Robust Machine Learning

Soumik Sarkar
Associate Professor
Department of Mechanical Engineering
Department of Computer Science (courtesy)
Iowa State University

MDL
September 4, 2020
Adversarial attacks on Deep learning models

- Deep learning is becoming a tool of choice for many autonomous perception and decision-making tasks, e.g., in self-driving cars
- However, vulnerabilities of deep learning models are alarming!
- Certification requires clear understanding of the vulnerabilities

Synthetic attacks: FGSM and variants


Physical world attacks


- **Attacks** – white box vs. black box, test time vs. train time, mostly gradient based
- **Defenses** – adversarial training, robust optimization
Pixel-space imperceptible attacks

Neural networks are vulnerable to adversarial attacks which are basically the inputs that are almost not distinguishable from the input data on which the model has been trained, but are classified incorrectly.

The above figure shows that a panda which is classified as ‘panda’ correctly by 57% confidence by a deep neural network model, after the addition of some Fast Gradient Sign Attack is classified incorrectly as a ‘gibbon’ whereas it is still a panda.

Formulation of gradient based attacks

• One of the first methods is Fast Gradient Sign Method (FGSM) which can be written as:

\[ x = x + \varepsilon \text{sgn}(\nabla_x L(\theta, x, y)) \]

In which L is the loss function, \(\varepsilon\) is the \(L_\infty\) norm bound of the perturbations, \(x\) is the input data, \(y\) is the data label and \(\theta\)s are the weights in the neural network.

• One of the most popular attack formulation - projected gradient descent (PGD) approach – an iterative optimization-based variant of FGSM

\[ x = x + \text{rand} + lr \ast \text{sgn}(\nabla_x L(\theta, x, y)) \]

In which rand is a random variable between \(-\varepsilon\) and \(\varepsilon\), and lr is the step size.
In terms of how much access the attacker has to the model attacks can be categorized into 3 different classes:

- **White box attacks:**
  The attacker has access to the full model

- **Black box attack:**
  The attacker only has access to the query and doesn’t know the model

- **Grey box attack**
  The attacker knows the architecture, takes a virtual model and performs a white box attack

Study of black box attack shows the notion of transferability which means attacks can transfer from one model to another and still be capable of fooling the network.
In terms of where the attacks can happen, adversarial attacks are classified into two different classes:

- **Attacks in Training set:**
  
  Attackers inject malicious data into the training set to subvert the normal operation of deep neural networks.

- **Attacks in Test set:**
  
  Injecting the adversaries in the test set to break the trained network.
Attack categories

- In terms of whether the attackers target a specific model attacks can be classified in two different classes:

  ❖ Untargeted attacks:

  An untargeted adversary can be defined as $A(X, M) \rightarrow \bar{X}$, where $A(.)$ is the adversarial function, $X$ is the input image, $\bar{X}$ is the adversarial example, and $M$ is the target model. $A$ is considered successful if $M(X) \neq M(\bar{X})$. Recent studies in this area came up with universal attacks that could satisfy the above property.

  ❖ Targeted attacks:

  A targeted adversary can be defined as $A(X, M, l) \rightarrow \bar{X}$, where $l$ is an additional target label, and $A$ is only considered successful if $M(\bar{X}) = l$.

Defense strategy: Adversarial training

Train with both normal and adversarial examples

\[ L = \frac{1}{(m - k) + 2k} \left( \sum_{i \in \text{CLEAN}} L(x_i, y_i) + 2 \sum_{i \in \text{ADV}} L(x_i, y_i) \right) \]

\( m \rightarrow \) minibatch  \( k \rightarrow \) no. of adv. examples

\( \gamma \rightarrow \) weighting

\( \xi \sim N(\mu, \sigma) \)

If \( \epsilon \) is fixed then robust model only works for that \( \epsilon \)

https://arxiv.org/abs/1611.01236

http://www.dsrg.stuorg.iastate.edu/fall-2018-talks/
Defense strategy: Robust Optimization

Regular 0-th loss: \[ E_{x, y \sim D} \left[ L \left( f(x), y \right) \right] \]

Robust 0-th loss: \[ E_{x, y \sim D} \left[ \max_{x' \in P(x)} L \left( f(x'), y \right) \right] \]

\( P(x) \) is a pre-defined perturbation set

1. What's a good \( P \)?

2. How do we train this model?

\[
\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \sum_{x' \in P(x_i)} L(f_{\theta}(x'), y_i) \]

Defense strategy: Robust Optimization

\[ \begin{align*}
\text{outer min} & \quad \phi_{x,y}(\theta) = \max_{x' \in P(x)} L(f_{\theta}(x'), y) \\
\text{inner max} & \quad \nabla_{\theta} \phi_{x,y}(\theta) = \nabla_{\theta} L(f_{\theta}(x^*), y)
\end{align*} \]

Dana\'s trick: we can obtain a gradient of \( \phi \) w.r.t. \( \theta \) by finding a constrained maximizer \( x^* \)

\[ x^* \rightarrow \text{Find a good attack} \]

https://arxiv.org/abs/1706.06083
Defense strategy: Robust Optimization

Algorithm:
1. Sample a data point \( x, y \)
2. Compute \( x^* \) for the robust loss \( \Phi_{x,y}(6) \)
3. Compute gradient \( g = \nabla_x L(f_0(x^*), y) \)
4. Update \( \theta \) with \( g \) \( \text{Repeat} \)

Inner loop:\n\[ \text{PGD} \quad T_c \left( x + y \text{sign} (\nabla L(\theta, y)) \right) \]

Basic iter

https://arxiv.org/abs/1706.06083
Certified defenses


## Attackers are winning!

<table>
<thead>
<tr>
<th>Defense</th>
<th>Dataset</th>
<th>Distance</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buckman et al. (2018)</td>
<td>CIFAR</td>
<td>0.031 ($\ell_\infty$)</td>
<td>0%*</td>
</tr>
<tr>
<td>Ma et al. (2018)</td>
<td>CIFAR</td>
<td>0.031 ($\ell_\infty$)</td>
<td>5%</td>
</tr>
<tr>
<td>Guo et al. (2018)</td>
<td>ImageNet</td>
<td>0.005 ($\ell_2$)</td>
<td>0%*</td>
</tr>
<tr>
<td>Dhillon et al. (2018)</td>
<td>CIFAR</td>
<td>0.031 ($\ell_\infty$)</td>
<td>0%</td>
</tr>
<tr>
<td>Xie et al. (2018)</td>
<td>ImageNet</td>
<td>0.031 ($\ell_\infty$)</td>
<td>0%*</td>
</tr>
<tr>
<td>Song et al. (2018)</td>
<td>CIFAR</td>
<td>0.031 ($\ell_\infty$)</td>
<td>9%*</td>
</tr>
<tr>
<td>Samangouei et al. (2018)</td>
<td>MNIST</td>
<td>0.005 ($\ell_2$)</td>
<td>55%**</td>
</tr>
<tr>
<td>Madry et al. (2018)</td>
<td>CIFAR</td>
<td>0.031 ($\ell_\infty$)</td>
<td>47%</td>
</tr>
<tr>
<td>Na et al. (2018)</td>
<td>CIFAR</td>
<td>0.015 ($\ell_\infty$)</td>
<td>15%</td>
</tr>
</tbody>
</table>

Semantic (but perceptible) attacks/edge-cases

Tesla death smash probe: Neither driver nor autopilot saw the truck

Report shows driver took a hands-off approach to driving

By Gareth Corfield 20 Jun 2017 at 21:58

A car correctly classified in daylight but misclassified at night.

A female classified incorrectly as male by just adding glasses.

The wreckage of Joshua Brown’s Tesla Model S (Pic: US NTSB)
A novel algorithm leveraging generative models to generate semantic adversarial examples for deep classifiers

Empirical and theoretical analysis of such semantic examples

# Semantic data augmentation

## Day to night

<table>
<thead>
<tr>
<th>Test Dataset</th>
<th>Classes</th>
<th>$M_1$ (Benign Data)</th>
<th>$M_2$ (Orig. Dataset)</th>
<th>$M_3$ (Semantic Augmentation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Benign (Day)</td>
<td>Bus</td>
<td>12.00</td>
<td>12.4</td>
<td>11.82</td>
</tr>
<tr>
<td></td>
<td>Car</td>
<td>8.59</td>
<td>8.30</td>
<td>8.55</td>
</tr>
<tr>
<td></td>
<td>Truck</td>
<td>10.43</td>
<td>9.6</td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td>mAP (Overall)</td>
<td>10.34</td>
<td>10.1</td>
<td>10.14</td>
</tr>
<tr>
<td>Original (Night)</td>
<td>Bus</td>
<td>8.74</td>
<td>12.8</td>
<td>10.30</td>
</tr>
<tr>
<td></td>
<td>Car</td>
<td>8.4</td>
<td>9.1</td>
<td>8.5</td>
</tr>
<tr>
<td></td>
<td>Truck</td>
<td>6.6</td>
<td>8.55</td>
<td>8.0</td>
</tr>
<tr>
<td></td>
<td>mAP (Overall)</td>
<td>7.91</td>
<td>10.15</td>
<td>9.0</td>
</tr>
</tbody>
</table>
A few success stories of reinforcement learning

Kohl and Stone, 2004
Ng et al, 2004
Tedrake et al, 2005
Kober and Peters, 2009

Silver et al, 2014 (DPG)
Lillicrap et al, 2015 (DDPG)

Mnih et al 2013 (DQN)
Mnih et al, 2015 (A3C)

Schulman et al, 2016 (TRPO + GAE)

Levine*, Finn*, et al, 2016 (GPS)

Silver*, Huang*, et al, 2016 (AlphaGo)
Attacks on deep RL agents

$l_\infty$ attack

+ .001 × \[ \text{sign}(\nabla_x J(\theta, x, y)) \]

action taken: down
original input

= 

action taken: noop
adversarial input

$l_1$ attack

+ .441 × \[ \text{argmax}_{i} \nabla_x J(\theta, x, y)_i \]

action taken: up
original input

= 

action taken: down
adversarial input

Huang, S., Papernot, N., Goodfellow, I., Duan, Y., & Abbeel, P., *Adversarial attacks on neural network policies*. ICLR 2017
FGSM attack example

Huang, S., Papernot, N., Goodfellow, I., Duan, Y., & Abbeel, P., Adversarial attacks on neural network policies. ICLR 2017
Defending deep RL agents

- Meta-learning based supervisory framework for robust policy learning – Meta-Learned Advantage Hierarchy (MLAH) algorithm
- In the presence of unknown adversaries
- Provably improved performance over state-of-the-art strategies such as PPO
- **Key idea** – We detect an attack by monitoring the learning performance, then automatically spawn another sub-policy that protects the nominal sub-policy from corrupt observation as well as learns to cope with the attack and (partially) recover performance!

![MLAH framework diagram](image)

Learning to cope with attacks

Figure 4. Illustration of MLAH agent’s coping behaviour under symmetrical mirror attack about the $y$-axis. The MLAH agent learns to use a different sub-policy that maps the adversarial observation to an optimal action that leads it to the goal.
Learning to cope with attacks

Legend:
- Red: Goal
- Orange: Actual Observation
- Blue: Adversarial Observation
Learning to cope with attacks

Coping with mirror attack

Legend:
- Goal
- Actual Observation
- Adversarial Observation
Action space attacks

- **Motivation**
  - Deep RL demonstrated to be good controllers
  - Cyber-physical systems are susceptible to actuation attacks
  - Most literature have studied state space attacks but not action space attacks
  - Given a certain budget, how to best distribute attacks to systems with multiple actuators?

- **Threat model**
  - Access to RL agent’s action stream
  - Access to RL agent’s training environment
  - Knowledge of RL agent’s architecture (white-box attack)

Myopic action space (MAS) attacks

- **Formulation**

  \[
  \min_{\delta_t} \quad R_{\text{adv}}(\delta_t) = R(s_t, a_t + \delta_t) + \sum_{j=t+1}^{T} R(s_j, a_j),
  \]

  subject to

  \[
  \|\delta_t\|_p \leq b, \quad s_{j+1} = E(s_j, a_j), \quad a_j = \Theta(s_j) \text{ (for } j = t, \ldots, T)\]

- **Myopic adversary**

  - At each time step, a perturbation is designed to minimize the agent’s reward at the current time step
Myopic action space (MAS) attacks

**Algorithm 1: Myopic Action Space (MAS) Attack**

1. Initialize nominal environment, $E_{nom}$, nominal agent $\pi_{nom}$ with weights, $\theta$
2. Initialize budget $b$
3. **while** $t \leq T$ **do**
4. Compute adversarial action $\hat{a}_{t+\frac{1}{2}}$ using $\nabla R_{adv}$
5. Compute $\delta_t = \hat{a}_{t+\frac{1}{2}} - a_t$, project $\delta_t$ onto ball of size $b$ to get $\delta_t'$
6. Compute projected adversarial action $\hat{a}_t = a_t + \delta_t'$
7. Step through $E_{nom}$ with $\hat{a}_t$ to get next state

- Projection step on line 5 ensures crafted perturbation is within a budget and also represents the allocation of perturbations across different action dimensions (across multiple actuators)
Look-ahead action space (LAS) attacks

- **Formulation**
  
  \[
  \min_{\Delta} \quad R_{adv}(\Delta) = \sum_{j=t}^{t+H} R(s_j, a_j + \delta_j) + \sum_{j=t+1}^{T} R(s_j, a_j)
  \]

  subject to \(\|\Delta\|_{p,q} \leq B, \Delta = [\delta_t, \delta_{t+1}, \ldots, \delta_H], \ s_{j+1} = E(s_j, a_j), \ a_j = \Theta(s_j)\)

- **Non-myopic adversary**
  
  - At each time step, adversary takes the dynamics of the agent into account and crafts a perturbation that minimizes reward up to a certain horizon
Algorithm 2: Look-ahead Action Space (LAS) Attack

1. Initialize nominal and adversary environments \( E_{nom}, E_{adv} \) with same random seed
2. Initialize nominal agent \( \pi_{nom} \) weights, \( \theta \)
3. Initialize budget \( B \), adversary action buffer \( A_{adv} \), horizon \( H \)
4. while \( t \leq T \) do
5. Reset \( A_{adv} \)
6. if \( H = 0 \) then
7. Reset \( H \) and \( B \)
8. while \( k \leq H \) do
9. Compute adversarial action \( \hat{a}_{t+\frac{1}{2},k} \) using \( \nabla R_{adv} \)
10. Compute \( \delta_{t,k} = \hat{a}_{t+\frac{1}{2},k} - a_{t,k} \)
11. Append \( \delta_{t,k} \) to \( A_{adv} \)
12. Step through \( E_{adv} \) with \( a_{t,k} \) to get next state
13. Compute \( ||\delta_{t,k}||_{\ell_p} \) for each element in \( A_{adv} \)
14. Project sequence of \( ||\delta_{t,k}||_{\ell_p} \) in \( A_{adv} \) on to ball of size \( B \) to obtain look-ahead sequence of budgets \( [b_{t,k}, b_{t,k+1} \ldots b_{t,k+H}] \)
15. Project each \( \delta_{t,k} \) in \( A_{adv} \) on to look-ahead sequence of budgets computed in the previous step to get sequence \( [\delta'_{t,k}, \delta'_{t,k+1} \ldots \delta'_{t,k+H}] \)
16. Compute projected adversarial action \( \hat{a}_t = a_t + \delta'_{t,k} \)
17. Step through \( E_{nom} \) with \( \hat{a}_t \)
18. \( B \leftarrow \max(0, B - \delta'_{t,k}); H \leftarrow H - 1 \)
Experimental studies

- Trained PPO agent in OpenAI Lunar Lander environment
Example of LAS attack on PPO agent
Example of LAS attack on DDQN agent
Some recent trends – understanding sources of vulnerabilities

• The vulnerability comes from latent feature space, so focusing on the latent space with respect to their vulnerability.  https://arxiv.org/abs/1908.04355

• Adversarial examples are not bugs, they are features  https://arxiv.org/pdf/1905.02175.pdf

• Connections between robustness and interpretability - Focusing on the adversarial examples which put the interpretability of the model at risk  https://openreview.net/pdf?id=Hyes70EYDB

• Do models have strong bias towards a dataset rather than the underlying task?  https://arxiv.org/abs/2002.04108

• Focusing on designing network architectures which provides better numerical stability  https://proceedings.icml.cc/static/paper_files/icml/2020/381-Paper.pdf
Backup Slides
Deep Reinforcement Learning

state $s_t$

brain

action $a_t$

reward $r_t$

IOWA STATE UNIVERSITY
Attacks on Deep RL agents

Primarily test-time attacks

FGSM attack example

Huang, S., Papernot, N., Goodfellow, I., Duan, Y., & Abbeel, P., *Adversarial attacks on neural network policies*. ICLR 2017
Poisoning of DRL state observation during training

- Sampled perturbation size & direction from uniform distribution $U(a,b)$
- Perturb state where $S_{i, adversary} = S_i + U(a,b)$ such that $\max_i |S_{i, adversary} - S_i| \leq \epsilon_{attack}$
- Experimented with:
  - White Noise Attacks where $a = -b$
  - Bias Attacks where $a \neq b$ and $a < b$
Meta-learned Advantage Hierarchy

- An online meta-learning framework with a master policy and arbitrary number of sub-policies

- Master policies learns to select different sub-policies under different situations (nominal/adversarial) using expected advantages of sub-policies as observation

- Simultaneously, sub-policies learn the best policy to follow under different situations (nominal/adversarial)

MLAH Algorithm

**Algorithm 1: MLAH**

**Input**: $\pi_{nom}$ and $\pi_{adv}$ sub-policies parameterized by $\theta_{nom}$ and $\theta_{adv}$; Master policy $\pi_{master}$ with parameter vector $\phi$.

1. Initialize $\theta_{nom}$, $\theta_{adv}$, $\phi$
2. **for** pre-training iterations [optional] **do**
   3. Train $\pi_{nom}$ and $\theta_{nom}$ on only nominal experiences.
   **end**
3. **for** learning life-time **do**
4.  **for** Time steps $t$ to $t + T$ **do**
5.   Compute $A_t$ over sub-policies (see eq. 4)
6.   select sub-policy to take action with $\pi_{master}$ using $A_t$ as observations
7. **end**
8. Estimate all $A_{GAE}$ for $\pi_{nom}$, $\pi_{adv}$ over $T$
9. Estimate all $A_{GAE}$ for $\pi_{master}$ over $T$ with respect to $A_t$ observations
10. Optimize $\theta_{nom}$ based on experiences collected from $\pi_{nom}$
11. Optimize $\theta_{adv}$ based on experiences collected from $\pi_{adv}$
12. Optimize $\phi$ based on all experiences with respect to $A_t$ observations
13. **end**
**MLAH advantages and disadvantages**

- **Pros:**
  - Advantage as observation for master policy acts a useful metric for detecting the presence of adversary
  - Reduces bias in learned value function baseline in the presence of adversary

  **Intuition:** Policy learns to map different policies to different states rather than resolving a single policy over multiple latent states

- **Cons:**
  - Proposed framework has a delayed response
MLAH analysis

Monotonic improvement of reward during learning (similar result as TRPO)

- **Proposition 1**: \( \hat{R}(\pi_{new}) \geq \hat{L}_{\pi_{old}}(\pi_{new}) - \frac{4\epsilon \gamma \alpha^2}{(1 - \gamma)^2} \)
  - \( \hat{R}(\pi_{new}) \) denotes the actual expected discounted rewards as a function of the new policy
  - \( \hat{L}_{\pi_{old}}(\pi_{new}) = L_{\pi_{old}}(\pi_{new}) + \delta - \hat{\delta} \) where \( L_{\pi_{old}}(\pi_{new}) \) is the approximated expected reward as a function of the new policy
    - \( \delta \) denotes the observed bias of the state value
    - \( \hat{\delta} \) Denotes observed bias in the expected discounted reward
  - \( \alpha \) is the total variation divergence for between the old and new policy
  - \( \gamma \) is the discount factor

\[
\bar{\epsilon} = \begin{cases} 
  \max_{s,a} |\hat{A}_\pi(s,a)| + (\gamma - 1)\delta, & \text{if } \hat{A}_\pi(s,a) \geq (1 - \gamma)\delta. \\
  -\max_{s,a} |\hat{A}_\pi(s,a)| + (1 - \gamma)\delta, & \text{if } 0 < \hat{A}_\pi(s,a) < (1 - \gamma)\delta. \\
  \max_{s,a} |\hat{A}_\pi(s,a)| + (1 - \gamma)\delta, & \text{if } \hat{A}_\pi(s,a) \leq 0 
\end{cases}
\]

- The actual expected discounted reward as a function of the new policy is at least greater or equals to the approximation of the reward
- This implies that using the conditioned policy (MLAH agent), we can expect a better reward than the approximation of the reward
MLAH analysis

MLAH should perform better given a sufficiently intelligent attacker

- **Proposition 2**: If \( \Delta \hat{\delta} < C \Delta V \), where \( C \geq \frac{(m-n)(1-m)(4\gamma\alpha^2+1-\gamma)}{(1-m+n)(1-\gamma)} \) and \( \Delta V = V_0 - V_1 \), then the conditioned policy (MLAH agent) has a higher lower bound of expected discounted reward compared to that of the unconditioned policy (Classic RL agent)
  - \( \Delta \hat{\delta} = \hat{\delta}_{unc} - \hat{\delta}_{con|0} \)
  - \( V_0 \) denotes expected state value under nominal conditions
  - \( V_1 \) denotes expected state value under adversarial conditions

Main Takeaways:

- Analysis of MLAH shows that proposed framework reduces bias in value function baseline of agent under adversarial attack
- Consequently, reducing bias improves the lower bound of expected rewards
Meta-agent can reliably detect (after certain initial training period) the presence of an (intermittently occurring) adversary via observing sub-policy advantages.
MLAH empirical results: performance evaluation

- Trained on Inverted-Pendulum environment with Vanilla PPO and (oracle) MLAH under intermittent $l_\infty$ bounded-stochastic attack

- MLAH-trained agent performs significantly better during nominal evaluation (and training)

- **Insight:** Vanilla PPO with adversarial training makes the learned policy less efficient
Results show that training an RL-agent under MLAH framework generally returns a higher reward as compared to a Vanilla RL-agent (PPO) when under adversarial attacks.

### MLAH empirical results: performance evaluation

Performance comparison under different (temporal) attack profiles

Table 2: Performance evaluation of Oracle-MLAH

<table>
<thead>
<tr>
<th>$m/n$</th>
<th>Normalized avg. training return</th>
<th>Normalized avg. evaluation return</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Vanilla</td>
<td>Oracle-MLAH</td>
</tr>
<tr>
<td>1.0/−</td>
<td>0.96 ± 0.03</td>
<td>0.96 ± 0.03</td>
</tr>
<tr>
<td>0.995/0.005</td>
<td>0.238 ± 0.082</td>
<td>0.553 ± 0.242</td>
</tr>
<tr>
<td>0.95/0.05</td>
<td>0.612 ± 0.08</td>
<td>0.677 ± 0.149</td>
</tr>
<tr>
<td>0.8/0.2</td>
<td>0.613 ± 0.043</td>
<td>0.728 ± 0.063</td>
</tr>
<tr>
<td>0.5/0.5</td>
<td>0.749 ± 0.093</td>
<td>0.764 ± 0.078</td>
</tr>
</tbody>
</table>

Results show that training an RL-agent under MLAH framework generally returns a higher reward as compared to a Vanilla RL-agent (PPO) when under adversarial attacks.