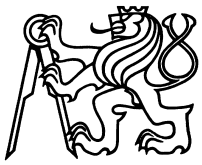
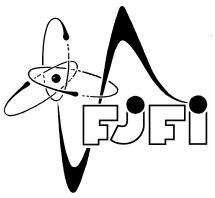


České vysoké učení technické v Praze
Fakulta jaderná a fyzikálně inženýrská



Robustní strojové učení a adversariální vzorky

Robust machine learning and adversarial examples

Bakalářská práce

Autor: **Pavel Jakš**

Vedoucí práce: **Mgr. Lukáš Adam, Ph.D.**

Akademický rok: 2021/2022

- Zadání práce -

- Zadání práce (zadní strana) -

Poděkování:
Chtěl bych zde poděkovat především svému školiteli - panu doktoru Adamovi - za pečlivost, ochotu, vstřícnost a odborné i lidské zázemí při vedení mé bakalářské práce.

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

V Praze dne 7. července 2022 Pavel Jakš

Název práce:

Robustní strojové učení a adversariální vzorky

Autor: Pavel Jakš

Obor: Matematická informatika

Druh práce: Bakalářská práce

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

Abstrakt: Algoritmy strojového učení vykazují přítomnost jevu, kde i malá perturbace vstupu algoritmu může způsobit velkou změnu predikce výstupu. Takto perturbovaným vstupům lze přisoudit název adversariální vzorky. Tento fenomén je demonstrován na příkladu klasifikace ručně psaných číslic pomocí konvolučních neuronových sítí. Předvedeny jsou metody jak pro generování těchto adversariálních vzorků, tak i pro obranu neuronové sítě proti těmto adversariálním útokům.

Klíčová slova: adversariální vzorky, cílená optimalizační metoda, CW, FGSM, I-FGSM, konvoluční neuronové sítě, PGD, robustní strojové učení, umělé neuronové sítě

Title:

Robust machine learning and adversarial examples

Author: Pavel Jakš

Abstract: Machine learning algorithms suffer from a phenomenon which occurs when small perturbation of an input of the algorithm causes a great change of the output prediction. Such perturbated inputs can be called adversarial examples. This phenomenon is demonstrated using the problem of classification of handwritten digits using convolutional neural networks. Methods to create such adversarial examples are shown as well as methods of defense of the neural network against such adversarial attacks.

Key words: adversarial examples, artificial neural networks, convolutional neural networks, CW, FGSM, I-FGSM, PGD, robust machine learning, targeted optimisation method

Obsah

Úvod

Pojem *neuronové sítě* představuje výpočetní jednotku, která svou univerzálností nachází uplatnění v mnoha disciplínách. Disciplínou, která s nástupem výpočetního výkonu posledních let, a tedy i širšího záběru metod strojového učení, zažila velmi vítaný pokrok, je oblast strojového vidění. Jednou z úloh této disciplíny je potom klasifikace ručně psaných číslic na obrázku. Tato úloha pak slouží k demonstraci metod a jevů popsaných v tomto textu.

Představeny budou v tomto textu základy umělých neuronových sítí, které svým charakterem představují nástroj na tvoření predikcí regresního či klasifikačního rázu. Tyto neuronové sítě lze potom na základě dat naučit, aby predikce byly užitečné a správné, tedy aby např. sítě klasifikovaly číslici na obrázku. Tento proces učení je prováděn převedením na *optimalizační* problém, který se následně řeší *numerickými gradientními metodami* také popsanými v tomto textu. V textu je též uvedeno srovnání těchto algoritmů učení.

Stěžejní část textu se potom zabývá jevem, který neuronové sítě provází. Jedná se o existenci tzv. *adversariálních vzorků*. Máme-li totiž vzorek, tedy vstup pro danou neuronovou síť, a to např. obrázek číslice, který daná neuronová síť správně klasifikuje, a přičteme-li k tomuto vzorku jistou perturbaci, která je velmi malá (myšleno ve zvolené l_p normě), potom se stane, že takto vytvořený vzorek je nesprávně klasifikován danou neuronovou sítí. Takto vytvořené vzorky pak nesou název adversariální, jelikož pro danou neuronovou síť představují hrozbu. Nesou totiž stejnou informaci, jako původní vzorek (též označovaný jako benigní), tedy v případě klasifikace číslic se jedná o obrázek, na kterém je napsána stejná číslice, ale neuronová síť tento adversariální vzorek špatně klasifikuje.

Metody, které vedou ke konstrukci takovýchto adversariálních vzorků, jsou nejčastěji založeny jednak na znalosti samotné neuronové sítě, proti které se adversariální vzorky tvoří, a jednak na znalosti optimalizovaného kritéria při učení neuronové sítě. Při tvorbě adversariálních vzorků potom dochází k maximalizaci vzdálenosti (ne nutně metrické) mezi predikcí dané neuronové sítě, je-li jí dán benigní vzorek, a predikcí hledaného adversariálního vzorku. Dále při tvorbě adversariálních vzorků dochází zajištění toho, aby vytvořený vzorek byl blízko benignímu vzorku ve smyslu zvolené l_p normy.

Poslední část textu uvádí metodu, jak danou neuronovou síť proti takovýmto adversariálním útokům bránit. Jedná se o metodu opět optimalizačního charakteru, která při učení neuronové sítě zohledňuje nejen body datové sady, která je použita k trénování dané neuronové sítě, nýbrž i celé jejich okolí ve smyslu l_p normy. Tím se v neuronové síti odrazí myšlenka, že nejen jeden daný bod z datové sady má být klasifikován tím či oním způsobem, ale i celé okolí ve smyslu l_p normy představuje tu samou třídu klasifikace.

Kapitola 1

Neuronové sítě

Neuronová síť je svým charakterem velmi přizpůsobivý výpočetní stroj vhodný pro řešení mnoha problémů. Mezi nejčastější problémy, jejichž řešením může být vhodná neuronová síť, patří *regrese*, čili předpovídání jedné skalární hodnoty na základě vstupu, či *klasifikace*, která má za cíl předpovědět třídu v němž se daný vstup nachází. Obecně tak neuronové síti odpovídá libovolně komplikované zobrazení $F : \mathbb{R}^{n_1 \times \ldots \times n_k} \to \mathbb{R}^{m_1 \times \ldots \times m_l}$. Pro případ regrese potom $l = 1$, $m_1 = 1$ a výstup F hraje roli predikované hodnoty, pro případ klasifikace je též $l = 1$, ale m_1 je rovno počtu tříd a výstup F je predikovanou pravděpodobnostní distribucí, která určuje s jakou pravděpodobností patří daný vstup příslušné třídě.

Samotná síť sestává z mnoha dílčích navzájem propojených částí, o nichž pojednávají následující pasáže této kapitoly.

1.1 Vrstva neuronů

Prvním základním konceptem, který stojí za pojmem neuronové sítě, je rozdělení výpočtu do vrstev. Takové vrstvy potom charakterizuje zobrazení $\phi : \mathbb{R}^{p_1 \times \ldots \times p_r} \to \mathbb{R}^{q_1 \times \ldots \times q_s}$, jehož předpis již lze snadno vyjádřit. Obrazy vstupů při zobrazení ϕ se potom nazývají *aktivace*.

1.1.1 Hustá vrstva

Prvním příkladem vrstev neuronů je tzv. *hustá vrstva* (angl. *dense layer* nebo *fully-connected layer*). Pro zobrazení ϕ platí, že zobrazuje vektory na vektory, tedy $r = s = 1$, a má předpis

$$\phi(u) = Wu + b, \tag{1.1}$$

kde $W \in \mathbb{R}^{q_1 \times p_1}$ je *matice vah* (z angl. *weight*) a $b \in \mathbb{R}^{q_1}$ je *vektor prahů* (z angl. *bias*).

Motivací za pojmenováním této vrstvy jako husté nebo též plně propojené je fakt, že každá složka vstupujícího vektoru ovlivňuje každou z výsledných aktivací, pokud tedy příslušný prvek matice vah není nulový.

1.1.2 Konvoluční vrstva

Pro představení dalšího typu vrstvy uveďme základní přehled o operaci konvoluce. Operace *konvoluce* je ve vší obecnosti operace mezi dvěma číselnými funkcemi g a h se stejným definičním oborem, jejíž výstupem je nová číselná funkce standardně označovaná jako $g * h$. Uveďme zde definici konvoluce pro reálné funkce definované na $\mathbb{R}^d$, tedy $g, h : \mathbb{R}^d \to \mathbb{R}$:

$$(g * h)(t) = \int_{\mathbb{R}^d} g(\tau) h(t - \tau) d\tau.$$

Důležitým předpokladem pro možnost konvoluce je samozřejmě existence integrálu na pravé straně.

Ačkoliv je konvoluce komutativní operací, nejen v kontextu strojového učení se mezi oběma funkcemi vstupujícími do konvoluce rozlišuje. Funkce vstupující jako první se nazývá vstup a druhá funkce se nazývá jádrem. Dále se v kontextu konvolučních sítí standardně objevují diskrétní funkce, které nabývají nenulových hodnot pouze v konečně mnoha bodech. Potom integrál přes $\mathbb{R}^d$ přechází v konečnou sumu:

$$(g * h)(i_1, ..., i_d) = \sum_{j_1} ... \sum_{j_d} g(j_1, ..., j_d) h(i_1 - j_1, ..., i_d - j_d). \tag{1.2}$$

Díky komutativitě konvoluce lze též psát:

$$(g * h)(i_1, ..., i_d) = \sum_{j_1} ... \sum_{j_d} g(i_1 - j_1, ..., i_d - j_d) h(j_1, ..., j_d). \tag{1.3}$$

Při aplikaci komutativity došlo k tzv. *překlopení jádra* (termín pochází z anglického kernel flipping). Za vynechání překlopení jádra lze dojít ke *křížové korelaci*:

$$(g * h)(i_1, ..., i_d) = \sum_{j_1} ... \sum_{j_d} g(i_1 + j_1, ..., i_d + j_d) h(j_1, ..., j_d). \tag{1.4}$$

Mnoho knihoven zabývajících se neuronovými sítěmi dle [1] implementují křížovou korelaci namísto konvoluce, ačkoliv tuto svou implementaci nazývají konvolucí.

Nejčastější užití konvoluce v neuronových sítích je při zpracování obrázků, které lze reprezentovat pomocí $C \times W \times H$ tenzorů, kde C značí počet kanálů obrázku (nejčastěji tři pro červenou, zelenou a modrou), W je šířka, H je výška obrázku. Uveďme předpis pro zobrazení ϕ, které odpovídá konvoluční vrstvě:

$$\forall j \in \{1, 2, ..., C_{out}\} \qquad \phi(u)_j = b_j + \sum_{i=1}^{C_{in}} K_{j,i} * u_i \tag{1.5}$$

kde $\phi : \mathbb{R}^{C_{in} \times W_{in} \times H_{in}} \to \mathbb{R}^{C_{out} \times W_{out} \times H_{out}}$ (C_{in} je počet vstupních kanálů, C_{out} počet výstupních kanálů, W_{in}, H_{in} jsou vstupní šířka a výška, W_{out}, H_{out} jsou výstupní šířka a výška), $b \in \mathbb{R}^{C_{out} \times W_{out} \times H_{out}}$ je práh, $K \in \mathbb{R}^{C_{out} \times C_{in} \times k_1 \times k_2}$ je tenzor konvolučních jader (k_1 a k_2 jsou rozměry konvolučního jádra).

Za povšimnutí stojí, že standardně $W_{out} \neq W_{in}$ a $H_{out} \neq H_{in}$, konkrétně při takto prosté implementaci konvoluční vrstvy platí:

$$W_{out} = W_{in} - k_1 + 1, \tag{1.6}$$

$$H_{out} = H_{in} - k_2 + 1. \tag{1.7}$$

1.1.3 Pooling vrstva

Pojem *pooling vrstvy* (bez překladu) se skrývá funkce, která reportuje souhrné statistiky vstupu. Například nejčastěji používanou pooling vrstvou je tzv. *max pooling* s parametry k_1, k_2 (angl. *kernel-size*), která při aplikaci na obrázek o rozměrech $C \times W_{in} \times H_{in}$ (počet kanálů, šířka, výška) v každém kanálu reportuje maximální hodnotu v blocích o rozměrech $k_1 \times k_2$. Potom zobrazení ϕ je zobrazení $\phi : \mathbb{R}^{C \times W_{in} \times H_{in}} \to \mathbb{R}^{C \times W_{out} \times H_{out}}$, kde platí:

$$W_{out} = \left\lceil \frac{W_{in}}{k_1} \right\rceil, \tag{1.8}$$

$$H_{out} = \left\lceil \frac{H_{in}}{k_2} \right\rceil, \tag{1.9}$$

a má předpis $\forall i \in \{1, ..., C\}, \forall j \in \{1, ..., W_{out}\}, \forall k \in \{1, ..., H_{out}\}$:

$$\phi(u)_{i,j,k} = \max\{u_{i,\mu,\nu} | (j-1)k_1 < \mu \leq jk_1, (k-1)k_2 < \nu \leq kk_2\}. \tag{1.10}$$

1.1.4 Aktivační vrstva

Aktivační vrstva označuje vrstvu, která slouží k omezení aktivací jiné vrstvy, aby byly v rozumných mezích. Např. jedná-li se o poslední vrstvu klasifikační neuronové sítě, pak aktivační vrstva zajišťuje, aby výsledné aktivace byly pravděpodobnostní distribucí.

Mezi často používané aktivační vrstvy patří funkce, jež vzniknou aplikací skalární funkce jedné proměnné $\sigma : \mathbb{R} \to \mathbb{R}$ na každý prvek vstupu zvlášť. Pro takové skalární funkce pak máme pojem aktivační funkce. Nejčastější aktivační funkce jsou následující:

- Sigmoid: $\sigma(z) = \frac{1}{1+e^{-z}}$,
- ReLU: $\sigma(z) = \max(0, z)$,
- LeakyReLU: $\sigma(z) = \max(0, z) + \alpha * \min(z, 0)$, kde $\alpha > 0$,
- Tanh: $\sigma(z) = tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$.

Další oblíbenou aktivační vrstvou je *softmax vrstva*. Ta má pro odpovídající funkci ϕ, která v tomto případě zobrazuje vektor na vektor stejných rozměrů (tedy $\phi : \mathbb{R}^{p_1} \to \mathbb{R}^{p_1}$) předpis:

$$\forall i \in \{1, 2, ..., p_1\} \quad \phi(u)_i = \frac{e^{u_i}}{\sum_{j=1}^{p_1} e^{u_j}}. \tag{1.11}$$

Užití této aktivační vrstvy je nasnadě. Jelikož prvky výsledné aktivace leží v intervalu $[0, 1]$ a sečtou se na 1, lze výstup takovéto aktivační vrstvy interpretovat jako pravděpodobnostní distribuci.

1.2 Hluboká dopředná neuronová síť

Nejjednodušším modelem neuronové sítě je *hluboká dopředná neuronová síť*, která je složením hustých a aktivačních vrstev. Konkrétně je odpovídající zobrazení F složením sudého počtu vrstev, kde na liché pozici je vrstva hustá a na sudé pozici je vrstva aktivační. Tedy $F : \mathbb{R}^{n_1} \to \mathbb{R}^{m_1}$.

Motivace za pojmenováním takovéhoto zobrazení jako hluboké dopředné neuronové sítě je následující: Pojmem *neuronová síť* se rozumí složení každé jednotlivé dvojvrstvy $\varphi = \phi_{activation} \circ \phi_{dense}$ (hustá vrstva ϕ_{dense} spojená s následující aktivační vrstvou $\phi_{activation}$) z mnoha tzv. *umělých neuronů* - dílčích výpočetních jednotek, které mají přepis

$$\varphi(u)_i = \sigma\left(b_i + \sum_{j=1}^{n} w_{i,j} u_j\right), \tag{1.12}$$

kde b_i je i-tá složka vektoru prahů husté vrstvy, $w_{i,j}$ je složka v i-tém řádku a j-tém sloupci matice vah husté vrstvy a σ je aktivační funkce příslušející aktivační vrstvě. Takto definovaný umělý neuron vzdáleně připomíná neuron v biologickém smyslu, neboť má mnoho vstupů a jeden výstup. Tímto způsobem zavedené umělé neurony jsou potom pospojovány v neuronovou síť.

Za pojmem *dopředná* v názvu hluboká dopředná neuronová síť stojí fakt, že informace plyne od vstupu první vrstvy až po aktivace poslední vrstvy v jediném směru, který je určen architekturou sítě.

Termín *hluboká* je potom zaveden pro sítě, které mají více než jednu dvojvrstvu.

1.3 Konvoluční neuronová síť

Pojmem *konvoluční neuronová síť* je myšleno složení vrstev neuronů, z nichž alespoň jedna je konvoluční. Standardně je konvoluční vrstva používána společně s aktivační vrstvou a pooling vrstvou, a tedy tvoří konvoluční trojvrstvu $\varphi = \phi_{pooling} \circ \phi_{activation} \circ \phi_{convolution}$, kde $\phi_{convolution}$ je konvoluční vrstva, $\phi_{activation}$ je aktivační vrstva a $\phi_{pooling}$ je pooling vrstva. Takovýchto trojvrstev může být v konvoluční síti několik za sebou a následovat může několik vrstev hustých spolu s aktivačními. Takto zavedená konvoluční trojvrstva má velmi vítanou vlastnost, totiž že výsledná síť je do určité míry invariantní vůči translacím vstupu [1].

Kapitola 2

Učení neuronové sítě

Předchozí kapitola představila neuronové sítě jakožto složení vrstev neuronů. Jednotlivé vrstvy jsou ovšem parametrizovány parametry, o nichž není jasné, jak je nastavit. Například hustá vrstva má za parametry matici vah W a vektor prahů b. Označme tedy písmenem θ vektor všech parametrů neuronové sítě a poznamenejme závislost zobrazení neuronové sítě na parametrech θ dolním indexem v F_θ. Hledání vhodných parametrů θ je potom označováno pojmem *učení neuronové sítě*.

Standardní přístup k učení je paradigma učení s učitelem. Tento pohled na učení neuronové sítě předpokládá existenci tzv. *trénovací sady dat* $\mathbb{T}$ (angl. *training dataset*), což je uspořádaná dvojice obsahující množinu *vzorků* $\mathbb{X} = \left\{x^{(i)} | i \in \{1, ..., N\}\right\}$ a k nim příslušné *značky* $\mathbb{Y} = \left\{y^{(i)} | i \in \{1, ..., N\}\right\}$, kde pojem vzorek představuje vstup neuronové sítě a pojem značka představuje správný výstup neuronové sítě; N je potom velikost trénovací sady $\mathbb{T}$. Trénovací sada pak hraje roli učitele.

2.1 Účelové funkce

Je-li pojem trenovací sady objasněn, lze přistoupit k termínu *účelové funkce* nebo též *ztrátové funkce*. Jedná se o reálnou funkci, která měří, jak moc se trénovaná neuronová síť mýlí ve svých predikcích na vzorcích trénovací sady. Úloha učení je potom převedena na úlohu optimalizace tohoto vhodně zvoleného kritéria.

Standardní účelová funkce je sestavena jako průměr dílčích ztrát, které neuronová síť dosahuje na vzorcích trénovací sady:

$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} L\left(F_\theta(x^{(i)}), y^{(i)}\right), \tag{2.1}$$

kde L značí konkrétní ztrátu pro daný vzorek a J je celková účelová funkce.

2.1.1 Střední kvadratická chyba

Jedna z klasických účelových funkcí je funkce *střední kvadratické chyby*. Je dána přepisem:

$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \|F_\theta(x^{(i)}) - y^{(i)}\|_2^2, \tag{2.2}$$

kde $\|\cdot\|_2$ je l_2 norma.

Výhodou této účelové funkce je fakt, že ji lze aplikovat na tenzory libovoných rozměrů. Nahlédneme-li na výraz v (2.2), $J(\theta)$ nabývá vždy nezáporné hodnoty a globální minimum 0 právě tehdy, když pro každý vzorek trénovací datové sady je předpověď neuronové sítě správná.

Další vlastností této účelové funkce je rozdíl v citlivosti na malé hodnoty výrazu $|F_\theta(x^{(i)})_j - y_j^{(i)}|^2$, ze kterých výsledná l_2 norma sestává, oproti jeho velkým hodnotám. Tj. pro $|F_\theta(x^{(i)})_j - y_j^{(i)}| < 1$ je výraz po umocnění na druhou ještě menší, kdežto pro $|F_\theta(x^{(i)})_j - y_j^{(i)}| > 1$ je výraz po umocnění ještě větší, což při aplikaci později popsaných algoritmů minimalizace ztráty, které využívají gradient účelové funkce, vede k větší toleranci malých odchylek, než kdyby byla použita l_1 norma.

2.1.2 Ztráta křížové entropie

Pro klasifikační problémy se ovšem standardně používá *ztráta křížové entropie*. Připomeňme, že u klasifikačního problému je výstup neuronové sítě pravděpodobnostní distribuce a značky jsou též pravděpodobnostní distribuce. Křížová entropie potom měří vzdálenost distribučních funkcí a má svůj původ v *Kullbackově-Leiblerově divergenci* D_{KL}. Máme-li dvě pravděpodobnostní distribuce f a g, pak křížová entropie $H(f, g)$ je rovna

$$H(f, g) = H(f) + D_{KL}(f, g), \tag{2.3}$$

kde $H(f)$ je entropie f. Pro diskrétní pravděpodobnostní distribuce máme:

$$H(f, g) = -\sum_i f_i \ln(f_i) + \sum_i f_i \ln\left(\frac{f_i}{g_i}\right) = -\sum_i f_i \ln(g_i). \tag{2.4}$$

Proto lze psát:

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N} H(y^{(i)}, F_\theta(x^{(i)})). \tag{2.5}$$

Onen výraz $H(y^{(i)}, F_\theta(x^{(i)}))$ v (2.5) lze tedy spočíst následovně:

$$H(y^{(i)}, F_\theta(x^{(i)})) = -\sum_{j=1}^{m} y_j^{(i)} \cdot \ln(F_\theta(x^{(i)})_j). \tag{2.6}$$

2.2 Algoritmus zpětného šíření chyby

Nejčastější metody učení neuronové sítě ve svém chodu pracují s gradientem účelové funkce podle parametrů neuronové sítě $\nabla_\theta J(\theta)$, který lze spočíst pomocí *algoritmu zpětného šíření chyby* (angl. *backpropagation*). Tento algoritmus však lze použít nejen v takto úzce specializovaném prostředí strojového učení, nýbrž i pro výpočet Jacobiho matice libovolné funkce (dle [1]).

Algoritmus stojí na opakované aplikaci *řetězového pravidla* pro výpočet derivace složené funkce. Proto zde řetězové pravidlo uveďme. Nechť $g : \mathbb{R}^n \to \mathbb{R}^m$ a $h : \mathbb{R}^m \to \mathbb{R}^p$, $a \in \mathbb{R}^n$, potom:

$$D(h \circ g)(a) = Dh(g(a))Dg(a), \tag{2.7}$$

kde D značí totální diferenciál. Zúžíme-li se na $p = 1$, dostáváme:

$$\nabla(h \circ g)(a) = \nabla h(g(a))Dg(a), \tag{2.8}$$

podíváme-li se na i-tou komponentu gradientu $h \circ g$:

$$\partial_i(h \circ g)(a) = \sum_{j=1}^{m} \partial_j h(g(a)) \cdot \partial_i g_j(a), \tag{2.9}$$

kde g_j značí j-tou komponentu vektorové funkce g. Tedy jak lze vidět v (2.8), pro algoritmus bude stěžejní násobení vektoru gradientu s maticí totálního diferenciálu.

Pro neuronové sítě algoritmus zpětného šíření chyby postupuje zpět celou neuronovou sítí a počítá dle řetězového pravidla parciální derivace účelové funkce dle parametrů neuronové sítě. V praxi je ovšem snadné natrefit na velmi složité neuronové sítě, které vedou k vyhodnocování mnoha podvýrazů v jednotlivých krocích algoritmu. Navíc mnoho takovýchto podvýrazů může být stejných. Proto je při implementaci namístě otázka, zda již vyhodnocené výrazy uložit do paměti, či je pokaždé vyhodnotit znovu. Je-li žádoucí co nejkratší doba běhu, pak je odpovědí vyhodnocené výrazy ukládat, neboť jejich získání z paměti počítače je mnohem rychlejší než opakované počítání. Ovšem při nedostatečné kapacitě paměti počítače není mnohdy možné ukládat všechny mezivýpočty, proto je implementováno jejich opakované počítání na úkor času běhu algoritmu.

2.3 Základní algoritmy učení

2.3.1 Gradientní sestup

Základním algoritmem pro učení neuronové sítě je *gradientní sestup* (angl. *gradient descent*). Opírá se o fakt, že gradient reálné funkce určuje směr největšího růstu dané funkce v daném bodě. Proto, máme-li účelovou funkci $J(\theta)$, kde θ jsou parametry neuronové sítě, má smysl tyto parametry aktualizovat proti směru gradientu funkce J následujícím způsobem:

$$\theta \leftarrow \theta - \epsilon \cdot \nabla_\theta J(\theta), \tag{2.10}$$

kde ϵ je tzv. *řád učení* (angl. *learning rate*) - kladné číslo, které určuje velikost jednoho kroku; jedná se o tzv. *hyper-parametr* neuronové sítě. Takovouto aktualizaci parametrů neuronové sítě lze provést několikrát, a to například tolikrát, dokud účelová funkce nedosáhne přijatelné hodnoty. Ideální by bylo, kdybychom gradientním sestupem dosáhli globálního minima účelové funkce, to ovšem není v žádném případě zaručeno, že se stane, gradientní sestup totiž dokáže nalézt pouze lokální minimum - ale to je pro reálné aplikace mnohdy dostačující.

2.3.2 Metoda hybnosti

Modifikací gradientního sestupu je tzv. *metoda hybnosti* [2]. Ta uvádí na scénu novou proměnnou - *rychlost* v (z angl. *velocity*), která je stejných rozměrů jako gradient účelové funkce a nese v sobě informaci o předchozích odhadech gradientu účelové funkce. Její role v algoritmu učení je následující:

$$v \leftarrow \alpha \cdot v - \epsilon \cdot \nabla_\theta J(\theta), \tag{2.11}$$

$$\theta \leftarrow \theta + v. \tag{2.12}$$

Užití hybnosti vede tedy k představení dalšího hyper-parametru, a to parametru $\alpha \in [0, 1)$, který určuje míru ovlivnění dalšího kroku předchozími odhady gradientu. Dle [1] jsou za hodnoty tohoto parametru nejčastěji volena čísla 0.5, 0.9 a 0.99.

Motivací k zavedení metody hybnosti je urychlení konvergence algoritmu, obzvlášť pro případy, kdy je gradient účelové funkce buď malý (má složky o malých velikostech v absolutní hodnotě), nebo příliš nestálý. V prvním případě přidává proměnná rychlosti výslednému kroku iterace na velikosti, tedy teoreticky urychluje konvergenci, a v druhém případě přidává konzistenci výsledným krokům iterací.

Je tu však riziko, že při nešťastném nastavení hyper-parametrů ϵ a α v průběhu učení nedojde k velkým změnám proměnné rychlosti. Může se tedy stát, nemíří-li prvně spočtený gradient správným směrem, že bude učení odsouzeno k neúspěchu.

2.3.3 Metoda Něstěrovovy hybnosti

Jinou modifikací gradientního sestupu, která je obdobou hybnosti, je *metoda Něstěrovovy hybnosti*. Ta má následující předpis iterace [3]:

$$v \leftarrow \alpha \cdot v - \epsilon \cdot \nabla_\theta J(\theta + \alpha \cdot v), \tag{2.13}$$

$$\theta \leftarrow \theta + v. \tag{2.14}$$

Zásadním rozdílem oproti metodě hybnosti je, kde se vyhodnocuje $\nabla_\theta J$. V tomto algoritmu se totiž gradient nevyhodnocuje θ, nýbrž v bodě $\theta + \alpha \cdot v$, tedy po aplikaci proměnné rychlosti na parametry. Za následek tento přístup má, že oproti klasické metodě hybnosti je gradient přesnější.

Nevýhodou této metody je ovšem vyšší výpočetní náročnost, jelikož se účelová funkce a její derivace musejí vyhodnocovat v jiném bodě.

2.4 Algoritmy s přizpůsobivým řádem učení

Existují další algoritmy, které pracují s proměnným řádem učení. Jedná se o *algoritmy s přizpůsobivým řádem učení*: *AdaGrad*, *RMSProp* a *Adam*. Tyto algoritmy přizpůsobují řád učení jednotlivým parametrům zvlášť.

2.4.1 AdaGrad

Algoritmus *AdaGrad* (z angl. *adaptive gradient*) dle [4] přizpůsobuje řád učení každému parametru jednotlivě, a to jeho škálováním nepřímo úměrně druhé odmocnině součtu všech hodnot gradientu, jež danému parametru v průběhu učení příslušel. To vede k tomu, že parametry, kterým přísluší velké hodnoty parciálních derivací účelové funkce, mají úměrně tomu rychlý úbytek v řádu učení, zatímco parametry, kterým přísluší malé hodnoty parciálních derivací učelové funkce, mají úměrně tomu pomalý úbytek v řádu učení. Celkový efekt tedy je, že se síť pohybuje rychleji ve směrech menšího spádu. Jedna iterace potom vypadá následovně:

$$g \leftarrow \nabla_\theta J(\theta), \tag{2.15}$$

$$r \leftarrow r + g \odot g, \tag{2.16}$$

$$\theta \leftarrow \theta - \frac{\epsilon}{\delta + \sqrt{r}} \odot g, \tag{2.17}$$

kde δ je malé číslo (např. 10^{-7}) pro numerickou stabilitu, $\odot$ značí Hadamardův součin a výraz zlomku a odmocniny na třetím řádku je myšlen po složkách.

Poznamenejme, že dle [6] algoritmus AdaGrad funguje dobře s řídkými gradienty. Nevýhoda tohoto algoritmu ovšem je jeho paměť - ve svých proměnných střádá velmi vzdálené hodnoty gradientu, což dle [1] mnohdy vede k předčasnému poklesu řádu učení.

2.4.2 RMSProp

Uveďme další algoritmus - *RMSProp* (zkratka angl. *root mean squared propagation*). Tento algoritmus nahrazuje součet přes všechny hodnoty gradientu exponenciálně tlumeným váženým průměrem, a

to způsobem, kde jedna iterace vypadá následovně [5]:

$$g \leftarrow \nabla_\theta J(\theta), \tag{2.18}$$

$$r \leftarrow \rho \cdot r + (1-\rho) \cdot g \odot g, \tag{2.19}$$

$$\theta \leftarrow \theta - \frac{\epsilon}{\delta + \sqrt{r}} \odot g, \tag{2.20}$$

Objevil se tu však nový hyper-parametr $\rho \in [0, 1)$ nazývaný *decay rate* (bez překladu).

RMSProp se ukazuje jako jeden z nejúspěšnějších algoritmů, proto je dnes v praxi jedním z nejpoužívanějších [1]. Důvodem je jeho vhodnost pro nekonvexní optimalizaci, která je v kontextu neuronových sítí naprosto zásadní. Oproti algoritmu AdaGrad, který se ukázal jako vhodný pro konvexní optimalizaci, se jedná o bezespornou výhodu.

2.4.3 Adam

Posledním představeným algoritmem je algoritmus *Adam*, který nese název z anglického *adaptive moments*, což přeloženo do češtiny zní jako přizpůsobivé momenty. V prvním přiblížení se jedná o kombinaci algoritmu RMSProp a metody hybnosti. Ve skutečnosti však je hybnost zakomponována již v následujícím, a to sice v odhadu prvního obecného momentu gradientu. Druhým aspektem, ve kterém se algoritmus liší od prostého RMSProp s hybností, jsou korekce prováděné na odhadech prvního a druhého obecného momentu gradientu. Jedna iterace algoritmu vypadá [6]:

$$g \leftarrow \nabla_\theta J(\theta), \tag{2.21}$$

$$s \leftarrow \rho_1 \cdot s + (1-\rho_1) \cdot g, \tag{2.22}$$

$$r \leftarrow \rho_2 \cdot r + (1-\rho_2) \cdot g \odot g, \tag{2.23}$$

$$\hat{s} \leftarrow \frac{s}{1-\rho_1^t}, \tag{2.24}$$

$$\hat{r} \leftarrow \frac{r}{1-\rho_2^t}, \tag{2.25}$$

$$\theta \leftarrow \theta - \frac{\epsilon}{\delta + \sqrt{\hat{r}}} \odot \hat{s}, \tag{2.26}$$

kde t je pořadí iterace a $\rho_1, \rho_2 \in [0, 1)$ jsou hyper-parametry nazvané *decay rate*.

2.5 Stochastické algoritmy učení

Výše zmíněné metody, jak je patrné z jejich předpisů, počítají gradient účelové funkce $\nabla_\theta J(\theta)$. Tento krok je ovšem velmi časově náročný, protože standardní trénovací sady mívají velmi mnoho vzorků. Při připomenutí (2.1) se výpočet sestává z N výpočtů dílčích gradientů:

$$\nabla_\theta J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta L\left(F_\theta(x^{(i)}), y^{(i)}\right). \tag{2.27}$$

Po aplikaci řetězového pravidla:

$$\nabla_\theta J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \nabla_\xi L\left(F_\theta(x^{(i)}), y^{(i)}\right) D_\theta F_\theta(x^{(i)}), \tag{2.28}$$

kde došlo k označení $L = L(\xi, \eta)$.

Proto je doporučenou praxí dle [1] aproximovat gradient účelové funkce $\nabla_\theta J(\theta)$ pomocí výpočtu na tzv. *mini-dávce* (z angl. *mini-batch*). Jedná se v každém kroku gradientního sestupu nebo jeho modifikací o to, že se z trénovací sady rovnoměrně vybere $M \ll N$ vzorků gradient se odhadne pomocí výpočtu na těchto M vzorcích:

$$\nabla_\theta J(\theta) \approx \frac{1}{M} \sum_{j=1}^{M} \nabla_\xi L\left(F_\theta(x^{(i_j)}), y^{(i_j)}\right) D_\theta F_\theta(x^{(i_j)}). \tag{2.29}$$

Číslo M lze vybírat dle [1] v řádu jednotek až stovek. Při aplikaci této aproximace během standardního gradientního sestupu se algoritmu říká *stochastický gradientní sestup* (angl. *stochastic gradient descent*), ovšem tento úkrok stranou lze provést i v případě ostatních představených algoritmů, ty však pro svou stochastickou variantu nemají speciální název.

Kapitola 3

Adversariální vzorky

Szegedy a spol. [7] objevili zvláštní chování klasifikační neuronové sítě, které spočívá v nesprávné klasifikaci mírně pozměněných vzorků trénovací sady neuronové sítě, kde ono mírné pozměnění nemění správnost příslušné značky.

Zaveďme funkci $C : \mathbb{R}^{m_1} \to \{1, 2, ..., m_1\}$, tedy funkci na prostoru značek, resp. výstupů klasifikační neuronové sítě, která přiřadí každému vektoru index odpovídající třídy, což lze formálně zapsat následovně:

$$C(y) = \text{argmax}_{i \in \{1,2,...,m_1\}} y_i. \tag{3.1}$$

Potom máme-li vzorek x a k němu příslušnou značku y říkáme, že $\tilde{x} = x + \Delta x$ je adversariální vzorek, pokud je splněno následující:

$$||\Delta x|| < \kappa \wedge C(F_\theta(\tilde{x})) \neq C(y), \tag{3.2}$$

kde $||.||$ je l_p norma a κ je malé číslo. Povšimněme si, že je-li původní vzorek špatně klasifikován, tedy $C(F_\theta(x)) \neq C(y)$, pak sám původní vzorek je adversariální.

Takto obecná definice adversariálních vzorků ovšem neposkytuje návod na jejich nalezení. Proto uveďme metody generování těchto adversariálních vzorků, které slouží k ověření jejich existence. Předtím ovšem pojmenujme neuronovou síť, která je terčem adversariálního útoku, jako *oběť* (angl. *victim*), dále pojmenujme strůjce takovéhoto adversariálního útoku jako *útočníka* (angl. *adversary*).

3.1 Metody generování adversariálních vzorků

Metody generování adversariálních vzorků se dělí na dvě kategorie dle míry znalosti útočníka o oběti. Nemá-li útočník znalost o oběti, hovoří se o tzv. *black-box metodě*. V opačném případě - má-li útočník kompletní znalost o oběti - se hovoří o tzv. *white-box metodě*. Tento text se zabývá pouze white-box metodami, neboť v black-box nastavení si může útočník natrénovat svou vlastní neuronovou síť a generovat adversariální vzorky proti ní - díky jevu *přenositelnosti* (angl. *transferability*) jsou tyto vzorky použitelné i proti původní síti [14].

Dále se metody generování adversariálních vzorků dělí na *cílené* (angl. *targeted*) a *necílené* (angl. *untargeted*). Cílené útoky generují vzorky $\tilde{x} = x + \Delta x$ tak, aby $C(F_\theta(\tilde{x})) = C(\tilde{y})$ pro pevně zvolenou značku $\tilde{y}$ různou od původní značky $C(y)$. Necílené útoky předem nevybírají značku za cíl, nýbrž požadavkem je jen, aby $C(F_\theta(\tilde{x})) \neq C(y)$. Necílené útoky nebývají tolik účinné jako cílené [13].

3.1.1 Fast gradient sign method

První metoda představená v [8] je známá pod zkratkou *FGSM* (z angl. *fast gradient sign method*). Jedná se o necílenou metodu s předpisem [10]:

$$\tilde{x} = x + \kappa \cdot \text{sign}(\nabla_x L(F_\theta(x), y)) \tag{3.3}$$

při zachování značení z minulých kapitol textu, označení sign pro znaménkovou funkci a κ pro velikost složek perturbace Δx, tedy κ je poloměr kulového okolí x v l_∞ normě.

3.1.2 Iterativní FGSM

Druhá metoda jde o krok dál, vzorec (3.3) aplikuje iterativně několikrát a generuje posloupnost $(\tilde{x}_n)_{n=0}^{K}$, kde K je počet iterací metody. Jedná se o metodu *I-FGSM* (z angl. *iterative fast gradient sign method*) představenou v [11] s předpisem:

$$\tilde{x}_0 = x \tag{3.4}$$

$$\tilde{x}_{n+1} = \text{Clip}_x^{\kappa}\{\tilde{x}_n + \gamma \cdot \text{sign}(\nabla_x L(F_\theta(x), y))\}, \tag{3.5}$$

kde funkce Clip omezuje výsledný součet, aby byl v κ-okolí původního vzorku x a zároveň platným vstupem pro neuronovou síť F_θ - například jsou-li vzorky obrázky, funkce Clip zajišťuje i to, aby hodnoty pixelů nebyly záporné či vyšší než 255. Počet iterací je ovšem dalším hyper-parametrem, který je nutno nastavit. Jedná se o necílenou metodu.

3.1.3 Projected gradient descent

Další metoda (necílená) nese název *PGD* (zkratka angl. *projected gradient descent*). Tato metoda je silnější variantou I-FGSM [10] a spočívá v náhodné inicializaci vzorku $\tilde{x}_0$ uvnitř κ-okolí původního vzorku a následných iteracích jako v I-FGSM [12].

3.1.4 Cílená optimalizační úloha

Čtvrtá metoda (cílená) nahlíží na generování adversariálních vzorků jako na optimalizační úlohu [7], [13]:

$$\tilde{x} = \text{argmin}_{\hat{x}} \left(\|\hat{x} - x\| + \lambda \cdot L(F_\theta(\hat{x}), \tilde{y}) \right), \tag{3.6}$$

kde $\tilde{y}$ značí cílenou nesprávnou značku, $\lambda > 0$ je zvoleno pomocí *line-search* algoritmu jako minimální hodnota, pro kterou platí, že $C(F_\theta(\tilde{x})) = C(\tilde{y})$. Tento optimalizační problém lze řešit algoritmem *L-BFGS* [9], resp. jeho variantou s vazbami (angl. *box-constrained L-BFGS*) nebo pomocí algoritmu *sign gradient descent*, což je algoritmus odvozený od standardního gradientního sestupu.

Na význam parametru λ lze nahlédnout následovně: Bude-li λ příliš velké, při řešení (3.6) nedojde k tomu, aby se $\tilde{x}$ podobalo původnímu vzorku x, a tedy nebude v κ-okolí x. Bude-li λ příliš malé, bude při řešení (3.6) kladen příliš velký důraz na první člen účelové funkce, takže řešením optimalizačního problému v (3.6) bude původní vzorek x. Je tedy třeba vhodné λ najít.

3.1.5 Carlini-Wagner

Následující metoda (necílená) nese název *CW* (*Carlini-Wagner*) a má předpis [13], [10]:

$$\tilde{x} = \text{argmin}_{\hat{x}} \left(\|\hat{x} - x\| - \lambda \cdot L(F_\theta(\hat{x}), y) \right), \tag{3.7}$$

kde $\lambda > 0$. K nastavení parametru λ lze přistoupit obdobně jako u předchozí metody a jeho hodnotu hledat pomocí *line-search* algoritmu, nebo hodnotu λ nastavit pevně jako *hyper-parametr* pro celou proceduru generování adversariálních vzorků. Při volbě hledání vhodné hodnoty parametru λ metodou line-search je zapotřebí hledat opět jeho minimální hodnotu, pro kterou bude platit, že $C(F_\theta(\tilde{x})) \neq C(y)$.

Lze si povšimnout jisté podoby CW útoku s cílenou optimalizační úlohou. Obě metody řeší optimalizační úlohu, kde první člen účelové funkce je vzdálenost argumentu minima od původního vzorku ve smyslu l_p normy a druhý člen je vážená účelová funkce, kde onu váhu reprezentuje hodnota λ, kterou ladíme. Zásadní rozdíl ovšem je ve znaménku, které stojí před členem obsahujícím účelovou funkci, a ve značce, která je druhým argumentem účelové funkce. Zatímco v případě cílené optimalizační úlohy se snažíme minimalizovat v jistém smyslu vzdálenost klasifikace adversariálního vzorku od cílené nesprávné značky, v případě CW útoku maximaluzujeme vzdálenost klasifikace adversariálního vzorku od správné značky, proto je v případě CW útoku druhý člen se znaménkem mínus.

Kapitola 4

Robustní učení neuronové sítě

Pro vývojáře neuronových sítí, ale i dalších modelů strojového učení je standardně jev adversariálních vzorků nežádoucí, proto vznikají pokusy o předejití existence těchto adversariálních vzorků. Souhrně se tyto snahy nazývají jako *robustní strojové učení*. Tento název je motivován faktem, že výsledkem těchto metod je model strojového učení, který je robustní vůči adversariálním útokům.

4.1 Optimalizační pohled

4.1.1 Formulace robustního učení jako optimalizačního problému

Jako se učení neuronové sítě převádí na optimalizační problém, tak i robustní učení neuronové sítě lze vyjádřit jako optimalizační problém. Ovšem v případě robustního učení je problém optimalizace dvojitý:

$$\theta = \operatorname{argmin}_{\hat{\theta}} \frac{1}{N} \sum_{i=1}^{N} \max_{\tilde{x} \in B(x^{(i)}, \kappa)} L\left(F_{\hat{\theta}}(\tilde{x}), y^{(i)}\right), \tag{4.1}$$

kde θ jsou hledané parametry neuronové sítě a $B(x^{(i)}, \kappa)$ je koule se středem v bodě $x^{(i)}$ a poloměrem κ ve vhodně zvolené normě.

Interpretace tohoto přístupu je následující. Ztrátová funkce L svým charakterem určuje ne nutně metrickou vzdálenost mezi predikcemi neuronové sítě a správnými značkami. Tohoto faktu využívají techniky generování adversariálních vzorků tím, že se tuto vzdálenost snaží zvětšit na malém okolí správně klasifikovaného vzorku. Robustní učení popsané v (4.1) má za cíl najít takové parametry neuronové sítě, které zaručují, že i na okolí vzorku trénovací sady nabývá ztrátová funkce nízkých hodnot, tedy predikce neuronové sítě jsou blízko značce původního vzorku na celém tomto okolí. Je ovšem nutné vyslovit předpoklad, který dává této metodě smysl, totiž, že se předpokládá, že na celém okolí původního vzorku platí, že značka libovolného bodu z tohoto okolí je tatáž jako původního vzorku.

4.1.2 Řešení optimalizačního problému robustního učení

Takto formulovaný optimalizační problém lze řešit též algoritmem gradientního sestupu i dalšími algoritmy od něho odvozenými, ovšem s tím rozdílem, že roli účelové funkce hraje:

$$\hat{J}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \max_{\tilde{x} \in B(x^{(i)}, \kappa)} L\left(F_{\theta}(\tilde{x}), y^{(i)}\right). \tag{4.2}$$

To přináší nutnost v každém kroku algoritmu minimalizace řešit maximalizační problém, tedy najít pro každé $i \in \{1, ..., N\}$ takový vzorek $\hat{x}^{(i)} = \text{argmax}_{\tilde{x} \in B(x^{(i)}, \kappa)} L\left(F_\theta(\tilde{x}), y^{(i)}\right)$, ve kterém dojde k vyčíslení gradientu účelové funkce $\nabla_\theta \hat{J}$. Tento maximalizační problém lze řešit jako hledání adversariálních vzorků, tedy například metodami I-FGSM či PGD představenými v předcházející kapitole. Následně se v takto nalezených bodech $\hat{x}^{(i)}$ vyčíslí gradient účelové funkce:

$$\nabla_\theta \hat{J}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \nabla_\xi L\left(F_\theta(\hat{x}^{(i)}), y^{(i)}\right) D_\theta F_\theta(\hat{x}^{(i)}) \tag{4.3}$$

a provede se krok vnějšího minimalizačního problému dle příslušného algoritmu. Jen poznamenejme, že tento algoritmus lze provést též na mini-dávce o $M \ll N$ vzorcích.

Kapitola 5

Srovnání algoritmů učení

5.1 Kritérium srovnávání

Pro účely srovnávání algoritmů učení neuronové sítě lze zvolit mnoho kritérií. Jedním z nich by mohl být samotný průběh účelové funkce v závislosti na počtu provedených iterací vybraného algoritmu, když všechny představené algoritmy mají iterativní charakter.

Jiným přístupem je užití tzv. *testovací sady* $\mathbb{S}$ (angl. *test dataset*). Svou strukturou testovací sada kopíruje sadu trénovací, jedná se tedy o uspořádanou dvojici množin vzorků $\mathbb{X} = \left\{x^{(i)} | i \in \{1, ..., S\}\right\}$ a značek $\mathbb{Y} = \left\{y^{(i)} | i \in \{1, ..., S\}\right\}$, kde S je velikost testovací sady.

Je-li neuronová síť svým charakterem síť klasifikační, pak lze sledovat podíl správných predikcí na testovacím datasetu vůči celkovému počtu vzorků. Výhodou tohoto přístupu je fakt, že při svém učení neuronová síť na vzorky testovacího datasetu nenarazila, což má za důsledek to, že lze očekávat stejnou úspěšnost sítě při její aplikaci. Tento přístup je využit v tomto textu.

5.2 Inicializace parametrů sítě a stochasticita algoritmu učení

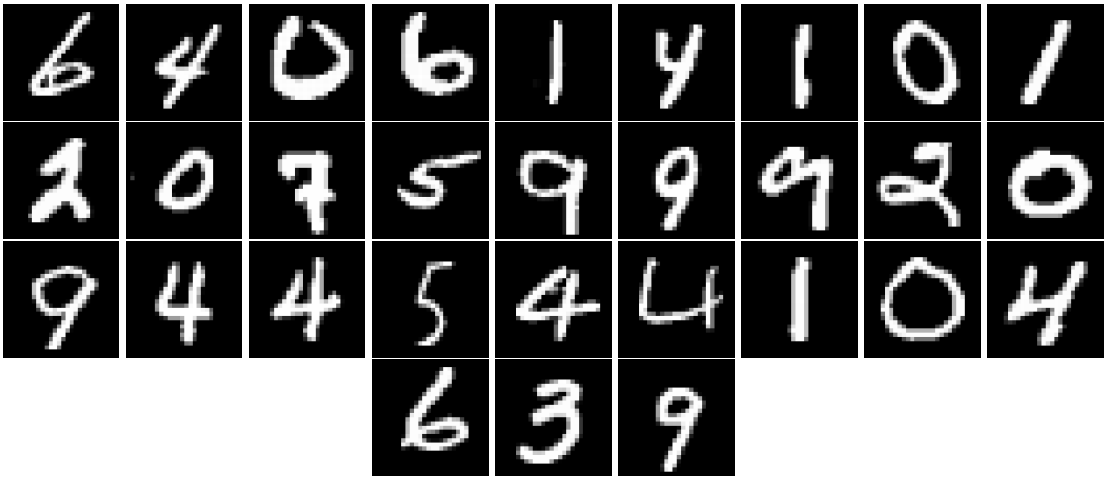
Nyní je namístě vyslovit poznámku o inicializaci parametrů neuronové sítě před samotným učením. Dle [1] je standardním postupem pro inicializaci vybírat hodnoty parametrů náhodně, a to z rovnoměrného rozdělení na rozumném intervalu. Konkrétní experimenty v tomto textu pracují s následujícím rozdělením vah, prahů a prvků konvolučních jader:

$$w_{i,j}, b_i, k_{i,j} \sim U\left(-\frac{1}{\sqrt{n}}, +\frac{1}{\sqrt{n}}\right), \tag{5.1}$$

kde n je v případě parametrů husté vrstvy počet sloupečků matice vah, v případě konvolučních vrstev je n rovno součinu počtu vstupních kanálů se součinem rozměrů konvolučních jader. Závěrem této poznámky tedy je, že inicializace parametrů neuronové sítě je náhodný proces. To má za důsledek fakt, že na proces učení neuronové sítě lze nahlížet očima statistika. Tento text konkrétně nahlíží na úspěšnost neuronové sítě na testovací sadě jako na náhodnou veličinu. Potom lze totiž porovnávat jednotlivé algoritmy na základě distribuční funkce této specifické náhodné veličiny.

5.3 Datová sada MNIST

Nedílnou ingrediencí pro srovnání algoritmů učení je samotná sada dat a k nim příslušný úkol, zda se jedná o klasifikaci či o regresi. Tato část textu se věnuje úkolu klasifikace ručně psaných číslic z čer-



Obrázek 5.1: Datová sada MNIST

nobílého obrázku. Sada dat, která je zde použita je nazvána MNIST [15]. Její trénovací sada $\mathbb{T}$ obsahuje 60 000 vzorků (a k nim odpovídajících značek) a testovací sada $\mathbb{S}$ obsahuje 10 000 vzorků (a k nim odpovídajících značek). Vzorky jsou ve své podstatě matice o rozměrech 28 řádků a 28 sloupečků, jejichž prvky jsou nezáporná celá čísla o hodnotě nejvýše 255. Tyto matice lze interpretovat jako obrázky.

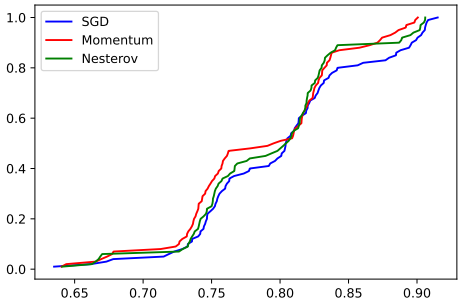
5.4 Výsledky

Přistupme nyní k samotnému srovnání algoritmů *stochastický gradientní sestup*, *metoda hybnosti* a *metoda Něstěrovovy hybnosti* (obě ve stochastické verzi). Pro srovnání těchto algoritmů byly provedeny následující dva experimenty: První se týká trénování jedné hluboké dopředné neuronové sítě těmito algoritmy pro úkol datové sady MNIST, jež je uvedena výše v textu, a to konkrétně aplikací 5 000 iterací algoritmu na nově inicializovanou síť. Pro stochastický gradientní sestup byl použit řád učení o hodnotě 10^{-2}, pro obě metody hybnosti byl použit řád učení 10^{-3} a koeficient $\alpha = 0.9$. Dále uveďme velikost mini-dávky $M = 30$ pro všechny tři algoritmy. V takovémto nastavení byly všechny tři algoritmy spuštěny stokrát. Na výsledné distribuční funkce lze nahlédnout v obrázku (5.2a). Z grafu lze vyčíst takřka zanedbatelný rozdíl mezi metodou hybnosti a metodou Něstěrovovy hybnosti. Dále graf vyjadřuje nemalou větší úspěšnost obyčejného stochastického gradientního sestupu.

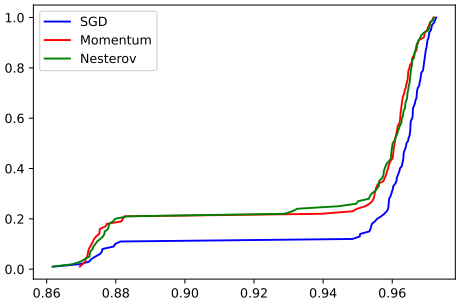
Druhý experiment je téměř totožný, jen je použit jiný model neuronové sítě, a to konkrétně se zakomponovanou konvolucí. Jinak je experiment totožný. Proto lze z Obr. (5.2b) odezřít výsledky, a to konkrétně, že stochastický gradientní sestup má v tomto nastavení lepší výkonnost.

Pro srovnání algoritmů *stochastický gradientní sestup*, *AdaGrad*, *RMSProp* a *Adam* lze využít podkladů na obrázku (5.2c), který zachycuje výsledky obdobných experimentů jako popsaných výše. Nastavení tohoto pokusu bylo následující: Pro stochastický gradientní sestup a algoritmus AdaGrad byl použit řád učení o hodnotě 10^{-2}, pro algoritmy RMSProp a Adam 10^{-3}. Pro AdaGrad bylo dále použito $\delta = 10^{-10}$, pro RMSProp $\delta = 10^{-8}$ a $\rho = 0.99$, pro Adam $\delta = 10^{-8}$, $\rho_1 = 0.9$ a $\rho_2 = 0.999$. Úkol byl stejný - natrénovat tentýž model dopředné neuronové sítě pro klasifikaci číslic datové sady MNIST za použití 5 000 iterací daného algoritmu. Učení sítě vždy proběhlo stokrát. Ze zmíněného obrázku vyplývá, že algoritmus AdaGrad je v tomto nastavení srovnatelný se stochastickým gradientním sestupem a že algoritmy RMSProp a Adam jsou minimálně pro toto specifické nastavení lepší.

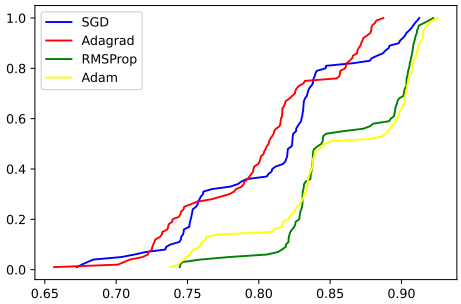
Dále se pro srovnání algoritmů *stochastický gradientní sestup*, *AdaGrad*, *RMSProp* a *Adam* lze opřít o výsledky vyobrazené na obrázku (5.2d). Ten zachycuje výsledky totožného nastavení jako obrázek



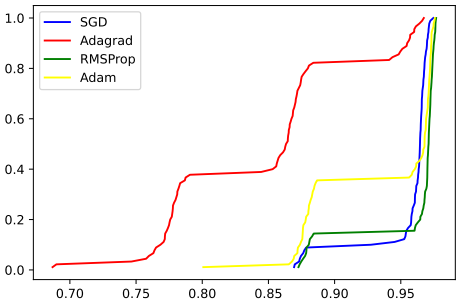
(a) Srovnání algoritmů učení I



(b) Srovnání algoritmů učení II



(c) Srovnání algoritmů učení III



(d) Srovnání algoritmů učení IV

Obrázek 5.2: Srovnání algoritmů učení
SGD - stochastický gradientní sestup; *Momentum* - metoda hybnosti; *Nesterov* - metoda Něstěrovovy hybnosti; *AdaGrad* - algoritmus AdaGrad; *RMSProp* - algoritmus RMSProp; *Adam* - algoritmus Adam.

(5.2c) jen s rozdílem použitého modelu. V tomto případě byl použit model konvoluční neuronové sítě. Jak lze nahlédnout, algoritmus AdaGrad byl pro tuto úlohu nevhodný. Algoritmus Adam dosáhl přijatelné úrovně neuronové sítě (tedy úspěšnost na testovací datové sadě vyšší než 95 %) zhruba v 60 % případů, algoritmus RMSProp zhruba v 90 % případů a stochastický gradientní sestup v 95 % případů. Ovšem kvalita přijatelně natrénovaných neuronových sítí byla v případě RMSProp vyšší než u stochastického gradientního sestupu.

Kapitola 6

Vygenerované adversariální vzorky

6.1 Srovnání metod

Přistupme nyní ke srovnání účinnosti metod generování adversariálních vzorků, které byly představeny výše, totiž metod *FGSM*, *I-FGSM*, *PGD*, *cílená optimalizační úloha* a *CW*.

6.1.1 Klasifikační úkol

Úkol, na kterém budeme metody srovnávat je tentýž jako v případě srovnání algoritmů učení. Jedná se tedy o problém klasifikace ručně psaných číslic datové sady MNIST [15]. Cílem metod bude tedy vytvořit perturbovaný obrázek číslice stejných rozměrů jako jsou obrázky oné datové sady s tím, že snahou bude vytvořit takovou perturbaci, pro kterou dojde po jejím přičtení k obrázku z datové sady k nesprávné klasifikaci.

6.1.2 Použitý model

Jelikož všechny tyto metody pracují na *white-box* principu, je nutné k jejich provedení mít k dispozici natrénovanou klasifikační neuronovou síť. Za tuto je zvolen model konvoluční neuronové sítě složené po řadě z vrstev:

1. Konvoluční vrstva s 1 vstupním kanálem, 32 výstupními kanály a jádrem konvoluce 3×3,
2. aktivační vrstva s funkcí ReLU,
3. konvoluční vrstva s 32 vstupními kanály, 32 výstupními kanály a jádrem konvoluce 3×3,
4. aktivační vrstva s funkcí ReLU,
5. max pooling vrstva s rozměry jádra 2×2,
6. konvoluční vrstva s 32 vstupními kanály, 64 výstupními kanály a jádrem konvoluce 3×3,
7. aktivační vrstva s funkcí ReLU,
8. konvoluční vrstva s 64 vstupními kanály, 64 výstupními kanály a jádrem konvoluce 3×3,
9. aktivační vrstva s funkcí ReLU,
10. max pooling vrstva s rozměry jádra 2×2,
11. hustá vrstva s 1024 vstupy a 200 výstupy,
12. aktivační vrstva s funkcí ReLU,
13. hustá vrstva s 200 vstupy a 10 výstupy,
14. aktivační vrstva s funkcí softmax.

Tento model byl trénován pomocí algoritmu RMSProp s řádem učení $\epsilon = 10^{-3}$, decay rate $\rho = 0.99$ a stabilizační konstantou $\delta = 10^{-8}$. Použito bylo 3250 iterací tohoto algoritmu a bylo dosaženo úspěšnosti 97.57% na testovací datové sadě.

Metoda	Úspěšnost
FGSM	40.4 %
I-FGSM	78.4 %
PGD	78.8 %
Cílená optimalizační metoda	100 %
CW	99.7 %

Tabulka 6.1: Úspěšnost metod generování adversariálních vzorků

6.1.3 Norma perturbace

Pro srovnání metod generování adversariálních vzorků byla zvolena *maximová*, tedy l_∞ norma. Za toleranci v této normě, resp. velikost perturbace byl zvolen ekvivalent 50 pixelových bodů, to jest $\kappa = \frac{50}{255}$, jelikož hodnoty pixelů obrázku jsou lineárně přeškálovány do intervalu $[0, 1]$.

Připomeňme, kde figuruje volba normy v předpisech jednotlivých metod generování adversariálních vzorků. Pro FGSM útok se volba l_∞ normy odráží v tom, že se k benignímu obrázku přičítá κ násobek znaménka gradientu, tedy z tohoto je nutně norma perturbace $\|\Delta x\|_\infty = \kappa$ a je možné splnit definici adversariálního vzorku. Pro I-FGSM a PGD se volba normy odráží ve funkci *Clip*, která zajišťuje, že norma perturbace $\|\Delta x\|_\infty \leq \kappa$. Pro cílenou optimalizační úlohu a CW útok je volba normy důležitá jednak pro množinu kde minimum z příslušné funkce hledáme, a jednak pro samotnou funkci, jejíž minimum metoda hledá.

6.1.4 Implementační detaily

Metoda FGSM byla implementována podle předpisu uvedeném v předchozích kapitolách.

Pro metody I-FGSM a PGD byl zvolen krok algoritmu $\gamma = 10^{-2}$ a počet iterací 62 podle návrhu v [11].

Cílená optimalizační byla implementována algoritmem sign gradient descent, s krokem 10^{-2} a 100 iteracemi, kde parametr λ byl volen nejprve jako $\lambda = 10^{-2}$ a při neúspěchu nahrazen dle pravidla

$$\lambda \leftarrow 10 \cdot \lambda. \tag{6.1}$$

Pro $\lambda \geq 100$ pak metoda vrátila poslední nalezený, byť stále správně klasifikovaný, vzorek. Takto jednoduše byl line-search algoritmus implementován z důvodu výpočetní náročnosti složitějších implementací algoritmu. Tato procedura byla aplikována pro všechny možné cílové značky a za výsledný vzorek položila ten nejbližší (ve smyslu l_∞ normy) původnímu benignímu vzorku z těch, které byly nesprávně klasifikovány.

Metoda CW byla implementována obdobně jako cílená optimalizační metoda pomocí sign gradient descentu s krokem 10^{-2} a 100 iteracemi. Parametr λ byl zvolen pevně jako $\lambda = 1$.

6.1.5 Výsledky

Bylo provedeno generování adversariálních vzorků všemi těmito metodami a za benigní vzorky bylo použito 1000 vzorků trénovací datové sady MNIST. Za úspěšnost metody uvažme procentuální podíl špatně klasifikovaných adversariálních vzorků. Potom lze uvést dle tabulky 6.1, že nejúspěšnější metodou v tomto nastavení je cílená optimalizační metoda. Nejméně úspěšnou metodou v tomto nastavení je naopak FGSM.

Na příklady vzorků vygenerovaných výše uvedenými metodami lze nahlédnout v obrázku 6.1. Jak je vidět, všechny metody při výše popsané volbě parametrů a implementačních detailů přidávají benigním

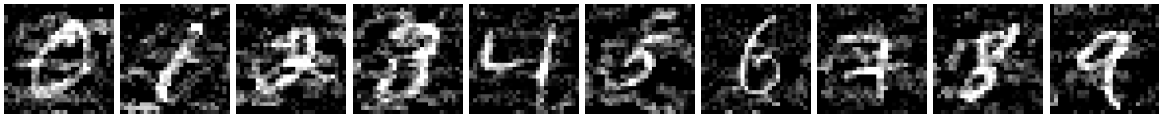
(a) Benigní vzorky

(b) FGSM

(c) I-FGSM

(d) PGD

(e) Cílená optimalizační metoda



(f) CW

Obrázek 6.1: Vzorky generované různými metodami hledání adversariálních vzorků

vzorkům do pozadí jakési artefakty. U vzorků vygenerovaných metodou CW ovšem dochází k nadměrnému zkreslení. Je to dáno nesprávnou volbou parametru λ. Diskuze ohledně tohoto typu útoku následuje v další sekci textu.

6.2 Analýza CW útoku

Podívejme se nyní na samotnou metodu CW útoku. V předcházející části textu došlo k pevnému výběru l_p normy, která figuruje v předpisu metody, a parametru λ. Zkusme proto zjistit, jaký vliv tyto volby mají na úspěšnost útoku a na kvalitu nalezených vzorků.

6.2.1 Vliv volby parametru λ

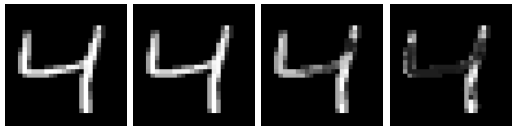
Z předpisu pro CW metodu lze odvodit, že větší hodnota λ napomáhá adversarialitě vzorku, tak jinak je spíše špatně klasifikován, než při volbě menší hodnoty λ. Je to ovšem za cenu vyšší vzdálenosti (ve smyslu použité l_p normy) nalezeného vzorku od původního benigního vzorku, tedy dojde k většímu zkreslení vzorku. Jak lze vyčíst z tabulky 6.2 a odvodit z obrázků 6.2 a 6.3, tato úvaha je správná.

Algoritmus	λ			
Norma	0.1	1	10	100
l_∞	99.6 %	99.7 %	99.8 %	99.8 %
l_1	0 %	32.5 %	88.9 %	91.7 %
l_2	23.9 %	93.6 %	100 %	100 %

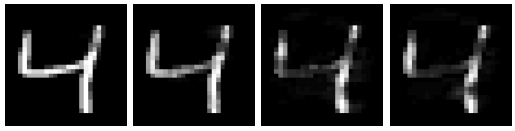
Tabulka 6.2: Úspěšnost CW útoku v závislosti na zvolených parametrech

(a) Benigní vzorek

(b) l_∞ útok



(c) l_1 útok



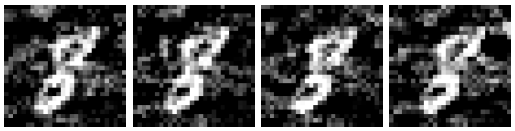
(d) l_2 útok

Obrázek 6.2: Výsledky CW útoku pro různé volby parametrů; λ je voleno po řadě 0.1, 1, 10 a 100

6.2.2 Vliv volby normy

Za normy byly v provedených numerických experimentech dosazeny l_∞, l_1 a l_2 normy. Jak lze nahlédnout v obrázcích 6.2 a 6.3, použití l_∞ normy vede k silnému zkreslení. Je to dáno faktem, že tato norma při derivování, které se používá pro numerické optimalizační algoritmy, vyzdvihne pouze jeden pixel obrázku, zbylé nechává netknuté. Použití l_1 či l_2 normy potom vede ke kultivovanějším obrázkům. Tím je myšleno, že vzniklé obrázky pozbývají náhodně vypadající artefakty v pozadí číslice.

(a) Benigní vzorek



(b) l_∞ útok

(c) l_1 útok

(d) l_2 útok

Obrázek 6.3: Výsledky CW útoku pro různé volby parametrů; λ je voleno po řadě 0.1, 1, 10 a 100

Kapitola 7

Robustně učené sítě

V rámci numerických experimentů, které byly provedeny v souvislosti s touto prací, bylo implementováno i robustní učení neuronové sítě podle optimalizačního přístupu uvedeného v jedné z předchozích kapitol. Následně pak byla touto implementací učena konvoluční neuronová síť stejné architektury, jež byla použita v předchozí kapitole s výsledky experimentů souvisejícími s generováním adversariálních vzorků.

Ohledně detailů implementace lze napsat, že za κ v (4.1) bylo voleno $\kappa = \frac{50}{255}$ a za normu určující kouli o poloměru κ okolo vzorků byla zvolena l_∞ norma. Dále zmiňme, že vnitřní maximalizační problém byl řešen algoritmem *projected gradient descent* (PGD) s krokem $\gamma = 10^{-2}$ a 62 iteracemi. Vnější minimalizace byla potom řešena algoritmem *RMSProp* s řádem učení $\epsilon = 10^{-4}$. Jeden krok algoritmu RMSProp byl pak prováděn pomocí mini-dávky o velikosti 30 vzorků. Těchto kroků pak bylo provedeno 5000.

S tímto nastavením bylo dosaženo úspěšnosti 97.16% na testovací datové sadě, tedy srovnatelné úspěšnosti s nerobustně učenou sítí. Narozdíl od nerobustního učení této architektury sítě, které se stejnými parametry trvalo zhruba 2.5 minuty, ovšem robustní učení trvalo více než 3 hodiny.

7.1 Adversariální útoky na robustně učenou síť

Na takto robustně naučenou neuronovou síť byly provedeny adversariální útoky za stejného nastavení, které prezentuje tabulka 6.1, proto lze nahlédnout na tabulku 7.1, odkud lze vyčíst, že útoky založené na FGSM (FGSM, I-FGSM a PGD) metodě jsou řádově méně úspěšné proti robustně trénované síti, než jsou proti standardně trénované síti. Avšak cílená optimalizační metoda a CW útok jsou proti robustně trénované síti stejně úspěšné jako proti standardně trénované síti. Je to dáno tím, že cílená optimalizační metoda a CW útok nemají ve svém předpisu zakomponováno ono κ-okolí okolo benigního vzorku ve vhodně zvolené l_p normě. Proto tyto metody mohou překročit hranice κ-okolí vzorků trénovací sady, před čímž z podstaty předpisu robustního učení síť není chráněna. Jak lze ovšem vidět z obrázku 7.1, CW útok generuje při tomto nastavení (l_∞ norma a $\lambda = 1$) vzorky velmi vzdálené původním benigním vzorkům.

7.1.1 CW útok na robustně učenou síť

Vzorky vzniklé l_∞ CW útokem na robustně učenou síť jsou, jak lze vidět v obrázcích 7.2 a 7.3, naprosto zkreslené a nepředstavují žádnou číslici, jak bylo nastíněno v úvodu této sekce textu, a to bez ohledu na volbu parametru λ. Pro změnu vzorky generované l_1 či l_2 CW útokem jsou čitelné číslice, jak lze nahlédnout v týž obrázcích. Při srovnání úspěšností CW útoků, které jsou vyneseny v tabulkách

Metoda	Úspěšnost
FGSM	5.7 %
I-FGSM	7 %
PGD	6.9 %
Cílená optimalizační metoda	100 %
CW	100 %

Tabulka 7.1: Úspěšnost metod generování adversariálních vzorků proti robustně naučené síti

(a) Benigní vzorky

(b) FGSM

(c) I-FGSM

(d) PGD

(e) Cílená optimalizační metoda

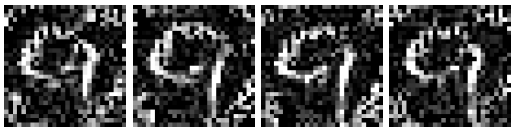
(f) CW

Obrázek 7.1: Vzorky generované různými metodami hledání adversariálních vzorků proti robustně naučené síti

Algoritmus	λ			
Norma	0.1	1	10	100
l_∞	100 %	100 %	100 %	100 %
l_1	0 %	1 %	88.4 %	100 %
l_2	0.2 %	88.7 %	99.9 %	100 %

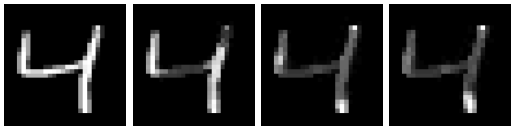
Tabulka 7.2: Úspěšnost CW útoku v závislosti na zvolených parametrech proti robustně naučené síti

(a) Benigní vzorek



(b) l_∞ útok

(c) l_1 útok

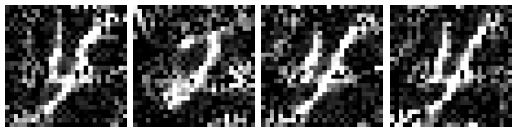


(d) l_2 útok

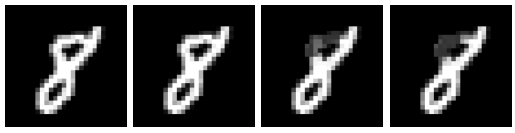
Obrázek 7.2: Výsledky CW útoku pro různé volby parametrů; λ je voleno po řadě 0.1, 1, 10 a 100

7.2 a 6.2, lze říci, že pro robustně naučenou síť je CW útok provedený v l_1 či l_2 normě méně úspěšný. Výjimkou je l_1 CW útok při volbě $\lambda = 100$.

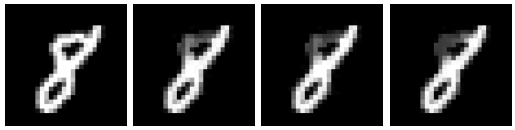
(a) Benigní vzorek



(b) l_∞ útok



(c) l_1 útok



(d) l_2 útok

Obrázek 7.3: Výsledky CW útoku pro různé volby parametrů; λ je voleno po řadě 0.1, 1, 10 a 100

Závěr

Úloha klasifikace ručně psaných číslic poskytla příležitost k srovnání algoritmů učení. Tyto výsledky pak jednak potvrzují oprávněnost oblíbenosti algoritmu RMSProp, jednak užitečnost stochastických variant těchto algoritmů pro jejich nižší výpočetní náročnost.

Stěžejní však je ověření existence adversariálních vzorků, jakožto fenoménu spojeného s klasifikačními neuronovými sítěmi. Byla provedena implementace pěti metod generování těchto adversariálních vzorků. Těchto pět metod potom je *fast gradient sign method (FGSM)*, *iterativní FGSM (I-FGSM)*, *projected gradient descent (PGD)*, *cílená optimalizační metoda* a *Carlini-Wagner (CW)*. Bylo provedeno porovnání jejich úspěšnosti, kde jako nejméně úspěšná vyšla metoda FGSM a jako nejvíce úspěšná cílená optimalizační metoda.

Dále bylo provedeno hlubší studium CW útoku, kde byly voleny v předpisu CW útoku (3.7) různé hodnoty parametru λ a různé l_p normy. Tyto volby pak měly vliv jak na úspěšnost CW útoku, tak na kvalitativní stránku obrázků vzniklých tímto typem generování adversariálních vzorků.

Zásadní potom bylo téma *robustního strojového učení*, kde bylo dosaženo pomocí algoritmu robustního strojového učení, který spočívá v řešení dvojitého optimalizačního problému, obrany proti útokům založených na metodě FGSM, totiž proti samotnému FGSM, dále však i I-FGSM a PGD. Pomocí tohoto algoritmu též došlo k částečné obraně proti CW útoku.

Literatura

[1] I. Goodfellow, Y. Bengio, A. Courville, *Deep Learning*. MIT Press, 2016.

[2] B. T. Polyak, *Some methods of speeding up the convergence of iteration methods*. USSR Computational Mathematics and Mathematical Physics, 1964.

[3] I. Sutskever, J. Martens, G. Dahl, G. Hinton, *On the importance of initialization and momentum in deep learning*. In ICML, 2013.

[4] J. Duchi, E. Hazan, Y. Singer, *Adaptive subgradient methods for online learning and stochastic optimization*. Journal of Machine Learning Research, 2011.

[5] G. Hinton, *Neural networks for machine learning*. Coursera, video lectures, 2012.

[6] D. Kingma, J. Ba, *Adam: A method for stochastic optimization*. In 'International Conference on Learning Representations', ICLR 2015.

[7] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, R. Fergus, *Intriguing properties of neural networks*. arXiv, 2014.

[8] I. Goodfellow, J. Shlens, C. Szegedy, *Explaining and Harnessing Adversarial Examples*. In 'International Conference on Learning Representations', ICLR 2015.

[9] J. Nocedal, S. Wright, *Numerical optimization*. Springer Science & Business Media, 2006.

[10] J. Liu, Q. Zhang, K. Mo, X. Xiang, J. Li, D. Cheng, R. Gao, B. Liu, K. Chen, G. Wei, *An efficient adversarial example generation algorithm based on an accelerated gradient iterative fast gradient*. Computer Standards & Interfaces, Volume 82, 2022.

[11] A. Kurakin, I. Goodfellow, S. Bengio, *Adversarial examples in the physical world*. arXiv 2016.

[12] A. Mądry, A. Makelov, L. Schmidt, D. Tsipras, A. Vladu, *Towards deep learning models resistant to adversarial attacks*. Stat 1050 9, 2017.

[13] N. Carlini, D. Wagner, *Towards evaluating the robustness of neural networks*. IEEE Symposium on Security and Privacy (SP), IEEE, 2017.

[14] N. Papernot, P. McDaniel, I. Goodfellow, *Transferability in machine learning: from phenomena to black-box attacks using adversarial samples*. arXiv 2016

[15] Y. Lecun, C. Cortes, C. J. Burges, *The mnist database of handwritten digits*. 1998.