FAST IS BETTER THAN FREE: REVISITING ADVERSARIAL TRAINING

ICLR 2020

论文主要成果概述

本文提出了一种加速对抗训练的方法：随机初始化的FGSM对抗训练

- 亮点：
  - 动机：Free对抗训练相比于普通的FGSM对抗训练而言具有鲁棒性的原因是什么？（主要区别在于Free会以某次迭代得到的扰动作为下次迭代的初始值）
  - 与多种训练的方法作比较：
    - PGD adversarial training；
    - Free FGSM adversarial training；
    - R+FGSM FROM TRAM `ER ET AL. (与本文方法较相似，但效果不好)。
  - 除随机初始化外，也也采用了循环学习率和混合精度等方法来加速训练；

PGD adversarial training

Algorithm 1 PGD adversarial training for T epochs, given some radius ϵ, adversarial step size α and N PGD steps and a dataset of size M for a network f_θ

```
for t = 1 ... T do
  for i = 1 ... M do
    // Perform PGD adversarial attack
    δ = 0 // or randomly initialized
    for j = 1 ... N do
      δ = δ + α · sign(∇_δ ℓ(f_θ(x_i + δ), y_i))
      δ = max(min(δ, ϵ), −ϵ)
    end for
    θ = θ − ∇_θ ℓ(f_θ(x_i + δ), y_i) // Update model weights with some optimizer, e.g. SGD
  end for
end for
```

for $t = 1 \ldots T$ **do**

  for $i = 1 \ldots M$ **do**

    // Perform PGD adversarial attack

    $\delta = 0$ *// or randomly initialized*

    for $j = 1 \ldots N$ **do**

      $\delta = \delta + \alpha \cdot \mathrm{sign}(\nabla_\delta \ell(f_\theta(x_i + \delta), y_i))$

      $\delta = \max(\min(\delta, \epsilon), -\epsilon)$

    end for

    $\theta = \theta - \nabla_\theta \ell(f_\theta(x_i + \delta), y_i)$ *// Update model weights with some optimizer, e.g. SGD*

  end for

end for

Free adversarial training

Algorithm 2 "Free" adversarial training for T epochs, given some radius ϵ, N minibatch replays, and a dataset of size M for a network f_θ

```
δ = 0
// Iterate T/N times to account for minibatch replays and run for T total epochs
for t = 1 ... T/N do
    for i = 1 ... M do
        // Perform simultaneous FGSM adversarial attack and model weight updates T times
        for j = 1 ... N do
            // Compute gradients for perturbation and model weights simultaneously
            ∇_δ, ∇_θ = ∇ℓ(f_θ(x_i + δ), y_i)
            δ = δ + ϵ · sign(∇_δ)
            δ = max(min(δ, ϵ), −ϵ)
            θ = θ − ∇_θ // Update model weights with some optimizer, e.g. SGD
        end for
    end for
end for
```

Fast adversarial training

Algorithm 3 FGSM adversarial training for T epochs, given some radius ϵ, N PGD steps, step size α, and a dataset of size M for a network f_θ

for $t = 1 \dots T$ **do**
  for $i = 1 \dots M$ **do**
    // Perform FGSM adversarial attack
    $\delta = \text{Uniform}(-\epsilon, \epsilon)$
    $\delta = \delta + \alpha \cdot \text{sign}(\nabla_\delta \ell(f_\theta(x_i + \delta), y_i))$
    $\delta = \max(\min(\delta, \epsilon), -\epsilon)$
    $\theta = \theta - \nabla_\theta \ell(f_\theta(x_i + \delta), y_i)$ *// Update model weights with some optimizer, e.g. SGD*
  end for
end for

R+FGSM FROM TRAM `ER ET AL.

While a randomized version of FGSM adversarial training was proposed by Tramèr et al. (2017), it was not shown to be as effective as adversarial training against a PGD adversary. Here, we note the two main differences between our approach and that of Tramèr et al. (2017).

1. The random initialization used is different. For a data point x, we initialize with the uniform distribution in the entire perturbation region with

$$x' = x + \text{Uniform}(-\epsilon, \epsilon).$$

In comparison, Tramèr et al. (2017) instead initialize on the surface of a hypercube with radius $\epsilon/2$ with

$$x' = x + \frac{\epsilon}{2}\text{Normal}(0, 1).$$

2. The step sizes used for the FGSM step are different. We use a full step size of $\alpha = \epsilon$, whereas Tramèr et al. (2017) use a step size of $\alpha = \epsilon/2$.

To study the effect of these two differences, we run all combinations of either initialization with either step size on MNIST. The results are summarized in Table 6.

R+FGSM与本文Fast方法对比

Table 6: Ablation study showing the performance of R+FGSM from Tramèr et al. (2017) and the various changes for the version of FGSM adversarial training done in this paper, over 10 random seeds.

Method	Step size	Initialization	Robust accuracy
R+FGSM (Tramèr et al., 2017)	0.15	Hypercube(0.15)	$34.58 \pm 36.06\%$
R+FGSM (+full step size)	0.30	Hypercube(0.15)	$26.53 \pm 32.48\%$
R+FGSM (+uniform init.)	0.15	Uniform(0.3)	$72.92 \pm 10.40\%$
Uniform + full (ours)	0.30	Uniform(0.3)	$86.21 \pm 00.75\%$

DAWNBench Improvements

Cyclic learning rates

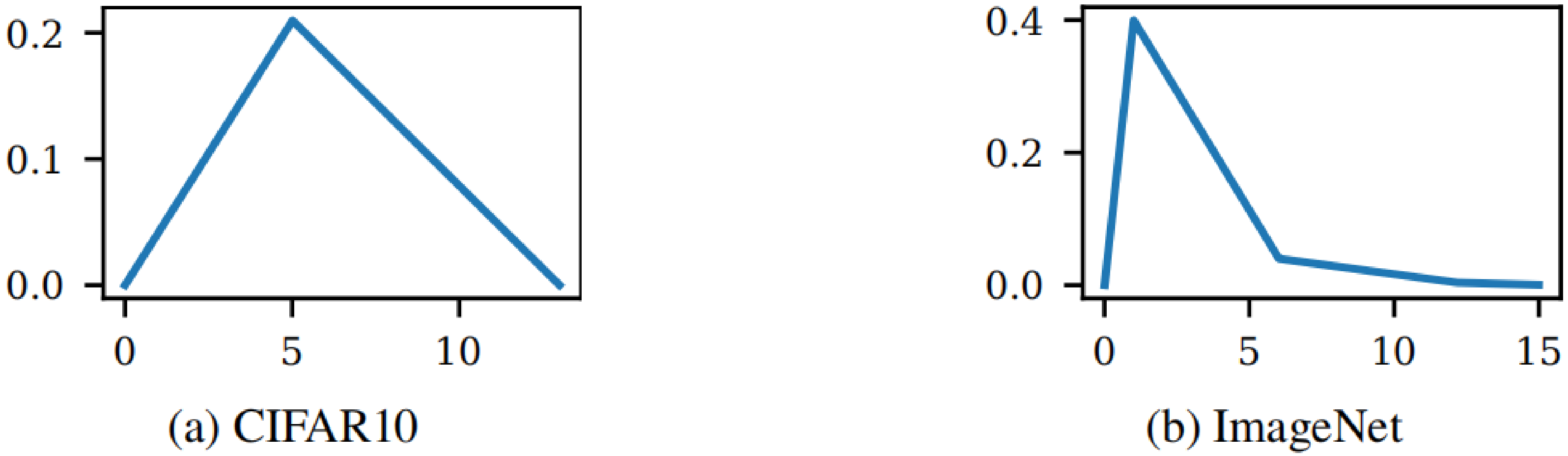

Figure 1: Cyclic learning rates used for FGSM adversarial training on CIFAR10 and ImageNet over epochs. The ImageNet cyclic schedule is decayed further by a factor of 10 in the second and third phases.

Mixed-precision arithmetic

used by Shafahi et al. (2019) but further strengthened with random restarts). Speedup with mixed-precision was incorporated with the Apex `amp` package at the `O1` optimization level for ImageNet experiments and `O2` without loss scaling for CIFAR10 experiments.[3]

Apex内的混合精度训练amp使用起来后，可以看到同样的数据，同样的batch size时，显存消耗减少到原来的60%，同时GPU-Util保持在较高值。在2080Ti的机器，batch size原来至多能达到12，使用apex.amp后可以达到24，效果显著。

DAWNBench Improvements 实验对比

Table 5: Time to train a robust ImageNet classifier using various fast adversarial training methods

Method	Precision	Epochs	Min/epoch	Total time (hrs)
FGSM (phase 1)	single	6	22.65	2.27
FGSM (phase 2)	single	6	65.97	6.60
FGSM (phase 3)	single	3	114.45	5.72
FGSM	single	15	-	14.59
Free ($m = 4$)	single	92	34.04	52.20
FGSM (phase 1)	mixed	6	20.07	2.01
FGSM (phase 2)	mixed	6	53.39	5.34
FGSM (phase 3)	mixed	3	95.93	4.80
FGSM	mixed	15	-	12.14
Free ($m = 4$)	mixed	92	25.28	38.76

Free FGSM、Fast FGSM、PGD结果对比

Table 1: Standard and robust performance of various adversarial training methods on CIFAR10 for $\epsilon = 8/255$ and their corresponding training times

Method	Standard accuracy	PGD ($\epsilon = 8/255$)	Time (min)
FGSM + DAWNBench			
+ zero init	85.18%	0.00%	12.37
+ early stopping	71.14%	38.86%	7.89
+ previous init	86.02%	42.37%	12.21
+ random init	85.32%	44.01%	12.33
+ $\alpha = 10/255$ step size	83.81%	46.06%	12.17
+ $\alpha = 16/255$ step size	86.05%	0.00%	12.06
+ early stopping	70.93%	40.38%	8.81
"Free" ($m = 8$) (Shafahi et al., 2019)[1]	85.96%	46.33%	785
+ DAWNBench	78.38%	46.18%	20.91
PGD-7 (Madry et al., 2017)[2]	87.30%	45.80%	4965.71
+ DAWNBench	82.46%	50.69%	68.8

END