Przykłady adwersarialne

Łukasz Pustelnik

2020/21

1 O projekcie

Projekt był realizowany we współpracy z GMUM, moim opiekunem był Przemysław Spurek, a w zespole byli: J. Tabor, P. Morawiecki, B. Wójcik. Projekt zestawia różne obrony adwersarialne przeciwko ataką typu „white-box".

2 Co to są przykłady adwersarialne?

Przykłady adwersarialne to dane wejściowe do modeli uczenia maszynowego, które są generowane, aby spowodować błąd modelu - intuicyjnie jest to coś rodzaju złudzenia optycznego dla systemów opartych na sztucznej inteligencji. Okazuje się, że bardzo łatwo można oszukać sieć neuronową, wystarczy do obrazu wejściowego dodać mały, aczkolwiek precyzyjnie wyliczony szum, który to spowoduje zmianę predykcji na błędną (Rysunek 1)

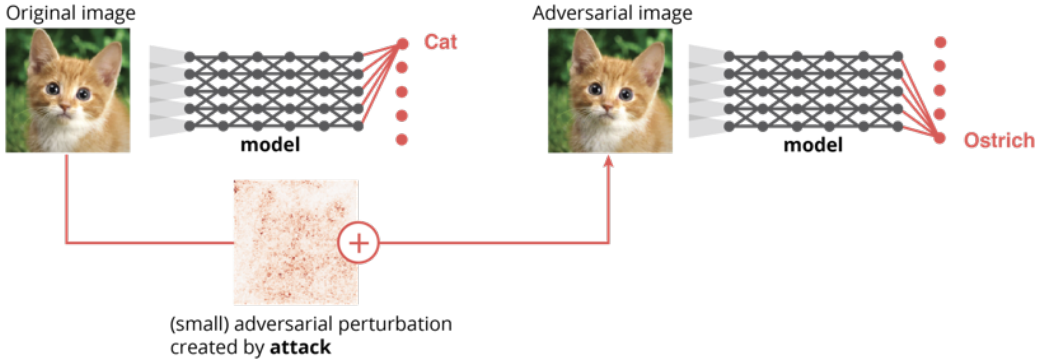


Figure 1: Przykład ataku adversarialnego. Po dodaniu szumu do obrazka poprawnie klasyfikowanego jako „kot" siec neuronowa zmienia predykcje na klasę „struś". Zauważmy, że ludzkie oko nie widzi różnicy między obrazkami, a sieć została oszukana

3 Ataki adwersarialne

3.1 Project Gradient Descent

Istnieje wiele różnych metod do tworzenia przykładów adwersarialnych, w tym projekcie korzystam metody Project Gradient Descent (PGD) jest to iteracyjny wariant ataku adwersarialnego FGSM. Metoda ta jest wolniejsza natomiast skuteczniejsza. PGD attack demonstration

$$x^0_{adv} = x$$
$$x^{t+1}_{adv} = \text{clip}_{[0,1]}\{x^t_{adv} + \alpha \ \text{sign}(\nabla_x J(x^t_{adv}, y_{true}))\}$$

4 Obrona Adwersarialna

4.1 Adversarial Training

Trening adwersarialny to intuicyjna metoda obrony, która ma na celu poprawę odporności sieci neuronowej poprzez włączenie przykładów adrwesarialnych do treningu. Formalnie jest to gra min - max, którą można

sformułować w następujący sposób:

$$\min_{\theta} \max_{D(x,x')<\eta} J(\theta, x', y)$$

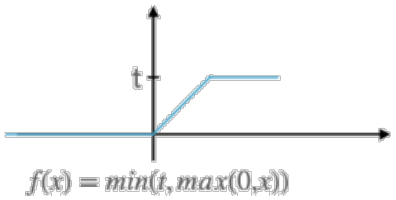
Trening Adwersariany ma też swoje wady np. obniża accuracy na „czystych" obrazkach

4.2 Dynamic Threshold Linear Layer

4.2.1 Pomysł

W przykładach adversarialnych często jest tak, że małe perturbacje kumulują się i w efekcie powodują zmianę predykcji na błędną klasę. Dlatego też chcąc ograniczyć ten efekt napisałem nową warstwę, która dla każdego neuronu:

1. Sortuje wartości z_i, wybiera 20% największych, które zsumowane dają parametr $t = \sum z_i$
2. Stosujemy spłaszczone ReLU, które spłaszcza również wartości większe niż wyliczony próg t



4.2.2 Rezultaty

DTL wyniki. Parametry ataku PGD to: $\epsilon = 0.3$, $\alpha = 0.1$ przy 30 iteracjach.
Eksperyment 1

- Base Model: 4 warstwy liniowe [784-256-128-64-10]
- DTL Model: 4 warstwy liniowe z dynamicznym progiem aktywacji [784-256-128-64-10]

	acccuracy	
model	clean image	pgd attack
Base model	96%	0%
DLT model	77%	8%

Obserwacje:

- base model całkowicie zmylony przez atak pgd
- stosując aktywacje z dynamicznym progiem 10% w modelu DTL 90% wag została wyzerowana, także model ten jest ciężko nauczyć ze względu na zanikający gradient stąd tylko 77% natomiast model ten uzyskał 8% odporności na atak

Eksperyment 2

- Base Conv Model: 2 warstwy konwolucyjne + 3 warstwy liniowe
- DTL Model: 2 warstwy konwolucyjne + 3 warstwy liniowe z dynamicznym progiem aktywacji

	acccuracy	
model	clean image	pgd attack
Base Conv model	99%	0%
DLT Conv model	89%	14%
Base Conv model AT	97%	86%
DLT Conv model AT	88%	79%

Obserwacje:

- base conv model całkowicie zmylony przez atak pgd, mimo dołożenia 2 warstw konwolucyjnych
- DLT conv model: dołożenie 2 warstw konwolucyjnych zwiększyło accuracy zarówno w przypadku czystych obrazków jak i adwersarialnych
- 14% modelu DTL na ataku pgd to wciąż mało, dlatego też dla obu modeli wykonałem trening adwersarialny i okazało się, że mimo obiecującego początku model DTL jest gorszy.

4.2.3 Podsumowanie

Pomysł z warstwą liniową z dynamicznym progiem aktywacji został porzucony, ze względu:

- na słabe wyniki, które najprawdopodobniej wynikają z zanikania gradientu, co więcej w większych modelach byłoby to jeszcze bardziej problematyczne.
- oraz duże koszta obliczeniowe. Próg jest wyliczany dla każdego neurona, tak więc ciężko byłoby to przełożyć na większe modele.

4.3 Sparse Neural Network

4.3.1 Pomysł

Pomysł na wykorzystanie sieci typu „sparse" do obrony adwersarialnego pochodzi od dobrego radzenia sobie z losowym szumem. [2]

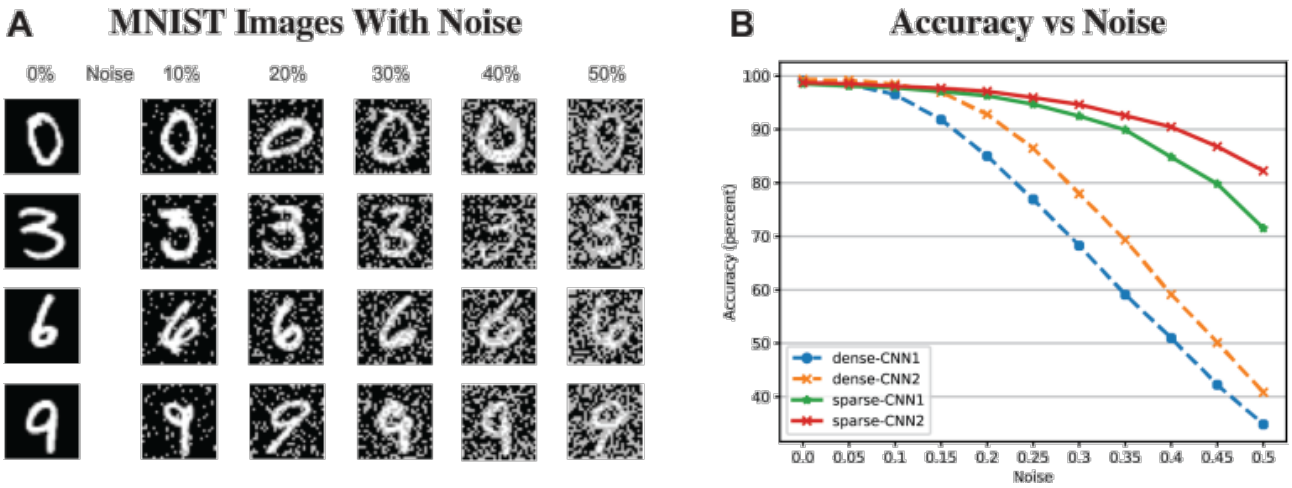


Figure 2: W przypadku sieci „sparse" losowy szumu nie ma tak dużego wpływy na accuracy modelu jak w przypadku zwykłych sieci

4.3.2 Rezultaty

SNN wyniki - notebook. Parametry ataku PGD to: $\epsilon = \frac{8}{255}$, $\alpha = \frac{2}{255}$ przy 20 iteracjach.
Modele:

- Klasyczy ResNet18
- Sparse ResNet18 to ResNet18 z podmienioną funkcją akwytacji na k-Winners z parameterm k=10%

	acccuracy	
model	clean image	pgd attack
ResNet18	92%	0%
Sparse ResNet18	90%	0%
ResNet18 AT	79%	47%
Sparse ResNet18 AT	67%	34%

Obserwacje:

- Niestety sieci typu „sparse" mimo dobrego radzenia sobie z losowym szumem nie dają rady w przypadku gdy szum dodany do obrazka jest precyzyjnie wyliczony przez atak adwersarialny. Model całkowicie zmylony tak jak i jego podstawowa wersja
- Co gorsza model „sparse" także słabiej wypada podczas treningu adwersarialnego

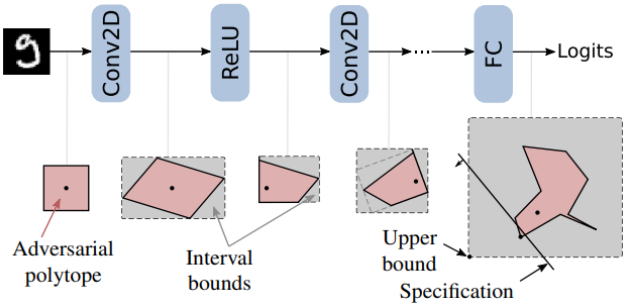
=

4.4 Interval Bound Propagation (IBP)

Reprodukcja pomysłu autorstwa DeepMind'u [1]

4.4.1 Pomysł

Zamykamy wejściowy obrazek w kostkę i uczymy sieć tak by cała kostka znalazła się po poprawnej stronie granicy decyzyjnej (Rysunek 4.4.1). W konsekwencji dodanie szumu mniejszego niż bok kostki nie spowoduje zmiany klasy, tak więc model będzie odporny na perturbacje adwersarialne. Rozwiązanie to działa bardzo dobrze, natomiast posiada ograniczenia wynikające ze sposobu uczenia.



4.4.2 Funkcja kosztu

W kontekście klasyfikacji pod wpływem niekorzystnych zaburzeń, rozwiązanie problemu optymalizacji dla każdej klasy docelowej $y \neq y_{true}$ daje zbiór logitów najgorszego przypadku, gdzie logit prawdziwej klasy jest równy jej dolnej granicy, a pozostałe są równe ich górnej granicy

$$z_{k,y}(\epsilon) = \begin{cases} \overline{z}_{k,y}(\epsilon) & y \neq y_{true} \\ \underline{z}_{k,y_{true}}(\epsilon) & \text{w.p.p} \end{cases}$$

Funkcja kosztu jest sumą dwóch składników, które są ważone parametrem κ

$$L = \kappa\, L_{fit} + (1 - \kappa)\, L_{spec}$$

gdzie:
L_{fit} to zwykła Cross-Entropy
L_{spec} to tak zwany „worst-case scenario”

4.4.3 Jak wygląda trening

Uczenie sieci interwałowych wymaga nowych warstw do propagacji oraz bardzo „powolnego” uczenia, ponieważ sieci neuronowe są bardzo wrażliwe na zmiany parametrów. Stąd dobrze jest zacząć trenowanie od parametru $\epsilon = 0$ i bardzo powoli go zwiększać, analogicznie dobrze jest zacząć od parametru $\kappa = 1$ i zmniejszać go powoli aż do $\frac{1}{2}$. Zwiększanie/zmniejszanie parametrów dla bardziej stabilnego treningu powinno odbywać się co batch.

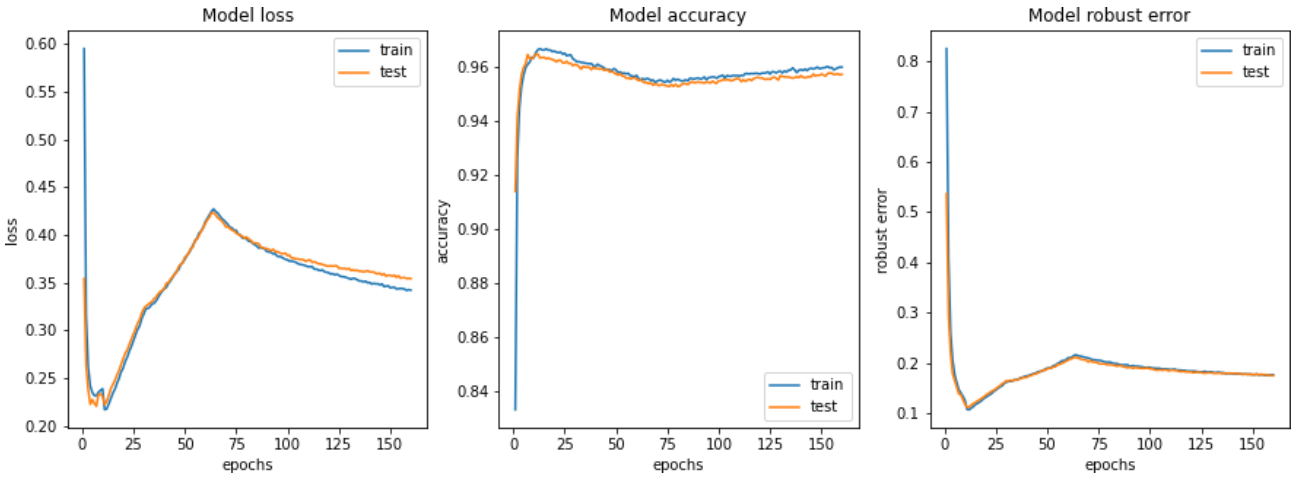


Figure 3: Trening IBP
. Wytrenowałem model, który ma odporność na perturbacje adwersarialne do $\epsilon = 0.3$

4.4.4 Rezultaty

IBP wyniki - notebook. Parametry ataku PGD to: $\epsilon = [0.2, 0.3, 0.4]$, $\alpha = 0.1$ przy 30 iteracjach.

	acccuracy			
model	clean image	$\epsilon = 0.2$	$\epsilon = 0.3$	$\epsilon = 0.4$
IBP model	95%	91%	89%	7%

Obserwacje:

- model zadziałał zgodnie z oczekiwaniami, został wyuczony do $\epsilon = 0.3$ i przy takim ataku uzyskał 89%
- zwiększając ϵ do 0.4 accuracy drastycznie spada do zaledwie 7%

4.4.5 Przykłady adwersarialne

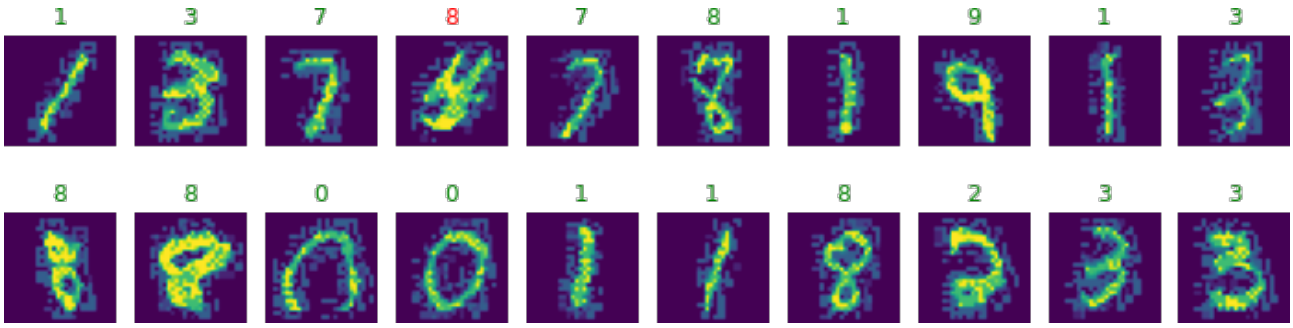


Figure 4: Atak PGD na model IBP z $\epsilon = 0.3$, $\alpha = 0.1$ przy 30 iteracjach

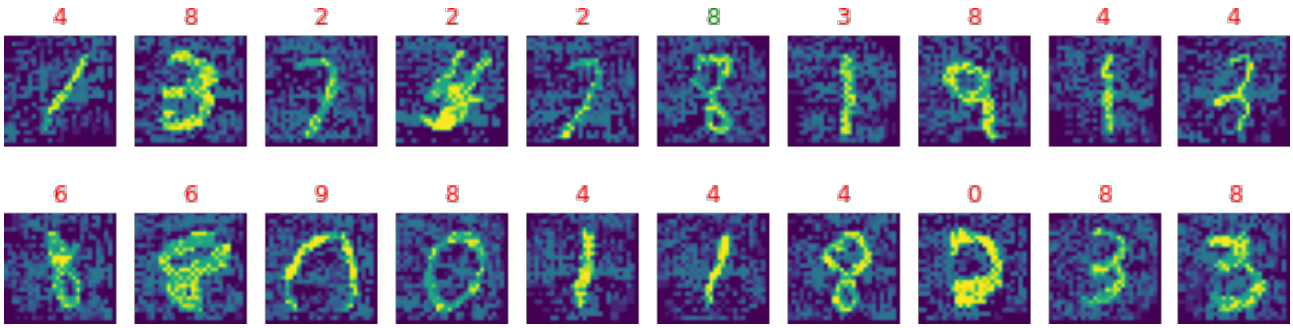


Figure 5: Atak PGD na model IBP z $\epsilon = 0.4$, $\alpha = 0.1$ przy 30 iteracjach

References

[1] Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovic, Timothy Mann, and Pushmeet Kohli. On the effectiveness of interval bound propagation for training verifiably robust models. *arXiv preprint arXiv:1810.12715*, 2018.

[2] Luiz Scheinkman Subutai Ahmad. How can we be so dense? the benefits of using highly sparse representations. 2019.