Indian Institute Of Technology - Delhi

Amar Nath and Shashi Khosla School of Information Technology (SIT)
Centre of Excellence in Cyber Security and Information Assurance (CoE-CSIA)

Finding Minimal Modifications in Deep Neural Networks.

Submitted by -
Tooba Khan (2021JCS2245)

Submitted to -
Dr. Subodh Vishnu Sharma
Dr. Kumar Madhukar

Contents

1 Adversarial Image Generation

1.1 Introduction

This section discusses the problem of finding adversarial images for a given Deep Neural Network and a set of input images. We propose a divide and conquer based approach to find minimal modifications in the given neural network and the translate the modifications into an adversarial input.

1.2 Terminology used

A deep neural network N has 1 input layer, several hidden layers and 1 output layer. We define two kinds of layers:

1. Neuron layers
2. Weight layers

Each weight layer originates from 1 neuron layer and ends up on next neuron layer. The modification is defined in terms of weight layers. Neuron layers are numbered as 0 for hidden layer 1, 1 for hidden layer 2 and so on. Similarly, weight layer 0 is the layer between input layer and hidden layer 1. This is shown in figure 1.

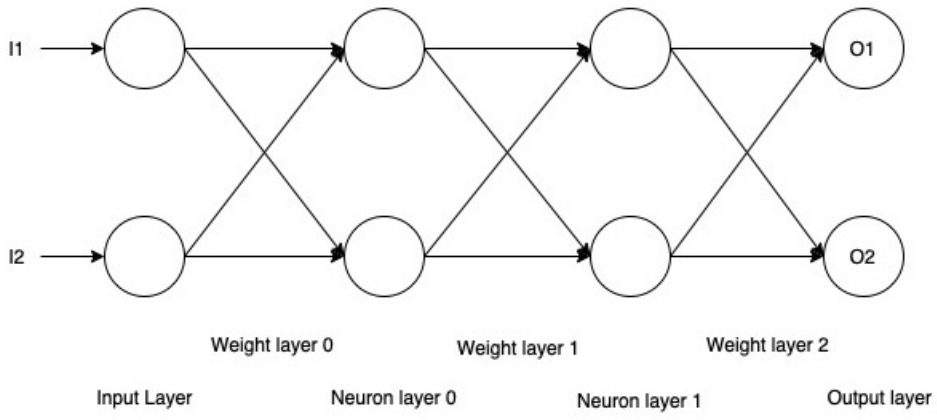


Fig 1 : Example network.

1.3 Implementation

Our technique to generate adversarial inputs is a two step process.

1.3.1 Step 1:

The first step is to find minimal modification in weight layer 0. A Deep neural network can have multiple layers and each layer can comprise of a large number of neurons. Finding modification in layer 0 of the network directly is an expensive process, both in terms of time and computation. To solve this problem, we find modification required in the middle weight layer of the network. This modification should satisfy the objective of adversarial inputs which means that the modification should be such that it causes a misprediction in the network. For example: If the original label of the input is Output_3, then the objective function should minimize the value of value of Output_3.

We maintain a track of which layer has to be modified in the current iteration, known as layer_to_modify. So, initially, $layer\_to\_modify = number\_of\_weight\_layers/2$.

Once we have modifications in the middle layer, the next step is to extract a sub-network from the original network. This is done by extracting neuron layers from input layer to layer_to_modify. Extracting a part of the network reduces the size of problem. In the next iteration, we find a minimal modification in $layer\_to\_modify/4$ on the extracted network. But, here the objective is different from the one in previous iteration. Iteration 2 onwards, our objective is not to minimize the value of true label, but now it is to maintain the neuron values of neuron layer $layer\_to\_modify$ found in the previous iteration, after the modification is applied.

This procedure is continued until $layer\_to\_modify$ becomes 0. At this stage, we have the modification required in the weight layer 0.

1.3.2 Step 2:

Once we have modifications required in the weight layer 0 of the network, we can translate the modifications into an adversarial input. First, we calculate the values of hidden layer 1, after applying modifications to layer 0. Then we use an optimizer to find out what is the minimum change which should be introduced in the input such that without modifying layer 0 weights, the neurons of hidden layer 1 can maintain their values.
For example, if the neurons in hidden layer 1 are labelled as h1,1 h1,2 h1,3 and so on, and after applying modifications to weight layer 0, the values of hidden layer 1 neurons are a1, a2, a3, respectively. Then, the query is to find an input I'=$[I_1 + \delta_1, \delta_2 + I_2, \delta_3 + I_3, ..]$ such that for original weight matrix for weight layer 0 the values of neurons in hidden layer 0 will be a1, a2, a3.

Where, δ_i is the modification in each pixel of the input image.
I=$[I_1, I_2, I_3, ..]$ is the original input.
and I_i is the i^th pixel in the Input image I.

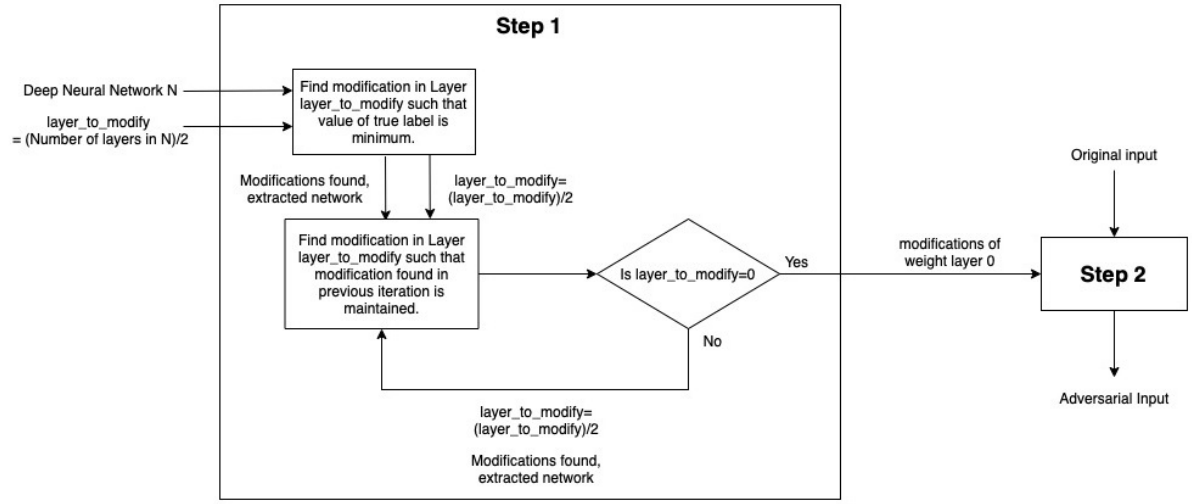


Fig 2 : Overall architecture of the adversarial input generation.

1.4 Example

A sample network is shown in Figure 3.The network has 1 input layer, 2 hidden layers and 1 output layer. It has 3 weight layers (as described previously).

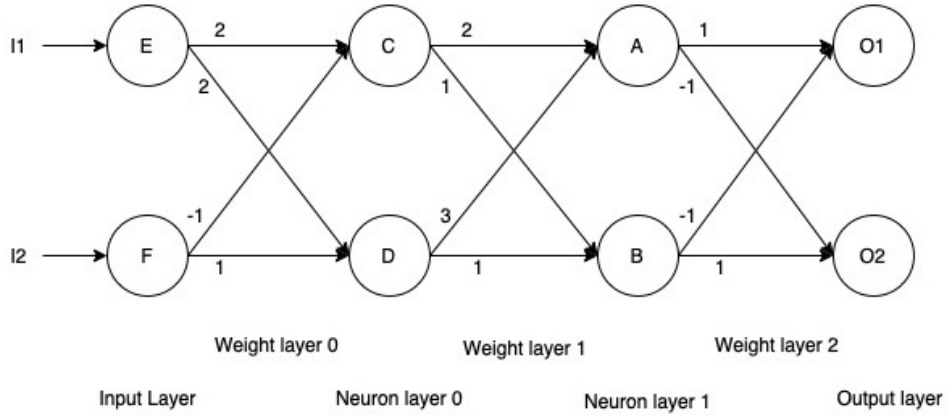


Fig 3 : Example network.

For an input [1, 1], the output of each neuron is shown in Figure 4. The output is [7, -7]

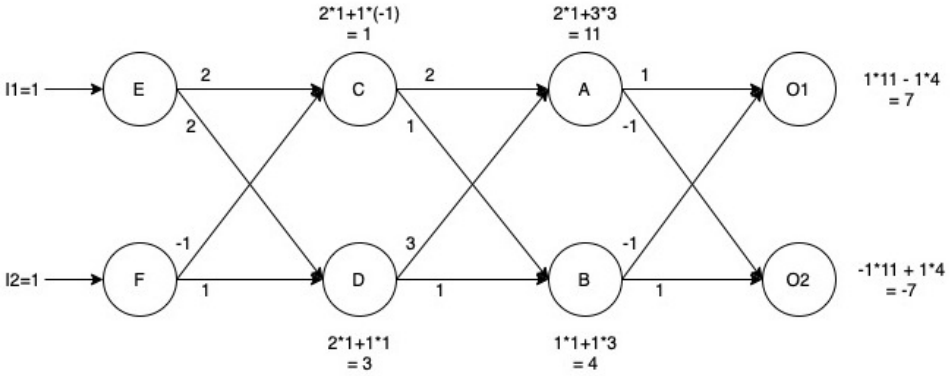


Fig 4 : Outputs of neurons for input [1, 1].

for this input. We want to generate an adversarial input such that the output neuron 2 has a value higher than the output neuron 1. To find such an adversarial input, we run the above algorithm.

Step 1: Finding modification in layer 0 by divide and conquer approach.
Iteration 1:
In the first iteration, number_of_weight_layers is 3. Since, layer_to_modify = (number_of_weight_layers)/2, layer_to_modify will be 1 in this iteration.
A possible modification in weight Layer 1 is:

-2, 0
-2, 0
If this modification is applied, the neurons A and B will get values 3 and 4 respectively. Now, using the values of A and B as 3 and 4, we get the values of output neurons as [-1, 1] as shown in figure 5. Thus, our modification is correct.

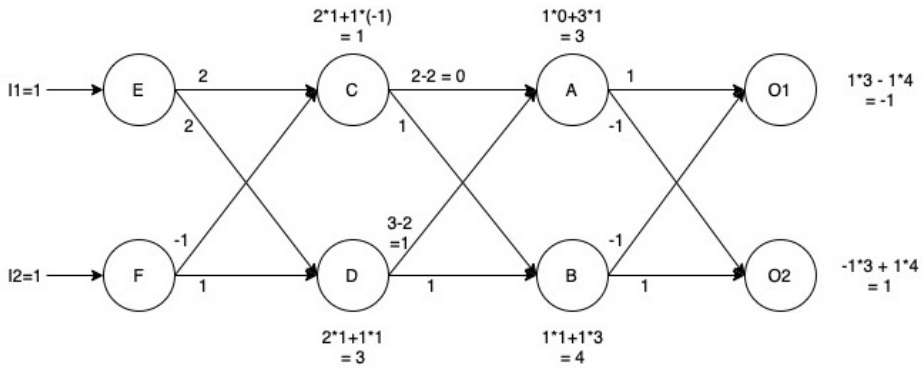


Fig 5 : Modification in weight layer 1.

Iteration 2:
In the second iteration, the network gets reduced. It is extracted from input layer to neuron layer=layer_to_modify. Thus, in this example, the reduced network looks like the one in figure 6.

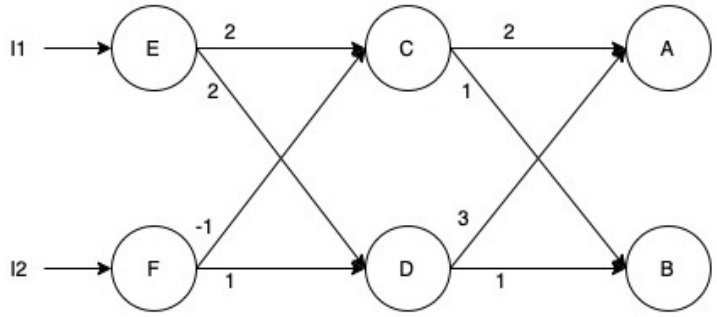


Fig 6 : Reduced network after first iteration.

It has 1 input layer + 2 layers of neurons. The layer_to_modify is calculated using the same formula and it's value in this iteration is 0. But iteration 2 onwards, the objective function changes. The objective function in iteration 1 was: Output_2 < Output_1. In this iteration, the objective function is, find a minimal change in the weight layer 0 such that neruon A=3 and neuron B=4.

One such modification is:
8, 0
0, -8
For the modifications above, the new weight edges are shown in Figure 7. It can be easily verified, that for the above modification, the neurons A and B will be 3 and 4 respectively. For the modification in layer 0, values of neurons C and D are 9, -5 respectively.

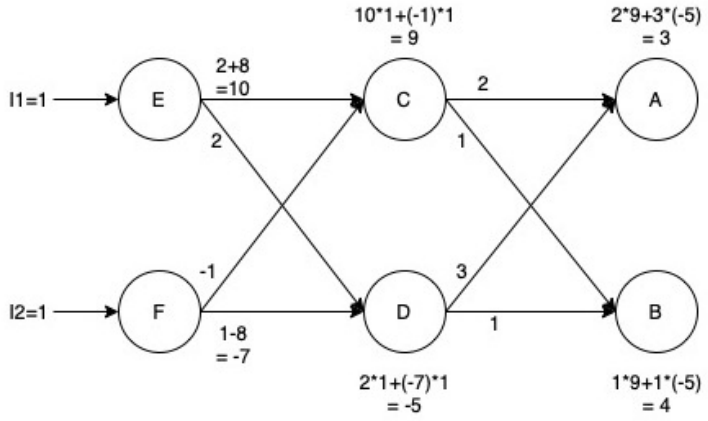


Fig 7 : Modification in weight layer 0.

This iteration gives us the modification in layer 0. We can now translate the modification to an adversarial input.

Step 2: Generating adversarial input using modifications of layer 0
In this step, the objective is to find the minimal modification required in the input such that The values of neurons C and D are 9, -5 respectively. This is visualized in figure 8.

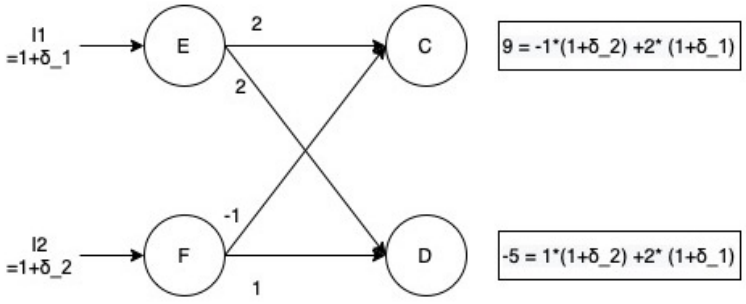


Fig 8 : Converting modifications of layer 0. to adversarial input.

If a delta is added to each input, the two linear equations will be:

$$9 = 2 * (1 + \delta_1) - 1 * (1 + \delta_2)$$

$$-5 = 2 * (1 + \delta_1) + 1 * (1 + \delta_2)$$

On solving the two equations above, we get $\delta_1 = 0$ and $\delta_2 = -8$. Thus, the final adversarial example is: [1, -7]. The output of each neuron and final output of network for the adversarial input generates, is shown in Figure 9 below:

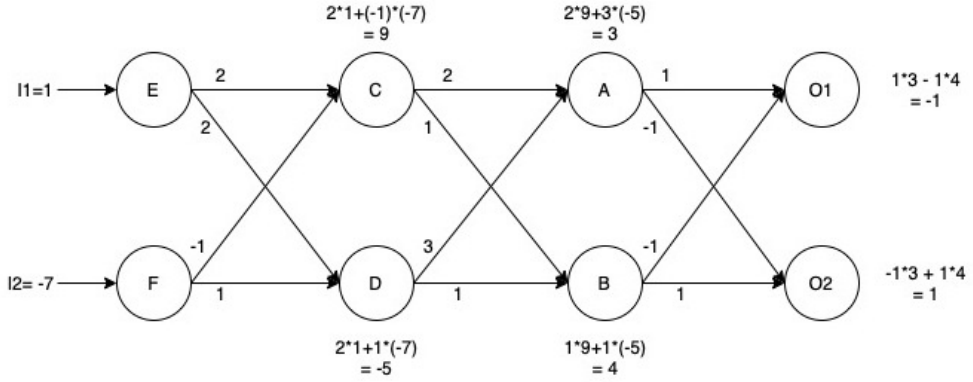


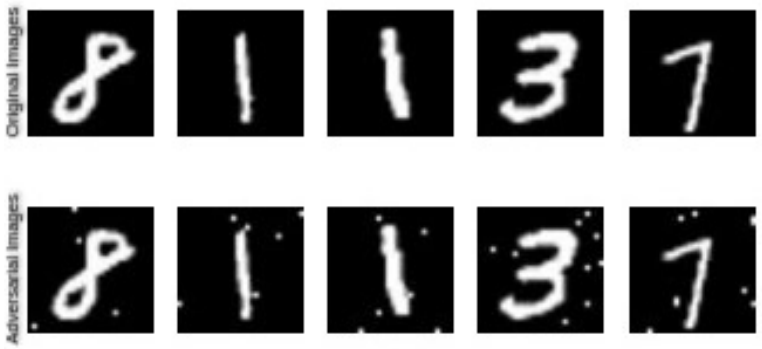
Fig 9 : Output for the Adversarial Input.

It can be seen that for the adversarial input generated, output of network is [-1, 1] and our objective is satisfied.

1.5 Results

Tests were conducted on 1000 randomly selected inputs from the MNIST data set and DNN trained on MNIST. The DNN had 1 input layer, 3 hidden layers and 1 output layer. When maximum L-∞ allowed was 1, and maximum number of pixels which can be changed were only 15 out of 784, adversarial inputs were found for 66% of them. Figure 10 shows some adversarial inputs generated by our technique.
Generation of an adversarial input took 1.7 seconds on an average.

(a)



(b)


Original Images
Adversarial Images

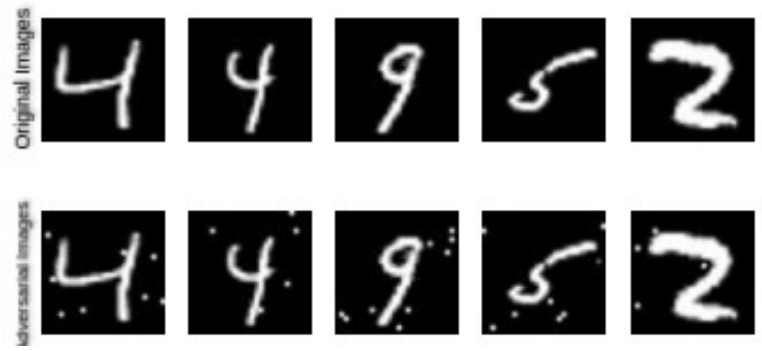
(c)


Original Images
Adversarial Images

(d)


Original Images
Adversarial Images

(e)



(f)

(g)

Fig 10: Adversarial Images generated when maximum modification was 1 and only 15 out of 784 pixels were allowed to be modified. In figures from (a) to (g) the first row shows original image and second row shows adversarial images generated. Originally, all images are of size 28x28 but for displaying, they have been resized to size 56x56.