



ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE  
Fakulta jaderná a fyzikálně inženýrská



# **Moderní metody robustního strojového učení**

## **Modern methods of robust machine learning**

Výzkumný úkol

Autor: **Bc. Pavel Jakš**  
Vedoucí práce: **Mgr. Lukáš Adam, Ph.D.**  
Konzultant: **Mgr. Vojtěch Čermák**  
Akademický rok: 2022/2023

## ZADÁNÍ VÝZKUMNÉHO ÚKOLU

Student: Bc. Pavel Jakš  
Studijní program: Matematická informatika  
Název práce (česky): Moderní metody robustního strojového učení  
Název práce (anglicky): Modern methods of robust machine learning

Pokyny pro vypracování:

- 1) Nastudovat literaturu v oblasti metrik vizuální podobnosti.
- 2) Nastudovat literaturu v oblasti tvorby adversariálních vzorků.
- 3) Nastudovat dokumentaci k relevantním knihovnám robustního strojového učení (RobustBench, Foolbox).
- 4) Implementace vybraných metrik vizuální podobnosti.
- 5) Využití naimplementovaných metod vizuální podobnosti pro tvorbu adversariálních vzorku.

Doporučená literatura:

- 1) Naveed Akhtar, Ajmal Mian, Navid Kardan, Mubarak Shah, Advances in adversarial attacks and defenses in computer vision: A survey. IEEE Access 9, 2021, 155161-155196.
- 2) W., Eric, F. Schmidt, Z. Kolter, Wasserstein adversarial examples via projected sinkhorn iterations. International Conference on Machine Learning, PMLR, 2019.
- 3) J. Rauber, R. Zimmermann, M. Bethge, W. Brendel, Foolbox: A Python toolbox to benchmark the robustness of machine learning models. Reliable Machine Learning in the Wild Workshop, 34th International Conference on Machine Learning, 2017.

Jméno a pracoviště vedoucího výzkumného úkolu:

Mgr. Lukáš Adam, Ph.D.

Katedra počítačů, Fakulta elektrotechnická, České vysoké učení technické v Praze, Karlovo náměstí 13, 121 35 Praha 2

Jméno a pracoviště konzultanta:

Mgr. Vojtěch Čermák

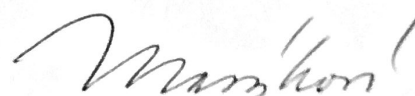
Katedra počítačů, Fakulta elektrotechnická, České vysoké učení technické v Praze, Karlovo náměstí 13, 121 35, Praha 2

Datum zadání výzkumného úkolu: 31.10.2022

Datum odevzdání výzkumného úkolu: 21.5.2023

Doba platnosti zadání je dva roky od data zadání.

V Praze dne 31. října 2022



vedoucí katedry

*Poděkování:*

Chtěl bych zde poděkovat především svému školiteli panu doktoru Adamovi za pečlivost, ochotu, vstřícnost a odborné i lidské zázemí při vedení mé diplomové práce. Dále děkuji svému konzultantovi panu magistru Čermákovi za jeho odborné rady.

*Čestné prohlášení:*

Prohlašuji, že jsem tuto práci vypracoval samostatně a uvedl jsem všechnu použitou literaturu.

V Praze dne 21. srpna 2023

Bc. Pavel Jakš

## Moderní metody robustního strojového učení

*Obor:* Matematická informatika

*Vedoucí práce:* Mgr. Lukáš Adam, Ph.D., Katedra počítačů, Fakulta elektrotechnická, České vysoké učení technické v Praze, Karlovo náměstí 13, 121 35 Praha 2.

*Konzultant:* Mgr. Vojtěch Čermák, Katedra počítačů, Fakulta elektrotechnická, České vysoké učení technické v Praze, Karlovo náměstí 13, 121 35 Praha 2.

[illegible]

**Klíčová slova:** klíčová slova (nebo výrazy) seřazená podle abecedy a oddělená čárkou

## Modern methods of robust machine learning

*Author:* Bc. Pavel Jakš

*Abstract:* Max. 10 lines of English abstract text. Max. 10 lines of English abstract text. Max. 10 lines  
of English abstract text. Max. 10 lines of English abstract text. Max. 10 lines of English abstract text.  
Max. 10 lines of English abstract text. Max. 10 lines of English abstract text. Max. 10 lines of English  
abstract text. Max. 10 lines of English abstract text. Max. 10 lines of English abstract text. Max. 10  
lines of English abstract text. Max. 10 lines of English abstract text. Max. 10 lines of English abstract  
text. Max. 10 lines of English abstract text. Max. 10 lines of English abstract text. Max. 10 lines of  
English abstract text. Max. 10 lines of English abstract text. Max. 10 lines of English abstract text.  
Max. 10 lines of English abstract text. Max. 10 lines of English abstract text. Max. 10 lines of English  
abstract text. Max. 10 lines of English abstract text. Max. 10 lines of English abstract text. Max. 10  
lines of English abstract text. Max. 10 lines of English abstract text.

**Key words:** keywords in alphabetical order separated by commas

# Obsah

<b>Úvod</b>	<b>7</b>
<b>1 Metriky vizuální podobnosti</b>	<b>8</b>
1.1 Metriky indukované $l_p$ normami . . . . .	8
1.2 MSE a RMSE . . . . .	9
1.3 Peak signal-to-noise ratio . . . . .	9
1.4 Wassersteinova vzdálenost . . . . .	9
1.5 Structural similarity index measure . . . . .	11
<b>2 Implementace metrik vizuální podobnosti</b>	<b>12</b>
2.1 Metriky založené na $l_p$ normách . . . . .	12
2.2 Modifikace Wassersteinovy vzdálenosti . . . . .	12
2.3 Structural dissimilarity . . . . .	14
<b>3 Adversariální vzorky a jejich tvorba</b>	<b>15</b>
3.1 Klasifikace v kontextu strojového učení . . . . .	15
3.2 Adversariální vzorky . . . . .	16
3.3 Tvorba adversariálních vzorků . . . . .	16
<b>4 Výsledky tvorby adversariálních vzorků</b>	<b>18</b>
4.1 Detaily implementace adversariálního útoku . . . . .	18
4.2 Srovnání útoků pro různé metriky . . . . .	19
<b>5 Robustnost neuronové sítě</b>	<b>23</b>
5.1 Přístup knihovny Foolbox . . . . .	23
5.2 Přístup knihovny RobustBench . . . . .	24
<b>Závěr</b>	<b>25</b>
<b>Literatura</b>	<b>26</b>
<b>Příloha</b>	<b>28</b>

# Úvod

Text úvodu....

# Kapitola 1

## Metriky vizuální podobnosti

Pod pojmem metrika na prostoru  $X$  si každý matematik představí zobrazení  $\rho : X \times X \rightarrow [0, +\infty)$  splňující

1.  $\rho(x, y) = 0 \iff x = y \quad \forall x, y \in X$ ,
2.  $\rho(x, y) = \rho(y, x) \quad \forall x, y \in X$ ,
3.  $\rho(x, z) \leq \rho(x, y) + \rho(y, z) \quad \forall x, y, z \in X$ .

Taková metrika může být na lineárním prostoru  $V$  nad číselným tělesem (pro naše účely zůstaňme nad  $\mathbb{R}$ ) snadno zadána pomocí normy, která je buď indukována skalárním součinem v případě pre-Hilbertových prostorů, nebo dána vlastnostmi, že se jedná o zobrazení  $\|\cdot\| : V \rightarrow [0, +\infty)$  a splňuje:

1.  $\|x\| = 0 \iff x = 0 \quad \forall x \in V$ ,
2.  $\|\alpha x\| = |\alpha| \cdot \|x\| \quad \forall \alpha \in \mathbb{R}, \forall x \in V$ ,
3.  $\|x + y\| \leq \|x\| + \|y\| \quad \forall x, y \in V$ .

Metriku potom získáme z normy následující konstrukcí:

$$\rho(x, y) = \|x - y\|,$$

tedy vzdálenost dvou vektorů je dána normou rozdílu vektorů. Snadno lze nahlédnout, že takto zadané zobrazení je metrika. S metrikami, které jsou tzv. indukované normami dle předchozího se setkáme.

### 1.1 Metriky indukované $l_p$ normami

Vzhledem k tomu, že obrázky, které jsou středem naší pozornosti, lze reprezentovat jako tenzory standardně o rozměrech  $C \times W \times H$ , kde  $C$  značí počet kanálů (nejčastěji kanály po řadě pro červenou, zelenou a modrou barvu),  $W$  označuje šířku a  $H$  výšku, tak lze na tyto tenzory vpustit  $L^p$  normy. Pro  $p \in [1, +\infty)$  je  $L^p$  norma z  $f \in L_p(X, \mu)$  definována vztahem:

$$\|f\|_p = \left( \int_X |f|^p d\mu \right)^{\frac{1}{p}}.$$



Pro naše obrázky lze za  $X$  vzít  $\{1, \dots, C\} \times \{1, \dots, W\} \times \{1, \dots, H\}$  a za  $\mu$  počítačící míru. Potom naše  $L^p$  norma přejde v  $l_p$  normu, která má pro naše obrázky, tedy tenzory  $x \in \mathbb{R}^{C \times W \times H}$ , tvar:

$$\|x\|_p = \left( \sum_{i=1}^C \sum_{j=1}^W \sum_{k=1}^H |x_{i,j,k}|^p \right)^{\frac{1}{p}}. \quad (1.1)$$

Trochu mimo stojí  $l_\infty$  norma, která má tvar pro tenzor  $x \in \mathbb{R}^{C \times W \times H}$ :

$$\|x\|_\infty = \max_{i \in \{1, \dots, C\}} \max_{j \in \{1, \dots, W\}} \max_{k \in \{1, \dots, H\}} |x_{i,j,k}|. \quad (1.2)$$

A úplně mimo stojí  $L_0$  norma, která svou povahou *není* norma ve smyslu výše uvedené definice, ale pro účely porovnávání obrázků se používá rozdíl obrázků v této pseudo-normě, proto ji zde zmiňuji:

$$\|x\|_0 = |\{x_{i,j,k} \neq 0\}|. \quad (1.3)$$

## 1.2 MSE a RMSE

Vzdálenosti, které mají blízko k metrikám indukovaným  $l_2$  normou, jsou *MSE* (z anglického *Mean Squared Error*) a *RMSE* (z anglického *Root Mean Squared Error*). Pro tenzory  $x, \tilde{x} \in \mathbb{R}^{C \times W \times H}$  mají definici:

$$\text{MSE}(x, \tilde{x}) = \frac{1}{CWH} \sum_{i=1}^C \sum_{j=1}^W \sum_{k=1}^H |x_{i,j,k} - \tilde{x}_{i,j,k}|^2 \quad (1.4)$$

$$\text{RMSE}(x, \tilde{x}) = \left( \frac{1}{CWH} \sum_{i=1}^C \sum_{j=1}^W \sum_{k=1}^H |x_{i,j,k} - \tilde{x}_{i,j,k}|^2 \right)^{\frac{1}{2}} \quad (1.5)$$

## 1.3 Peak signal-to-noise ratio

Vzdálenost označená zkratkou *PSNR* z anglického *peak signal-to-noise ratio* vyjadřuje vztah mezi obrázkem  $x \in \mathbb{R}^{C \times W \times H}$  a jeho pokažením  $\tilde{x} \in \mathbb{R}^{C \times W \times H}$  za přidání šumu. Definice je následující:

$$\text{PSNR}(x, \tilde{x}) = 10 \cdot \log_{10} \left( \frac{l^2}{\text{MSE}(x, \tilde{x})} \right), \quad (1.6)$$

$$= 20 \cdot \log_{10} \left( \frac{l}{\text{RMSE}(x, \tilde{x})} \right), \quad (1.7)$$

kde  $l$  je dynamický rozsah obrázků, tedy rozdíl mezi maximální možnou hodnotou pixelů a minimální možnou hodnotou pixelů. Jedná se tedy o transformaci metriky *MSE*. Samotná hodnota *PSNR* ovšem není metrická vzdálenost. Vždyť budou-li se obrázky  $x$  a  $\tilde{x}$  blížit k sobě, hodnota *PSNR*( $x, \tilde{x}$ ) poroste do nekonečna.

## 1.4 Wassersteinova vzdálenost

Bud'  $(M, d)$  metrický prostor, který je zároveň *Radonův*. Zvolme  $p \in [1, +\infty)$ . Potom máme *Wassersteinovu  $p$ -vzdálenost* mezi dvěma pravděpodobnostními mírami  $\mu$  a  $\nu$  na  $M$ , které mají konečné  $p$ -té

momenty, jako:

$$W_p(\mu, \nu) = \left( \inf_{\gamma \in \Gamma(\mu, \nu)} \mathbb{E}_{(x,y) \sim \gamma} d(x, y)^p \right)^{\frac{1}{p}}, \quad (1.8)$$

kde  $\Gamma(\mu, \nu)$  je množina všech sdružených pravděpodobnostních měr na  $M \times M$ , které mají po řadě  $\mu$  a  $\nu$  za marginální pravděpodobnostní míry [5].

Jak to souvisí s obrázky? Přes dopravní problém. Pod pravděpodobnostní distribucí  $\mu$  či  $\nu$  na  $X$  si lze představit rozložení jakési hmoty o celkové hmotnosti 1. Sdružená rozdělení  $\gamma \in \Gamma(\mu, \nu)$  potom odpovídají transportnímu plánu, kde  $\gamma(x, y)$  d  $x$  d  $y$  vyjadřuje, kolik hmoty se přesune z  $x$  do  $y$ . Tomu lze přiřadit nějakou cenu  $c$ , totiž kolik stojí přesun jednotkové hmoty z  $x$  do  $y$ :  $c(x, y)$ . V případě *Wassersteinovy vzdálenosti* za cenu dosadíme  $c(x, y) = d(x, y)^p$ , tedy  $p$ -tou mocninu vzdálenosti mezi  $x$  a  $y$ . Potom cena celkového dopravního problému s transportním plánem  $\gamma$  bude:

$$c_\gamma = \int c(x, y) \gamma(x, y) \, d x \, d y \quad (1.9)$$

$$= \int c(x, y) \, d \gamma(x, y) \quad (1.10)$$

a optimální cena bude:

$$c = \inf_{\gamma \in \Gamma(\mu, \nu)} c_\gamma. \quad (1.11)$$

Po dosazení:

$$c = \inf_{\gamma \in \Gamma(\mu, \nu)} \int c(x, y) \, d \gamma(x, y) \quad (1.12)$$

$$= \inf_{\gamma \in \Gamma(\mu, \nu)} \int c(x, y) \gamma(x, y) \, d x \, d y \quad (1.13)$$

$$= \inf_{\gamma \in \Gamma(\mu, \nu)} \mathbb{E}_{(x,y) \sim \gamma} c(x, y) \quad (1.14)$$

$$= \inf_{\gamma \in \Gamma(\mu, \nu)} \mathbb{E}_{(x,y) \sim \gamma} d(x, y)^p \quad (1.15)$$

$$= W_p(\mu, \nu)^p \quad (1.16)$$

Dostáváme tedy interpretaci, že  $p$ -tá mocnina *Wassersteinovy vzdálenosti* odpovídá ceně dopravního problému.

Pro obrázky má tato konstrukce následující uplatnění: Obrázky je třeba chápat jako diskrétní pravděpodobnostní rozdělení, proto je třeba je normalizovat, aby součet prvků tenzoru obrázku byl roven 1. Pak střední hodnota v definici *Wassersteinovy vzdálenosti* přejde ve váženou sumu cen, tedy  $p$ -tých mocnin vzdáleností mezi jednotlivými pixely.

Jak je to barevnými obrázky, tedy s obrázkem, které mají více než jeden kanál? Zde lze uplatnit následující dva přístupy:

1. Normovat celý obrázek na jedničku, tedy všechny kanály dohromady, a tím pádem i definovat vzdálenost mezi jednotlivými kanály,
2. Normovat každý kanál zvlášť na jedničku, počítat *Wassersteinovu metriku* pro každý kanál zvlášť a následně vybrat nějakou statistiku výsledných vzdáleností, např. průměr.

## 1.5 Structural similarity index measure

Zkratka *SSIM* pochází z anglického *structural similarity index measure*. Tato metrika se při výpočtu indexu dvou obrázků  $x$  a  $\tilde{x}$  dívá na podokna, ze kterých vybere jisté statistiky a z nich vytvoří index pro daná podokna obrázků. Potom se jako celkový index bere průměr přes tato okna. Uved' me vzorce pro výpočet indexu SSIM pro případ, že máme jediné okno, které splývá s obrázkem, které pro jednoduchost zvolme jednokanálové, tedy černobílé. Označme  $N = W \times H$  počet pixelů v obrázku a indexujme prvky matice obrázku jediným číslem. Potom definujeme pro obrázky  $x$  a  $\tilde{x}$  následující:

$$\begin{aligned}\mu_x &= \frac{1}{N} \sum_{i=1}^N x_i, \\ \mu_{\tilde{x}} &= \frac{1}{N} \sum_{i=1}^N \tilde{x}_i, \\ \sigma_x^2 &= \frac{1}{N-1} \sum_{i=1}^N (x_i - \mu_x)^2, \\ \sigma_{\tilde{x}}^2 &= \frac{1}{N-1} \sum_{i=1}^N (\tilde{x}_i - \mu_{\tilde{x}})^2, \\ \sigma_{x\tilde{x}} &= \frac{1}{N-1} \sum_{i=1}^N (x_i - \mu_x)(\tilde{x}_i - \mu_{\tilde{x}}).\end{aligned}$$

Potom:

$$\text{SSIM}(x, \tilde{x}) = \frac{(2\mu_x\mu_{\tilde{x}} + C_1)(2\sigma_{x\tilde{x}} + C_2)}{(\mu_x^2 + \mu_{\tilde{x}}^2 + C_1)(\sigma_x^2 + \sigma_{\tilde{x}}^2 + C_2)}, \quad (1.17)$$

kde  $C_1, C_2$  jsou konstanty pro stabilitu dělení volené kvadraticky úměrně dynamickému rozsahu. Můžeme si povšimnout, že  $\text{SSIM}(x, \tilde{x})$  není metrická vzdálenost. Budou-li obrázky stejné, nevyjde 0, nýbrž 1. Může se také stát, že SSIM vrátí zápornou hodnotu, která může vzniknout členem  $\sigma_{x\tilde{x}}$ . Jak volíme celkový SSIM pro barevné obrázky? Jako průměr přes kanály.

## Kapitola 2

# Implementace metrik vizuální podobnosti

V minulé kapitole jsme viděli přehled metod, jak přistoupit k porovnávání dvou různých obrázků. Předvedli jsme, jak vyčíslit rozdíl mezi dvěma obrázky. Ne vždy se ovšem jedná o metriku vesmýslu matematickém, což pro tvorbu adversariálních vzorků je záhodno, a ne vždy lze takovou vzdálenost přímočaře spočítat. Proto uvedme, je-li to nutné, příslušné úkroky stranou, které nám umožní hledat adversariální vzorky, a to pokud možno v krátkém čase. Samotnou výslednou implementaci v podobě kódu v jazyce python lze nalézt v příloze v části týkající se souboru *metrics.py*.

### 2.1 Metriky založené na $l_p$ normách

Implementovat klasické  $l_p$  normy je snadné, a tedy i metriky jimi indukované. MSE a RMSE jsou též snadné na implementaci. Vlastně i PSNR. Metriku vizuální podobnosti PSNR je třeba ovšem ošetřit, neboť, jak již bylo poznamenáno, budou-li se obrázky  $x$  a  $\tilde{x}$  blížit k sobě, hodnota  $\text{PSNR}(x, \tilde{x})$  poroste do nekonečna. Proto zkusme vzít konstrukci, kde prohodíme roli dynamického rozsahu  $l$  (peak signal) s rolí šumu (noise), dostaneme tedy, co lze nazvat noise to peak signal ratio (NPSR):

$$\text{NPSR}(x, \tilde{x}) = 20 \cdot \log_{10} \left( \frac{\text{RMSE}(x, \tilde{x})}{l} \right), \quad (2.1)$$

$$= -\text{PSNR}(x, \tilde{x}). \quad (2.2)$$

Při dvou obrázcích blížících se k sobě bude tedy NPSR klesat, a to neomezeně.

### 2.2 Modifikace Wassersteinovy vzdálenosti

Abychom mohli s Wassersteinovou metrikou nakládat například v počítači, je nutné tuto metriku spočítat. Podíváme-li se do definice, znamená to vyřešit optimalizační problém. Byť bychom se omezili hledání vzdáleností dvou vektorů o rozměru  $q$ , měli bychom problém s časovou složitostí nejlépe  $O(q^3 \log q)$  [6]. A to je hodně. Proto se podívejme, jak Wassersteinovu vzdálenost spočítat rychleji, byť za ztráty přesnosti.

Omezme se na prostory konečné dimenze. Potom mějme za úkol spočítat Wassersteinovu (zvolme  $p = 1$ ) vzdálenost vektorů  $\mu, \nu \in \mathbb{R}^q, \mu^T \mathbf{1}_q = \nu^T \mathbf{1}_q = 1$ , kde  $\mathbf{1}_q$  je vektor rozměru  $q$  složený pouze z jedniček. Potom  $\mu, \nu$  lze chápat jako diskrétní pravděpodobnostní rozdělení. Označme jako  $U(\mu, \nu)$  množinu všech matic  $P \in \mathbb{R}^{q \times q}, P_{i,j} \geq 0$  takových, že  $P \mathbf{1}_q = \mu$  a  $P^T \mathbf{1}_q = \nu$ . Jako matici  $C$  označme zadanou matici cen, která splňuje, že reprezentuje metriku. To znamená, že  $C_{i,j} \geq 0, C_{i,j} = 0 \iff i = j$ ,

$C_{i,j} = C_{j,i}$  a  $C_{i,k} \leq C_{i,j} + C_{j,k}$ . Potom lze napsat:

$$W(\mu, \nu) \equiv W_1(\mu, \nu) = \min_{P \in U(\mu, \nu)} \langle P, C \rangle, \quad (2.3)$$

kde  $\langle P, C \rangle = \sum_{i,j=1}^q P_{i,j} C_{i,j}$ .

Přejdeme nyní od Wassersteinovy metriky k tzv. duální Sinkhornově metrice. Ta je pro pevně zvolené  $\lambda > 0$  definována následovně:

$$W^\lambda(\mu, \nu) = \langle P^\lambda, C \rangle, \quad (2.4)$$

$$\text{kde } P^\lambda = \operatorname{argmin}_{P \in U(\mu, \nu)} \langle P, C \rangle - \frac{1}{\lambda} H(P), \quad (2.5)$$

kde  $H(P)$  je entropie pravděpodobnostního rozdělení  $P$ , tedy

$$H(P) = - \sum_{i,j=1}^q P_{i,j} \log(P_{i,j}).$$

Jedná se tedy o regularizovaný dopravní problém. Tato úprava Wassersteinovy metriky je, jak se přesvědčíme, mnohem lépe vyčíslitelná. Nejdříve se ovšem podívejme na intuici za touto úpravou.

Začneme s mírnou úpravou původního optimalizačního problému definujícího Wassersteinovu vzdálenost: Pro  $\alpha > 0$  definujme jakési  $\alpha$  okolí rozdělení  $\mu\nu^T$  (sdružené pravděpodobnostní rozdělení s marginálními  $\mu$  a  $\nu$ , kde  $\mu$  a  $\nu$  jsou nezávislá rozdělení) ve smyslu *Kullback-Leiblerovy divergence*

$$U_\alpha(\mu, \nu) = \{P \in U(\mu, \nu) | KL(P || \mu\nu^T) \leq \alpha\}. \quad (2.6)$$

Připomeňme definici Kullback-Leiblerovy divergence:

$$KL(\tilde{P} || \hat{P}) = \sum_{i,j=1}^q P_{i,j} \log \frac{P_{i,j}}{Q_{i,j}}.$$

Pro dané  $P \in U(\mu, \nu)$  lze na kvantitu  $KL(P || \mu\nu^T)$  nahlédnout jako na informaci mezi veličinami s rozděleními  $\mu$  a  $\nu$ . Tedy  $U_\alpha(\mu, \nu)$  vybírá ta rozdělení, která nesou malou vzájemnou informaci mezi  $\mu$  a  $\nu$  (ve smyslu menší než  $\alpha$ ). Dle [6] lze tuto úpravu ospravedlnit pomocí *principu maximální entropie*.

Potom lze definovat následující Sinkhornovu metriku:

$$W^\alpha(\mu, \nu) = \min_{P \in U_\alpha(\mu, \nu)} \langle P, M \rangle. \quad (2.7)$$

Jaký je vztah mezi Sinkhornovou metrikou  $W^\alpha$  a duální Sinkhornovou metrikou  $W^\lambda$ ? Přes téma duality matematického programování. Zatímco ve  $W^\alpha$  figuruje parametr  $\alpha$  v omezení definičního oboru, kde optimalizujeme, tak ve  $W^\lambda$  figuruje parametr  $\lambda$  jako Lagrangeův multiplikátor příslušné vazby.

Článek [6] poskytuje též nahlédnutí na fakt, že  $W^\lambda$  a  $W^\alpha$  jsou skutečně metriky.

Tento úrok stranou pomocí entropické regularizace původního problému lineárního programování, jehož vyřešení je nutné pro výpočet Wassersteinovy vzdálenosti, poskytuje úlevu v oblasti časové složitosti pro výpočet.

Konečný numerický algoritmus pro výpočet duální Sinkhornovy metriky potom vypadá následovně: Na vstupu algoritmus dostává pravděpodobnostní rozdělení  $\mu$  a  $\nu$ , jejichž vzdálenost je hledaná, dále matici  $C$  a regularizační parametr  $\lambda$ .

1.  $I = \mu > 0$  - tj. do proměnné  $I$  uložíme indexy, kde rozdělení  $\mu$  je nenulové.

2.  $\tilde{\mu} = \mu[I]$  - do proměnné  $\tilde{\mu}$  uložíme právě nenulové prvky  $\mu$ .
3.  $\tilde{C} = C[I, :]$  - do proměnné  $\tilde{C}$  uložíme příslušné řádky matice cen.
4.  $K = \exp(-\lambda * \tilde{C})$  - jako matici  $K$  vezmeme matici, která vznikne po prvcích jako exponenciála matice  $-\lambda M$ .
5.  $u = \text{ones}(\text{len}(\tilde{\mu})) / \text{len}(\tilde{\mu})$  - do proměnné  $u$  uložíme rovnoměrné rozdělení délky  $\tilde{\mu}$ .
6.  $\hat{K} = \text{diag}(1/\tilde{\mu}) @ K$
7. Opakujme:  $u = 1/(\hat{K} @ (v/(K^T @ u)))$  - dokud není dosaženo vhodné zastavovací kritérium.
8.  $v = v/(K^T @ u)$ .
9.  $W^\lambda(\mu, v) = \text{sum}(u * ((K * \tilde{C}) @ v))$ .

Algoritmus byl napsán, aby syntakticky odpovídal programovacímu jazyku *python*, který využívá knihoven jako je *numpy* či *PyTorch*.

## 2.3 Structural dissimilarity

Nyní potřebujeme z indexu SSIM vykřesat metiku, resp. alespoň aby byla splněna podmínka, že když se dva obrázky blíží k sobě, tak jejich vzdálenost klesá. K tomu může dobře posloužit konstrukce *DSSIM* (structural dissimilarity):

$$\text{DSSIM}(x, \tilde{x}) = \frac{1 - \text{SSIM}(x, \tilde{x})}{2}. \quad (2.8)$$

Bohužel nezískáváme ryzí metiku, neboť není splněna trojúhelníková nerovnost. Máme ale vlastnost, že

$$\text{DSSIM}(x, \tilde{x}) = 0 \iff x = \tilde{x}, \quad (2.9)$$

která plyne z vlastnosti

$$\text{SSIM}(x, \tilde{x}) = 1 \iff x = \tilde{x}. \quad (2.10)$$

## Kapitola 3

# Adversariální vzorky a jejich tvorba

### 3.1 Klasifikace v kontextu strojového učení

Mějme za úkol klasifikovat jakési vzorky do  $m$  tříd. Např. mějme za úkol na základě černobílého obrázku s číslicí říci, jaká že číslice je na daném obrázku vyobrazená. Máme-li dostatečný počet vzorků, o kterých víme, do jaké třídy náleží, můžeme využít různých metod strojového učení. Pro konkrétnost zvolme metodu neuronových sítí. To znamená, že vytvoříme zobrazení  $F_\theta : X \rightarrow Y$ , kde za  $X$  bereme množinu všech možných vzorků, v případě klasifikace číslic na obrázku právě všechny možné obrázky příslušného rozměru. Dále za  $Y$  bereme množinu všech diskretních pravděpodobnostních rozdělení na třídách, tedy v případě klasifikace číslic může být:

$$Y = \left\{ y \in \mathbb{R}^{10} \mid \forall i = 1, \dots, 10 : y_i \geq 0 \wedge \sum_{i=1}^{10} y_i = 1 \right\}.$$

$F_\theta$  je potom daná neuronová síť parametrizovaná pomocí parametrů  $\theta$ . Vhodné parametry  $\theta$  se potom volí pomocí procesu *učení*, což je řešení následujícího optimalizačního problému:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} J(\theta), \quad (3.1)$$

kde  $J$  je funkce na parametrech určující, jak moc je neuronová síť špatná má-li dané parametry  $\theta$ . Za funkci  $J$  se standardně volí agregace typu průměr či součet dílčí ztrátové funkce  $L$ , která určuje, jak moc se neuronová síť mýlí na konkrétním vzorku. K tomu je tedy potřeba mít trénovací datovou sadu sestávající z dostatečného počtu vzorků se správnými odpověďmi, tedy značkami. Budiž trénovací datová sada označena  $\mathbb{T} = (\mathbb{X}, \mathbb{Y})$ , kde  $\mathbb{X} = (x^{(i)} \in X)_{i=1}^N$  a  $\mathbb{Y} = (y^{(i)} \in Y)_{i=1}^N$ . Potom:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^N L(y^{(i)}, F_\theta(x^{(i)})). \quad (3.2)$$

Jelikož prvky  $Y$  jsou diskretní pravděpodobnostní distribuce, za dílčí ztrátovou funkci lze vzít *křížovou entropii*:

$$L(y, \hat{y}) = - \sum_{i=1}^m y_i \log(\hat{y}_i). \quad (3.3)$$

Procesu učení se zde nemusíme věnovat. Jedná se o řešení optimalizačního problému (3.1).

Poslední objekt, který si zde v úvodní sekci kapitoly zadefinujeme, je samotná klasifikace. Jedná se o funkci  $C : Y \rightarrow \{1, 2, \dots, m\}$  definovanou přepisem:

$$C(y) = \underset{i \in \{1, \dots, m\}}{\operatorname{argmax}} y_i. \quad (3.4)$$

### 3.2 Adversariální vzorky

Jsme-li vybaveni metrikou  $\rho$  na prostoru vzorků  $X$ , lze přistoupit k uvedení konceptu adversariálních vzorků. Nejprve však definujme vzorek *benigní*. Jedná se o takový vzorek  $x \in X$ , že  $C(F_\theta(x)) = C(y)$ , kde  $y$  je pravdivá třída vzorku  $x$ . Máme-li takový benigní vzorek  $x$ , pak *adversariální vzorek* k němu, je takový vzorek  $\tilde{x}$ , že vzorek  $\tilde{x}$  je podobný benignímu vzorku  $x$ , ale je špatně klasifikovaný. Podobnost lze matematicky vyjádřit způsobem, že vzdálenost  $\tilde{x}$  od  $x$  je malá ve smyslu:  $\rho(x, \tilde{x}) < \kappa$ , kde  $\kappa$  je pevně zvolený číselný práh, neboli poloměr kulového okolí, ve kterém adversariální vzorek hledáme. Špatnou klasifikaci lze pak vyjádřit:  $C(F_\theta(\tilde{x})) \neq C(F_\theta(x))$ . Je-li sám vzorek  $x$  špatně klasifikovaný, pak je sám vlastně adversariálním vzorkem.

Problematika adversariálních vzorků představuje v oboru strojového učení úskalí, neboť vhodně zvolený adversariální vzorek dokáže v praxi, kde algoritmy strojového učení mohou sehrávat roli při automatizaci bezpečnostně kritických úkolů, daný algoritmus, rozbít.

Zatím jsme adversariální vzorky představili pouze jako teoretický koncept. V následujících sekcích textu se podíváme na jejich konkrétní tvorbu a v jedné z následujících kapitol i na praktickou ukázkou takových adversariálních vzorků.

### 3.3 Tvorba adversariálních vzorků

Dostaneme-li benigní vzorek  $x$  s pravdivou značkou  $y$  a máme-li za úkol k němu pro daný klasifikátor najít adversariální vzorek  $\tilde{x}$ , mějme v první řadě na mysli, že chceme, zachovat podobnost  $x$  a  $\tilde{x}$ , tak aby oba vzorky měly stejnou třídu reprezentovanou značkou  $y$ . Tedy jedna část úkolu bude minimalizovat  $\rho(x, \tilde{x})$ . V další části úkolu máme na výběr ze dvou možností.

Lze k tomuto úkolu přistoupit tak, že si vybereme falešnou značku  $\tilde{y}$ , která reprezentuje jinou třídu než  $y$ . Dále se budeme snažit pohybovat s  $\tilde{x}$  tak, abychom  $F_\theta(\tilde{x})$  přiblížili k  $\tilde{y}$ , budeme tedy minimalizovat  $L(\tilde{y}, F_\theta(\tilde{x}))$ , kde  $L$  je dílčí ztrátová funkce na množině značek. Máme tedy dvě hodnoty, které chceme minimalizovat. Jak je ale dát dohromady? Tradice hovoří o zavedení nezáporného parametru  $\lambda$ , který nám dá:

$$\tilde{x} = \underset{\tilde{x}}{\operatorname{argmin}} \rho(x, \tilde{x}) + \lambda \cdot L(\tilde{y}, F_\theta(\tilde{x})). \quad (3.5)$$

Zavedený parametr  $\lambda$  pak hraje roli v určování toho, zda požadujeme, aby výsledek (3.5) byl velmi blízký  $x$ , nebo aby tento výsledek byl jistě nesprávně klasifikovaný. Tento způsob je představen v původním článku, který osvětluje problematiku adversariálních vzorků [7].

Druhý přístup spočívá v tom, že se snažíme namísto minimalizace ztráty k falešné značce maximalizovat ztrátu od původní značky  $y$ . Tedy:

$$\tilde{x} = \underset{\tilde{x}}{\operatorname{argmin}} \rho(x, \tilde{x}) - \lambda \cdot L(y, F_\theta(\tilde{x})). \quad (3.6)$$

Tento přístup se nazývá metoda CW (*Carlini-Wagner*, [8]).

Jak nastavit parametr  $\lambda$ ? Bud' můžeme parametr  $\lambda$  chápat jako hyper-parametr, tedy jako něco, co musíme ručně ladit, nebo lze hledat optimální hodnotu parametru  $\lambda$  vhodně vybraným kritériem. Myšlenka je potom následující: Chtějme najít adversariální vzorek co nejbližší původnímu benignímu vzorku. Potom v optimalizačním problému (3.5) nebo (3.6) potřebujeme, aby byl kladen větší důraz na první člen  $\rho(x, \tilde{x})$ , tedy aby  $\lambda$  bylo co nejmenší. Zároveň ale potřebujeme, aby výsledek byl nesprávně klasifikován. Proto optimální hodnota parametru  $\lambda$ , označme ji jako  $\lambda^*$ , můžeme v případě CW útoku získat řešením

$$\lambda^* = \underset{\lambda}{\operatorname{argmin}} \rho(x, \tilde{x}^\lambda), \quad (3.7)$$

$$C(F_\theta(\tilde{x}^\lambda)) \neq C(y), \quad (3.8)$$



kde

$$\tilde{x}^\lambda = \operatorname{argmin}_{\hat{x}} \rho(x, \hat{x}) - \lambda \cdot L(y, F_\theta(\hat{x})). \quad (3.9)$$

Z intuice lze na  $\rho(x, \tilde{x}^\lambda)$  nahlédnout jako na monotónní funkci v proměnné  $\lambda$ . Proto na hledání  $\lambda$  lze užít metodu půlení intervalů.

## Kapitola 4

# Výsledky tvorby adversariálních vzorků

### 4.1 Detaily implementace adversariálního útoku

Pro předvedení adversariálního útoku jsme zvolili metodu *CW* s pevným parametrem  $\lambda$ . Pro připomenutí: Metoda útoku *CW* je založena na řešení problému

$$\tilde{x} = \underset{\hat{x}}{\operatorname{argmin}} \rho(x, \hat{x}) - \lambda \cdot L(y, F_{\theta}(\hat{x})). \quad (4.1)$$

Tento problém jsme se rozhodli řešit *znaménkovým gradientním sestupem* (v angl. literatuře uváděný jako *sign gradient descent*), a to s pevným počtem iterací (100) a s pevným krokem  $10^{-2}$ , který je pro obrázky lineárně přeškálované do intervalu  $[0, 1]$  akorát dostačující. Jako ukázkovou úlohu jsme zvolili klasifikaci číslic na datové sadě *MNIST* [9] (jednokanálové obrázky o rozměru  $28 \times 28$ ), která je v komunitě strojového učení notoricky známá. Pro řešení původního problému klasifikace jsme natrénovali (algoritmem *RMSProp* [10]) jednoduchou konvoluční neuronovou síť, která na testovací datové sadě dosáhla úspěšnosti 96,9%. Inicializace startovního bodu znaménkového gradientního sestupu probíhala v dvojím režimu: V normálním a speciálním. Normální inicializace spočívala v náhodné inicializaci každého pixelu v okolí hodnoty 0,5 na základě realizace náhodné veličiny s rovnoměrným rozdělením  $U(0, 3; 0, 8)$ . Jak ale uvidíme z výsledků později, pro jednu z metrik je tato inicializace zcela nevyhovující, proto došlo i k implementaci speciální inicializace, kde byl za počáteční bod gradientního sestupu pro daný benigní vzorek  $x$  vzat právě tento vzorek  $x$ . Na samotnou implementaci *CW* útoku lze nahlédnout v Listing 4.1.

Listing 4.1: adversarials.py

```
def cw_batch(
2     model: nn.Module,
    benign_examples: torch.Tensor,
4     labels: torch.Tensor,
    c_lambda: float,
6     metric: Metric,
    special_init: bool = False
8 ) -> torch.Tensor:
    if special_init:
10         adversarial_examples = benign_examples
    else:
12         adversarial_examples = 0.5 * torch.ones(benign_examples.shape) \
            + 0.3 * (2 * torch.rand(benign_examples.shape) - 1)
14     loss_fn = nn.CrossEntropyLoss(reduction='sum')
    step_size = 1e-2
16     for _ in range(100):
        adversarial_examples.requires_grad = True
18         if adversarial_examples.grad is not None:
            adversarial_examples.grad.zero_()
```

```

20     benign_examples.requires_grad = True
21     if benign_examples.grad is not None:
22         benign_examples.grad.zero_()
23     metrics = metric(benign_examples, adversarial_examples)
24     loss = metrics.sum() \
25         - c_lambda * loss_fn(
26             model(adversarial_examples),
27             torch.tensor(labels, dtype=torch.long)
28         )
29     loss.backward()
30     adversarial_examples = (adversarial_examples \
31         - step_size * adversarial_examples.grad.apply_(
32             lambda x: 1 if x >= 0 else -1
33         )).detach()
34     return adversarial_examples

```

## 4.2 Srovnání útoků pro různé metiky

Přístupme nyní k vyhodnocení experimentu, který spočíval v následujícím: Pro vybranou metiku vizuální podobnosti a pevně zvolené  $\lambda$ , které se v logaritmu mění lineárně, tj. prochází hodnoty  $\lambda \in \{10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^2, 10^3\}$ , spustit generování adversariálních vzorků pro 20 benigních vzorků z datové sady MNIST a zjistit pro danou metiku a dané  $\lambda$  průměrnou úspěšnost útoku, dále ale i průměrnou vizuální podobnost (v té samé metrice, která byla použita v optimalizačním problému) vygenerovaných vzorků a následně jednotně pro všechny metiky průměrnou  $l_2$  vzdálenost vygenerovaných vzorků od původních benigních.

Tento experiment vyžaduje, aby implementace dané metiky byla automaticky derivovatelná nástroji, které nabízí knihovna *PyTorch*. To bohužel nesplňuje Wassersteinova vzdálenost, resp. její modifikace duální Sinkhornovy metiky, a to z důvodu jejího iterativního charakteru. Proto byla tato metrika vynechána z následujícího experimentu.

V Tabulce (4.1), která se v textu nachází níže, jsou již uvedeny čísla reprezentující výše představené sledované údaje. Jelikož metrika vizuální podobnosti DSSIM je založena na indexu SSIM, který je počítán postupně v podoknech a následně agregován, je u této metiky uvedena i velikost tohoto podokna.

Tabulka 4.1: Výsledky CW útoku na konvoluční neuronovou síť

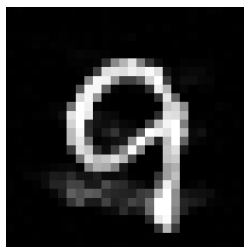
Začátek tabulky				
Metrika	Parametr $\lambda$	Úspěšnost útoku	Prům. vzdálenost	Prům. $l_2$ vzdálenost
DSSIM 5	1e-03	0.35	0.1700	3.5686
	1e-02	0.80	0.2061	8.7937
	1e-01	1.00	0.2243	10.8962
	1e+00	0.95	0.2282	11.2006
	1e+01	1.00	0.2359	12.1091
	1e+02	1.00	0.2397	11.8446
	1e+03	1.00	0.2443	11.3087
DSSIM 13	1e-03	0.00	0.0032	0.5658
	1e-02	0.55	0.0091	1.2746
	1e-01	0.80	0.0218	2.0332
	1e+00	0.90	0.0325	2.6771
	1e+01	1.00	0.0387	3.0795
	1e+02	1.00	0.0439	3.3408

	1e+03	1.00	0.0514	3.6996
DSSIM 21	1e-03	0.00	0.0002	0.2379
	1e-02	0.65	0.0068	1.1227
	1e-01	0.85	0.0179	1.9304
	1e+00	0.90	0.0245	2.3356
	1e+01	1.00	0.0319	2.8478
	1e+02	1.00	0.0351	3.0299
	1e+03	1.00	0.0402	3.2757
DSSIM 28	1e-03	0.00	0.0010	0.3576
	1e-02	0.75	0.0088	1.4044
	1e-01	0.90	0.0157	1.9415
	1e+00	1.00	0.0229	2.4947
	1e+01	1.00	0.0272	2.7499
	1e+02	1.00	0.0314	2.9653
	1e+03	1.00	0.0351	3.1641
$l_1$	1e-03	0.00	3.9127	0.1612
	1e-02	0.00	3.9545	0.1625
	1e-01	0.00	3.9264	0.1619
	1e+00	0.00	3.9351	0.1623
	1e+01	0.75	15.6376	1.9642
	1e+02	0.85	20.6205	2.3534
	1e+03	1.00	27.6782	3.1174
$l_2$	1e-03	0.00	0.1615	0.1615
	1e-02	0.00	0.1613	0.1613
	1e-01	0.00	0.1611	0.1611
	1e+00	0.75	1.5334	1.5334
	1e+01	0.95	2.1603	2.1603
	1e+02	1.00	2.4959	2.4959
	1e+03	1.00	2.6871	2.6871
$l_\infty$	1e-03	1.00	1.1404	15.6282
	1e-02	1.00	1.1585	15.6439
	1e-01	1.00	1.1527	15.6620
	1e+00	1.00	1.1629	15.5520
	1e+01	1.00	1.1511	15.7462
	1e+02	1.00	1.1573	15.6811
	1e+03	1.00	1.1606	15.5801
$l_\infty$ speciální	1e-03	1.00	0.7640	9.3911
	1e-02	1.00	0.7600	9.3717
	1e-01	1.00	0.7650	9.4483
	1e+00	1.00	0.7680	9.3663
	1e+01	1.00	0.7620	9.3416
	1e+02	1.00	0.7690	9.4340
	1e+03	1.00	0.7590	9.4688
NPSR	1e-03	0.00	$-\infty$	0.0000
	1e-02	0.00	$-\infty$	0.0000
	1e-01	0.00	$-\infty$	0.0000
	1e+00	0.00	$-\infty$	0.0000

	1e+01	0.00	$-\infty$	0.0052
	1e+02	0.05	$-\infty$	0.0437
	1e+03	0.15	$-\infty$	0.1282
Konec tabulky				

Ohledně počtu generovaných vzorků v experimentu lze poznamenat, že kvůli výpočetní náročnosti DSSIM na dostupných výpočetních zařízeních nebylo možné toto číslo navýšit.

Na Obrázku (4.1) lze spatřit adversariální vzorek vygenerovaný  $l_2$  útokem s hodnotou parametru  $\lambda = 10$ . Jeho  $l_2$  vzdálenost od původního vzorku je potom 1,75. Toto je typický zástupce adversariálních vzorků: V obrázku jsou jakési artefakty, které pro lidské oko nepředstavují zatemnění informace, že se jedná o obrázek devítky. Klasifikátor v podobě naší neuronové sítě ovšem tyto artefakty interpretoval jinak, a to tak, že obrázek špatně klasifikoval.

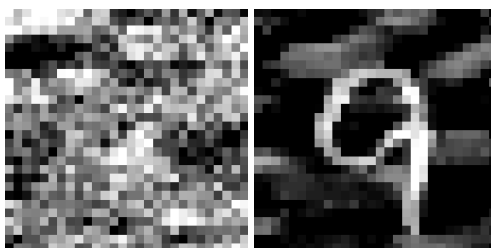


Obrázek 4.1: Příklad adversariálního vzorku

Celkově lze z tabulky vyčíst, že například metrika vizuální podobnosti NPSR, tedy logaritmická transformace přeškálované metriky  $l_2$ , je naprosto nevhodná pro tvorbu adversariálních vzorků. Toto je na první pohled způsobeno charakterem derivace logaritmu, jelikož tato derivace je pro argument klesající k nule až neomezená a v gradientních metodách optimalizace po čase vždy převládá při optimalizaci účelové funkce derivaci druhého členu v (3.6).

Z tabulky jako král úspěšnosti vychází útok prováděný s metrikou  $l_\infty$ . To je ovšem za cenu velkého znečištění obrázku, jak vidíme v Obrázku () a jak lze odvodit z průměrné  $l_2$  vzdálenosti vygenerovaného vzorku od původního benigního, která je až řádově větší než v případě např.  $l_2$  útoku s  $\lambda = 10^3$ , jenž má mimochodem též úspěšnost 100%. Na Obrázku (4.2) lze spatřit vlevo vzorek vygenerovaný  $l_\infty$  útokem s normální inicializací a vpravo vzorek vygenerovaný  $l_\infty$  útokem se speciální inicializací. Oba vzorky jsou značně poškozené, tedy mají velkou vzdálenost od původního benigního vzorku. To je dáno faktem, že při optimalizaci v CW útku člen  $\rho(x, \hat{x})$  záleží vlastně na jediném pixelu, proto při každém kroku znaménkového gradientního sestupu algoritmus přiblíží k původnímu vzorku maximálně pouze jeden jediný pixel. Ohledně  $l_\infty$  útoku lze též poznamenat, že ať v nastavení s normální inicializací, nebo v nastavení se speciální inicializací zdánlivě nezávisí na parametru  $\lambda$ . Souvisí to s předchozím. Lze vlastně říci, že skoro pro všechny indexy  $i$  (výjimkou je vždy jeden jediný) platí  $\partial_{\hat{x}_i} \rho(x, \hat{x}) = 0$ . Proto se při znaménkovém gradientním sestupu výsledek k původnímu vzorku prakticky vůbec nepřibližuje.

Další komentář se bude věnovat obdobnému problému, který vyvstal pro DSSIM útok s parametrem velikosti výpočetního podokna 5. Podle Obrázku (4.3) je okolí číslice silně znečištěno artefakty. Kupodivu samotná číslice a její bezprostřední okolí je relativně nedotknuté. Tento fenomén se vyskytl hlavně u volby velikosti výpočetního podokna 5. Vysvětlení zní jednoduše. Neboť je původní benigní obrázek dále od číslice roven v každém pixelu nule, pak SSIM index příslušných dvou podoken dále od číslice vychází konstantně skoro nula, a to kvůli členu  $\sigma_{x\tilde{x}}$  v čitateli. Není úplně nula díky konstantě  $C_2$ , která zde vystupuje kvůli dělení. Ta je ovšem dostatečně malá, aby byla převážena druhým členem gradientu.



Obrázek 4.2: Příklady vzorků vygenerovaných  $l_\infty$  útokem



Obrázek 4.3: Příklad vzorku vygenerovaného DSSIM útokem

## Kapitola 5

# Robustnost neuronové sítě

Předvedli jsme jev existence adversariálních vzorků a z jeho vlastní povahy je zřejmé, že se jedná o nežádoucí jev. Na tento fenomén potom odpovídá *robustní strojové učení*, které se ve své podstatě snaží při učení neuronové sítě danou neuronovou sít' naučit tak, aby, pokud možno, k existenci adversariálních vzorků nedocházelo. Obor robustního strojového učení potom nabízí řadu metod, které k tomu mohou dopomoci. Otázkou ale je, jak takovou metodu zařadit mezi ostatní ve smyslu jejího srovnání s ostatními metodami. Potřebujeme proto ideálně číselné vyjádření toho, jak si na tom daná neuronová sít', která byla učena daným algoritmem robustního strojového učení, stojí ve smyslu náchylnosti na přítomnost jevu adversariálních vzorků.

### 5.1 Přístup knihovny Foolbox

Jednou z programovacích knihoven, která se snaží osvětlit tuto tematiku a přinést nástroj pro výše nastíněné měření robustnosti modelu neuronové sítě, je knihovna *Foolbox* [3]. Její přístup spočívá v implementaci pěti základních stavebních kamenů pro tvorbu adversariálních vzorků. Jsou to:

- *Model*, implementace rozhraní, které zajišťuje kompatibilitu knihovny s populárními knihovnami strojového učení (jako je i *PyTorch*);
- *kritérium*, totiž pravidlo, podle kterého se rozhoduje, zda daný vzorek je adversariální či nikoliv;
- *metrická vzdálenost*, to jest funkce, která vyjadřuje velikost perturbace potenciálního adversariálního vzorku (rozdíl  $\tilde{x} - x$ , užijeme-li zavedeného značení);
- *algoritmus útoku*, způsob, jakým budou potenciální adversariální vzorky vyráběny;
- samotná *adversariální perturbace*, což je výsledek algoritmu útoku.

Za komentář stojí, jaká že kritéria mohou být užita k určení adversariality vzorku. S jedním jsme se již setkali v minulých kapitolách, totiž kritérium *nesprávné klasifikace*, které spočívá ve vyhodnocení vzorku jako adversariálního právě při určení modelu, že daný vzorek je v jiné třídě než původní vzorek, podle kterého je vzorek adversariální tvořen. Nemusíme zůstat pouze u tohoto kritéria. Další kritéria lze odvodit při hlubším studiu pravděpodobnostního rozdělení, které model produkuje. Takže např. kritérium *top-k nesprávné klasifikace* spočívá v tom, že vzorek je adversariální, pokud původní třída není mezi  $k$  nejpravděpodobnějšími třídami.

Za zmínku též stojí fakt, že knihovna *Foolbox* implementuje celou řadu různých adversariálních útoků.

## 5.2 Přístup knihovny RobustBench

Další programovací knihovnou, která se věnuje tématu robustnosti neuronových sítí je knihovna *RobustBench* [4]. Tato knihovna jde o krok dál než knihovna Foolbox, neboť její snahou je vyvinout jednotnotný test, který pro všechna nastavení produkuje jediné číslo, které lze tudíž hladce porovnat s ostatními výsledky. Těchto testů je několik druhů, totiž pro datové sady *CIFAR-10*, *CIFAR-100* [11] a *ImageNet* [12]. Následně podle zkoumané metriky jsou testy pro  $l_\infty$  či  $l_2$  útoky. Výsledné číslo se potom nazývá *robustní úspěšnost* (z angl. *robust accuracy*), jehož vyhodnocení spočívá ve vyčíslení průměrné úspěšnosti klasifikace poškozených vzorků zkoumaným modelem v daném nastavení experimentu. Tyto poškozené vzorky jsou postupně generovány procesem *AutoAttack* [13], který spočívá v postupném provádění čtyř typů adversariálních útoků. Nejprve benigní vzorky projdou úpravou v podobě *projected gradient descent* (PGD) [14] s adaptivní velikostí kroku a ztrátou křížové entropie. Dále vzorky, které zůstanou správně klasifikované projdou obdobně procesem PGD, ale s jinou ztrátovou funkcí, a to ztrátou rozdílu podílů hodnot funkce logit. Poté je proveden *cílený FAB útok* [15], následně *black-box square attack* [16]. Následně se výsledky agregují v již zmíněnou adversariální úspěšnost. Tím pádem lze metody robustního strojového učení mezi sebou porovnávat.



# **Závěr**

Text závěru....

# Literatura

- [1] N. Akhtar, A. Mian, N. Kardan, M. Shah: *Advances in adversarial attacks and defenses in computer vision: A survey*. IEEE Access 9, 2021, 155161-155196.
- [2] W. Eric, F. Schmidt, Z. Kolter: *Wasserstein adversarial examples via projected sinkhorn iterations*. International Conference on Machine Learning, PMLR, 2019.
- [3] J. Rauber, R. Zimmermann, M. Bethge, W. Brendel: *Foolbox: A Python toolbox to benchmark the robustness of machine learning models*. Reliable Machine Learning in the Wild Workshop, 34th International Conference on Machine Learning, 2017.
- [4] F. Croce, M. Andriushchenko, V. Schwag, E. Debenedetti, N. Flammarion, M. Chiang, P. Mittal, M. Hein: *RobustBench: a standardized adversarial robustness benchmark*. Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2), 2021
- [5] L. Vaserstein, *Markov processes over denumerable products of spaces, describing large systems of automata*. Problemy Peredači Informacii 5, 1969.
- [6] M. Cuturi, *Sinkhorn Distances: Lightspeed Computation of Optimal Transport*. Advances in Neural Information Processing Systems 26, 2013.
- [7] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, R. Fergus, *Intriguing properties of neural networks*. arXiv, 2014.
- [8] N. Carlini, D. Wagner, *Towards evaluating the robustness of neural networks*. IEEE Symposium on Security and Privacy (SP), IEEE, 2017.
- [9] Y. Lecun, C. Cortes, C. J. Burges, *The mnist database of handwritten digits*. 1998.
- [10] G. Hinton, *Neural networks for machine learning*. Coursera, video lectures, 2012.
- [11] A. Krizhevsky, G. Hinton: *Learning multiple layers of features from tiny images*. Technical Report, 2009.
- [12] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei: *Imagenet: A large-scale hierarchical image database*. IEEE conference on computer vision and pattern recognition, pages 248–255, Ieee, 2009.
- [13] F. Croce, M. Hein: *Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks*. In ICML, 2020.
- [14] A. Mańdry, A. Makelov, L. Schmidt, D. Tsipras, A. Vladu, *Towards deep learning models resistant to adversarial attacks*. Stat 1050 9, 2017.

- [15] F. Croce, M. Hein: *Minimally distorted adversarial examples with a fast adaptive boundary attack*. ICML, 2020.
- [16] M. Andriushchenko, F. Croce, N. Flammarion, M. Hein: *Square attack: a query-efficient black-box adversarial attack via random search*. ECCV, 2020.

# Příloha

metrics.py

```
1 from typing import Union
2 from abc import abstractmethod
3 import itertools
4
5
6 import torch
7 import torch.nn as nn
8
9
10 class Transform(nn.Module):
11     """
12     Class to encapsulate transformation of data
13     """
14
15     pass
16
17
18 class Identity(Transform):
19     """
20     Identity transformation
21     """
22
23     def forward(self, x: torch.Tensor) -> torch.Tensor:
24         return x
25
26
27 class Metric(nn.Module):
28     """
29     Module that encapsulates implementation of a mathematical concept of metric
30     With possibility of a transformation applied
31     """
32
33     def __init__(self, transform: Union[Transform, None] = None):
34         """
35         Constructor
36         :param transform: Transformation to be applied
37         """
38         super().__init__()
39         if transform is None:
40             self.transform = Identity()
41         else:
42             self.transform = transform
43
44     def forward(self, x: torch.Tensor, y: torch.Tensor) -> torch.Tensor:
45         transformed_x, transformed_y = self.transform(x), self.transform(y)
46         return self.compute(transformed_x, transformed_y)
47
48     @abstractmethod
49     def compute(self, x: torch.Tensor, y: torch.Tensor) -> torch.Tensor:
50         """
51         Actual computation of the metric distance
52         :param x: input No. 1
```

```

52         :param y: input No. II
53         :return: Tensor of batch of metrics
54         """
55         pass
56
57
58 class Norm(nn.Module):
59     """
60     Encapsulation of a norm
61     """
62
63     @abstractmethod
64     def forward(self, x: torch.Tensor) -> torch.Tensor:
65         """
66         :param x: Tensor of which norm shall be computed (shape: Batch x the_rest)
67
68         :return: Tensor of batch of norms
69         """
70         pass
71
72
73 class LpNorm(Norm):
74     """
75     Implementation of a L_p norm for a given positive integer p
76     """
77
78     def __init__(self, p: int):
79         """
80         :param p: positive integer
81         """
82         super().__init__()
83         if p < 1:
84             raise ValueError("p must be greater than 1")
85         self.p = p
86
87     def forward(self, x: torch.Tensor) -> torch.Tensor:
88         """
89         :param x: Tensor of which norm shall be computed (shape: Batch x the_rest)
90
91         :return: Tensor of batch of norms
92         """
93         return (x.abs() ** self.p).sum(dim=tuple(range(1, x.ndim))) ** (1 / self.p)
94
95
96 class L2Norm(LpNorm):
97     """
98     Special case of a LpNorm - L_2
99     """
100
101     def __init__(self):
102         super().__init__(2)
103
104
105 class L1Norm(LpNorm):
106     """
107     Special case of a LpNorm - L_1
108     """
109     def __init__(self):
110         super().__init__(1)
111
112
113 class LinfNorm(Norm):
114     """
115     Implementation of a L_infty norm
116     """
117
118     def forward(self, x: torch.Tensor) -> torch.Tensor:

```

```

120         """
121         :param x: Tensor of which norm shall be computed (shape: Batch x the_rest)
122
123         :return: Tensor of batch of norms
124         """
125         out = x.abs()
126         for _ in range(1, out.ndim):
127             out = out.max(dim=1)[0]
128         return out
129
130 class L0Norm(Norm):
131     """
132     Implementation of a L_0 norm
133     """
134
135     def forward(self, x: torch.Tensor) -> torch.Tensor:
136         """
137         :param x: Tensor of which norm shall be computed (shape: Batch x the_rest)
138
139         :return: Tensor of batch of norms
140         """
141
142         return ((x != torch.zeros(x.shape)) * 1).sum(dim=tuple(range(1, x.ndim)))
143
144 class MetricFromNorm(Metric):
145     """
146     Encapsulation of metric distance which is derived as a norm of a difference
147     """
148
149     def __init__(self, norm: Norm, transform: Union[Transform, None] = None):
150         """
151         Implementation of a constructor
152         :param norm: Norm
153         :param transform: Transformation to be applied
154         """
155         super().__init__(transform)
156         self.norm = norm
157
158     def compute(self, x: torch.Tensor, y: torch.Tensor) -> torch.Tensor:
159         """
160         Implementation of the actual computation using the given norm
161         :param x: Tensor x
162         :param y: Tensor y
163         :return: Tensor of batch of metrics
164         """
165         out = self.norm(x - y)
166         return out
167
168
169 class LpMetric(MetricFromNorm):
170     """
171     Metric devived from a L_p norm
172     """
173
174     def __init__(self, p, transform: Union[Transform, None] = None):
175         """
176         Implementation of a constructor
177         :param p: Positive integer p
178         :param transform: Transformation to be applied
179         """
180         super().__init__(LpNorm(p), transform)
181
182
183 class L2Metric(LpMetric):
184     """

```

```

186     Implementation of a L_2 metric derived from L_2 norm
187     """
188
189     def __init__(self, transform: Union[Transform, None] = None):
190         super().__init__(2, transform)
191
192 class L0Metric(MetricFromNorm):
193     """
194     Implementation of a L_0 metric derived from L_0 norm
195     """
196
197     def __init__(self, transform: Union[Transform, None] = None):
198         super().__init__(L0Norm(), transform)
199
200
201 class LinfMetric(MetricFromNorm):
202     """
203     Implementation of a L_infty metric derived from L_infty norm
204     """
205
206     def __init__(self, transform: Union[Transform, None] = None):
207         super().__init__(LinfNorm(), transform)
208
209
210 class MeanSquaredError(Metric):
211     """
212     Implementation of MSE as a metric
213     """
214
215     def __init__(self, transform: Union[Transform, None] = None):
216         super().__init__(transform)
217
218     def compute(self, x: torch.Tensor, y: torch.Tensor) -> torch.Tensor:
219         return ((x - y) ** 2).mean(dim=tuple(range(1, x.ndim)))
220
221
222 class RootMeanSquaredError(Metric):
223     """
224     Implementation of RMSE as a metric
225     """
226
227     def __init__(self, transform: Union[Transform, None] = None):
228         super().__init__(transform)
229
230     def compute(self, x: torch.Tensor, y: torch.Tensor) -> torch.Tensor:
231         return ((x - y) ** 2).mean(dim=tuple(range(1, x.ndim))) ** (1 / 2)
232
233
234 class PeakSignalToNoiseRatio(Metric):
235     """
236     Implementation of PSNR
237     """
238
239     def __init__(self, transform: Union[Transform, None] = None, l: float = 1):
240         """
241         Constructor
242
243         :param l: peak signal of the image
244         """
245         super().__init__(transform)
246         self.l = l
247         self.rmse = RootMeanSquaredError()
248
249     def compute(self, x: torch.Tensor, y: torch.Tensor) -> torch.Tensor:
250         out = 20 * torch.log10(self.l / self.rmse(x, y))
251         return out
252

```

```

254 class NoiseToPeakSignalRatio(Metric):
255     """
256     Implementation of NPSR
257     """
258
259     def __init__(self, transform: Union[Transform, None] = None, l=1):
260         """
261         Constructor
262
263         :param l: peak signal of the image
264         """
265         super().__init__(transform)
266         self.l = l
267         self.rmse = RootMeanSquaredError()
268
269     def compute(self, x: torch.Tensor, y: torch.Tensor) -> torch.Tensor:
270         out = 20 * torch.log10(self.rmse(x, y) / self.l)
271         return out
272
273
274 class StructuralDissimilarity(Metric):
275     """
276     Implementation of Structural Dissimilarity.
277     Computed as (1 - SSIM) / 2
278     """
279
280     def __init__(self, transform: Union[Transform, None] = None, window_size=100, k_1=1e
281 -2, k_2=3e-2, l=1):
282         """
283         Constructor
284
285         :param window_size: number, size of the sliding window in which SSIMs are
286         computed
287         :param k_1: component of the first constant (for safe division)
288         :param k_2: component of the second constant (for safe division)
289         :param l: peak signal, component of the safe division constants
290         """
291         super().__init__(transform)
292         self.window_size = window_size
293         self.c_1 = (k_1 * l) ** 2
294         self.c_2 = (k_2 * l) ** 2
295
296     def compute(self, x: torch.Tensor, y: torch.Tensor) -> torch.Tensor:
297         if x.ndim != 4 or y.ndim != 4:
298             raise ValueError("Not an image")
299         if x.shape != y.shape:
300             raise ValueError("Given images of different shapes")
301
302         batch = x.shape[0]
303         channels = x.shape[1]
304         width = x.shape[2]
305         height = x.shape[3]
306
307         num_width_windows = width - self.window_size + 1 if width > self.window_size else
308         1
309         num_height_windows = height - self.window_size + 1 if height > self.window_size
310         else 1
311
312         windows_indexes = [
313             [
314                 range(i_w_start, min(i_w_start + width, i_w_start + self.window_size)),
315                 range(i_h_start, min(i_h_start + height, i_h_start + self.window_size))
316             ]
317             for i_w_start, i_h_start in itertools.product(range(num_width_windows), range
318 (num_height_windows))
319         ]

```



```

316     x_windows = torch.zeros(len(windows_indexes), batch, channels, min(self.
window_size, width), min(self.window_size, height))
    y_windows = torch.zeros(len(windows_indexes), batch, channels, min(self.
window_size, width), min(self.window_size, height))
318     for i, indexes in enumerate(windows_indexes):
        x_windows[i, :, :, :] += torch.index_select(torch.index_select(x, 2, torch
.tensor(indexes[0], dtype=torch.int)), 3, torch.tensor(indexes[1], dtype=torch.int))
320        y_windows[i, :, :, :] += torch.index_select(torch.index_select(y, 2, torch
.tensor(indexes[0], dtype=torch.int)), 3, torch.tensor(indexes[1], dtype=torch.int))

322        x_means = x_windows.mean(dim=(3, 4))
        y_means = y_windows.mean(dim=(3, 4))
324        x_variances = x_windows.var(dim=(3, 4), unbiased=True)
        y_variances = y_windows.var(dim=(3, 4), unbiased=True)
326        x_means_expanded = x_means \
            .reshape(num_width_windows * num_height_windows, batch, channels, 1, 1) # \
328            # .expand(num_width_windows * num_height_windows, batch, channels, min(self.
window_size, width), min(self.window_size, height))
        y_means_expanded = y_means \
330            .reshape(num_width_windows * num_height_windows, batch, channels, 1, 1) # \
            # .expand(num_width_windows * num_height_windows, batch, channels, min(self.
window_size, width), min(self.window_size, height))
332        bessel = (min(self.window_size, width) * min(self.window_size, height))
        bessel = bessel / (bessel - 1)
334        # bessel = 1
        cov = bessel * ((x_windows - x_means_expanded) * (y_windows - y_means_expanded)).
mean(dim=(3, 4))

336        out = (2 * x_means * y_means + self.c_1) * (2 * cov + self.c_2) \
338            / ((x_means ** 2 + y_means ** 2 + self.c_1) * (x_variances + y_variances +
self.c_2))
        return (1 - out.mean(dim=(0, 2))) / 2
340

342 class WassersteinApproximation(Metric):
    """
344     Implementantation of dual-Sinkhorn divergence
    """
346
    def __init__(self, transform:Union[Transform, None] = None, regularization: float =
5, iterations: int = 250, verbose: bool = False):
348         """
        Constructor
350
        :param regularization: regularization coefficient of the entropy term in the
optimization problem
352        :param iterations: fixed number of iterations
        :param verbose: whether to be noisy or not - for debugging reasons
354        """
        super().__init__(transform)
356        self.regularization = regularization
        self.iterations = iterations
358        self.verbose = verbose

360    def compute(self, x:torch.Tensor, y:torch.Tensor) -> torch.Tensor:
    if self.verbose:
362        print('ENTERING WASSERSTEIN')

364        if x.ndim != 4 or y.ndim != 4:
            raise ValueError("Not a batch of images")
366        if x.shape != y.shape:
            raise ValueError("Given images of different shapes")
368        if any(x.flatten() < 0) or any(y.flatten() < 0):
            raise ValueError("Given images are with negative values")

370        batch = x.shape[0]

```

```

372     channels = x.shape[1]
373     width = x.shape[2]
374     height = x.shape[3]

375     if channels > 1:
376         raise NotImplementedError("Wasserstein not implemented for multi-channel
377 images")

378     if (x < 0).sum() > 0 or (y < 0).sum() > 0:
379         raise ValueError("Images must be given with non-negative entries.")

380     # Normalization -> into probability distribution
381     x_norm, y_norm = (x / x.sum(dim=(2, 3), keepdim=True)).reshape(batch, width *
382 height), \
383 (y / y.sum(dim=(2, 3), keepdim=True)).reshape(batch, width * height)

384     cost_matrix = torch.tensor(
385         [
386             [
387                 abs(i // width - j // width) + abs(i % width - j % width)
388                 for j in range(width * height)
389             ]
390             for i in range(width * height)
391         ]
392     )
393     # cost_matrix = cost_matrix / cost_matrix.sum()

394     dists = torch.zeros(batch)

395     for i in range(batch):
396         dists[i] += self.compute_vectors_distance(x_norm[i].flatten(), y_norm[i].
397 flatten(), cost_matrix)
398         if self.verbose:
399             print(f'-COMPUTED DISTANCE {i + 1 } out of {batch}')
400         if self.verbose:
401             print('LEAVING WASSERSTEIN')
402         return torch.Tensor(dists)

403 def compute_vectors_distance(self, x, y, cost_matrix, retain_all_iterations: bool =
404 False):
405     indices = (x != 0)
406     x_non_zero = x[indices]
407     x_non_zero_dim = x_non_zero.shape[0]

408     # Algorithm from paper https://proceedings.neurips.cc/paper/2013/file/af21d0c97db2e27e13572cbf59eb343d-Paper.pdf

409     # u_vector_prev = torch.ones(x_non_zero_dim) / x_non_zero_dim
410     u_vector = torch.ones(x_non_zero_dim) / x_non_zero_dim
411     K_matrix = (- self.regularization * cost_matrix[indices, :]).exp()
412     K_tilde_matrices = torch.diag(1 / x_non_zero) @ K_matrix

413     if self.verbose:
414         print('-STARTING ITERATIONS')

415     if not retain_all_iterations:
416         for _ in range(self.iterations):
417             u_vector = 1 / (K_tilde_matrices @ (y / (K_matrix.transpose(0, 1) @
418 u_vector)))
419             v_vector = y / (K_matrix.transpose(0, 1) @ u_vector)
420             dist = (u_vector * ((K_matrix * cost_matrix[indices, :]) @ v_vector))
421             return dist.sum()
422     else:
423         dists = []
424         for _ in range(self.iterations):
425             u_vector = 1 / (K_tilde_matrices @ (y / (K_matrix.transpose(0, 1) @
426 u_vector)))

```

```
432         v_vector = y / (K_matrix.transpose(0, 1) @ u_vector)
433         dist = (u_vector * ((K_matrix * cost_matrix[indices, :]) @ v_vector))
434         dists.append(dist.sum())
    return dists
```