret and audit the model with respect to individual inputs and entire datasets, and find vulnerability against adversarial attacks. We demonstrate that flip points can help identify mistakes made by a model, improve the model's accuracy, and reveal the most influential features for classifications.

Keywords: deep learning, homotopy optimization, decision boundaries, interpretability

1. Introduction

A deep learning model, after it is trained, can be viewed as a function that takes certain types of inputs (e.g., images) and maps them to outputs (e.g., classes). It is, therefore, a "deep learning function" (Strang, 2019). Understanding the behavior of such functions is commonly referred to as "interpretation" in the deep learning literature and it is considered a hard problem because of the complexity of the models.

Here, we take the mathematical view that any classification model is defined by the decision boundaries it defines between output classes, e.g., a linear regression model is defined by a hyperplane. This view also holds for deep learning classifiers, although their decision boundaries are geometrically complex. For definiteness, we focus our experiments on deep learning models for image classification, although our results apply to other models. We show how points on the decision boundaries can be computed and used to yield insight about deep learning functions.

1.1. Related work

Some methods interpret the output of a model by providing a saliency map showing the influence of individual pixels in the classification of an image. Most of these methods rely on first-order derivatives of the loss function and some post processing of the derivatives to investigate the sensitivity of the classification to a step in one particular direction. However, a deep learning model is typically a nonlinear non-convex function, and in some cases, the direction to the decision boundary may have a large angle with the direction of first-order derivatives (Yousefzadeh and O'Leary, 2019). Hence, those interpretation methods may be unreliable, as shown empirically by Ghorbani et al. (2019). Recently, Singla et al. (2019) considered the effect of second-order derivatives, which is an improvement over first-order methods, but still an approximation. Here, we determine the direction to the decision boundaries (rather than approximating them) by actually computing points on those boundaries, using a global optimization approach.

Studies on adversarial robustness are relevant, because changing the label of images requires crossing the decision boundaries. Shafahi et al. (2019) used high-dimensional geometry to derive a bound on the distance between an image and its nearest flip point under certain probabilistic assumptions. However, many of the adversarial methods do not directly minimize the distance to the decision boundaries (e.g., Jetley et al. (2018), Fawzi et al. (2017), and Madry et al. (2018)). One of the well-known attacks is DeepFool (Moosavi-Dezfooli et al., 2016) which takes gradient-based steps until it crosses a decision boundary, but the objective function is not minimizing the distance. The method proposed by Brendel et al. (2018) relies on random perturbation of the image, which has significant computational limitations.

Some authors study decision boundaries through simplifying assumptions. For example, Ribeiro et al. (2016) assume that decision boundaries can be locally approximated by hyperplanes. They apply multiple random perturbations to an image until enough images are obtained with two opposite labels. They then approximate the boundary by linear regression. The simplifications embedded in this approach can lead to explanations that contradict the classification of a model. Fawzi et al. (2018) also showed that decision boundaries of deep learning models are curved and not reliably approximable with hyperplanes.

Another approach, proposed by Chen et al. (2019) is to train an image classifier using prototypes such that the trained model is interpretable with respect to the prototypes. This approach although insightful, is not applicable to interpret general models not trained with prototypes.

Other methods interpret a model using counterfactuals. For example, for a given image and a trained model, Goyal et al. (2019) find a similar image with another label (i.e., a counterfactual) and identify spatial regions in those images such that changing them flips the classification from one label to the other. Although informative, this approach can be prohibitively expensive in practice, as it requires pairwise comparison of images.

Koh and Liang (2017) and Koh et al. (2019) have used influence functions to guide the perturbation and interpret black-box models with emphasis on finding the importance of individual points in the training data, but their method cannot be used to explain outputs of the models or to investigate the decision boundaries.

There are studies on expressivity and approximability of deep learning models such as Blcskei et al. (2019), and also on geometry of decision boundaries such as Shamir et al. (2019), but those approaches are not yet applicable to image classification models.

1.2. Our plan

In Section 2, we give a mathematical definition of the flip point closest to a given input, and we discuss its computation in Section 3. For a family of neural networks with a tunable activation function, we propose a homotopy algorithm for this computation. In Section 4, we discuss how flip points provide valuable information to the user of a deep learning model, and in Section 5 we discuss the insights they provide to model designers. Section 6 illustrates our ideas on two datasets. Section 7 is our discussions, and we conclude in Section 8.

2. Defining flip points for deep learning models

We denote the deep learning function by $\mathcal{N}$ and assume that its output is a continuous function of its inputs. We use $\boldsymbol{x}$ for the vector of inputs and $\boldsymbol{z}$ for the vector of outputs, where $\boldsymbol{z} = \mathcal{N}(\boldsymbol{x})$. Each element of $\boldsymbol{z}$ corresponds to a specific class and the element of $\boldsymbol{z}$ with the largest value is the classification of model.

We define a *flip point* to be any point on the decision boundary of a model. Decision boundaries partition the domain of a classification function into classes. Decision boundaries have been previously considered in various studies, e.g., by Elsayed et al. (2018), however, finding the minimum distance to decision boundaries has been considered intractable.

Binary classification case. To make the concept of flip points clear, let's consider an image classification model with 2 output classes. For definiteness, we refer to the output as a classification of "cancerous" or "noncancerous". For convenience, we assume that the output $\boldsymbol{z}(\boldsymbol{x})$ is normalized (perhaps using $\mathrm{softmax}$) so that the two elements sum to one. Since $z_1(\boldsymbol{x}) + z_2(\boldsymbol{x}) = 1$, we can specify the classification by a single output: $z_1(\boldsymbol{x}) > \frac{1}{2}$ is a classification of "cancerous", and $z_1(\boldsymbol{x}) < \frac{1}{2}$ is a classification of "noncancerous". If $z_1(\boldsymbol{x}) = \frac{1}{2}$, then the classification is undefined, which would correspond to a flip point.

Now, given an output $z_1(\boldsymbol{x}) \neq \frac{1}{2}$ for a particular input $\boldsymbol{x}$, we want to investigate how changes in $\boldsymbol{x}$ can change the classification, for example, from "cancerous" to "noncancerous". In particular, it would be useful to find the *least change* in $\boldsymbol{x}$ that makes the classification change by reaching a decision boundary. This can be formulated as an optimization problem.

Since the output of the deep learning model is continuous, $\boldsymbol{x}$ lies in a region of points whose output z_1 is greater than $\frac{1}{2}$, and the boundary of this region is continuous. So what we really seek is a nearby point on that boundary, and we call points on the boundary *flip points*. So given $\boldsymbol{x}$ with $z_1(\boldsymbol{x}) > \frac{1}{2}$, we seek a nearby point $\hat{\boldsymbol{x}}$ with $z_1(\hat{\boldsymbol{x}}) = \frac{1}{2}$.[1]

Multi-classification case. We now formalize our optimization problem for a general model with many output classes: "For a given input $\boldsymbol{x}$, we would like to find the closest flip point to it between classes i and j."

This requires the output of the model to be equal for classes i and j and the rest of output elements to be less than those two elements.[2] The optimization problem to find such a point can be formulated as

1. One technical point: Because z_1 is continuous, there will be a point arbitrarily close to $\hat{\boldsymbol{x}}$ for which z_1 is less than $1/2$ and the classification becomes "noncancerous" *unless* $\hat{\boldsymbol{x}}$ is a local minimizer of the function z_1. In this extremely unlikely event, we will have the gradient $\bigtriangledown z_1(\hat{\boldsymbol{x}}) = \boldsymbol{0}$ and the second derivative matrix positive semidefinite, and $\hat{\boldsymbol{x}}$ will not be a boundary point. In practice, this is not likely to occur.
2. Here, we define the closest flip point between classes i and j in a strict way, where the point is only a flip point between those two classes. In general, a flip point can be a flip between multiple classes.

$$\min_{\hat{\boldsymbol{x}}} \|\hat{\boldsymbol{x}} - \boldsymbol{x}\|, \tag{1}$$

subject to:

$$\boldsymbol{z} = \mathcal{N}(\hat{\boldsymbol{x}}), \tag{2}$$

$$\boldsymbol{z}_i = \boldsymbol{z}_j, \tag{3}$$

$$\boldsymbol{z}_i > \boldsymbol{z}_k, \forall k \notin \{i, j\}, \tag{4}$$

$$\hat{\boldsymbol{x}} \text{ be within its feasible bounds.} \tag{5}$$

Equation 1 is our objective function, which minimizes the distance between the optimization variable and the input $\boldsymbol{x}$. $\|.\|$ is a norm appropriate to the data.[3] Equation 2 relates the input of the deep learning function to its output. Equation 3 ensures the output of model is equal for classes i and j. Equation 4 ensures that model output for all other classes have smaller values than those two classes. Finally, equation 5 ensures the closest flip point is within some feasible bounds, because typically the domain of $\mathcal{N}$ is a bounded space. For example, pixel values of images are defined within a range. We denote the solution to this problem as $\hat{\boldsymbol{x}}^{c}{}_{(i,j)}$.

Equations 2 through 5 define the basic constraints for a flip point between classes i and j. Other constraints can be added as well, if necessary. For example, we might want to fix certain patches in an image, or we might want to allow only certain colors in an image to change (Laidlaw and Feizi, 2019). In the case of inputs with discrete features, we can add the discrete constraints to the problem or add regularization terms to the objective function using the techniques described by Nocedal and Wright (2006). It is also possible to modify these constraints to find flip points between multiple classes, i.e., the locations where multiple decision boundaries intersect. More details about our optimization problem and its possible extensions are explained in Appendix E.

3. How flip points are computed

3.1. General approach

We define the numerical process of solving 1 through 5 to compute closest flip points by an off-the-shelf optimization solver $\mathcal{F}$:

$$\hat{\boldsymbol{x}}^{c}{}_{(i,j)} = \mathcal{F}(\boldsymbol{x}, \mathcal{N}, \mathbf{x_0}, \mathcal{C}, \mathbf{i}, \mathbf{j}). \tag{6}$$

The inputs to $\mathcal{F}$ include the trained model $\mathcal{N}$, the starting point $\mathbf{x_0}$ (perhaps chosen to be $\boldsymbol{x}$), and a set of constraints $\mathcal{C}$, including but not limited to the ones defined by equations 2 through 5. Understandably, this optimization problem is considered hard and intractable in the deep learning literature. Besides the non-convexity and high dimensionality of variables, the main issue here is the issue of vanishing and exploding gradients in deep learning models. This is a well-known phenomenon explainable by the flow of gradients through the layers of the networks. When computing the gradients for our optimization problem, this can lead to a badly scaled and uninformative

3. Shafahi et al. (2019) have investigated the effect of different norms in measuring the distance for image classifiers.

gradient matrix and eventually make our optimization problem ill-conditioned.[4] The other trouble in solving this optimization problem is satisfying its constraints. Sometimes finding any feasible solution can be challenging.

Despite these difficulties, an off-the-shelf optimization algorithm $\mathcal{F}$ might be able to efficiently find the solution for our problem. But, it is also possible for such algorithms to fail, or they might find an inferior local minimizer (too far from the input). Here, we develop a homotopy optimization algorithm for this problem which may be advantageous over other algorithms. Our homotopy starts by applying $\mathcal{F}$ to an easier model and gradually transforming it to the original model, each time using the previously obtained flip point as our starting point for $\mathcal{F}$. This way, it follows a path of feasible solutions for the intermediate models which makes it likely to find a feasible solution for our original model.

In the following section, we formulate a deep learning function, and then, we develop our homotopy method.

3.2. Our deep learning function

Consider a feed-forward neural network $\mathcal{N}$ with n_x inputs, m layers, and n_m outputs. We have weight matrices $\boldsymbol{W}^{(k)}$ and bias vectors $\boldsymbol{b}^{(k)}$ for each layer $k = 1, \ldots, m$. The output of layer k in the network is denoted by $\boldsymbol{y}^{(k)}$. The activation function used in the neurons is the error function

$$y = activation(c|\sigma) = \text{erf}(\frac{c}{\sigma}) = \frac{1}{\sqrt{\pi}} \int_{-\frac{c}{\sigma}}^{+\frac{c}{\sigma}} e^{-t^2} dt, \tag{7}$$

where c is the result of applying the weights and bias to the neuron's inputs. The tuning parameter σ is constant among the neurons on each layer and is optimized during the training process. Hence, for the whole network, we have a vector of tuning parameters, $\boldsymbol{\sigma}$, where each element of it corresponds to one hidden layer in the network. While erf is not a very common choice for activation function, it has been shown that its performance in terms of accuracy can be comparable to other activation functions (Ramachandran et al., 2018). Mobahi (2016) has also reported success in using the erf for training recurrent neural networks.

We note that when σ is small, the activation function resembles a step function, while when σ is large, it resembles a linear function, as shown in Figure 1, so erf captures the behavior of common activation functions (Shalev-Shwartz and Ben-David, 2014). Since erf is the antiderivative of the Gaussian function, both the output of neurons and their gradients will be bounded.

Defining $\boldsymbol{y}^{(0)} = \boldsymbol{x}$, the output for layers 1 through $m-1$ is

$$\boldsymbol{y}^{(i)} = \text{erf}\left(\frac{\boldsymbol{y}^{(i-1)}\boldsymbol{W}^{(i)} + \boldsymbol{b}^{(i)}}{\sigma_i}\right). \tag{8}$$

Then

$$\boldsymbol{y}^{(m)} = \boldsymbol{y}^{(m-1)}\boldsymbol{W}^{(m)} + \boldsymbol{b}^{(m)}, \tag{9}$$

4. For flip point computation, we are concerned about the gradients of outputs with respect to inputs, while in neural network literature, this issue of "vanishing and exploding gradients" usually concerns the training process and the gradient of the loss function with respect to the training parameters (Bengio et al., 1994; Hanin, 2018). In both cases, the "vanishing and exploding gradients" phenomenon can be studied by investigating individual matrices in the chain rule formulation of the gradient matrix.

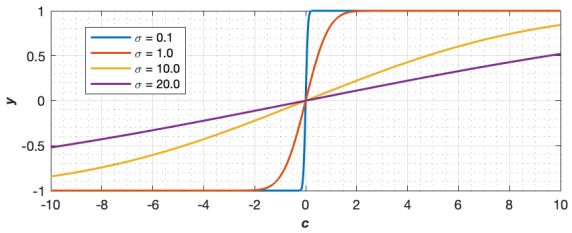

Figure 1: Shape of erf function as σ varies.

and the output of the network is

$$\boldsymbol{z} = \mathcal{N}(\boldsymbol{x}) = \text{softmax}\left(\boldsymbol{y}^{(m)}\right).^{5} \tag{10}$$

We formulate the derivatives and derive a bound on the Lipschitz constant of $\mathcal{N}$ in Appendix C.

3.3. Homotopy method

Homotopy optimization methods are usually used to solve "hard" optimization problems, by transforming the problem into an "easy" form with some desired properties, and gradually transforming the problem back to its original form, while tracing a path of solutions (Nocedal and Wright, 2006; Dunlavy and O'Leary, 2005). Such approach has to be tailored and designed in order to overcome the specific challenges of the problem at hand (Mobahi and Fisher, 2015a,b; Dunlavy et al., 2005). Examples of using homotopy for training deep learning models include the work by Mobahi (2016) for training recurrent neural networks and the work by Chen and Hao (2019) for training fully connected networks. This is quite different than our focus: solving optimization problems about the trained models.

Our homotopy method starts by transforming the original model $\mathcal{N}$ to a desirable form where:

1. The gradients of the neurons flow through the network by ensuring that they are bounded away from zero,

2. Our starting point is a feasible solution for the transformed model.

This is performed with Algorithm 1 and we denote the transformed model by $\mathcal{N}^h$.

Parameter τ defines the lower bound for the gradient of individual neurons. In Algorithm 1, line 1 computes a scalar γ such that the derivative of the erf for it is equal to τ. Lines 3 through 12 tune the $\boldsymbol{\sigma}$s, layer by layer, starting from the first layer and ending at the last hidden layer. Line 4 bounds the individual gradients between τ and 1. Choosing the $\sigma_k^h \geq \frac{2}{\sqrt{\pi}}$ ensures the gradients of neurons are upper bounded by 1. This relationship can be easily derived by setting the maximum derivative of erf less than or equal to τ.

Choosing $\sigma_k^h = \frac{1}{\gamma}\left\|\boldsymbol{y}^{(k-1)}\boldsymbol{W}^{(k)} + \boldsymbol{b}^{(k)}\right\|_\infty$ would make the gradients of all the neurons in layer k lower bounded by τ, if $\frac{1}{\gamma}\left\|\boldsymbol{y}^{(k-1)}\boldsymbol{W}^{(k)} + \boldsymbol{b}^{(k)}\right\|_\infty \leq \frac{2}{\tau\sqrt{\pi}}$. If we obtain $\sigma_k^h > \frac{2}{\tau\sqrt{\pi}}$, it implies that the variation of inputs is relatively large among the neurons on that layer. In such situations, we calculate the σ_k^h separately for each neuron on that layer (lines 5 through 10), and use a non-uniform

5. Some recent studies suggest replacing the softmax function with other functions; for example, Wang and Osher (2019) suggest using a graph Laplacian-based interpolating function.

Algorithm 1 Algorithm for homotopy transformation of network (with erf activation)

Inputs: $\mathcal{N}, \boldsymbol{x}, \tau, i, j$
Output: $\boldsymbol{\sigma}^h$ and $\boldsymbol{b}^{h(m)}$

1: $\gamma = \sqrt{log(\frac{2}{\tau\sqrt{\pi}})}$
2: $\boldsymbol{y}^{(0)} = \boldsymbol{x}$
3: **for** $k = 1$ to $m-1$ **do**
4: $\quad \sigma_k^h = \max(\frac{2}{\sqrt{\pi}}, \frac{1}{\gamma}\|\boldsymbol{y}^{(k-1)}\boldsymbol{W}^{(k)} + \boldsymbol{b}^{(k)}\|_\infty)$
5: $\quad$ **if** $\sigma_k^h > \frac{2}{\tau\sqrt{\pi}}$ **then**
6: $\quad\quad \boldsymbol{c} = \boldsymbol{y}^{(k-1)}\boldsymbol{W}^{(k)} + \boldsymbol{b}^{(k)}$
7: $\quad\quad$ **for** $t = 1$ to n_k **do**
8: $\quad\quad\quad \sigma_{k,t}^h = \max(\frac{2}{\sqrt{\pi}}, \frac{1}{\gamma}|c_t|)$
9: $\quad\quad$ **end for**
10: $\quad$ **end if**
11: $\quad \boldsymbol{y}^{(k)} = \text{erf}(\frac{\boldsymbol{y}^{(k-1)}\boldsymbol{W}^{(k)} + \boldsymbol{b}^{(k)}}{\sigma_k^h})$
12: **end for**
13: $\min_{\boldsymbol{b}^{h(m)}} \|\boldsymbol{b}^{h(m)} - \boldsymbol{b}^{\mathcal{N}(m)}\|_2$, subject to:
$\quad$ (1) $\boldsymbol{y}^{(m)} = \boldsymbol{y}^{(m-1)}\boldsymbol{W}^{(\boldsymbol{m})} + \boldsymbol{b}^{h(m)}$,
$\quad$ (2) $y_i^{(m)} = y_j^{(m)}$,
$\quad$ (3) $\forall\, l \neq i, j \mid y_i^{(m)} > y_l^{(m)}$
14: **return** $\boldsymbol{\sigma}^h, \boldsymbol{b}^{h(m)}$

σ_k^h in the homotopy algorithm for that layer. Line 11 computes the output of each layer after the σ is tuned for that layer.

The second goal of homotopy transformation is to make the starting point a feasible flip point for the transformed model. By changing $\boldsymbol{b}^{\mathcal{N}(m)}$ to $\boldsymbol{b}^{h(m)}$, computed at line 13 of Algorithm 1, the input $\boldsymbol{x}$ actually becomes a flip point for the transformed network. Having a starting point that is feasible with respect to flip point constraints considerably facilitates the optimization process and makes it more likely to follow a path of feasible solutions, ending with a feasible solution for the original model. The optimization problem on line 13 of the algorithm is a convex quadratic programming problem and can be solved by standard algorithms.

Overall, the parameters that we changed in $\mathcal{N}$ to obtain the $\mathcal{N}^h$ are $\boldsymbol{\sigma}$'s and $\boldsymbol{b}$'s. We show in Appendix D.1 that the model remains Lipschitz continuous as the result of this transformation and the change in its Lipschitz constant depends on the value of τ. In our numerical experiments, we have used different values of τ ranging between 10^{-4} and 10^{-7}.

The homotopy method gradually transforms the model $\mathcal{N}^h$ back to its original form, through a series of iterations. This is performed with Algorithm 2.

We start with $\mathcal{N}^h$ and $\boldsymbol{x}$ which is a flip point for it. At each iteration, we transform the model one step towards its original form and find the closest flip point for it, using the flip point from the previous iteration as the starting point and an off-the-shelf optimization. This way, we follow a path of flip points starting from $\boldsymbol{x}$, and ending with the closest flip point $\hat{\boldsymbol{x}}^c$ for the original model.

Algorithm 2 Homotopy algorithm to find the closest flip point to a given input between two classes

Inputs: $\mathcal{N}$, $\boldsymbol{x}$, η, τ, $\mathcal{C}$, i, j

Output: Closest flip point to $\boldsymbol{x}$

1: Compute $\boldsymbol{\sigma}^h$ and $\boldsymbol{b}^{h(m)}$ using Algorithm 1 with inputs $(\mathcal{N}, \boldsymbol{x}, \tau, i, j)$
2: $\hat{\boldsymbol{x}}^{c,0} = \boldsymbol{x}$
3: **for** $k = 1$ to η **do**
4: $\quad \boldsymbol{\sigma}^k = \boldsymbol{\sigma}^h + k(\frac{\boldsymbol{\sigma}^{\mathcal{N}} - \boldsymbol{\sigma}^h}{\eta})$
5: $\quad \boldsymbol{b}^{k(m)} = \boldsymbol{b}^{h(m)} + k(\frac{\boldsymbol{b}^{\mathcal{N}(m)} - \boldsymbol{b}^{h(m)}}{\eta})$
6: $\quad$ Replace $\boldsymbol{\sigma}^k$ and $\boldsymbol{b}^{k(m)}$ in $\mathcal{N}$, to obtain $\mathcal{N}^k$
7: $\quad \hat{\boldsymbol{x}}^{c,k} = \mathcal{F}(\boldsymbol{x}, \mathcal{N}^k, \hat{\boldsymbol{x}}^{c,k-1}, \mathcal{C}, i, j)$
8: **end for**
9: **return** $\hat{\boldsymbol{x}}^{c,\eta}$ as the closest flip point to $\boldsymbol{x}$

In Appendix D, we analyze the homotopy path and investigate the effect of step size on the model and its output. We show that the change in the model output and the Lipschitz constant of the model at each homotopy step is governed by the size of the step.

Therefore, if we choose the step size small enough, we can ensure that the change in model output is small and the solution of each step is close enough to the solution from the previous step. This makes it likely for the homotopy method to find a feasible solution for the original model, under some basic conditions that we discuss in Appendix D.4.

The parameter η defines the number of iterations that Algorithm 2 uses to transform the network back to its original form, assuming that step sizes are uniform. In our numerical experiments, we have achieved best results with η ranging between 1 and 15. Choosing $\eta = 1$ is equivalent to not using the homotopy algorithm and directly applying $\mathcal{F}$ to the original network with starting point $\boldsymbol{x}$. It is also possible to use adaptive steps which we discuss in Appendix D.5.

More details about the algorithm are provided by Yousefzadeh (2019, pp.25-29). Extension of our homotopy algorithms for models with ReLU activation is presented in Appendix F.

4. How flip points provide valuable information to the user

We now explain how flip points can be used to improve the performance and interpretation of neural networks.

Determine the least change in $\boldsymbol{x}$ that alters the classification of the model. The vector $\hat{\boldsymbol{x}}^c - \boldsymbol{x}$ is a clear explanation of the minimum change in the input that can make the outcome different. This is insightful information that can be provided along with the output. For an image, this would reveal the least changes in the pixels that can change the classification.

Assess the trustworthiness of the classification for $\boldsymbol{x}$. In our numerical examples we show that the numerical value of the output of an image classification model, when the last layer is defined by the $\mathrm{softmax}$ function, does not indicate how sure we should be of the correctness of the output. In fact, many mis-classifications correspond to very high $\mathrm{softmax}$ values. This has been previously observed by Nguyen et al. (2015) and Guo et al. (2017). The reason behind it is related to the loss function used for model training which is usually cross entropy. Cross entropy incurs a loss when a training point is *very close* to a decision boundary. Elsayed et al. (2018) and Jiang et al. (2019) suggested adding a penalty term to the training loss to ensure the decision boundaries maintain a larger

margin to training points. Still, the overall distribution of distance of training points to decision boundaries can have a large variation (as we have observed in several models) and the minimum distance maintained by the training loss function can be considered a small distance compared to that distribution.

We observe this in our numerical experiments. The distances of incorrectly classified points to their flip points tend to be small compared to the distances for correct classifications, implying that closeness to a flip point is indicative of how sure we can be of the correctness of a classification. Still, the relatively small margin to decision boundaries can be large enough for the points to have a large $\mathrm{softmax}$ score. Therefore, the actual distance to decision boundary can be more informative than the $\mathrm{softmax}$ score.

Gal and Ghahramani (2016) propose using information from training using dropout to assess the uncertainty of classifications. Their method is restricted to this particular training method, does not provide the likely correct classification, and is more expensive than the method we propose. Another approach, proposed by Guo et al. (2017) constructs a calibration model, trained separately on a validation set, and appends it as a post-processing component to the network. Also, Lakshminarayanan et al. (2017) used ensembles of neural networks, trained adversarially with pre-calculated scoring rules, in order to estimate the uncertainty in classifications.

Using flip points can be viewed as a direct method to assess the trustworthiness of classifications, even when models are calibrated or trained adversarially. Therefore, flip point assessment is not necessarily in competition with other methods in the literature.

Identify ambiguity in the classification of $\boldsymbol{x}$ due to measurement uncertainties. Often, some of the inputs to a neural network are measured quantities which have associated uncertainties. When the difference between $\boldsymbol{x}$ and its closest flip point is less than the uncertainty in the measurements, then the classification made by the model is quite possibly incorrect, and this information should be communicated to the user.

Gain insight into important feature combinations using PCA analysis of the flip points. The direction from a single data point to the closest flip point provides sensitivity information. Using PCA analysis, we can extend this insight to an entire dataset or to subsets within a dataset,

We form a matrix with one row $\hat{\boldsymbol{x}}^c - \boldsymbol{x}$ for each data point. PCA analysis of this matrix identifies the most influential directions for flipping the outputs in the dataset and thus the most influential features. This procedure provides clear and accurate interpretations of the neural network model for groups of data points. One can use nonlinear PCA or auto-encoders to enhance this approach. Alternatively, for a given data point, PCA analysis of the directions from the data point to a collection of boundary points can give insight about the shape of the decision boundary.

5. How flip points can improve the training and security of the model

Flip points may provide useful information about the models and their training.

Estimate the influence of individual training points in shaping the model. It is shown previously that some training points are more influential in shaping the models (Katharopoulos and Fleuret, 2018) and there are redundancies in some training sets (Birodkar et al., 2019). Training points that are geometrically dominated with respect to decision boundaries by other training points of same class are expected to have less influence on shaping the decision boundaries. In our numerical experiment on MNIST, we see that points that are close to their flip points are more influential on the testing accuracy of model.

Koh and Liang (2017) and Koh et al. (2019) use influence functions to relate individual classifications of a trained model to training points that are most influential for that classification. Although this approach provides insightful, its reliance on small perturbations of training data and local gradient information of the loss function is a limitation.

Generate synthetic data to improve accuracy and to shape the decision boundaries. We can add flip points to the training set as *synthetic data* to move the output boundaries of a neural network insightfully and effectively. Suppose that our trained neural network correctly classifies a training point $\boldsymbol{x}$ but that there is a nearby flip point $\hat{\boldsymbol{x}}^c$. We generate a synthetic data point by adding $\hat{\boldsymbol{x}}^c$ to the training set, using the same classification as that for $\boldsymbol{x}$. Retraining the network will then tend to push the classification boundary further away from $\boldsymbol{x}$.

Using flip points to alter the decision boundaries can be performed not just to improve the accuracy of a model but also to change certain traits adopted by the trained network. For example, if a model is biased for or against certain features of the inputs, one can alter that bias using synthetic data. There are studies in the literature that have used synthetic data (but not flip points) to improve the accuracy, e.g., Jaderberg et al. (2014). There is also a line of research that has used perturbations of the inputs in order to make the trained models robust, e.g., Tsipras et al. (2019). However, using *flip points* as *synthetic data* is novel and can potentially benefit the studies on robustness of networks, too.

Understand adversarial influence. Flip points also provide insight for anyone with adversarial intentions. First, these points can be used to understand and exploit possible flaws in a trained model. Second, adding flip points with incorrect labels to the training data will effectively distort the class boundaries in the trained model and can diminish its accuracy or bias its results (Shafahi et al., 2018; Zhu et al., 2019; Gupta et al., 2019). Our methods may be helpful in studying adversarial attacks such as the problems studied by Schmidt et al. (2018), Sinha et al. (2018), Madry et al. (2018), Katz et al. (2017), and Ilyas et al. (2019).

6. Results

In our numerical results, we use feed-forward neural networks with 12 layers and $\mathrm{softmax}$ on the output layer.[6] When calculating flip points, we measure the distance in equation (1) using the 2-norm. Calculating the closest flip points is quite fast, under 1 second for the MNIST and a few seconds for the CIFAR-10 datasets, using a 2017 MacBook. More information about our models are provided in Appendix A.

6.1. MNIST

The MNIST dataset (LeCun et al., 1998) has 10 output classes, the digits 0 through 9. We could use pixel data as input to the models, but, for efficiency, we choose to represent each image using the Haar wavelet basis. The 100 most significant wavelets are chosen by pivoted QR decomposition (Golub and Van Loan, 2012) of the matrix formed from the wavelet coefficients of all images in the training set. The wavelet transformation applies convolutions of various widths to the input data and leads to significant compression of the input data, from 784 features to 100, allowing us to use smaller networks. This idea, independent of flip points, is valuable whenever working with image

6. Keep in mind that all classification models with continuous output have decision boundaries, regardless of the architecture of the model (number of layers, activation function, etc.), the training set, and the training regime (regularization, etc.).

data, because it removes the redundancies in the input space. Using pixel input instead of wavelet coefficients would yield interpretation traits similar to those that we present here on these datasets.

We train two networks, NET1 and NET2, using half of the training data (30,000 images) for each. Table 1 shows the accuracy of each network in the 2-fold cross validation. Accuracy could be improved using more wavelet coefficients, but these networks are adequate for our purposes.

Table 1: Classification accuracies for NET1 and NET2 for MNIST.

TRAINED NETWORK	ACCURACY ON 1ST HALF OF TRAINING SET	ACCURACY ON 2ND HALF OF TRAINING SET	ACCURACY ON TESTING SET
NET1	100%	97.62%	97.98%
NET2	97.56%	100%	97.64%

For each of the images in the training set, we calculate the flip points between the class predicted by the trained neural networks and each of the other 9 classes.

Flip points identify alternate classifications. Some images are misclassified and close to at least one flip point. For all of these points, the correct label is identified by the closest of the 9 flip points (or one of those tied for closest after rounding to 4 decimal digits). For example, the image shown in Figure 2, from the second half of the MNIST training set, is an "8" mistakenly classified as "3" by NET1 with softmax score of 98%. Its distances to the closest flip points are shown in Table 2. Assuming that we do not know the correct label for this image, we would report the label as "3", with the additional explanation that there is low trust in this classification (because of closeness to the flip point), and the correct label might be "8".

Figure 2: MNIST image mistakenly classified as "3" by NET1.

Table 2: Distance to closest flip points between class "8" and other classes, for image in Figure 2.

CLASS	0	1	2	4	5	6	7	8	9
DISTANCE	1.27	1.32	0.58	2.16	0.56	1.45	1.51	**0.16**	0.90

Flip points provide better measure of trust than softmax. Many practitioners use the softmax output as a measure of confidence in the correctness of the output. As illustrated in Figure 3, the softmax scores range between 31% and 100% for the mistakes by NET1 and NET2, and range between 37% and 100% for correct classifications, providing no separation between the groups. If softmax were a good proxy for distance, then the data would lie close to a straight line. Instead, most of the mistakes have small distance but large softmax score: more than 73% of the mistakes have 80% or more softmax score. Hence, softmax cannot identify mistakes. Fortunately, the fig-

ure shows that the distance to the closest flip point is a much more reliable indicator of mistakes: mistakes almost always correspond to small distances.

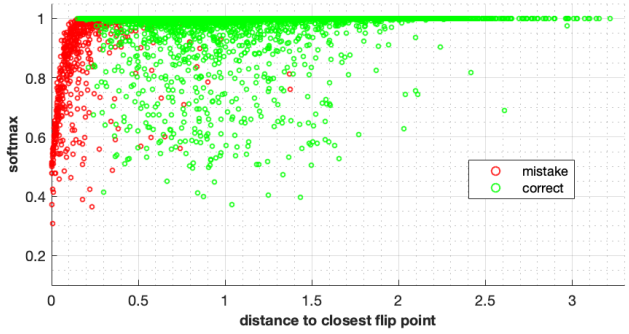

Figure 3: For the MNIST data, a large $\mathrm{softmax}$ score says nothing about the reliability of the classification. In contrast, distance to the closest flip point is a much more reliable indicator.

Flip points identify influential training points. Images that are correctly classified but are relatively close to a flip point are the most influential ones in the training process. To verify this, consider the first half of the MNIST training set, and order the images by their distances to their nearest $\hat{\boldsymbol{x}}^c$ for NET1. We then consider using neural networks trained using a subset of this data.

Data points at most 0.75 from a flip point form a subset of $9,463$ images, about 15% of the training set. A model trained on this subset achieves 97.9% accuracy on the testing set. Training with a subset of $9,463$ images randomly chosen from the training set on average (50 trials) achieves 96.2% accuracy on the testing set. A subset of same size from the images farthest from their flip points achieves only 90.6% accuracy on the testing set. These trends hold for all distance thresholds (Figure 4). This confirms that distance to the flip point is in fact related to influence in the training process.

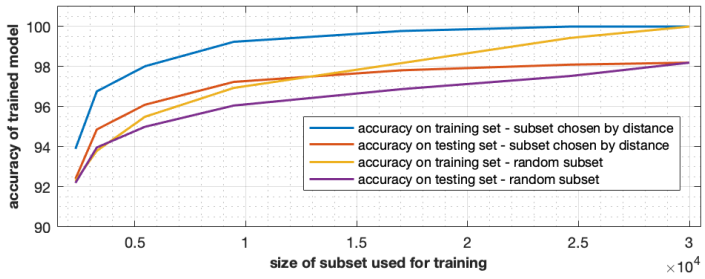

Figure 4: Accuracy of models trained on MNIST subsets.

We note that the model learns the entire training set with 100% accuracy when trained on about 16,000 images chosen by the distance measure. In contrast, it only achieves 98.8% accuracy when trained on a randomly chosen subset of the same size. Also note that flip points are computed by solving a non-convex optimization problem, so we cannot guarantee that we have indeed found the *closest* flip point. Nevertheless, the computation seems to provide very *useful* flip points, validated by the small distances achieved by some flip points and by the results shown in Figures 3 – 4.

Flip points improve the training of the network. We append to the entire training set a flip point for each mistake of NET1 and NET2 in the training set, labeled with the correct label for the

mistake. The resulting model achieves 100% accuracy on the appended training set and 98.6% on the testing set, an improvement over the 98.2% accuracy of the original network. This technique of appending synthetic images to the training set may be even more helpful for datasets with limited training data.

6.2. CIFAR-10

We now consider two classes of "airplanes" and "ships" in the CIFAR-10 dataset (Krizhevsky, 2009). We flip all the training images along their vertical axis and append their flipped version to the original training set. This time, we perform 2D wavelet decomposition on images using the Daubechies-1 wavelet basis and choose 2,500 of the wavelet coefficients, using pivoted QR factorization. Training the resulting model leads to 85.75% testing accuracy. We then calculate the flip points for all the images in both sets. Observations that we reported for MNIST generally apply here, too. So we focus our discussion on the directions to flip points and PCA analysis of them.

Figure 5 shows an image in the testing set that is mistakenly classified as an airplane, along with its closest flip point. We have computed the closest flip point in the wavelet space. It is interesting that the 1-norm distance between the image and its closest flip point in the pixel space is 242, and the differences are hard to detect by eye. Note that pixel values can range between 0 and 255.

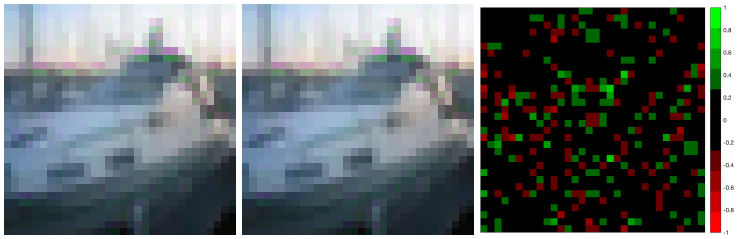

Figure 5: A ship image misclassified as airplane (left), its flip point (middle), and their difference (right).

In the pixel space, the matrix of directions between the misclassified images and their closest flip points has many columns that are uniformly close to zero. This accounts for nearly one third of the pixels. Therefore, we can investigate the mistakes by looking at a subset of 3,072 pixels. Figure 6 shows the average values of change that flip misclassified images to their correct class. We can see some patterns, specifically in the colors and regions of image.

The matrix of directions that flip misclassified images to their correct class (in the pixel space) has about 58% sparsity, if we disregard pixel changes of less than 0.5 (in the scale of 0 to 255). The first three principal components of these directions have the pattern shown in Figure 7. These patterns clearly reveal which regions of images are more capable to correct the classification of misclassified images. Note that usually changes with largest positive values are next to changes with largest negative value, signifying that these directions correspond to patterns in images, not isolated pixels.

Finally, we note that great similarity exists between the directions for the correct classifications in the training and testing sets. Investigating other principal components can provide additional insights.

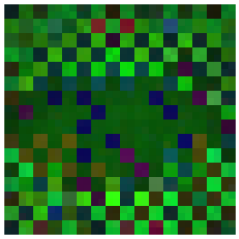

Figure 6: Average change in pixel values for flipping misclassified images to their correct class. Some pixels are not involved and some pixels are involved only with a specific color. This is an RGB image where pixel values are magnified with a 25 factor.

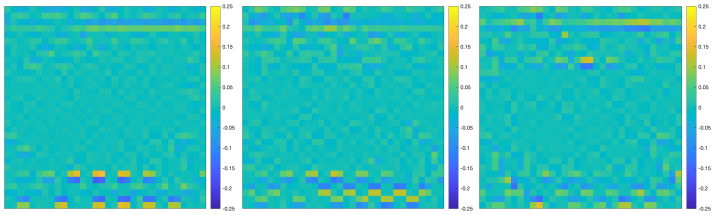

Figure 7: First three (left to right) principal components of directions that flip misclassified airplanes to their correct class.

7. Discussions

7.1. Comparison with alternative methods.

Investigating the decision boundaries of a trained neural network using flip points, and especially, efficiently computing and interpreting the closest flip point to an input, are new contributions of this paper.

Many studies consider the optimization problem of finding closest flip points intractable. Some methods take small steps from an input until they cross a decision boundary, but do not aim to find the closest boundary point (Moosavi-Dezfooli et al., 2016). Other methods rely on random perturbation of the inputs which makes them prohibitively expensive and unlikely to find the closest boundary point. For example, Ribeiro et al. (2016) generates random perturbations of the image that would produce labels on both sides of the decision boundary. Ribeiro et al. (2018) reports that finding a sensible amount of random perturbation is challenging. Finding points on the decision boundaries of a network is a nonlinear optimization problem, and solving it by random perturbations can be inefficient and unreliable.

7.2. A note on computational cost

If we compute the closest flip point using an optimization algorithm that requires derivatives, then each iteration of the algorithm requires the derivative of the output of the model with respect to the input. For any given neural network, the cost of computing this derivative is slightly less than computing the derivative of the loss function of the network for *a single input*, with respect to the

weight parameters of the first layer. In the training of a network, we have to store the derivatives of the loss function, not only for the weight parameters of the first layer, but also for all the other layers. This computation is repeated for each of the training points for each epoch of training. Thus, the derivative computation for finding the closest flip point for a set of inputs is less expensive than the derivative computation for a comparable number of training datapoints.

The number of iterations it takes to find the closest flip point would vary, depending mainly on the location of input and its distance from the closest decision boundary, the output surface of the network in that vicinity, and the optimization method. However, all those factors are present in the training of the network, as well, in a much more complex fashion, especially because the number of training parameters of a network is often orders of magnitude larger than the number of input features. This gives a clear idea how inexpensive computation of the closest flip point is, compared to the training process of the same network. For example, computing the closest flip point for each image of our model trained on MNIST takes about a second on a MacBook.

8. Conclusion

We studied the problem of model interpretation for deep learning models. Specifically, for classification models, we considered their decision boundaries and defined the concept of flip points, points on those boundaries. We posed an optimization problem to systematically study the decision boundaries. Since this problem is considered intractable to solve, we designed a homotopy algorithm which tries to overcome the issue of vanishing and exploding gradients by transforming the network and following a path of feasible solutions in order to find the feasible flip point for the original model.

We showed that the closest flip point to a given input can provide accurate explanations about changes in the input that can flip the output from one class to another. The distance of an input to the closest flip point proved to be an effective measure of trust we can have in the correctness of the output. Most importantly, analyzing the directions of data points to the decision boundaries of the models provided useful information about their patterns of behavior as mathematical functions. Analyzing the patterns in those directions can identify the influential features in the inputs for each output class, and also reveal the patterns that models have learned to distinguish each class from others. Adding flip points as synthetic data boosted the accuracy of a model, but we noted that it can also be used adversarially.

The concepts used in designing our homotopy algorithm can be used to solve other similar optimization problems incorporating deep learning functions in their objective and/or constraints.

Acknowledgments

The authors thank Thomas Goldstein and Furong Huang for helpful comments and discussions. The authors also thank anonymous reviewers. R.Y. thanks Hossein Mobahi for his helpful comments.

References

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

Vighnesh Birodkar, Hossein Mobahi, and Samy Bengio. Semantic redundancies in image-classification datasets: The 10% you don't need. *arXiv preprint arXiv:1901.11409*, 2019.

Wieland Brendel, Jonas Rauber, and Matthias Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. In *International Conference on Learning Representations*, 2018.

Helmut Blcskei, Philipp Grohs, Gitta Kutyniok, and Philipp Petersen. Optimal approximation with sparsely connected deep neural networks. *SIAM Journal on Mathematics of Data Science*, 1(1): 8–45, 2019.

Chaofan Chen, Oscar Li, Daniel Tao, Alina Barnett, Cynthia Rudin, and Jonathan K Su. This looks like that: Deep learning for interpretable image recognition. In *Advances in Neural Information Processing Systems*, pages 8928–8939, 2019.

Qipin Chen and Wenrui Hao. A homotopy training algorithm for fully connected neural networks. *Proceedings of the Royal Society A*, 475(2231):20190662, 2019.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (ELUs). *arXiv preprint arXiv:1511.07289*, 2015.

Daniel M Dunlavy and Dianne P O'Leary. Homotopy optimization methods for global optimization. Technical report, Sandia National Laboratories, 2005.

Daniel M Dunlavy, Dianne P O'Leary, Dmitri Klimov, and Devarajan Thirumalai. HOPE: A homotopy optimization method for protein structure prediction. *Journal of Computational Biology*, 12 (10):1275–1288, 2005.

Gamaleldin Elsayed, Dilip Krishnan, Hossein Mobahi, Kevin Regan, and Samy Bengio. Large margin deep networks for classification. In *Advances in Neural Information Processing Systems*, pages 842–852, 2018.

Alhussein Fawzi, Seyed-Mohsen Moosavi-Dezfooli, and Pascal Frossard. The robustness of deep networks: A geometrical perspective. *IEEE Signal Processing Magazine*, 34(6):50–62, 2017.

Alhussein Fawzi, Seyed-Mohsen Moosavi-Dezfooli, Pascal Frossard, and Stefano Soatto. Empirical study of the topology and geometry of deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3762–3770, 2018.

Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *International Conference on Machine Learning*, pages 1050–1059, 2016.

Amirata Ghorbani, Abubakar Abid, and James Zou. Interpretation of neural networks is fragile. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3681–3688, 2019.

Gene H Golub and Charles F Van Loan. *Matrix Computations*. JHU Press, Baltimore, 4th edition, 2012.

Yash Goyal, Ziyan Wu, Jan Ernst, Dhruv Batra, Devi Parikh, and Stefan Lee. Counterfactual visual explanations. In *International Conference on Machine Learning*, pages 2376–2384, 2019.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International Conference on Machine Learning*, pages 1321–1330, 2017.

Neal Gupta, W Ronny Huang, Liam Fowl, Chen Zhu, Soheil Feizi, Tom Goldstein, and John P Dickerson. Strong baseline defenses against clean-label poisoning attacks. *arXiv preprint arXiv:1909.13374*, 2019.

Boris Hanin. Which neural net architectures give rise to exploding and vanishing gradients? In *Advances in Neural Information Processing Systems*, pages 580–589, 2018.

Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. Adversarial examples are not bugs, they are features. In *Advances in Neural Information Processing Systems*, pages 125–136, 2019.

Max Jaderberg, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Synthetic data and artificial neural networks for natural scene text recognition. In *NeurIPS Deep Learning Workshop*, 2014.

Saumya Jetley, Nicholas Lord, and Philip Torr. With friends like these, who needs adversaries? In *Advances in Neural Information Processing Systems*, pages 10749–10759, 2018.

Yiding Jiang, Dilip Krishnan, Hossein Mobahi, and Samy Bengio. Predicting the generalization gap in deep networks with margin distributions. In *International Conference on Learning Representations*, 2019.

Angelos Katharopoulos and Francois Fleuret. Not all samples are created equal: Deep learning with importance sampling. In *International Conference on Machine Learning*, pages 2525–2534, 2018.

Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Towards proving the adversarial robustness of deep neural networks. In *1st Workshop on Formal Verification of Autonomous Vehicles*, pages 19–26, 2017.

Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *International Conference on Machine Learning*, pages 1885–1894, 2017.

Pang Wei W Koh, Kai-Siang Ang, Hubert Teo, and Percy S Liang. On the accuracy of influence functions for measuring group effects. In *Advances in Neural Information Processing Systems*, pages 5255–5265, 2019.

Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

Cassidy Laidlaw and Soheil Feizi. Functional adversarial attacks. In *Advances in Neural Information Processing Systems*, pages 10408–10418, 2019.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems*, pages 6402–6413, 2017.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018.

Hossein Mobahi. Training recurrent neural networks by diffusion. *arXiv preprint arXiv:1601.04114*, 2016.

Hossein Mobahi and John W Fisher. On the link between Gaussian homotopy continuation and convex envelopes. In *International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition*, pages 43–56. Springer, 2015a.

Hossein Mobahi and John W Fisher. A theoretical analysis of optimization by Gaussian continuation. In *AAAI Conference on Artificial Intelligence*, 2015b.

Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: A simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2574–2582, 2016.

Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 427–436, 2015.

Jorge Nocedal and Stephen Wright. *Numerical Optimization*. Springer, New York, 2nd edition, 2006.

Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2018.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should I trust you?: Explaining the predictions of any classifier. In *International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144. ACM, 2016.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-precision model-agnostic explanations. In *AAAI Conference on Artificial Intelligence*, pages 1527–1535, 2018.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive Modeling*, 5(3):1, 1988.

Ludwig Schmidt, Shibani Santurkar, Dimitris Tsipras, Kunal Talwar, and Aleksander Madry. Adversarially robust generalization requires more data. In *Advances in Neural Information Processing Systems*, pages 5019–5031, 2018.

Ali Shafahi, W Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. Poison frogs! Targeted clean-label poisoning attacks on neural networks. In *Advances in Neural Information Processing Systems*, pages 6103–6113, 2018.

Ali Shafahi, W Ronny Huang, Christoph Studer, Soheil Feizi, and Tom Goldstein. Are adversarial examples inevitable? In *International Conference on Learning Representations*, 2019.

Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge University Press, 2014.

Adi Shamir, Itay Safran, Eyal Ronen, and Orr Dunkelman. A simple explanation for the existence of adversarial examples with small Hamming distance. *arXiv preprint arXiv:1901.10861*, 2019.

Sahil Singla, Eric Wallace, Shi Feng, and Soheil Feizi. Understanding impacts of high-order loss approximations and features in deep learning interpretation. In *International Conference on Machine Learning*, pages 5848–5856, 2019.

Aman Sinha, Hongseok Namkoong, and John Duchi. Certifiable distributional robustness with principled adversarial training. In *International Conference on Learning Representations*, 2018.

Gilbert Strang. *Linear Algebra and Learning from Data*. Wellesley-Cambridge Press, 2019.

Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy. In *International Conference on Learning Representations*, 2019.

Bao Wang and Stanley J Osher. Graph interpolating activation improves both natural and robust accuracies in data-efficient deep learning. *arXiv preprint arXiv:1907.06800*, 2019.

Roozbeh Yousefzadeh. *Interpreting Machine Learning Models and Application of Homotopy Methods*. PhD thesis, University of Maryland, College Park, 2019.

Roozbeh Yousefzadeh and Dianne P O'Leary. Investigating decision boundaries of trained neural networks. *arXiv preprint arXiv:1908.02802*, 2019.

Chen Zhu, W Ronny Huang, Hengduo Li, Gavin Taylor, Christoph Studer, and Tom Goldstein. Transferable clean-label poisoning attacks on deep neural nets. In *International Conference on Machine Learning*, pages 7614–7623, 2019.

Appendix A. Information about models used in our numerical examples

Here, we provide more information about the models we have trained and used in Section 6. We have used fully connected feed-forward neural networks with 12 hidden layers. The number of neurons for the models used for each data set is shown in Table A1. The activation function we have used in the neurons is the tunable error function explained in Section 3.3. We also used $\mathrm{softmax}$ on the output layer, and cross entropy for the training loss function. We used Tensorflow for training the networks, with Adam optimizer and learning rate of 0.001.

Table A1: Number of nodes in neural network used for each data set.

DATA SET	MNIST	CIFAR-10
INPUT LAYER	100	2,500
LAYER 1	500	1,000
LAYER 2	500	500
LAYER 3	500	455
LAYER 4	400	410
LAYER 5	300	365
LAYER 6	250	320
LAYER 7	250	275
LAYER 8	250	230
LAYER 9	200	185
LAYER 10	150	140
LAYER 11	150	95
LAYER 12	100	50
OUTPUT LAYER	10	2

Appendix B. Code

Our code will be available at https://github.com/roozbeh-yz/flip_homotopy_erf.

Appendix C. Mathematical properties of model and analysis of the homotopy method

Here, we first derive the derivatives of the output of our model discussed in Section 3.2 and investigate its Lipschitz continuity. We then study how the properties of model changes as we perform the homotopy transformation.

C.1. Derivatives of the output with respect to input

The derivative of the outputs of the network with respect to its inputs is a Jacobian matrix, $\mathrm{Jac}(\boldsymbol{z}, \boldsymbol{x})$, when the network has more than one input feature and more than one output class. The computation of the derivatives is analogous to the back-propagation approach commonly used to compute the gradients with respect to the training parameters of the networks (Rumelhart et al., 1988). Hence, we calculate the derivatives, layer by layer, and use the chain rule. We first take the derivative of the output of the first layer with respect to the inputs:

$$\mathrm{Jac}\left(\boldsymbol{y}^{(1)}, \boldsymbol{x}\right)_{n_1,n_x} = \frac{2}{\sigma[1]\sqrt{\pi}}\left(\mathbb{1}_{n_x,1}\; e^{-\left(\frac{\boldsymbol{x}\,\boldsymbol{W}^{(1)}+\boldsymbol{b}^{(1)}}{\sigma[1]}\right)^2_{1,n_1}}\right)^T \odot {\boldsymbol{W}^{(1)}_{n_x,n_1}}^T. \tag{11}$$

In general, we can write the Jacobian for the output of any hidden layer in terms of the Jacobian for the layer above it. This recursive relation is expressed by

$$\mathrm{Jac}\left(\boldsymbol{y}^{(i)}, \boldsymbol{x}\right)_{n_i,n_x} = \frac{2}{\sigma[i]\sqrt{\pi}}\left(\mathbb{1}_{n_x,1}\; e^{-\left(\frac{\boldsymbol{y}^{(i-1)}\,\boldsymbol{W}^{(i)}+\boldsymbol{b}^{(i)}}{\sigma[i]}\right)^2_{1,n_i}}\right)^T \odot \left({\boldsymbol{W}^{(i)}_{n_{i-1},n_i}}^T\; \mathrm{Jac}\left(\boldsymbol{y}^{(i-1)}, \boldsymbol{x}\right)_{n_{i-1},n_x}\right). \tag{12}$$

We continue this process until we reach the last output layer, which has a softmax function instead of the activation function. We use the chain rule to obtain the Jacobian as:

$$\mathrm{Jac}(\boldsymbol{z}, \boldsymbol{x})_{n_m,n_x} = \left(\left(\boldsymbol{z}^T_{1,n_m}\; \mathbb{1}_{1,n_m}\right) \odot \left(\boldsymbol{\delta}_{n_m,n_m} - \mathbb{1}_{n_m,1}\; \boldsymbol{z}_{1,n_m}\right)\right)\left({\boldsymbol{W}^{(m)}_{n_{m-1},n_m}}^T\; \mathrm{Jac}\left(\boldsymbol{y}^{(m-1)}, \boldsymbol{x}\right)_{n_{m-1},n_x}\right). \tag{13}$$

C.2. Lipschitz continuity of the deep learning function

To evaluate the Lipschitz constant of the model, we derive an upper bound on the norm of the derivatives of the output of the network. For that, we write the chain rule decomposition of equation (13) and compute the maximum norm of each matrix in the decomposition. The derivative will be bounded by the product of the norms. Consider equation (11) as the first step. We have

$$\begin{aligned} \|\,\mathrm{Jac}\left(\boldsymbol{y}^{(1)}, \boldsymbol{x}\right)\|_\infty &\leq \frac{2}{\sigma[1]\sqrt{\pi}}\,\|\left(\mathbb{1}_{n_x,1}\; e^{-\left(\frac{\boldsymbol{x}\,\boldsymbol{W}^{(1)}+\boldsymbol{b}^{(1)}}{\sigma[1]}\right)^2_{1,n_1}}\right) \odot \boldsymbol{W}^{(1)}\|_\infty \\ &\leq \frac{2}{\sigma[1]\sqrt{\pi}}\,\|\boldsymbol{W}^{(1)}\|_\infty. \end{aligned} \tag{14}$$

because $0 \leq \mathbb{1}_{n_x,1}\; e^{-\left(\frac{\boldsymbol{x}\,\boldsymbol{W}^{(1)}+\boldsymbol{b}^{(1)}}{\sigma[1]}\right)^2_{1,n_1}} \leq 1$.

Continuing this process down the layers of the network, using equations (12) and (13), we can easily bound the derivative of output. We note that for each hidden layer in the network,

$$\mathbb{1}_{n_x,1}\; e^{-\left(\frac{\boldsymbol{y}^{(i-1)}\,\boldsymbol{W}^{(i)}+\boldsymbol{b}^{(i)}}{\sigma[i]}\right)^2_{1,n_i}}$$

is bounded between 0 and 1, and can be dropped.

We obtain the upper bound on the Lipschitz constant as

$$\|\,\mathrm{Jac}\left(\boldsymbol{z}, \boldsymbol{x}\right)\|_\infty \leq \frac{2^{m-1}}{\pi^{(m-1)/2}\prod_{i=1}^{m-1}\sigma[i]}\prod_{i=1}^{m}\|\boldsymbol{W}^{(i)}\|_\infty. \tag{15}$$

Appendix D. Analysis of homotopy path

Here, we analyze how the properties of model change during the homotopy transformation.

D.1. Lipschitz continuity after homotopy transformation

Our Algorithm 2 ensures $\forall i, \frac{2}{\sqrt{\pi}} \leq \sigma^h[i] \leq \frac{2}{\tau\sqrt{\pi}}$, where $0 < \tau \ll 1$ because it is a lower bound on gradients of neurons. Therefore for the model transformed with homotopy, $\mathcal{N}^h$, we can bound the right hand side of equation 15 as

$$\tau^{m-1} \prod_{i=1}^{m} \|\boldsymbol{W}^{(i)}\|_\infty \leq \frac{2^{m-1}}{\pi^{(m-1)/2} \prod_{i=1}^{m-1} \sigma^h[i]} \prod_{i=1}^{m} \|\boldsymbol{W}^{(i)}\|_\infty \leq \prod_{i=1}^{m} \|\boldsymbol{W}^{(i)}\|_\infty. \tag{16}$$

Equation 16 shows that the Lipschitz constant of the model after homotopy transformation, κ^h, can not be drastically different than the Lipschitz constant of the original model, κ.

We note that the bounds on the Lipschitz continuity of model has an inverse linear relationship with σ. In Algorithm 2, we make a linear transformation between σ^h and σ. Therefore, choosing the η large enough would ensure the Lipschitz continuity of the model has a small change at each iteration of the homotopy algorithm.

D.2. Effect of b

Lipschitz constant of the model is not affected by change in $\boldsymbol{b}$ as shown in equations 15 and 16, but model output changes.

Let's consider the $\boldsymbol{b}^{k(m)}$ at some iteration k of the homotopy algorithm and let's denote $\boldsymbol{\delta b} = \boldsymbol{b}^{k+1(m)} - \boldsymbol{b}^{k(m)}$. Given an input $\boldsymbol{x}$, the output of the model at iteration k will be $\boldsymbol{z}^k = \mathcal{N}^k(\boldsymbol{x})$. If we replace $b^{k(m)}$ with $b^{k+1(m)}$ to obtain $\mathcal{N}^{k+1}$, we have

$$\boldsymbol{z}^{k+1} = \mathcal{N}^{k+1}(\boldsymbol{x}) = e^{\boldsymbol{y}^{k+1(m)}} / \Sigma e^{\boldsymbol{y}^{k+1(m)}} = e^{\boldsymbol{y}^{k(m)} + \delta b} / \Sigma e^{\boldsymbol{y}^{k(m)} + \boldsymbol{\delta b}}$$

Therefore for an output class i, we have

$$z^{k+1}[i] = e^{y^{k(m)}[i]} \frac{e^{\delta b[i]}}{e^{\boldsymbol{y}^{k(m)}} e^{\boldsymbol{\delta b}^T}}$$

and

$$z^{k}[i] = e^{y^{k(m)}[i]} \frac{1}{e^{\boldsymbol{y}^{k(m)}} \mathbb{1}},$$

where $\mathbb{1}$ is a vector of ones.

We note that the exponential function is strictly positive. Therefore, it is easy to see that the Jacobian matrix of $\boldsymbol{z}^{k+1}$ w.r.t. $\boldsymbol{\delta b}$ is a symmetric square matrix with non-negative diagonals and non-positive off-diagonals.

The ratio of change from $z^k[i]$ to $z^{k+1}[i]$ is governed by $\boldsymbol{\delta b}$. On the other hand, in Algorithm 2, $\boldsymbol{\delta b} = k(\frac{\boldsymbol{b}^{\mathcal{N}(m)} - \boldsymbol{b}^{h(m)}}{\eta})$. Therefore, using η, we can control the δb and conseuently, the change from $\boldsymbol{z}^k$ to $\boldsymbol{z}^{k+1}$. Later, we will explain the possibility of using an adaptive approach to move along the homotopy path.

D.3. Effect of σ on model output

Changing the σ affects the Lipschitz continuity of the model, as we discussed earlier. Regarding its effect on the output of the model, we have to consider how the output of neurons change as we modify the σ.

To understand this effect, we write the Taylor expansion of erf activation

$$\mathrm{erf}(\frac{x}{\sigma}) = \frac{2x}{\sqrt{\pi}\sigma} - \frac{2x^3}{3\sqrt{\pi}\sigma^3} + \mathcal{O}(\frac{x^5}{\sigma^5})$$

Also, the derivative of erf activation w.r.t. σ is

$$\frac{d}{d\sigma}\,\mathrm{erf}(\frac{x}{\sigma}) = -\frac{2xe^{x^2/\sigma^2}}{\sqrt{\pi}\sigma^2}$$

Clearly, if at each iteration of homotopy transformation, we change the σ in small enough increments, the output of each neuron will not have drastic changes. Because the output of each neuron is bounded between 0 and 1, and this recursion repeats at all layers, the overall changes in the output of the model can be adequately small.

Using η, we can control how much σ changes at each iteration of homotopy algorithm.

D.4. Following the path and finding a feasible solution

The model remains Lipschitz continuous during the entire homotopy algorithm, and we know how its Lipschitz constant changes during this process. By controlling the size of steps, we can ensure the change in the output of model is small at each step. Moreover, our homotopy algorithm starts with a feasible solution. Therefore, if we choose the size of homotopy steps small enough, we can be hopeful to find a feasible solution at each step of the algorithm and eventually to find a feasible flip point for the original model.

We note that following a continuous path requires the domain to be continuous and assumes that the homotopy path does not exit the limits of the domain. Exiting the boundaries of the domain is theoretically possible, but unlikely to encounter, especially for image classification models. This is because the inputs (images) are typically far from the limits of the domain (pixel bounds), while in comparison, decision boundaries of models are very close to the inputs. The domain for image classification models can also be considered continuous. One might be able to provide a guarantee for finding a feasible solution for the original model, but such guarantee requires further analysis.

The property of our homotopy method to start with a feasible solution and try to maintain the feasibility can be valuable because, in some cases, finding any boundary point can be challenging.

In practice, if the homotopy path reaches a domain limit, by default it will try to move in the other dimensions that are not binding. If that fails, we can try to do a trust-region search to jump to an alternative path, or we can add a singularity near the failed path and restart the homotopy algorithm. In such cases, using a global solver for $\mathcal{F}$ may be helpful. We note that $\boldsymbol{x}$ is by definition inside the limits of the domain, so the initiation of homotopy algorithm will not fail.

D.5. Adaptive steps instead of uniform steps

We note that the steps along the homotopy path can be adaptive, instead of uniform steps based on η. In the adaptive approach, if the model fails to find a feasible solution using the solution of

previous iteration as its starting point, the algorithm would retrieve the step and make a smaller step from the previous iteration. Similarly, certain strategies can be used for increasing the size of steps. Also, in our homotopy method, we change the homotopy parameter monotonically (lines 4 and 5 of Algorithm 2), but we note that allowing homotopy parameters to change non-monotonically might be advantageous in some cases (Nocedal and Wright, 2006). These strategies regarding adaptive steps can be the subject of future studies.

D.6. Practical notes about $\mathcal{F}$

Since we are tracing a path of solutions and we expect the solution of each homotopy iteration to be relatively close to the solution from previous iteration, it would make sense to use a trust-region optimization algorithm for $\mathcal{F}$. At each iteration of homotopy method, such $\mathcal{F}$ would use the starting point and expand the radius of its trust-region until it finds a solution. Also, we do not necessarily have to follow a continuous homotopy path and $\mathcal{F}$ can have a global search approach inside a ball centered at its starting point. As long as $\mathcal{F}$ finds a feasible solution at each iteration, the homotopy algorithm can proceed.

Appendix E. Discussions about our optimization problem

E.1. Global optimality

Because our optimization problem is non-convex, there is no guarantee that we actually find its global minimizer. We take several measures to increase our chance.

1. By preprocessing the images with wavelets, we reduce the dimensionality of feature space, which reduces the dimensionality of our optimization problem and makes us more likely to find the global minimizer.

2. We use tunable erf activation functions to make our models smaller and to deal with vanishing and exploding gradients.

3. For comparison, we solve our optimization problem with standard global optimization algorithms and see that our algorithm finds better or similar minimizers.

We note that our optimization problem has a reference point which is its input, $\boldsymbol{x}$. So, as soon as we find any flip point $\hat{\boldsymbol{x}}^c$, we have reduced our search domain to the ball centered at $\boldsymbol{x}$ with radius $\|\boldsymbol{x} - \hat{\boldsymbol{x}}^c\|$.

If the solution we find is a local minimizer of our optimization problem, then the global minimizer will be inside that ball. If the solution we find is the global minimizer of our optimization problem, then all the points inside such ball will have the same classification by the model $\mathcal{N}$ which is useful to measure the adversarial robustness.

E.2. Uniqueness of solution and distance to decision boundary

About the uniqueness, it is possible for multiple flip points to have the same distance to an input. In any case, the closest distance will be unique. If such points are connected, finding one point can lead us to other points, using the degeneracies in the Jacobian matrix. If other closest flip points are not contiguous, one could find them by adding a term to the objective function to put a singularity at the point already found. In our experiments, we have not encountered such cases.

E.3. Extensions of our optimization problem

Deep learning models may have quantified outputs. As an example, suppose that a deep learning model is used to determine the appropriate dosage of a medicine. In this application, a flip point might be the closest point to a given input that changes the dose by a given amount.

In general, for any input $\boldsymbol{x}$ and any model $\mathcal{N}$, we could seek the least changes in $\boldsymbol{x}$ in order to obtain a specific output $\boldsymbol{z}$. In other words, to interpret the models we could try to formulate and solve specific optimization problems about them, beyond finding points on the decision boundaries. The homotopy method we develop finds the closest flip points, but conceptually, it is a method to solve optimization problems involving deep learning functions.

Appendix F. Extension of homotopy method for models with ReLU activations

In general, our homotopy method can be summarized in the following pseudo code.

Algorithm F1 Homotopy algorithm to find the closest flip point to a given input between two classes

Inputs: trained model $\mathcal{N}$, input $\boldsymbol{x}$, lower bound on flow of gradients τ, class i, class j, constraints set $\mathcal{C}$
Output: Closest flip point to $\boldsymbol{x}$

1: Transform the trained model by tuning its activation functions, layer by layer, ensuring that for $\boldsymbol{x}$, gradients of individual neurons are lower bounded by $\tau \rightarrow$ model $\mathcal{N}^{\dagger}$
2: Change the bias parameter for the last layer of $\mathcal{N}^{\dagger}$, such that $\boldsymbol{x}$ becomes a feasible flip point $\rightarrow$ model $\mathcal{N}^{h}$ which is the "easy" model for the initiation of homotopy
3: Derive a Lipschitz constant for the original model $\mathcal{N}$ the transformed model $\mathcal{N}^{h}$, and also the intermediate models
4: Choose the size of homotopy steps based on the Lipschitz constants
5: Gradually transform the $\mathcal{N}^{h}$ into its original form $\mathcal{N}$, through a series of iterations, at each step solving the optimization problem for the intermediate model $\mathcal{N}^{i}$ using the flip point from previous step as the starting point, and satisfying $\mathcal{C}$, tracing a path of flip points
6: **return** the final solution, $\hat{\boldsymbol{x}}^{c}$, the closest flip point for $\mathcal{N}$

Algorithm 1 presented in Section 3.3 is a specific implementation of lines 1 through 4 of Algorithm F1, for models with erf as their activation function. Algorithm 2 which transforms the model back to its original form is the implementation of line 5 of Algorithm F1.

Similarly, the extension of our homotopy transformation for models with ReLU activation function is presented in Algorithms F2 and F3.

Algorithm F2 Algorithm for homotopy transformation of network (ReLU activation)

Inputs: $\mathcal{N}, \boldsymbol{x}, \tau, i, j$
Output: $\boldsymbol{\alpha}^h$ and $\boldsymbol{b}^{h(m)}$

$\boldsymbol{\alpha}^h = \boldsymbol{\alpha}$
$\boldsymbol{y}^{(0)} = \boldsymbol{x}$
for $k = 1$ to $m - 1$ **do**
  if $\| - \boldsymbol{y}^{(k-1)}\boldsymbol{W}^{(k)} - \boldsymbol{b}^{(k)}\|_\infty > 0$ **then**
    $\alpha^h[k] = \tau$
  end if
  $\boldsymbol{y}^{(k)} = PReLU(\boldsymbol{y}^{(k-1)}\boldsymbol{W}^{(k)} + \boldsymbol{b}^{(k)}, \alpha^h[k])$
end for
$\min_{\boldsymbol{b}^{h(m)}} \|\boldsymbol{b}^{h(m)} - \boldsymbol{b}^{\mathcal{N}(m)}\|_2$, subject to:
(1) $\boldsymbol{y}^{(m)} = \boldsymbol{y}^{(m-1)}\boldsymbol{W}^{(\boldsymbol{m})} + \boldsymbol{b}^{h(m)}$,
(2) $y_i^{(m)} = y_j^{(m)}$,
(3) $\forall\, l \neq i, j \mid y_i^{(m)} > y_l^{(m)}$
return $\boldsymbol{\alpha}^h, \boldsymbol{b}^{h(m)}$

Algorithm F3 Homotopy algorithm for calculating closest flip point (ReLU activation)

Inputs: $\mathcal{N}, \boldsymbol{x}, \eta, \tau, \mathcal{C}, i, j$
Output: Closest flip point to $\boldsymbol{x}$

Compute $\boldsymbol{\alpha}^h$ and $\boldsymbol{b}^{h(m)}$ using Algorithm F2 with inputs $(\mathcal{N}, \boldsymbol{x}, \tau, i, j)$
$\hat{\boldsymbol{x}}^{c,0} = \boldsymbol{x}$
for $k = 1$ to η **do**
  $\boldsymbol{\alpha}^k = \frac{\eta - k}{\eta}\boldsymbol{\alpha}^h$
  $\boldsymbol{b}^{k(m)} = \boldsymbol{b}^{h(m)} + k(\frac{\boldsymbol{b}^{\mathcal{N}(m)} - \boldsymbol{b}^{h(m)}}{\eta})$
  Replace $\boldsymbol{\alpha}^k$ and $\boldsymbol{b}^{k(m)}$ in $\mathcal{N}$, to obtain $\mathcal{N}^k$
  $\hat{\boldsymbol{x}}^{c,k} = \mathcal{F}(\boldsymbol{x}, \mathcal{N}^k, \hat{\boldsymbol{x}}^{c,k-1}, \mathcal{C}, i, j)$
end for
return $\hat{\boldsymbol{x}}^{c,\eta}$ as the closest flip point to $\boldsymbol{x}$

In Algorithm F2, we use the parametric ReLU activation defined as:

$$PReLU(x, \alpha) = \mathbb{1}(x < 0)(\alpha\, x) + \mathbb{1}(x \geq 0)(x). \tag{17}$$

We note that instead of PReLU activations, it is possible to use exponential linear units (ELU) (Clevert et al., 2015), where

$$ELU(x, \alpha) = \mathbb{1}(x < 0)(\alpha\, e^x - 1) + \mathbb{1}(x \geq 0)(x) \tag{18}$$

This might have an advantage over PReLU, because the derivative of ELU is continuous over its entire domain, while the derivative of PReLU has a point of discontinuity at zero. Possible advantage of using ELU in Algorithm F2 requires empirical experiments in future research.