Deep Learning Practice: Homework3

张驰　　Academy for Advanced Interdisciplinary Studies

Apr 2, 2019

1 Goals

Learn the basic principle of generating adversarial examples.

1.1 Background

By adding small perturbations to an image, we can force a wrong classification of a trained neural network. The problem is formulated as, finding $\widehat{x}$ $s.t. D(x, \widehat{x}) \leq \epsilon$, and $argmax_j P(y_j|\hat{x}, \theta) \neq y_{true}$ (untargeted), or $argmax_j P(y_j|\hat{x}, \theta) = y_{target}(targeted)$, $where P(y|x, \theta)$ is a classifier model parameterized by θ and ϵ is the maximum allowed noise. And $\widehat{x}$ is the generated adversarial examples.

- In this experiment we target 100 images from CIFAR-10 dataset and use a VGG-like model.
- We evaluate the generated examples with perturbation scale and success rate, i.e. the probability of generated adversarial examples being misclassified by the model

2 Q1: Generating untargeted adversarial examples

- Run the baseline code and record the perturbation scale and success rate.

We firstly run the baseline. For more information, I modify the config as follow:

```
# In attack.py
acc, loss, adv, x, logits = attack.generate_graph(pre_noise, data, gt, label)
# In train.py
 _, accuracy, loss_batch, adv_examples,ori_image, a, summary = sess.run([train, acc, loss,
                                        adv, logits, merged],feed_dict=feed_dict)

cv2.imwrite('../../ori_image/'+'{}.png'.format(idx),ori_image.astype('uint8').reshape(32, 32
                                        , 3))
cv2.imwrite('../../adv_image/'+'{}.png'.format(idx),adv_examples.astype('uint8').reshape(32,
                                         32, 3))

accuracy_gt=acc_gt.eval(feed_dict={placeholders['data']:adv_examples,placeholders['gt']:
                                        labels})
```

I demonstrate more values to view the details. Table.1 is the final result:

Table. 1: Result of Baseline

Result
Success rate of this attack is 0.99
Noise norm of this attack is 24.92516326904297

However, let us see the adversatial figure and original figure after 500 epochs. The Figure.1 shows five contrast experiments. The left is original image with its label and the right is advesatial image with its label Comparing with the result of baseline, we can find that small perturbations on samples can have a big impact on the classification results. Noise norm is about 25, which is to say there is only about 5 pixels changed in the experiment.

Now, we are aware that generated examples with perturbation are bad for our ideal result. So, we must take some measures to defend the interference of adversatial samples.

3 Q2: Adding loss terms

Total variation is defined as $\sum_{i,j}(y_{i+1,j} - y_{i,j})^2 + (y_{i,j+1} - y_{i,j})^2$ where i,j denotes row and column index of a pixel in the image. Total variation loss encourages images to be made up of piece-wise constant patches.

3.1 Results analysis for total variation

- Adding averaged total variation loss(explained in Homework2) and see how the perturbation scale and success rate changes.

```
def add_variation(noise):
  rel = tf.zeros(noise.shape)
  for k in range(1, noise.shape[1]):
    rel += tf.square(noise[0, k, :, :] - noise[0, k - 1, :, :]) + tf.square(noise[0, :, k, :] -
                                                    noise[0, :, k - 1, :])

  return tf.reduce_mean(rel)
```

Table.2 is the result of adding total variation loss:

Table. 2: Result of adding total variation loss

Result
Success rate of this attack is 0.7899999999999998
Noise norm of this attack is 36.90948104858398

It can be seen that after adding total variation loss, the success rate of this attack is reduced by about 20%. TV loss smoothes the images, thereby limiting the noise distribution of adversarial samples. So, we can concluded that TV loss can improve the robustness of samples and improve the defend adversarial attacks.

3.2 Results analysis for l2 loss between adversarial examples and the original images

- To obtain minimum distortion, add l2 loss between adversarial examples and the original images. You may construct your loss as， $L = crossentropyloss(preds, gt) + c(\widehat{x} - x)^2$ Rerun the experiment to see how success rate changes with different c value, e.g. 1, 0.1, 0.01 etc.

Adding l2 loss is quite easy, but we should make clear the loss is the images without normalization as follow:

```
loss = tf.losses.softmax_cross_entropy(target_one_hot, logits) *(-1)+ 0.1*tf.reduce_mean(tf.
                                                    square(x-x_round))
```

Table.3 are the results of different values of C:

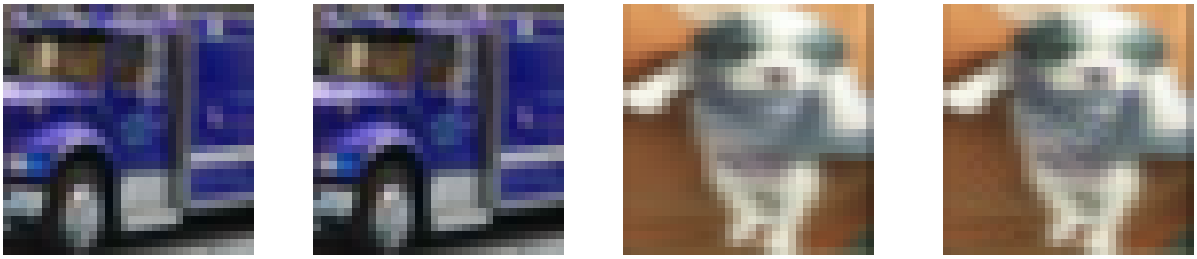
Table. 3: Result of adding l2 loss

Result	Value of C	whether adding tv loss
Success rate of this attack is 0.88 Noise norm of this attack is 19.40403938293457	0.01	No
Success rate of this attack is 0.79 Noise norm of this attack is 32.22861099243164	0.01	Yes
Success rate of this attack is 0.73 Noise norm of this attack is 10.287617683410645	0.1	No
Success rate of this attack is 0.71 Noise norm of this attack is 15.223876953125	0.1	Yes
Success rate of this attack is 0.48 Noise norm of this attack is 1.822910189628601	1	No
Success rate of this attack is 0.49 Noise norm of this attack is 2.0136361122131348	1	Yes

The success of attack is reducing as C increaing, and noise norm is the same. The combination of l2 loss and tv loss do not improve the result too much for C=1 while it is helpful for C=0.01 and slightly improving the result for C=0.1, though the noise norm is larger. The reason may be related to the magnitude for C and tv loss has certain restriction for l2 loss.

L2 regularization between the original image and the adversarial examples means that the gap between the original image and adversarial examples should not be too large, to be specific, the nosie can not be too large. We should add as little noise as possible in case of misclassification. In fact, this operation is also a adversatial process, that is, to ensure the adversatial samples are misclassified as much as possible and the sample and the original image are as close as possible.

Finally, Figure.2 verifies our assumption. The differences between the original image and the adversarial examples are very small.



(a) ori.truck (b) adv.truck (c) ori.cat (d) adv.dog

Figure. 2: Adversitial sample and original image using L2 loss

4 Q3: Whether augmentation helps defending adversarial attacks?

4.1 Results analysis for affine transformation

- Implement one or some of the augmentation techniques, e.g. affine transformation, adding salt and pepper noise, bluring etc. on the generated adversarial examples. Evaluate the augmented adversarial examples and find out whether these examples still get misclassified.

The point of this experiment was to figure out when to do the augmentation technique. In fact, it is for adversarial examples. So we should do augmentation for adversarial examples after 500 epoch and see the success rate.

Table.4 is the result:

Table. 4: Result of affine transformation

Result
Success rate of this attack is 0.89
Noise norm of this attack is 7286.65625

We can find 10% reduction in result. And after affine transformation, the difference between images is huge. So it changed the original image a lot, I do not think it is a good way.

5 Q4: Generating targeted adversarial examples

- In targeted attack scenario, please assign a target label for each image. After training, the corresponding examples will be classified as this label. You need to modify the loss term and rerun the generating process. You may also increase the number of epochs in each run.

After these experiment, based on the analysis, I find some rules. In untargeted attack scenario, we thought noise was undirected. After 500 epoch, it would tend to any label except ground truth, but I found it would be label of other image that is similar to original image, such as cat is similar to dog, automibile is similar to truck. So it seems apparently untargeted, but it has target in training. Figure.3 and Figure.4 show the details.

```
e:48/100,  1 loss: 3.949 acc: 1.000000 logits: 0.998738 index: 2
e:48/100,  2 loss: 3.525 acc: 1.000000 logits: 0.761989 index: 2
e:48/100,  3 loss: 3.796 acc: 1.000000 logits: 0.998693 index: 2
e:48/100,  4 loss: 3.877 acc: 1.000000 logits: 0.999240 index: 2
e:48/100,  5 loss: 3.885 acc: 1.000000 logits: 0.994599 index: 2
e:48/100,  6 loss: 3.861 acc: 1.000000 logits: 0.971986 index: 2
e:48/100,  7 loss: 3.862 acc: 1.000000 logits: 0.975869 index: 2
```

Figure. 3: Detail Analysis

```
e:48/100, 494 loss: -12.777 acc: 0.000000 logits: 0.999798 index: 4
e:48/100, 495 loss: -7.572 acc: 0.000000 logits: 0.999932 index: 4
e:48/100, 496 loss: -5.948 acc: 0.000000 logits: 0.999862 index: 4
e:48/100, 497 loss: -7.389 acc: 0.000000 logits: 0.999666 index: 4
e:48/100, 498 loss: -11.579 acc: 0.000000 logits: 0.999932 index: 4
e:48/100, 499 loss: -8.983 acc: 0.000000 logits: 0.999309 index: 4
e:48/100, 500 loss: -9.451 acc: 0.000000 logits: 0.999936 index: 4
```

Figure. 4: Detail Analysis

However, we set targeted label for each image. Here are the key codes as follow:

```
#In common.py
nr_epoch = 3000
#In attack.py   adversatial samples approaches the specified label
loss = tf.losses.softmax_cross_entropy(target_one_hot, logits)
#In dataset.py
def instance_generator(self):
  img = np.concatenate((img_r, img_g, img_b), axis = 2)
  label = self.samples_mat['Y'][i]
  target=self.target_label(label)

  yield img.astype(np.float32), np.array(label, dtype=np.int32), np.array(target, dtype=np.int32
                                                                          )

#In train.py
  accuracy_ta = acc_gt.eval(feed_dict={placeholders['data']: adv_examples, placeholders['gt']:
                                                                          target})
```

```
accuracy_gt = acc_gt.eval(feed_dict={placeholders['data']: adv_examples, placeholders['gt']:
                                     labels})
ta_succ = (idx * succ + 1 - accuracy_ta) / (idx + 1)
succ = (idx * succ + 1 - accuracy_gt) / (idx + 1)
```

Table.5 is the final result.

Table. 5: Result of targeted attack

Result
Success rate of target attack is 0.99
Success rate of this attack is 1.0
Noise norm of this attack is 32.489326477050785

After 3000 epoch, the targeted attack reach its goal (Figure.5, Figure.6), the final result is indeed the targeted label. still get a little different in pixel. It is only changed by 5 or 6 pixel on average. Once again, a small attack on pixel can change the outcome of a classification, that is remarkable. Neural networks must improve robustness by some means

```
e:99/100, 1 loss: 8.876 acc: 1.000000 logits: 0.999495 index: 1 target: [3]
e:99/100, 2 loss: 7.454 acc: 1.000000 logits: 0.987788 index: 1 target: [3]
e:99/100, 3 loss: 10.463 acc: 1.000000 logits: 0.994394 index: 1 target: [3]
e:99/100, 4 loss: 9.729 acc: 1.000000 logits: 0.999637 index: 1 target: [3]
e:99/100, 5 loss: 8.405 acc: 1.000000 logits: 0.996569 index: 1 target: [3]
e:99/100, 6 loss: 10.060 acc: 1.000000 logits: 0.984728 index: 1 target: [3]
```

Figure. 5: Tagrget Attack Start Result

```
e:99/100, 2995 loss: 0.001 acc: 0.000000 logits: 0.998586 index: 3 target: [3]
e:99/100, 2996 loss: 0.105 acc: 0.000000 logits: 0.899924 index: 3 target: [3]
e:99/100, 2997 loss: 0.029 acc: 0.000000 logits: 0.971028 index: 3 target: [3]
e:99/100, 2998 loss: 0.289 acc: 0.000000 logits: 0.749325 index: 3 target: [3]
e:99/100, 2999 loss: 0.037 acc: 0.000000 logits: 0.963736 index: 3 target: [3]
e:99/100, 3000 loss: 0.105 acc: 0.000000 logits: 0.899924 index: 3 target: [3]
```

Figure. 6: Tagrget Attack Final Result

In practice, we find that the noise after each 500 epoch will not affect the initial classification of the next image. So there comes a question: does the final noise have an effect on the whole images, or just a local effect for single image.

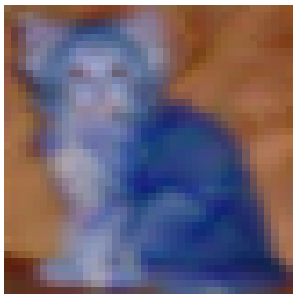
(a) ori.automobile

(b) adv.truck

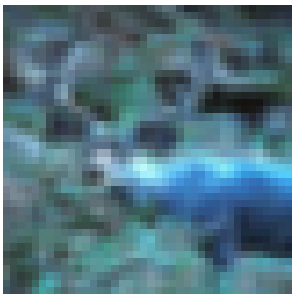
(c) ori.horse

(d) adv.deer

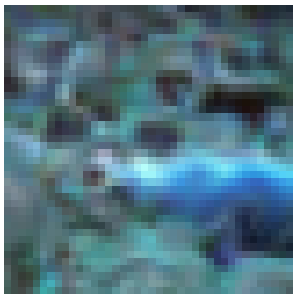
(e) ori.cat



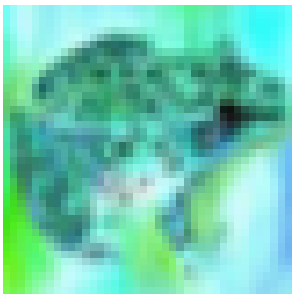
(f) adv.dog



(g) ori.deer



(h) adv.automobile



(i) ori.frog

(j) adv.truck

Figure. 1: Original figure and adversarial figure after 500 epochs.(Only slightly different)