A comparative study of Zeroth Order Optimization attacks on the MNIST dataset

Tanmay Ambadkar
Department of Computer Science and Engineering
Pennsylvania State University
University Park, PA, 16803
ambadkar@psu.edu

Abstract

This paper presents a comparative study of Zeroth Order Optimization (ZOO) attacks on the MNIST dataset. ZOO is a type of adversarial attack that aims to fool a machine learning model by modifying the input in such a way that the model produces a wrong prediction. The study compares the Newton and Adam ZOO methods and evaluates their effectiveness on the MNIST dataset, which is a widely used benchmark for machine learning models. The results of the study show that ZOO methods can successfully fool machine learning models on the MNIST dataset. Additionally, the study shows the performance of different ZOO methods, with some methods being faster than others.

1 Introduction

The following assignment explores zoo Adam and zoo Newton's attacks on the MNIST dataset. The model being attacked was ResNet18. We implement the untargeted attack in python. The boilerplate code has been provided.

I have implemented the patchwise coordinate descent algorithm to speed up the operation. I have used a window size of 4x4.

I will explain all code that has been used to implement these attacks in the following subsections:

1.1 Loss Function

The loss function is used to find the difference between the true label and the next label with the highest confidence. We find the log of the probabilities and implement the following function

$$f(x) = max\{log[F(x)]_{t0} - maxlog[F(x)]_i, -k) \tag{1}$$

Where $F(x)_i$ is the probability of the class i. The ResNet model provides logits instead of class probabilities. We use the log_softmax function in PyTorch to do the same. The code implementation is below

```
def loss_fn(logits, label):

    log_probs = F.log_softmax(logits, dim=1).reshape(-1,)
    label_log_prob = log_probs[label].item()
    log_probs[label] = -1e8
    max_log_prob = log_probs.max()
```

Preprint. Under review.

```
    return max(label_log_prob - max_log_prob, 0)
```

1.2 Gradient and Hessian function

The gradient function calculates the gradient using the loss function to change the patch in the image. We use the loss function mentioned in the prior section to calculate the gradient.

For creating the small deviation, we create a matrix of zeros and fill the patch with the value of h = 0.0001. The following is the python implementation of the same:

```
def gradient(network, image, label, row, col):

    constant = torch.zeros(*image.shape).to(device)
    constant[0, 0, row:row+4, col:col+4] = 0.0001
    diff = loss_fn(network(image+constant), label) - loss_fn(network(image-constant)
    diff = diff/0.0002
    return diff.cpu()
```

The hessian function is a modification of the gradient function by changing the numerator and denominator.

```
def hessian(network, image, label, row, col):

    constant = torch.zeros(*image.shape).to(device)
    constant[0,0,row:row+5, col:col+5] = 0.0001
    diff = loss_fn(network(image+constant), label) + loss_fn(network(image-constant)
    diff = diff/0.0001**2
    return diff.cpu()
```

The hessian function is used in the Zoo-Newton method only. It adds a considerable delay due to the addition of multiple forwards passes through the model.

1.3 Newton's attack method

We select a step_size of 0.01 and use the loss function as the convergence condition. When the loss function becomes 0, we break out of the loop. This guarantees 100% success rate as every attack converges.

We select a random row and column and use that to find a patch for calculating the gradient and hessian value. If the hessian value is less than 0, we use the gradient as our update value. Otherwise, we divide the gradient with the hessian value and use that as our update step.

The following is the code implementation of newtons method.

```
def zoo_attack_newton(network, image, label):
    step_size = 0.01
    while loss_fn(network(image), label)>0:
        row = random.randint(0, 31-9)
        col = random.randint(0, 31-9)
        grad = gradient(network, image, label, row, col)
        hess = hessian(network, image, label, row, col)
        if hess<=0:
            delta = -step_size*grad
        else:
            delta = -step_size*grad/hess
        image[0, 0, row:row+4, col:col+4] = image[0, 0, row:row+4, col:col+4]
        + delta.to(device)

    return image
```

1.4 Adams attack method

The adam method uses the Adam optimizer to find the update step value. It uses coordinate-wise update values so we create a matrix of M and v and T, indexed by the row and column. We use the algorithm as described by the paper, except we change the coordinate-based descent to patch-based for faster convergence.

The following code implements adam's attack.

```
def zoo_attack_adam(network, image, label):
    betas = (0.9,0.999)
    eps = 1e-08
    step_size = 0.01
    M = torch.zeros(*image.shape[2:])
    v = torch.zeros(*image.shape[2:])
    T = torch.zeros(*image.shape[2:])
    delta=1e8
    last_image = torch.clone(image)
    logits = network(image)
    while loss_fn(logits, label)>0:

        row = random.randint(0, 31-9)
        col = random.randint(0, 31-9)
        grad = gradient(network, image, label, row, col)
        T[row:row+4, col:col+4]+=1

        M[row:row+4, col:col+4] = betas[0]*M[row:row+4, col:col+4]
        + (1-betas[0])*grad

        v[row:row+4, col:col+4] = betas[1]*v[row:row+4, col:col+4]
        + (1-betas[1])*grad**2

        M_hat = M[row:row+4, col:col+4]/(1-betas[0]**T[row:row+4, col:col+4])

        v_hat = v[row:row+4, col:col+4]/(1-betas[1]**T[row:row+4, col:col+4])

        delta = -step_size * M_hat/(v_hat**0.5 + eps)
        image[0, 0, row:row+4, col:col+4] = image[0, 0, row:row+4, col:col+4]
        + delta.to(device)
        torch.cuda.empty_cache()

    print()
    return image
```

2 Results

Both attacks make sure to converge due to our convergence criterion. Thus, every attack yields a 100% success rate. However, there is a significant time difference between both methods. Because the newton method has 3 more forward passes per iteration per image, it is slower than Adams. I use an 8-core CPU with 16GB ram to test both models. The adam attack on average ran for 1 minute (tested for 50 images). The newtons attack ran for 2min 37 secs. I could not test the model on a GPU because of memory issues.

Some images from adam's attack can be found in Figure 1.

The figures can be seen with some occlusion. The original paper also reports similar images. The Newton method yields the following images.

Figure 1: Attacked MNIST images using Adam

Figure 2: Attacked MNIST images using Newton

We see that newtons images are more occluded. However, it is easy for a human to identify that both sets of images have been occluded.

2.1 Discussion

The method we have implemented guarantees 100% success because every image converges such that the loss function is always 0. This means that it will always find an image where the label predicted is different from the true label.

One drawback of our method is that we cannot fix the number of iterations for all images. This is because if the loss function is greater than 0, it means that the model can still correctly identify the true label, meaning our attack was not successful. During execution, we measured upwards of 30000 steps on average for optimizing an image. This means that our method is not as optimal as that implemented by the authors. They use 1500 steps for every image and get almost 100% success.

3 Conclusion

The following assignment assesses the different methods used for ZOO attacks. We experiment with both Adam's and Newton's methods with patchwise coordinate descent to compare the results. We got a 100% success rate due to our convergence criterion but yielded significantly more optimization steps than the authors. The time for a successful attack ranged from 1 to 3 mins on CPU only. During our experiments, we found that newtons method took significantly longer than Adams's method (Can be seen in the author's results too). Thus, we successfully implemented the ZOO attack using Newton and Adams method.