Black-Box Adversarial Attack with Transferable Model-based Embedding

Yannis Karmim, Joseph Louët, Marc Treü

Sorbonne Université Sciences

TL;DR

Les principales contributions de l'article sont les suivantes:

- Créer un générateur pour la construction d'attaque black-box adverse.
- Tirer avantage de l'espace latent créé par le générateur pour réduire le nombre de requêtes minimales nécessaire pour tromper un classifieur.
- Modèle d'attaque adverse facilement transférable sur d'autres datasets.

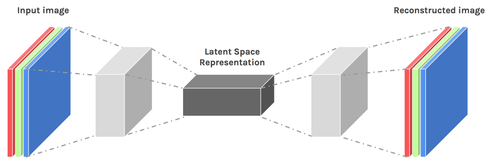
Introduction

L'utilisation des modèles de réseaux de neurones cause des préoccupations majeures de sécurité. Ces modèles sont connus pour être vulnérable aux attaques adversaires qui peuvent tromper ces réseaux pour donner de mauvaises prédictions. Ces méthodes d'attaques peuvent être divisées en **deux catégories**. La première si l'attaquant a accès aux paramètres du réseau (**white-box attack**). Et la seconde si l'attaquant n'y a pas accès (**black-box attack**). On peut essayer de tromper notre réseau en l'amenant vers une classe voulue (*targeted attack*), ou plus simplement vers n'importe quelle autre classe que le vrai label (*untargeted attack*). Les attaques black-box peuvent être elles même divisées en **deux catégories**. Les méthodes **transfert-based** utilise un modèle pré-entraîné puis génère des exemples adversaires sur ce dernier afin d'attaquer le modèle inconnu visé. Les méthodes **score-based** nécessite une loss oracle et l'interrogation multiple du réseau visé pour pouvoir estimer son gradient. L'attaquant peut ensuite appliquer les techniques d'attaque white-box avec un gradient approché. Les attaques transfert-based n'ont pas un taux de réussite élevé. Tandis que le taux de réussite des attaques score-based ont des performances proches des attaques white-box mais elles nécessitent un grand nombre de requêtes. L'idée est de regrouper l'avantage des méthodes **score-based** et **transfer-based**.

Modèle proposé

La méthode proposée **TREMBA** [1] se décompose en deux étapes :

- Entraîner un encodeur-décodeur qui génère efficacement des perturbations adversaires sur le réseau source avec une faible dimension d'espace latent.
- Rechercher une solution sur l'espace latent grâce aux méthodes NES [2].



Générateur

Considérons un classifieur F avec $x \in [0,1]^{dim(x)}$ et $F(x)$ la sortie de notre classifieur avant la couche de softmax. Notre objectif est de trouver une petite perturbation δ tel que $\arg\max F(x+\delta) \neq y$ ou $\arg\max F(x+\delta) = t$ pour une classe t donnée.
Soit $\mathcal{G}$ **le générateur** des perturbations divisé en deux parties : l'**encodeur** $\mathcal{E}$ et le **décodeur** $\mathcal{D}$
Pour une **attaque non visée**, nous entraînons le générateur en minimisant la loss suivante :

$$\mathcal{L}_{\text{untarget}}(x_i, y_i) = \max\left(F_s(\varepsilon \tanh(\mathcal{G}(x_i)) + x_i)_{y_i} - \max_{j \neq y_i} F_s(\varepsilon \tanh(\mathcal{G}(x_i)) + x_i)_j, -\kappa\right)$$

Pour une **attaque visée**, nous entraînons le générateur en minimisant la loss suivante :

$$\mathcal{L}_{\text{target}}(x_i, t) = \max\left(\max_{j \neq y_i} F_s(\varepsilon \tanh(\mathcal{G}(x_i)) + x_i)_j - F_s(\varepsilon \tanh(\mathcal{G}(x_i)) + x_i)_t, -\kappa\right)$$

Encodeur
ConvReLUBN(16,3,3)
ConvReLUBN(32,3,3)
MaxPool(2,2)
ConvReLUBN(64,3,3)
ConvReLUBN(64,3,3)
MaxPool(2,2)
ConvReLUBN(128,3,3)
ConvReLUBN(128,3,3)
MaxPool(2,2)
ConvReLUBN(32,3,3)
ConvReLUBN(8,3,3)
MaxPool(2,2)

Décodeur
ConvReLUBN(32,3,3)
DeconvReLUBN(64,3,3)
ConvReLUBN(128,3,3)
DeconvReLUBN(128,3,3)
ConvReLUBN(128,3,3)
DeconvReLUBN(64,3,3)
ConvReLUBN(32,3,3)
DeconvReLUBN(16,3,3)
ConvReLUBN(3,1,1)

Figure: Architecture du générateur pour ImageNet

NES

La méthode NES permet d'approcher le gradient d'un réseau dont on ne connaît pas les paramètres. Dans l'article on approche le gradient de la loss du réseau qu'on attaque par rapport à l'espace latent z construit dans le générateur, au lieu que ce soit sur l'espace des images x.

Algorithm 1: Attaque sur l'espace latent

Input: réseau visé F; image x et son label y ou la classe visée t; encodeur $\mathcal{E}$; decodeur $\mathcal{D}$; limite de perturbation ϵ

Output: perturbation δ

$z_0 = \mathcal{E}(x)$

for $t = 1$ *to* T **do**

  Sample $v_1, v_2, \ldots, v_b \sim \mathcal{N}(z_{t-1}, \sigma^2)$

  Calculate $\mathcal{L}_i = \mathcal{L}_{\text{untarget}}(x, y)$ or $\mathcal{L}_{\text{target}}(x, t)$

  Update

  $z_t = z_{t-1} - \frac{\eta}{b}\sum_{i=1}^{b} \mathcal{L}_i \nabla_{z_{t-1}} \log \mathcal{N}(\nu_i | z_{t-1}, \sigma^2)$

end

return $\delta = \epsilon \tanh \mathcal{D}(z_T)$

Résultats

Dans nos expérimentations, on a prit un générateur entraîné à attaquer des modèles neuronaux sur ImageNet, et on a évalué son efficacité sur un nouveau dataset **15Scene**.

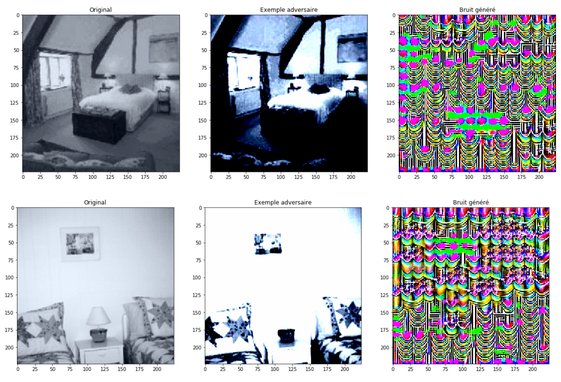


Figure: Exemples de perturbation généré

Dans la colonne de gauche l'image originale, celle du milieu l'exemple adversaire généré, et dans celle de droite, le bruit a l'origine de la perturbation.

On a fine-tuné un VGG16 sur **15Scene**, et on a appliqué la méthode **TREMBA** de façon black-box. On a limité à 500 requêtes, pour être comparable à ce qui avait été fait sur Google API.

Réseaux	Success	Queries	All Queries*
VGG16	71,5%	29.25	164.76

Table: Résultat

All Queries*: Si on compte aussi dans la moyenne des requêtes les échecs (qui ont donc atteint la limite de 500 requêtes)

Conclusion

- Méthode efficace de black-box attack, facilement transférable sur d'autres données.
- Article intéressant puisqu'il assemble deux différentes catégories d'attaques adverse.
- Modèle bien détaillé, les différentes étapes sont bien expliquées tant au niveau théorique que pour l'implémentation.

References

[1] Tong Zhang Zhichao Huang.
Black-box adversarial attack with transferable model-based embedding.
ICLR 2020, 2020.

[2] Tobias Glasmachers Yi Sun Jan Peters Daan Wierstra, Tom Schaul and Jürgen Schmidhuber.
Natural evolution strategies.
The Journal of Machine Learning Research, 2014.

SCIENCES SORBONNE UNIVERSITÉ

Code Source

https://github.com/marc-treu/AMAL_TREMA