CSCI 4261/6961 Pedagogical Project Assignment

Haniel Campos, Gretchen Forbush, Alex Kim

December 2, 2020

1 Introduction and Overview

1.1 Generating Adversarial Attacks

This is a subsection. In the realm of adversarial attacks, there are two distinct types: targeted and untargeted. Given target model, M, and image, I, with true class, X:

- Untargeted Attacks - goal: have M misclassify I as a class other than X
  - Benefit: faster
  - Drawback: not as reliable
- Targeted Attacks - goal: have M misclassify I as a target class, Y
  - Benefit: more successful attack method
  - Drawback: costly (time)

1.2 Benign vs. Attack Systems

As we have seen in our studies, benign systems (those without attacks) have a variation of the following goal

$$argmin_f \sum_{x_i \in D} l(f(x_i), y_i)$$

where f is a prediction, l is the loss function, D is the input data, x_i is an element in the data, and y_i is the target label

In an attack, the system is changed slightly by these two steps:

1. The input data (D) is changed from D to D'
2. A goal is set for the attack so that $f(x_i)$ no longer outputs y_i. This will change the loss inputs from $l(f(x_i), y_i)$ to $l(f(x_i), y_i')$ with $y_i \neq y_i'$.

1.3 Adversarial Perturbation

Step 1 can be performed through a method called "adversarial perturbation."

- Untargeted attacks: maximize loss between $f(x)$ and $f(x')$ until the prediction is incorrect
- Targeted attacks: maximize loss between $f(x)$ and $f(x')$ AND minimize the loss between $f(x')$ and y' until $f(x') = y'$

There are two types of adversarial perturbation:

1. Single-step - add noise once and be done e.g. Fast Gradient Sign Method

$$\text{Untargeted: } x' = x + \epsilon.sign(\nabla_x l(x, y))$$

$$\text{Targeted: } x' = x - \epsilon.sign(\nabla_x l(x, y_{target}))$$

   - Benefit: fast
   - Drawbacks: often easier to detect; focuses on maximizing the loss over minimizing perturbation

2. Multi-step - make a small perturbation at each iteration e.g. Fast Gradient Sign Method

$$\text{Untargeted: } x'_0 = x \implies x'_{N+1} = Clip_{x,\epsilon}\{x'_N + \alpha.sign(\nabla_x l(x'_n, y)\}$$

$$\text{Targeted: } x'_0 = x \implies x'_{N+1} = Clip_{x,\epsilon}\{x'_N + \alpha.sign(\nabla_x l(x'_n, y_{target})\}$$

   - Benefit: more successful
   - Drawbacks: more expensive (computationaly)

2 Problem Set

Using the starter code from `adv_problemset_starter.ipynb`, complete the following tasks:

1. Write a function `adversarial_attack(images, labels, eps)` that performs a single-step untargeted adversarial attack on the images provided.
   - `images` is the set of input images provided to the model
   - `labels` is the set of true labels provided by the dataset
   - `eps` is a hyperparameter used in the calculations

   HINTS:

   - Use the loss object `keras.losses.CategoricalCrossentropy()` to calculate the loss for each iteration. Make sure to indicate that the loss is being calculated from the logits. See `https://www.tensorflow.org/api_docs/python/tf/keras/losses/CategoricalCrossentropy` for more details.
   - Consider converting the inputs to `tensor`s to take advantage of certain built-in functions (e.g. tf.GradientTape()). Just do not forget to convert back to `nparray`s on output!

2. Run the adversarial attack function on the training data with different values of epsilon to determine an appropriate value.
   - Evaluate the model on these attack images against the true labels and output the base accuracy.
   - Use the model to predict the outputs of the adversarial images.
   - Use the pre-defined method `display_images(images, predicted_labels, true_labels)` to display some of the predictions.

3. Write a function `multistep_adversarial_attack(images, labels, eps, a, T)` that performs a multistep untargeted adversarial attack on the images provided.
   - `images` is the set of input images provided to the model
   - `labels` is the set of true labels provided by the dataset
   - `eps` is a hyperparameter used in the calculations
   - `a` is a parameter that defines the step-size by $\alpha_t = (1 - a * t)^{-1}$
   - `T` is the number of steps taken per input image

HINTS:

- Use the same loss object `keras.losses.CategoricalCrossentropy()` to calculate the loss for each iteration.
- You should not have to change too much from the single-step algorithm!

4. Run the adversarial attack function on the training data with different values of epsilon, a, and T to determine an appropriate value.
   - Evaluate the model on these attack images against the true labels and output the base accuracy.
   - Use the model to predict the outputs of the adversarial images.
   - Use the pre-defined method `display_images(images, predicted_labels, true_labels)` to display some of the predictions.
5. Compare the results from step 2 to those from step 4.
   - Are they as you expected?
   - What is the runtime complexity of both of your algorithms?
   - How might you expect a targeted attack to perform compared to the untargeted ones you implemented?
   - What would you expect the time complexity of a targeted (single-step and multistep) attack to be?