Adversarial Attack

Rishitosh Kumar Singh

February 9, 2021

1 Learning Rule

Consider a trained three-layered (L-M-N) artificial neural network (ANN) of conventional neurons. The trained ANN consists of L number of input neurons, M number of hidden neurons, and N number of output neurons. The net potential of m^{th} conventional hidden neuron is defined as

$$V_m = \sum_{l=1}^{L} w_{lm} x_l + w_{0m} x^0 \tag{1}$$

and its output is defined as

$$Y_m = \quad f(V_m) \tag{2}$$

Similarly, the net potential of n^{th} output neuron is defined as

$$V_n = \sum_{m=1}^{M} w_{mn} Y_m + w_{0n} x^0 \tag{3}$$

and its output is defined as

$$Y_n = \quad f(V_n) \tag{4}$$

Let Y_n^D be the desired output and Y_n^{Adv} be desired adversary output of the n_{th} output neuron. Since the network's already trained, $Y_n^D - Y_n$ will be very small. Adversarial error due to output (e_n^Y) can be calculated as $e_n^Y = Y_n^{Adv} - Y_n$. Adversary's goal is to generate a slightly perturbed version of x^{Adv}. To make small perturbations in the image, error e^x is introduced. For l_{th} input this error can be calculated as $e_l^x = x_l^{Adv} - x_l$. The cost function (E) can be calculated using the mean square of both e^Y, and e^x adversarial errors and expressed as

$$E = \quad \frac{1}{2N} \sum_{n=1}^{N} (e_n^Y)^2 + \lambda \frac{1}{2L} \sum_{l=1}^{L} (e_l^x)^2 \tag{5}$$

where, λ is real number in range $(0, \infty)$ for target adversarial attack and for non-target adversarial attack, $\lambda = 0$. The goal is to keep reducing error (E) by updating input (x). The update rule adds the error gradient to input as

$$x^{old} = x^{new} + \Delta x \tag{6}$$

For particular input (x_l), Δx_l can be calculated as

$$\Delta x_l = \frac{\eta}{N} \sum_{m=0}^{M} \delta_m w_{lm} + \lambda \frac{\eta}{L} e_l^x \tag{7}$$

where,

$$\delta_m = \left\{ \sum_{n=0}^{N} \delta_n w_{mn} \right\} f'(V_m)$$

, and

$$\delta_n = e_n^Y f'(V_n)$$

The eqs. (5) to (7) is implemented as described in algorithms 2 to 4.

Algorithm 1: Generate Adversarial Image

input : Desired input (x^{Adv}), Desired adversary output (Y^{Adv}), η, λ, Epochs
output: NULL
$x \leftarrow x^{Adv}$
while *Epochs* ***is not*** *0* **do**
  Forward-pass(x)
  Backward-adversarial-pass(Y^{Adv})
  Update-inputs($x, x^{Adv}, \eta, \lambda$)
  Epochs $\leftarrow$ Epochs -1
end

Algorithm 2: Forward-pass

input : Input Vector (x)
output: Predicted Output Vector (y)
for $i \leftarrow 0$ **to** *num_layers* -1 **do**
  foreach *neuron, m* ***in*** *layer$_i$* **do**
    $V_m^i \leftarrow \sum_{l=1}^{len(x)} w_{lm} x_l + w_{0m} x^0$
    $Y_m^i \leftarrow f(V_m^i)$
  end
  $x \leftarrow Y^i$
end

2 Performance Evaluation

Training ANN

An ANN with structure [784−20−10] with sigmoid activation function is constructed and compiled with stochastic gradient descent (SGD) as optimizer and mean square error (MSE) as loss function. The network consists of one hidden layer with 20 neurons and one output layer with 10 neurons.

The compiled network is trained on a subset of MNIST dataset consisting of random 600 training samples and 100 testing samples. The network was able to achieve training accuracy of 96% and testing accuracy of 89%. Figure 1 illustrates testing performance of trained network.

Algorithm 3: Backward-adversarial-pass

input : Predicted Output Vector (y)
output: NULL
for $i \leftarrow num\_layers - 1$ **to** 0 **do**
  if $layer_i$ ***is*** *Output-Layer* **then**
    foreach *neuron*, n ***in*** $layer_i$ **do**
      $\Delta_n^i \leftarrow \left\{ e_n^i \cdot f'(V_n^i) \right\}$
    end
  end
  else
    foreach *neuron*, m ***in*** $layer_i$ **do**
      $e_m^i \leftarrow 0$
      foreach *neuron*, n ***in*** $layer_{i+1}$ **do**
        $e_m^i \leftarrow e_m^i + \left\{ w_{mn} \cdot \Delta_n^{i+1} \right\}$
      end
      $\Delta_m^i \leftarrow \left\{ e_m^i \cdot f'(V_m^i) \right\}$
    end
  end
end

Algorithm 4: Update-inputs

input : Input Vector (x), Desired input (x^D), η, λ
output: NULL
for $i \leftarrow 0$ **to** $len(x) - 1$ **do**
  $x_i \leftarrow x^i + \frac{\eta}{N} \left\{ \Delta_i^0 + \lambda \cdot (x_i^D - x_i) \right\}$
  `// N is the number of output neurons`
  $x_i \leftarrow x^i + \eta \left\{ \frac{1}{N} \Delta_i^0 + \frac{\lambda}{L} \cdot (x_i^D - x_i) \right\}$
end

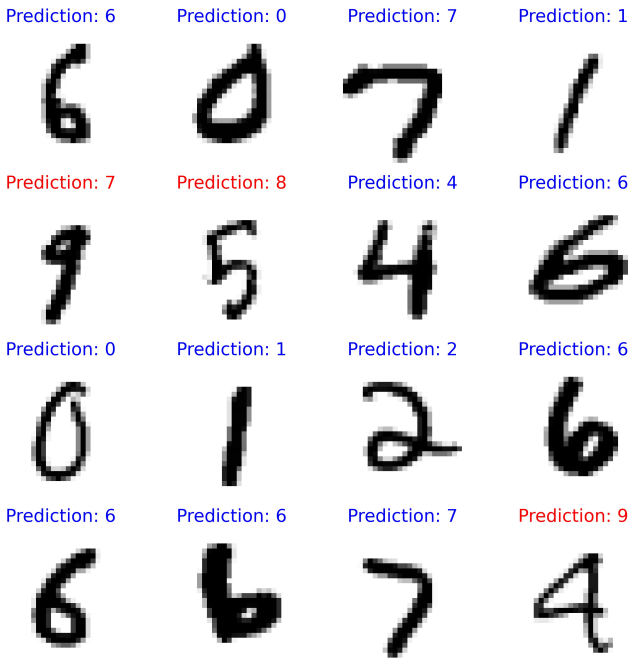
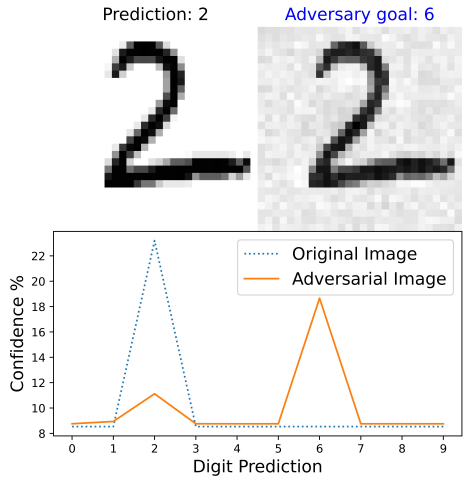


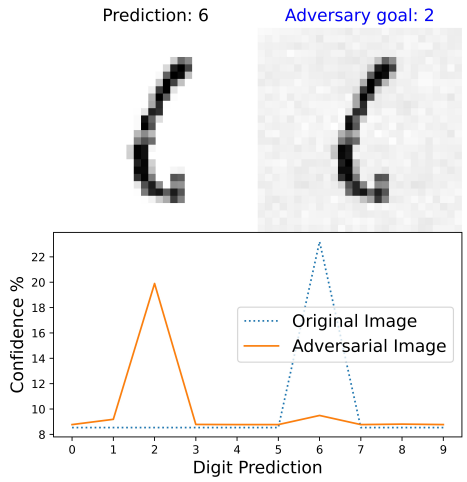
Figure 1: Predictions of trained network on test set

Target Adversarial Attack

For generating adversarial images, algorithms 1 to 4 are used where adversary just need to pass target image (x^{Adv}) and goal (Y^{Adv}) in algorithm 1. For generating adversarial images $\lambda = 0.8$ is considered. A noise will be generated as an adversarial image if $\lambda = 0$ is considered. Figure 2 consists of all generated adversarial images using which target adversarial attack was successful. In fig. 2 it can also be observed how prediction confidence changed when adversarial image is used. Some generated images as shown in fig. 3 were not able to fool the trained network, but it is worth to notice that network confidence in correctly predicting particular image is decreased, so they can be termed as incomplete attacks.



(a)



(b)

Figure 2: Some original images (in left) and generated adversarial images (in right) with their prediction (below)

3 Conclusion

In this paper, a gradient-based adversarial image generation algorithm is proposed. Using the proposed algorithm, the adversary can generate adversarial images for target and non-target adversarial attacks. Neural networks are a

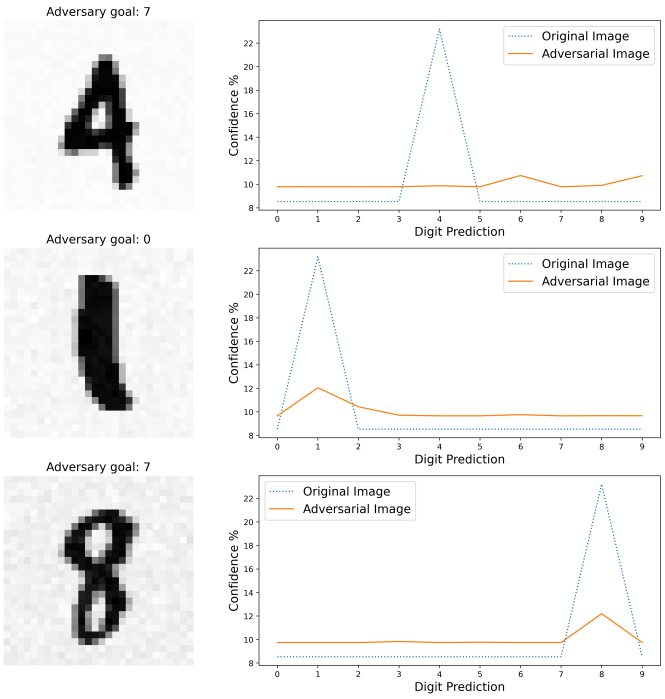


Figure 3: Some generated adversarial images (in left) using which attack was not successful with its prediction (in right)

black box, what a network learns is still unknown, which makes them more vulnerable to be duped. This paper demonstrated that tricking neural networks using adversarial examples is an easy task. Just like an ANN, all complex neural networks such as CNN are also susceptive to adversarial attacks. Instead of employing neural networks in every possible scenario, a tremendous amount of research is required to protect them from these attacks.

Appendices

Derivation of learning rules

Consider a trained three-layered neural network with structure $(L - M - N)$ consisting of L inputs, M hidden neurons, and N output neurons. The output of m^{th} neuron Y_m is computed by activation function of net potential (V_m), where V_m and Y_m are defined respectively as

$$V_m = \sum_{l=1}^{L} w_{lm} x_l + w_{0m} x^0 \tag{.1}$$

and

$$Y_m = \quad f(V_m) \tag{.2}$$

Similarly, the respective net potential (V_n) and output (Y_n) of n^{th} output neuron are given as

$$V_n = \sum_{m=1}^{M} w_{mn} Y_m + w_{0n} x^0 \tag{.3}$$

and

$$Y_n = \quad f(V_n) \tag{.4}$$

Let Y_n^{Adv} be the desired adversary output at n^{th} neuron, then the error, e_n^Y at n^{th} output neuron is calculated through the difference between Y_n and Y_n^{Adv}, which is expressed as

$$e_n^Y = Y_n^D - Y_n \tag{.5}$$

and let x^{Adv} be the desired adversarial image, then the error e_l^x at l^{th} input is calculated through the difference between x_l and x_l^{Adv}, which is expressed as

$$e_l^x = x_l^{Adv} - x_l \tag{.6}$$

The cost function E (MSE) can be calculated by

$$E = \frac{1}{2N} \sum_{n=1}^{N} (e_n^Y)^2 + \lambda \frac{1}{2L} \sum_{l=1}^{L} (e_l^x)^2 \tag{.7}$$

During adversarial image generation, inputs are adaptable instead of network weights. The gradient-decent based backpropagation is used to minimize the cost function. The current input x^{old} is updated to x^{new} using Δx as

$$x^{old} = x^{new} + \Delta x \tag{.8}$$

where Δx is promotional to negative gradient of cost function $(\nabla_x E)$.

$$\begin{aligned} \Delta x &= -\eta \nabla_x E \\ &= -\eta \cdot \frac{\partial E}{\partial x} \end{aligned} \tag{.9}$$

For input x_l, $-\partial E / \partial x_l$ is derived using chain rule of derivation.

$$-\frac{\partial E}{\partial x_l} = \frac{1}{N} \sum_{n=0}^{N} \left\{ e_n^Y \cdot f'(V_n) \cdot \frac{\partial V_n}{\partial x_l} \right\} + \frac{\lambda}{L} e_l^x \tag{.10}$$

Now, substituting eq. (.10) in eq. (.9) yields

$$\Delta x_l = \frac{\eta}{N} \left\{ \sum_{n=0}^{N} \delta_n w_{mn} \right\} \nabla_{x_l} Y_m + \lambda \frac{\eta}{L} e_l^x \tag{.11}$$

where $\delta_n = e_n^Y f'(V_n)$

Now using eqs. (.1) and (.2), $\nabla_{x_l} Y_m$ can further be simplified using chain rule of derivation

$$\begin{aligned} \nabla_{x_l} Y_m &= \sum_{m=0}^{M} \left\{ f'(V_m) \frac{\partial V_m}{\partial x_l} \right\} \\ &= \sum_{m=0}^{M} f'(V_m) w_{lm} \end{aligned} \tag{.12}$$

Now, substituting eq. (.12) in eq. (.11) yields

$$\Delta x = \frac{\eta}{N} \sum_{m=0}^{M} \delta_m w_{lm} + \lambda \frac{\eta}{L} e_l^x \tag{.13}$$

where,

$$\delta_m = \left\{ \sum_{n=0}^{N} \delta_n w_{mn} \right\} f'(V_m)$$

References