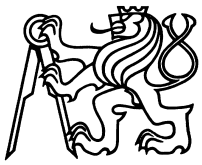
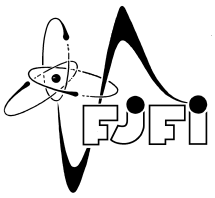


České vysoké učení technické v Praze
Fakulta jaderná a fyzikálně inženýrská



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: Abstrakt max. na 10 řádků.

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

Title:

Robust machine learning and adversarial examples

Author: Pavel Jakš

Abstract: Max. 10 lines of English abstract text.

Key words: keywords in alphabetical order separated by commas

Obsah

Úvod

Pojem neuronové sítě představuje výpočetní jednotku, která svou univerzálností nachází uplatnění v mnoha disciplínách.

Kapitola 1

Neuronové sítě

Neuronová síť je svým charakterem velmi přizpůsobivý výpočetní stroj vhodný pro řešení mnoha problémů. Mezi nejčestější 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 $F : \mathbb{R}^{n_1,\dots,n_k} \to \mathbb{R}^{m_1,\dots,m_l}$. Pro případ regrese potom $l = 1$, $m = 1$ a výstup F hraje roli predikované hodnoty, pro případ klasifikace je též $l = 1$, ale m 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 $\phi : \mathbb{R}^{p_1,\dots,p_r} \to \mathbb{R}^{q_1,\dots,q_s}$, 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 $W \in \mathbb{R}^{q_1,p_1}$ 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.

Nečastější 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:

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

kde $\phi : \mathbb{R}^{C_{in}, W_{in}, H_{in}} \to \mathbb{R}^{C_{out}, W_{out}, H_{out}}$ (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), $b \in \mathbb{R}^{C_{out}, W_{out}, H_{out}}$ je práh, $K \in \mathbb{R}^{C_{out}, C_{in}, k_1, k_2}$ 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 $C \times W \times H$ (počet kanálů, šířka, výška) v každém kanálu reportuje maximální hodnotu v blocích o rozměrech $k_1 \times k_2$. Potom zobrazení ϕ je zobrazení $\phi : \mathbb{R}^{C,W,H} \to \mathbb{R}^{C,W_{out},H_{out}}$, kde platí:

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

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

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

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

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:

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

Užití této aktivační vrstvy je na snadě. 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

1.3 Konvoluční neuronová síť

Kapitola 2

Učení neuronové sítě

Standardní přístup k *učení neuronové sítě*, což je termín, kterým se označuje vhodné nalezení parametrů neuronové sítě, 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* $\mathbb{X} = \{x^{(i)} | i \in \hat{N}\}$ a k nim příslušné *značky* $\mathbb{Y} = \{y^{(i)} | i \in \hat{N}\}$, kde pojem vzorek představuje vstup neuronové sítě jakožto zobrazení 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 součet nebo průměr dílčích ztrát, které neuronová síť dosahuje na vzorcích trénovací sady:

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

případně:

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

kde $x^{(i)}$ je i-tý vektor trénovací sady, $y^{(i)}$ je i-tý vektor trénovacích značek, N je velikost trénovací sady, F_θ neuronová síť jakožto funkce $F_\theta : Dom_{F_\theta} \subset \mathbb{R}^n \to \mathbb{R}^m$ parametrizovaná parametry θ, L značí konkrétní ztrátu pro daný vzorek a J je celková účelová funkce. V tomto textu se držme tvaru v (2.2).

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

$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{m} (F_\theta(x^{(i)})_j - y_j^{(i)})^2 \tag{2.3}$$

nebo

$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \|F_\theta(x^{(i)}) - y^{(i)}\|_2^2, \tag{2.4}$$

kde $x^{(i)}$ je i-tý vektor trénovací sady, $y^{(i)}$ je i-tý vektor trénovacích značek, N je velikost trénovací sady, F_θ neuronová síť jakožto funkce $F_\theta : Dom_{F_\theta} \subset \mathbb{R}^n \to \mathbb{R}^m$ parametrizovaná parametry θ a $\|\cdot\|_2$ je L^2 norma.

Další účelová funkce, která nachází uplatnění v klasifikačních problémech, se vypočte pomocí křížové entropie:

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

kde H označuje právě onu křížovou entropii mezi pravděpodobnostními distribucemi. Připomeňme, že klasifikační neuronová síť produkuje diskrétní pravděpodobnostní distribuce, a proto lze na výstup takovéto neuronové sítě a její značky (také pravděpodobnostní distribuce) aplikovat křížovou entropii. Onen výraz v (2.5) lze spočíst následovně:

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

přičemž $x^{(i)}$ je i-tý vektor trénovací sady, $y^{(i)}$ je i-tý vektor trénovacích značek, N je velikost trénovací sady, F_θ neuronová síť jakožto funkce $F_\theta : Dom_{F_\theta} \subset \mathbb{R}^n \to \mathbb{R}^m$ parametrizovaná parametry θ.

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]).

Pro celkový popis algoritmu zaveďme pojem *výpočetního grafu*. Nechť vrcholy grafu představují proměnné, a to libovolných rozměrů, hrany grafu nechť jsou barevné a orientované, kde barva značí jednu z prováděných operací a orientace značí, jaká proměnná vznikla ze které pomocí dané operace.

Pojem výpočetního grafu lze ilustrovat následujícím příkladem: Nechť proměnná u je číslo a proměnné v a w vektory stejných rozměrů a platí, že proměnnou u lze získat jako $u = v \cdot w$. Potom tomuto příkladu náleží výpočetní graf o třech vrcholech, a to vrcholech proměnných v, w a u, a dvou hranách - první z v do u o barvě odpovídající tomu býti prvním argumentem skalárního součinu a druhá z w do u o barvě odpovídající tomu býti druhým argumentem skalárního součinu.

Dále je zapotřebí uvést *řetězové pravidlo* pro výpočet derivace složené funkce, o které se algoritmus opírá. Nechť $g : \mathbb{R}^n \to \mathbb{R}^m$ a $h : \mathbb{R}^m \to \mathbb{R}^p$, $a \in \mathbb{R}^n$, potom:

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

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

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

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. Vrcholy výpočetního grafu jsou ovšem libovolných rozměrů. Potom lze dané proměnné urovnat do vektorů a spočíst gradient opět násobením vektoru gradientu s maticí totálního diferenciálu a následně převést vypočtený gradient zpět do příslušného tvaru.

Nyní lze nahlédnout na výpočet funkce jejíž gradient je žádoucí spočíst, například účelové funkce neuronové sítě, pomocí výpočetního grafu. Potom algoritmus zpětného šíření chyby postupuje po výpočetním grafu od výsledné proměnné k listovým vrcholům a aplikuje řetězové pravidlo.

V praxi je ovšem snadné natrefit na velmi složité výpočetní grafy, které vedou k vyhodnocování mnoha podvýrazů. Navíc mnoho takovýchto podvýrazů může být stejných. Při implementaci je tedy 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. Opačný přístup lze uplatnit při nedostatku paměti stroje.

2.3 Algoritmy učení

Základním algoritmem pro učení neuronové sítě je *gradientní sestup* (angl. *gradient descent*). Opírá se o fakt, že gradient reálné funkce určuje směr největšího spádu 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 další 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.

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.

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

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

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

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.

Algoritmus *AdaGrad* 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 by potom mohla vypadat 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 - v proměnné r si pamatuje velmi vzdálené hodnoty gradientu, což dle [1] mnohdy vede k předčasnému poklesu řádu učení. Proto je namístě uvést další algoritmus - *RMSProp*. Tento algoritmus nahrazuje součet přes všechny hodnoty gradientu exponenciálně tlumeným váženým průměrem, a to způsobem, kde jedna iterace vypadá následovně [5]:

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

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

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

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

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 pomocí prahu 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 δ je malé číslo (např. 10^{-7}) pro numerickou stabilitu, $\odot$ značí Hadamardův součin, výraz zlomku a odmocniny na šestém řádku je myšlen po složkách, t je pořadí iterace a $\rho_1, \rho_2 \in [0, 1)$ jsou hyperparametry nazvané *decay rate*.

2.4 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 (2.2) se výpočet sestává z N výpočtů dílčích gradientů:

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

kde $x^{(i)}$ je i-tý vektor trénovací sady, $y^{(i)}$ je i-tý vektor trénovacích značek, N je velikost trénovací sady, F_θ neuronová síť jakožto funkce $F_\theta : Dom_{F_\theta} \subset \mathbb{R}^n \to \mathbb{R}^m$ parametrizovaná parametry θ, L značí konkrétní ztrátu pro daný vzorek a J je celková účelová funkce.

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:

$$\nabla_\theta J(\theta) \approx \sum_{j=1}^{M} \nabla_\theta L(F_\theta(x^{(i_j)}), y^{(i_j)}), \tag{2.28}$$

kde $x^{(i)}$ je i-tý vektor trénovací sady, $y^{(i)}$ je i-tý vektor trénovacích značek, M je velikost mini-dávky, $i_j \sim U\{1, N\}$ jsou indexy vzorků vybraných do mini-dávky, F_θ je neuronová síť jakožto funkce $F_\theta : Dom_{F_\theta} \subset \mathbb{R}^n \to \mathbb{R}^m$ parametrizovaná parametry θ, L značí konkrétní ztrátu pro daný vzorek a J je celková účelová funkce.

Číslo M lze vybírat dle [1] v řádu jednotek až stovek. Při aplikaci této apriximace 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 algoritmech, 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. Zjištění lze formálně zapsat následovně:

$$(\exists x, y \in \mathbb{T})(\exists \Delta x \in \mathbb{R}^n, \|\Delta x\| < \kappa)(F_\theta(x) = y \wedge F_\theta(x + \Delta x) \neq y), \tag{3.1}$$

kde κ je malé číslo a $\|\cdot\|$ je L_p norma. Takovým vzorkům $\tilde{x} = x + \Delta x$ se říká *adversariální vzorky*.

Pro konkrétní vzorek x a příslušnou značku y definujeme množinu adversariálních vzorků jako

$$\widetilde{\mathbb{X}}_x = \{\tilde{x} \in \mathbb{R}^n | F_\theta(\tilde{x}) \neq y \wedge \|\tilde{x} - x\| < \kappa\}. \tag{3.2}$$

Takto obecná definice adversariálních vzorků ovšem neposkytuje návod na jejich nalezení. Proto uveďme metody generování těchto adversariálních vzorků. 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 [15].

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 $F_\theta(\tilde{x}) = \tilde{y}$ pro pevně zvolenou značku $\tilde{y}$ různou od původní značky $y = F_\theta(x)$. Necílené útoky předem nevybírají značku za cíl, nýbrž požadavkem je jen, aby $F_\theta(\tilde{x}) \neq F_\theta(x)$. Necílené útoky nebývají tolik účinné jako cílené [14].

První metoda představená v [8] je známá pod zkratkou *FGSM* (z angl. *fast gradient sign method*). Jedná se o necílenou metodu, která využívá mnoho-dimenzionální lineární vztahy neuronové sítě [10] a má předpis:

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

při zachování značení z minulých kapitol textu, označení *sign* pro znaménkovou funkci a γ pro velikost složek perturbace Δx.

Druhá metoda jde o krok dál, vzorec (3.3) aplikuje iterativně několikrát a generuje posloupnost $(x_n)_{n=0}^{K}$, kde K je počet iterací metody. Jedná se o metodu *I-FGSM* (z angl. *iterative fast gradient sign method*) představenou v [12] s předpisem:

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

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

kde funkce $Clip$ omezuje výsledný součet, aby byl v κ-okolí původního vzorku x a zároveň v definičním oboru neuronové sítě F_θ - například jsou-li vzorky obrázky, funkce $Clip$ zajišťuje, aby hodnoty pixelů nebyly záporné či vyšší než 255. Počet iterací je ovšem dalším hyper-parametrem, který je nutno nastavit. Jedná se tedy o necílenou metodu.

Třetí metoda (cílená) nahlíží na generování adversariálních vzorků jako na optimalizační úlohu [7], [14]:

$$\tilde{x} = \arg \min_{\hat{x} \in Dom_{F_\theta}} \lambda \cdot \|\hat{x} - x\| + L(F_\theta(\hat{x}), \tilde{y}), \tag{3.6}$$

kde $\lambda > 0$, Dom_{F_θ} je definičním oborem F_θ, $\tilde{y}$ značí cílenou nesprávnou značku. Tento optimalizační problém lze řešit algoritmem *L-BFGS* [9], resp. jeho variantou s vazbami (angl. *box-constrained L-BFGS*).

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 [13].

Následující metoda (necílená) má opět optimalizační charakter. Jmenuje se *CW* (*Carlini-Wagner*) a má předpis [14], [10]:

$$\tilde{x} = \arg \min_{\hat{x} \in Dom_{F_\theta}} \|\hat{x} - x\| - c \cdot L(F_\theta(\hat{x}), y), \tag{3.7}$$

kde $c > 0$.

Kapitola 4

Robustní učení neuronové sítě

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 $\mathbb{X} = \{x^{(i)} | i \in \hat{S}\}$ a značek $\mathbb{Y} = \{y^{(i)} | i \in \hat{S}\}$, 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

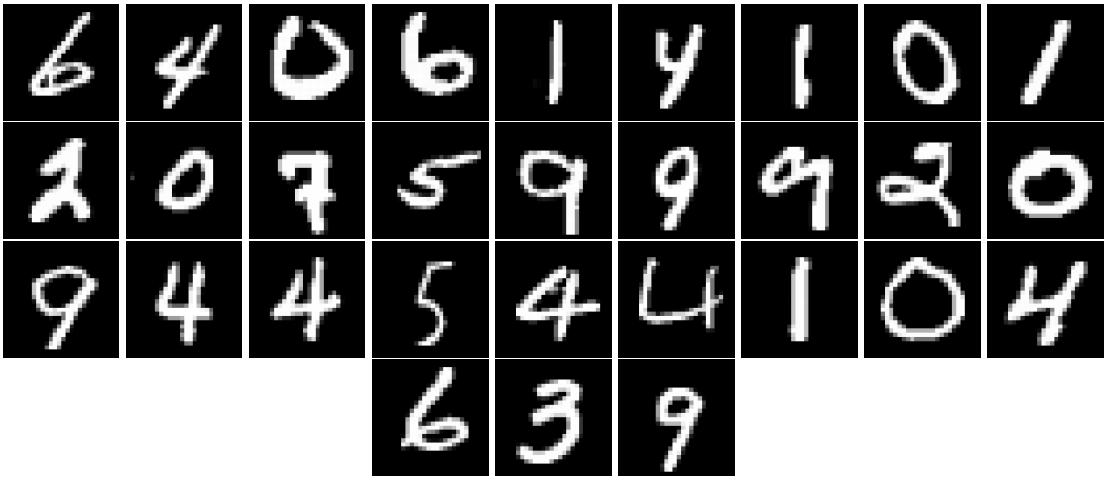
Nyní je namístě vyslovit poznámku o inicializaci parametrů neuronové sítě před samotným učením. Dle [1] je standardním postupem pro inicializaci vybírat hodnoty parametrů náhodně, a to z rovnoměrného rozdělení na rozumném intervalu. Konkrétní experimenty v tomto textu pracují s následujícím rozdělením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

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

kde n je v případě vah počet sloupečků matice vah, v případě prahů velikost vektoru prahů. 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 černobílého obrázku. Sada dat, která je zde použita je nazvána MNIST [16]. Její trénovací sada $\mathbb{T}$ obsahuje



Obrázek 5.1: Datová sada MNIST

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.

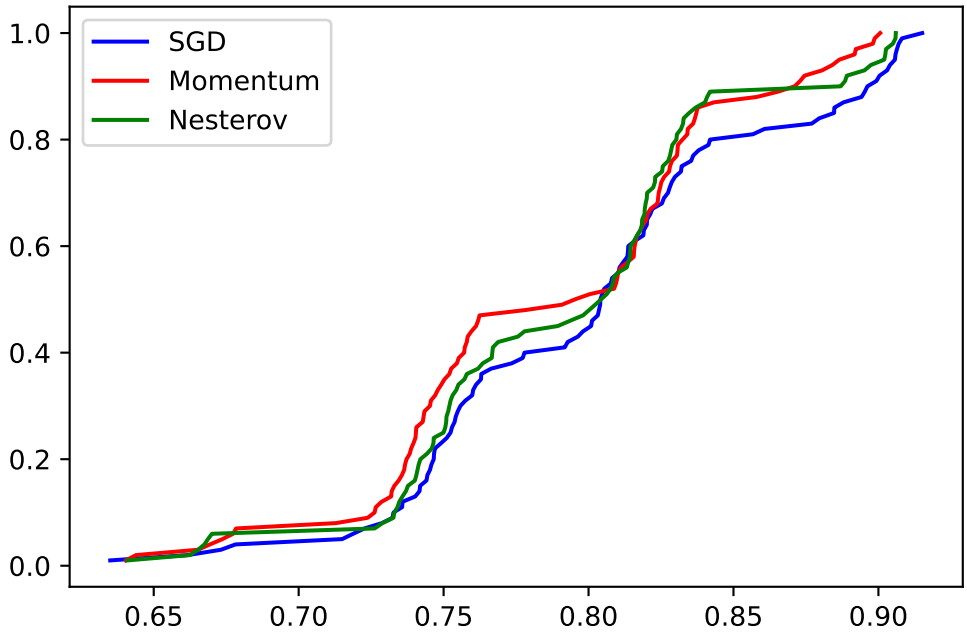
5.4 Výsledky

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 (5.2). 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. (5.3) odezřít výsledky, a to konkrétně, že stochastický gradientní sestup má v tomto nastavení lepší výkonnost.

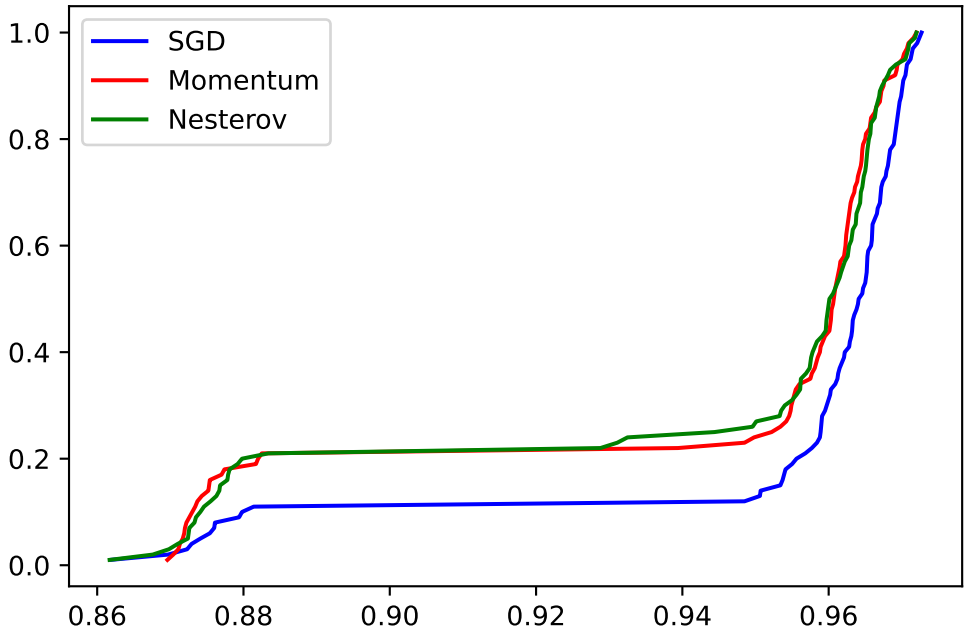
Pro srovnání algoritmů *stochastický gradientní sestup*, *AdaGrad*, *RMSProp* a *Adam* lze využít podkladů na obrázku (5.4), který zachycuje výsledky obdobných experimentů jako popsaných výše. Nastavení tohoto pokusu bylo následující: Pro stochastický gradientní sestup a algoritmus AdaGrad byl použit 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 (5.5). Ten zachycuje výsledky totožného nastavení jako obrázek (5.4) jen s rozdílem použitého modelu. V tomto případě byl použit model konvoluční neuronové sítě. Jak

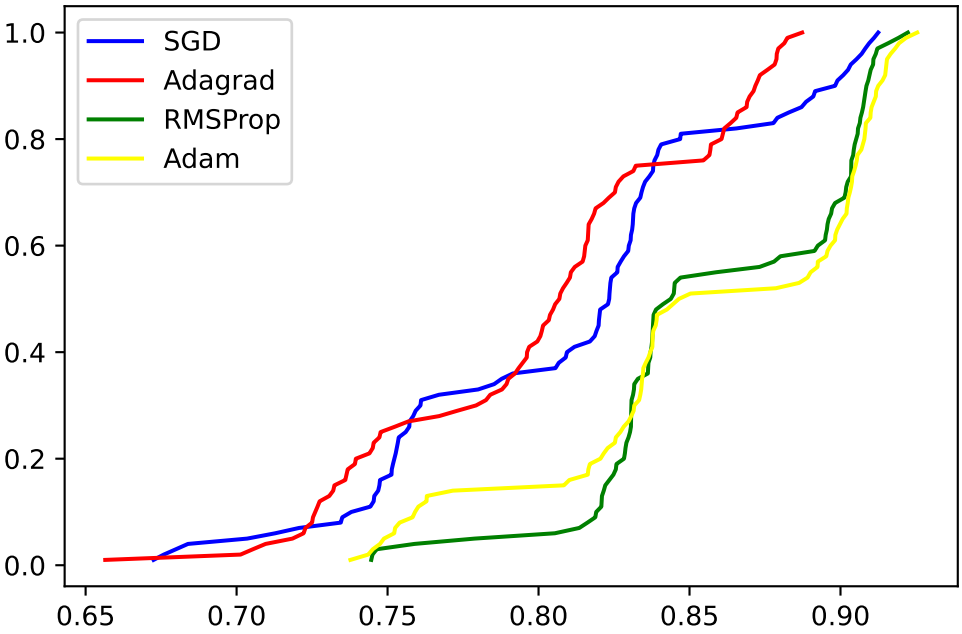


Obrázek 5.2: Srovnání algoritmů učení I
Simple - stochastický gradientní sestup; *Momentum* - metoda hybnosti; *Nesterov* - metoda Něstěrovovy hybnosti.

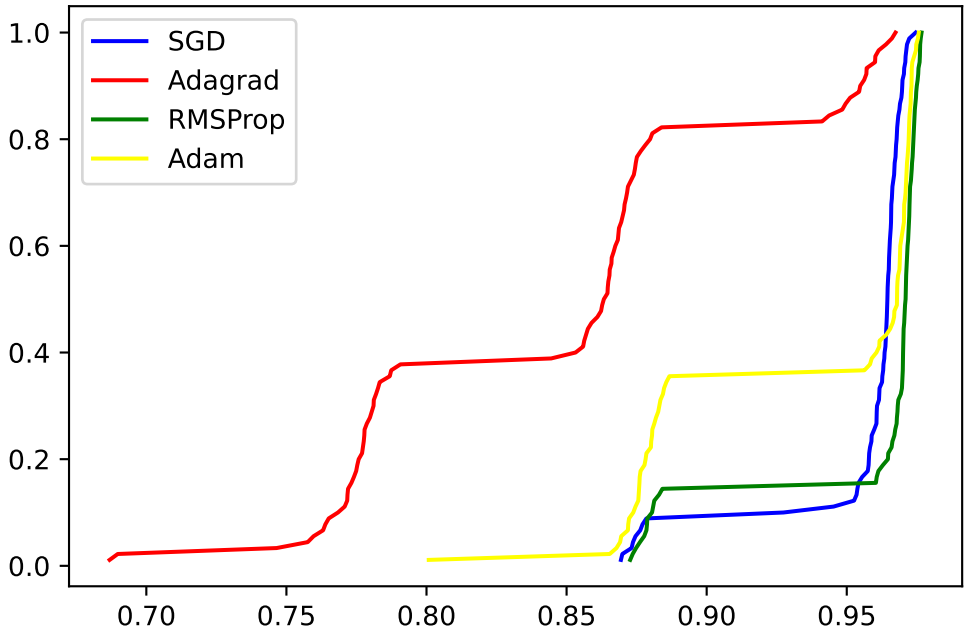
lze nahlédnout, algoritmus AdaGrad byl pro tuto úlohu nevhodný. Algoritmus Adam dosáhl přijatelné úrovně neuronové sítě (tedy úspěšnost na testovací datové sadě vyšší než 95 %) zhruba v 60 % případů, algoritmus RMSProp zhruba v 90 % případů a stochastický gradientní sestup v 95 % případů. Ovšem kvalita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



Obrázek 5.3: Srovnání algoritmů učení II
Simple - stochastický gradientní sestup; *Momentum* - metoda hybnosti; *Nesterov* - metoda Něstěrovovy hybnosti.



Obrázek 5.4: Srovnání algoritmů učení III
SGD - stochastický gradientní sestup; *AdaGrad* - algoritmus AdaGrad; *RMSProp* - algoritmus RMSProp; *Adam* - algoritmus Adam



Obrázek 5.5: Srovnání algoritmů učení IV
SGD - stochastický gradientní sestup; *AdaGrad* - algoritmus AdaGrad; *RMSProp* - algoritmus RMSProp; *Adam* - algoritmus Adam

Závěr

Text závěru....

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] Y. Li, B. Wu, Y. Feng, Y. Fan, Y. Jiang, Z. Li, S. Xia, *Semi-supervised robust training with generalized perturbed neighborhood*. Pattern Recognition, Volume 124, 2022.

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

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

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

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

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

[17] T. Weng, H. Zhang, P. Chen, J. Yi, D. Su, Y. Gao, C. Hsieh, L. Daniel, *Evaluating the Robustness of Neural Networks: An Extreme Value Theory Approach*. In 'International Conference on Learning Representations', ICLR 2018.