A Meta Learning Approach for Adversarial Attacks

Shaik Mohammed Sayeed CS19BTECH11004
Gantasala Naga Aneesh Ajaroy CS19BTECH11010
Peddi Naga Hari Teja CS19BTECH11021
Vemulapalli Aditya CS19BTECH11025
Appanagari Sathwik Chakravarthi ES19BTECH11008

Abstract

Previously in the preliminary report we briefly wrote about adversarial attacks, few shot learning, meta learning and model agnostic metal learning read from different papers. In this mid-term report, we elaborated upon MAML and adversarial attacks. We tried to experiment on existing adversarial attacks and MAML models. We separately experimented on these models with an idea of integrating them at later point of time to reach our goal.

1. Adversarial Attacks

1.1 Introduction

With the explosion of data, known as big data, and increase in computation power, machine learning especially deep learning has advanced tremendously in the field of computer vision, NLP etc., At the same time, it is imperative that we can ensure the security and robustness of deep learning models.

One such vulnerability of security in deep learning models is "adversarial attacks". It includes adversarial examples, input images formed by applying small perturbations to examples from the dataset, such that the perturbed input results in the model outputting an incorrect answer with high probability. The perturbed image is the same as input image to the naked eye but the model misclassifies it as another.

1.2 Types

An adversarial attack can be targeted and untargeted. In targeted adversarial attack, the perturbation is applied to image to make the model predict an intended output decided before, where as in untargeted attack the image is perturbated just to make model misclassify the image to another lass with high probability

1.3 Adversarial perturbation

The noise that is added to the image to create the adversarial image is called adversarial perturbation.

Perturbation(noise) can be computed through one or multiple steps. One-step perturbation involves a single addition of noise, it is fast but is sometimes ineffective and perturbed image can be identified. Where as in multi-step, input is perturbed iteratively, it is hard to detect but is time and cost consuming.

1.4 Threat models

Models for adversarial attacks are classified into three types namely black-box, grey-box and white-box.

In black-box models the adversary creator didn't know the parameters and structure of neural net. He gives images and gets predictions which he uses as basis for creating surrogate model and then creating perturbation. It has the highest protection against adversarial attacks

In grey-box models, the structure of neural net is known but not weights. The same procedure for black-box is followed but due to additional info, it is more vulnerable to attacks

In white-box models, the full access to structure and weights is known and is most suspectable to attacks and hard to detect attacks.

The black and grey box models have many defenses proposed which are ineffective against white-box models.

Some notations

$x^{|}$, $y^{|}$ are adversarial image and its prediction
$||_p$ is Lp-norm
J (θ, $x^{|}$, $y^{|}$) is loss optimization and θ are weights
c is perturbation const

1.5 Adversarial attack algorithms

L – BFGS Algorithm

It works on the principle of minimizing the sum of Lp-norm between image and its perturbation, and optimization loss
Minimize
$c.|| x - x^{|} ||_p + J(\theta, x^{|}, y^{|})$

Fast gradient sign method
It is an one step iterative method where perturbation is updated along the gradient of loss optimization
$x^{|} = x + c.(\text{delta}(J(\theta, x, y))$ for untargeted
$x^{|} = x - c.(\text{delta}(J(\theta, x, y^{|}))$ for targeted

BIM and PGD algorithm
It optimizes the Fast gradient sign method by running finer iteratives of small size over T iterations
$x^{|}_{t+1} = x_t + a.(\text{delta}(J(\theta, x, y))$
a = T.c (very complex algo see papers in reference)

Some other algorithms are Momentum iterative attack algorithm, Carlini and wagner attack algorithm, Jacobian based map approach, Uniserval adverbial attack algorithm etc.

1.6 Adversarial defense algorithms

Denoising Ensemble
Here the perturbed image passes through denoising ensemble to recreate the original image. It consists of a set of auto encoders that try to remove specific noises in the image

Random noising
In this method a random self-ensemble is created which adds perturbation at each neural layer in both training and testing phases which stabilises the output of perturbed images over DNNs

2. Results replicated from literature

To get a better understanding of adversarial attacks, we tried to replicate the results from the paper using this github code.

On the MNIST dataset we tried to evaluate AdvGAN using generative adversarial networks in white-box and black-box setting. On different target models we used AdvGAN to generate adversarial examples. These examples are generated under an L_∞ bound of 0.3 on MNIST.

AdvGAN has advantages such as computational efficiency, performs faster than efficient FGSM etc over the white box and black box attacks. Along with white box attacks AdvGAN can be used to attack in semi-whitebox setting and black box setting with high success rate where as FGSM and other optimization methods can perform only in whitebox setting.

3. Model Agnostic Meta Learning (MAML)

3.1 Introduction

The goal of this approach is that we want to perform better on a new task with only using a few data samples for training without overfitting, given the model is trained on a variety of learning tasks.

Here a general meta learning algorithm is proposed, which can be applied to any of the learning problems (i.e., model agnostic) and model that is trained using gradient descent approach.

The model is easily suitable to different architectures and problem settings and a variety of loss functions.

Essentially, we want to learn an internal representation so that it can be suitable for many tasks so that slight changes of the parameters can produce good results.

3.2 Meta-Learning Problem Setup

The meta-learning problem treats the entire tasks as training examples.

Let model be f: inputs x -> output **a**
A generic version of learning task is

$$\mathcal{T} = \{\mathcal{L}(\mathbf{x}_1, \mathbf{a}_1, \ldots, \mathbf{x}_H, \mathbf{a}_H), q(\mathbf{x}_1), q(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{a}_t), H\}$$

$\mathcal{L}$ - Loss Function

$q(\mathbf{x}_1)$ - distribution over initial observations

$q(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{a}_t)$ - transition distribution

H - episode length

$H = 1$ for i.i.d supervised learning problems.

Samples of length H are generated by the model.

The loss term
$\mathcal{L}(\mathbf{x}_1, \mathbf{a}_1, \ldots, \mathbf{x}_H, \mathbf{a}_H) \rightarrow \mathbb{R}$, gives a task specific feedback

$p(\mathcal{T})$ - distribution of tasks our model should adopt

In the setting of K- shot learning, our model is trained to learn a new task $\mathcal{T}_i$ which is drawn from

$\mathcal{T} = \{\mathcal{L}(\mathbf{x}_1, \mathbf{a}_1, \ldots, \mathbf{x}_H, \mathbf{a}_H), q(\mathbf{x}_1), q(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{a}_t), H\}$ from only K samples drawn from q_i, $\mathcal{L}_{\mathcal{T}_i}$ - feedback or loss, and then it is tested on new samples from $\mathcal{T}_i$.

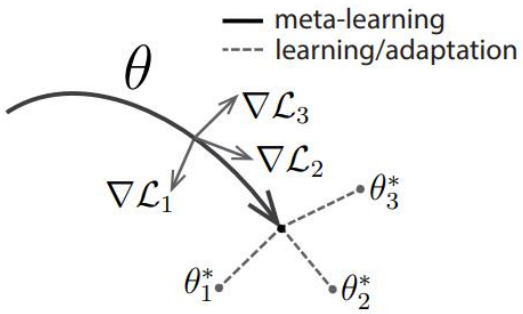
Test error changes using new data from q_i with respect to parameters is observed and used to improve the model f.

On effect, test error on sampled tasks $\mathcal{T}_i$ is used as a training error for the meta-learning process.

Meta - testing phase: After meta-training, new tasks are sampled from p(T), and meta-performance of the model is found by it's performance after learning from K samples.

3.3 A Model-Agnostic Meta-Learning Algorithm

Some internal representations are more transferrable than others is the intuition behind the approach we are taking. We want to find model parameters such that they are sensitive to task, so that by making small changes in the parameters we can huge improvements on the loss function of any task drawn from $p(\mathcal{T})$when altered in the direction of the gradient of that loss.



Assumptions
- loss function is smooth so that gradient-based learning techniques can be used

Let parameter vector be θ

Let the model be parametrized function - f_θ

The model parameters θ will change to θ_i' when the model is adapting to a new task.

Here θ_i' is computed using one or more gradient descent updates on task $\mathcal{T}_i$.

Example, one gradient update looks like

$$\theta_i' = \theta - \alpha\nabla_\theta\mathcal{L}_{\mathcal{T}_i}(f_\theta)$$

α - is the step size which can be either meta-learned or can be fixed as a hyperparameter.

The model parameters are trained by optimizing for the performance of $f_{\theta_i'}$ with respect to θ across tasks sampled from $p(\mathcal{T})$.

Meta - objective is

$$\min_\theta \sum_{\mathcal{T}_i\sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'}) = \sum_{\mathcal{T}_i\sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta-\alpha\nabla_\theta\mathcal{L}_{\mathcal{T}_i}(f_\theta)})$$

This method essentially tries to optimize model parameters so that using a small number of gradient steps on a new task we can get maximally effective behaviour on that task.

We use SGD for meta-optimization across tasks,

$$\theta \leftarrow \theta - \beta\nabla_\theta \sum_{\mathcal{T}_i\sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$$

β - meta step size

Generic Full Algorithm is

Require: $p(\mathcal{T})$: distribution over tasks
Require: α, β: step size hyperparameters
1: randomly initialize θ
2: **while** not done **do**
3: Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
4: **for all** $\mathcal{T}_i$ **do**
5: Evaluate $\nabla_\theta\mathcal{L}_{\mathcal{T}_i}(f_\theta)$ with respect to K examples
6: Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha\nabla_\theta\mathcal{L}_{\mathcal{T}_i}(f_\theta)$
7: **end for**
8: Update $\theta \leftarrow \theta - \beta\nabla_\theta \sum_{\mathcal{T}_i\sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$
9: **end while**

Gradient through a gradient is involved during meta-gradient update. Computationally, this requires an additional backward pass-through f to compute Hessian-vector products.

3.4 Meta-Learning Algorithm for Supervised Regression and Classification:

Assume Horizon H = 1 so the model takes single input and gives single output.

$\mathcal{T}_i$ generates K i.i.d. observations x from q_i .

For regression tasks using the mean-squared error(MSE) the loss function is

$$\mathcal{L}_{\mathcal{T}_i}(f_\phi) = \sum_{\mathbf{x}^{(j)}, \mathbf{y}^{(j)} \sim \mathcal{T}_i} \|f_\phi(\mathbf{x}^{(j)}) - \mathbf{y}^{(j)}\|_2^2$$

$\mathbf{x}^{(j)}, \mathbf{y}^{(j)}$ are an input/output pair which are sampled from the task $\mathcal{T}_i$.

K input/output pairs are given for learning for each task during K-shot regression tasks.

The cross-entropy loss for discrete classification tasks is

$$\mathcal{L}_{\mathcal{T}_i}(f_\phi) = \sum_{\mathbf{x}^{(j)}, \mathbf{y}^{(j)} \sim \mathcal{T}_i} \mathbf{y}^{(j)} \log f_\phi(\mathbf{x}^{(j)}) + (1 - \mathbf{y}^{(j)}) \log(1 - f_\phi(\mathbf{x}^{(j)}))$$

K-shot classification tasks use K input/output pairs from each class, so NK data points for N-way classification.

Algorithm for MAML for Few-Shot Supervised Learning

Require: $p(\mathcal{T})$: distribution over tasks
Require: α, β: step size hyperparameters
1: randomly initialize θ
2: **while** not done **do**
3: Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
4: **for all** $\mathcal{T}_i$ **do**
5: Sample K datapoints $\mathcal{D} = \{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}\}$ from $\mathcal{T}_i$
6: Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ using $\mathcal{D}$ and $\mathcal{L}_{\mathcal{T}_i}$ in Equation (2) or (3)
7: Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
8: Sample datapoints $\mathcal{D}_i' = \{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}\}$ from $\mathcal{T}_i$ for the meta-update
9: **end for**
10: Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$ using each $\mathcal{D}_i'$ and $\mathcal{L}_{\mathcal{T}_i}$ in Equation 2 or 3
11: **end while**

4. References

[1] https://arxiv.org/pdf/1412.6572.pdf
[2] https://arxiv.org/pdf/1712.09665.pdf
[3] https://arxiv.org/pdf/1801.02610.pdf
[4] https://arxiv.org/pdf/1703.03400.pdf
[5] https://proceedings.neurips.cc/paper/2020/file/cfee398643cbc3dc5eefc89334cacdc1-Paper.pdf