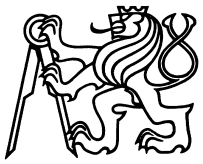
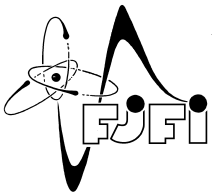


České vysoké učení technické v Praze
Fakulta jaderná a fyzikálně inženýrská



Interpretovatelné odhady modelů s aplikací pro pohyb robotů

Interpretable estimates of models with application to robot movement

Diplomová práce

Autor: **Stanislav Novotný**

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

Konzultant: **Doc. Ing. Václav Šmídl, Ph.D.**

Akademický rok: 2021/2022

Poděkování:
Chtěl bych zde poděkovat především svému školiteli Mgr. Lukáši Adamovi, Ph.D. za pečlivost, ochotu, trpělivost, vstřícnost a odborné i lidské zázemí při vedení tohoto výzkumného úkolu. Dále děkuji svému konzultantovi Doc. Ing. Václavu Šmídlovi, Ph.D. za odborný dohled.

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

V Praze dne 7. července 2020 Stanislav Novotný

Název práce:

Interpretovatelné odhady modelů s aplikací pro pohyb robotů

Autor: Stanislav Novotný

Obor: Matematické inženýrství

Druh práce: Diplomová práce

Vedoucí práce: Mgr. Lukáš Adam, Ph.D., Ústav teorie informace a automatizace Pod Vodárenskou věží 4 182 00, Praha 8

Konzultant: Doc. Ing. Václav Šmídl, Ph.D., Ústav teorie informace a automatizace Pod Vodárenskou věží 4 182 00, Praha 8

Abstrakt:

Klíčová slova:

Title:

Interpretable estimates of models with application to robot movement

Author: Stanislav Novotný

Abstract:

Key words:

Obsah

Úvod

Kapitola 1

Teoretický podklad

V této kapitole si nejdříve představíme strojové učení, což je podskupina umělé inteligence. Seznámíme se s tím, jak využíváme optimalizaci v hlavních nástrojích strojového učení - regresi a klasifikaci. Dále se podíváme na neuronové sítě a metody optimalizace.

1.1 Strojové učení

1.1.1 Optimalizace

Při optimalizaci [4] se snažíme maximalizovat nebo minimalizovat určitou funkci na dané množině. Tento fenomén se v praxi projevuje jako například maximalizování zisku na základě předpovědi vývoje trhu, hledání nejkratší nebo nejrychlejší cesty atp. Matematicky formulováno řešíme maximalizaci/minimalizaci f na množině X.

$$\begin{aligned} &\text{minimalizuj } f(\mathbf{w}) \\ &\text{vzhledem k } \mathbf{w} \in X. \end{aligned}$$

Tímto pokryjeme oba dva problémy, jelikož

$$\begin{aligned} &\text{minimalizuj } f(\mathbf{w}) \\ &\text{vzhledem k } \mathbf{w} \in X \end{aligned}$$

je ekvivalentní

$$\begin{aligned} &\text{maximalizuj } -f(\mathbf{w}) \\ &\text{vzhledem k } \mathbf{w} \in X. \end{aligned}$$

Při optimalizaci se snažíme najít tzv. globální minimum, což je bod $\mathbf{w}^* \in X$ splňující

$$f(\mathbf{w}^*) \leq f(\mathbf{y}) \quad \forall \mathbf{y} \in X.$$

Hledání globálních minim je však velice obtížné. Nejčastěji se spokojíme s lokálním minimem, bodem $\mathbf{w}^* \in X$, pro který existuje epsilonové okolí tak, že je splněno

$$f(\mathbf{w}^*) \leq f(\mathbf{y}) \quad \forall \mathbf{y} \in B(\mathbf{w}^*, \epsilon) \cap X,$$

kde $B(\mathbf{w}^*, \epsilon) = \{\mathbf{y} : \|\mathbf{w}^* - \mathbf{y}\| < \epsilon\}$ je epsilonové okolí bodu $\mathbf{w}^*$.
Z matematické analýzy známe:

Věta 1.1.1. Nechť f je diferencovatelná na $\mathbb{R}^n$, která má v bodě $\mathbf{w}^*$ lokální minimum, pak $\nabla f(\mathbf{w}^*) = 0$. Je-li f konvexní, pak každý bod $\mathbf{w}^*$, pro který platí $\nabla f(\mathbf{w}^*) = 0$, je globálním minimem f.

Důkaz. Důkaz viz. [10] strana 14. □

Ve snaze nalézt lokální minimum funkce tedy budeme hledat body $\mathbf{w}^*$, ve kterých je $\nabla f(\mathbf{w}^*) = 0$, těmto bodům říkáme stacionární body. Hledáme je proto, jelikož v těchto bodech může být lokální extrém.

1.1.2 Regrese a klasifikace

Regrese [9] a klasifikace [8] jsou nástroje strojového učení, které na základě předem známých dat vracejí předpokládaný odhad pro zcela nová data. Mějme n neznámých vzorků $\mathbf{x}_1, \ldots, \mathbf{x}_n$, které mohou představovat například n obrázků, kde každý má rozlišení 28x28 pixelů, čili každý jeden vzorek je vektor o 784 složkách. Dále mějme vektor již známých dat $y_1, \ldots, y_n$, kde každý prvek vektoru představuje číslo, které má určitý obrázek s ručně psaným číslem představovat. Uvažujeme funkci $predict(\mathbf{x}, \mathbf{w})$, u které hledáme neznámou proměnnou $\mathbf{w}$ tak, abychom ztotožnili vzorky $\mathbf{x}_i$ se známými daty y_i, kde $i \in n$. Cílem je tedy nastavit funkci $predict(\mathbf{x}, \mathbf{w})$ tak, aby při použití nových vzorků správně predikovala známá data. Regrese předpovídá spojitou veličinu, zatímco klasifikace vrací pouze konečné množství stavů. Příkladem regrese může být například odhad ceny domu v závislosti na jeho ploše a vybavení. Klasifikace je například rozpoznávání číslic a písmen v ručně psaném textu.
Při zachování stejného značení se snažíme minimalizovat střední odchylku mezi neznámými vzorky a známými daty, řešíme tedy problém

$$\text{minimalizuj}_{\mathbf{w}} \ \frac{1}{n} \sum_{i=1}^{n} L(predict(\mathbf{x}_i, \mathbf{w}), y_i), \tag{1.1}$$

kde L je tzv. ztrátová funkce, která určuje, jakým způsobem chceme odchylku měřit.

1.1.2.1 Lineární regrese

Zabýváme-li se lineární regresí, funkce $predict(\mathbf{x}_i, \mathbf{w})$ nabývá tvaru

$$predict(\mathbf{x}_i, \mathbf{w}) = \mathbf{w}^T \mathbf{x}_i$$

a ztrátová funkce odpovídá střední kvadratické odchylce

$$L(\hat{y}, y) = (\hat{y} - y)^2.$$

Problém minimalizace tedy vypadá takto

$$\text{minimalizuj}_{\mathbf{w}} \ \frac{1}{n} \sum_{i=1}^{n} (\mathbf{w}^T \mathbf{x}_i - y_i)^2. \tag{1.2}$$

Snažíme se nalézt minimum výrazu

$$\frac{1}{n}\left[(\mathbf{w}^T\mathbf{x}_1 - y_1)^2 + (\mathbf{w}^T\mathbf{x}_2 - y_2)^2 + \ldots + (\mathbf{w}^T\mathbf{x}_n - y_n)^2\right] \tag{1.3}$$

Je také možné pracovat s maticovou formou. Uvažujme matici $\mathbf{X}$, jejíž i-tý řádek tvoří transpozice vzorku $\mathbf{x}_i$

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_n^T \end{pmatrix}, \ \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}.$$

Minimalizaci (1.2) pak můžeme převést na

$$\text{minimalizuj}_{\mathbf{w}} \ \ \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2 , \tag{1.4}$$

kde $\|\cdot\|$ značí Eukleidovskou normu. Jelikož i-tá komponenta vektoru $\mathbf{X}\mathbf{w} - \mathbf{y}$ má tvar

$$\mathbf{x}_i^T\mathbf{w} - y_i, \quad i \in n,$$

po aplikaci druhé mocniny l_2 normy na celý vektor $\mathbf{X}\mathbf{w} - \mathbf{y}$ obdržíme výraz

$$(\mathbf{x}_1^T\mathbf{w} - y_1)^2 + (\mathbf{x}_2^T\mathbf{w} - y_2)^2 + \ldots + (\mathbf{x}_n^T\mathbf{w} - y_n)^2. \tag{1.5}$$

Z (1.3) a (1.5) je vidět, že minimalizace (1.2) a (1.4) jsou si ekvivalentní, protože minimum nezávisí na čísle $\frac{1}{n}$.

Věta 1.1.2. $\|\cdot\|^2$ je konvexní zobrazení $\mathbb{R}^n \to \mathbb{R}_+$.

Důkaz. $\mathbb{R}^n$ konvexní množina. Nejdříve ukážeme, že $\|\cdot\|$ je konvexní. Z definice normy platí

$$\|k\mathbf{x}\| = |k| \, \|\mathbf{x}\| \ \text{ a } \ \|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\| \ \ \forall k \in \mathbb{R}, \ \ \forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n,$$

pak $\forall \theta \in [0, 1]$

$$\|\theta\mathbf{x} + (1-\theta)\mathbf{y}\| \leq \|\theta\mathbf{x}\| + \|(1-\theta)\mathbf{y}\| = \theta \, \|\mathbf{x}\| + (1-\theta) \, \|\mathbf{y}\|$$

Nechť $f(z) = z^2$ a $g(\mathbf{x}) = \|\mathbf{x}\|$ obě dvě funkce jsou konvexní, $f(z)$ je neklesající na oboru hodnot $g(x)$, to jest na $[0, +\infty)$. Složení obou funkcí $f \circ g = \|\cdot\|^2$ je tedy konvexní. □

Věta 1.1.2 společně s tím, že (1.4) je kombinace kvadratického a lineárního zobrazení tedy říká, že řešíme konvexní kvadratický problém. Položíme-li derivaci výrazu (1.4) rovnu nule, dostáváme

$$2\mathbf{X}^T(\mathbf{X}\mathbf{w} - \mathbf{y}) = 2\mathbf{X}^T\mathbf{X}\mathbf{w} - 2\mathbf{X}^T\mathbf{y} = 0.$$

Řešením této rovnice je

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y},$$

což je analytické řešení za předpokladu existence inverzní matice.

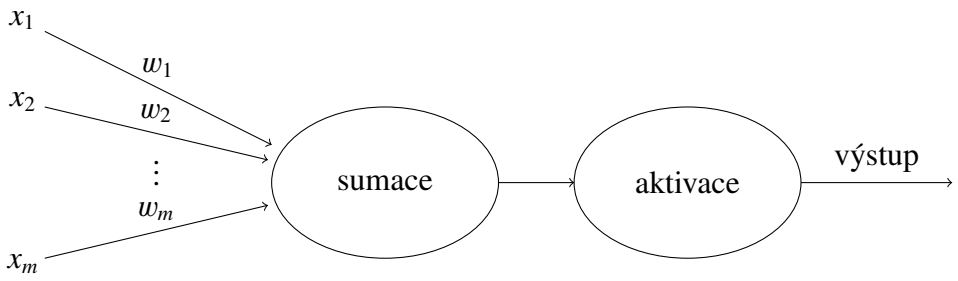
V praktické části se budeme zabývat výhradně regresí, proto zde klasifikaci nezmiňujeme.

1.2 Neuronové sítě

Jak již samotný název napovídá, neuronové sítě [1] jsou inspirované chováním našich řídících center, mozků. V lidském mozku se nacházejí buňky, které se s časem neregenerují jako ostatní buňky v našem těle, každá z nich typicky navazuje spojení s 1 000 - 10 000 ostatními. Tyto buňky se nazývají neurony. Síla lidské mysli pochází z počtu neuronů a množství spojení mezi nimi. Umožňuje nám to přemýšlet, pamatovat si věci a uplatňovat naše zkušenosti na nové úkoly. Neuron bychom mohli zjednodušeně rozdělit na čtyři části.

- Dendrity, výběžky které přijímají signály.
- Jádro zpracovávající vstupní signál.
- Axon, který změní zpracovaný vstupní signál na výstupní.
- Synapse, výběžky navazující elektrochemické spojení s ostatními neurony.

Převedli bychom základní ideologii neuronu do počítače, vypadala by takto



Obrázek 1.1: Jednoduchý model neuronové vrstvy

Na obrázku 1.1 x_i představují různé vstupy, které se násobí jejich váhami w_i, představující dendrity. Váha udává důležitost jednotlivých vstupů. V nejjednoduším případě můžeme všechny takto získané hodnoty sečíst a následně pomocí aktivační funkce určit, zda se tento neuron aktivuje nebo ne. Sumace společně s aktivací představují úlohu jádra a axonu. Symbolicky zapsáno, výstup tohoto neuronu bude vypadat takto:

$$y = \mathbf{1}\left(w_1 x_1 + w_2 x_2 + \ldots + w_m x_m\right) \tag{1.6}$$

Příklad 1. Příklad jednoduché neuronové sítě Mějme černobílý obrázek o rozlišení 28 x 28 pixelů, na kterém je vyobrazeno číslo mezi 0 a 9. Stejně tak jako lidský mozek rozpozná, o které číslo se jedná, budeme chtít to samé po programu. Každému pixelu je přiřazen jeden neuron vstupní vrstvy, který si můžeme představit jako buňku obsahující reálné číslo od nuly do jedné podle světelné intenzity přiřazeného pixelu. Těmto číslům budeme říkat aktivační čísla. Nula představuje černou a jednička bílou. Všech těchto 784 neuronů představují vstupní vrstvu neuronové sítě. Ve výstupní vrstvě bude 10 neuronů představujících čísla od nuly do devíti.

Mezi vstupní a výstupní vrstvou bude navázaných několik skrytých vrstev, které se budou zaměřovat na to, aby rozeznaly určité vzory. Číslice 9 je tvořena kroužkem a ocáskem, ze vstupu vezmeme všechna aktivační čísla přiřazená jednotlivým pixelům, každý neuron ze skryté vrstvy je navázán na všechny předchozí pomocí vah. Každé aktivační číslo vynásobíme příslušnou váhou a sešteme přes všechny neurony, tak určíme, jestli se v konkrétní části obrázku nachází vzor. Jelikož tento součet může být libovolné reálné číslo, využijeme aktivační funkce sigmoid $\sigma(z) = \frac{1}{1+e^{-z}}$, která nám výslednou hodnotu promítne do intervalu [0,1]. Tím získáme hodnotu, která určí, jak moc je příslušný součet relevantní. Pokud chceme upravit důležitost určitého neuronu, můžeme k sumě předtím, než na ni zapůsobíme aktivační funkcí přidat jakékoliv číslo, které se nazývá bias. Váhy tedy udávají, jaký vzor bude neuron ve skryté vrstvě detekovat. Bias udává, jak velký součet musí vážená suma mít, aby se pak neuron stal relevantním. Takto to vypadá pro každý jeden neuron ze skryté vrstvy zvlášť. Kdybychom měli 2 skryté vrstvy, každá obsahující 16 neuronů, počet všech vah v celé neuronové síti by byl 784*16+16*16+16*10 = 12 960. Počet parametrů biasu 16+16+10=42. Celkem tedy 13 002 proměnných pro takto jednoduchou síť.

Označíme vektor vstupních dat $\mathbf{x}^0$, matici vah j-té neuronové vrstvy $\mathbf{W}^j$, bias j-té neuronové vrstvy $\mathbf{b}^j$, aktivační funkci j-té neuronové vrstvy $\mathbf{l}^j$ Funkce *predict* této neuronové sítě bude vypadat následovně

$$\mathbf{x}^0 \to \mathbf{l}^1(\mathbf{W}^1\mathbf{x}^0 + \mathbf{b}^1) = \mathbf{y}^1 \to \mathbf{l}^2(\mathbf{W}^2\mathbf{y}^1 + \mathbf{b}^2) = \mathbf{y}^2 \to \mathbf{l}^3(\mathbf{W}^3\mathbf{y}^2 + \mathbf{b}^3) = \mathbf{y}^3.$$

Zde první dvě aktivační funkce $\mathbf{l}^1$ a $\mathbf{l}^2$ mohou představovat nelineární nebo po částech lineární zobrazení a aktivační funkce výstupní vrstvy $\mathbf{l}^3$ pak například sigmoid zmíněný v Příkladu 1. Jako příklad po částech lineární aktivační funkce si můžeme uvést například ReLU definovanou jako $\mathbf{l}(x) = \max\{0, x\}$. Příkladem jiné nelineární aktivační funkce než je sigmoid může být Swish definovaná jako $\mathbf{l}(x) = \frac{x}{1+e^{-x}}$.

Zavedeme nyní náš optimalizační problém. Mějme n vstupních dat $\mathbf{x}^0_1, \ldots, \mathbf{x}^0_n$ a požadovaný výstup $\mathbf{y} \in \mathbb{R}^n$. Označíme

$$\mathbf{x} = \begin{pmatrix} \mathbf{x}^0 \\ 1 \end{pmatrix}, \mathbf{w}^T = \left(\mathbf{W}^1, \mathbf{b}^1, \mathbf{W}^2, \mathbf{b}_2 \ldots\right).$$

Vektor $\mathbf{w}^T$ je seskládán ze všech parametrů jednotlivých neuronových vrstev v celé síti. Dále mějme predikci neuronové sítě $predict(\mathbf{x}_i, \mathbf{w})$ závislou na vstupních datech a parametrech $\mathbf{w}$, které se snažíme určit. Optimalizační problém má tvar:

$$\text{minimalizuj}_{\mathbf{w}} \ \frac{1}{n}\sum_{i=1}^{n} L\left(predict(\mathbf{x}_i, \mathbf{w}), y_i\right), \tag{1.7}$$

kde funkce L označuje ztrátovou funkci. Označíme

$$\frac{1}{n}\sum_{i=1}^{n} L\left(predict(\mathbf{x}_i, \mathbf{w}), y_i\right) =: \ f(\mathbf{w}).$$

Ztrátovou funkci označíme závislou pouze na parametru $\mathbf{w}$, jelikož $\mathbf{x}$ i $y_1, \ldots, y_n$ jsou předem známé.

Nejběžnější vrstva využívaná v neuronových sítích je tzv. dense vrstva, která jednoduše udělá lineární kombinaci všech vstupů a přidá bias.

Definice 1.2.1 (Dense vrstva)**.** Mějme obecně matici paramerů $\mathbf{W}^i$. Jako výstup i-té dense vrstvy označíme

$$\mathbf{y}^i = \mathbf{l}^i\,(\mathbf{W}^i\mathbf{x} + \mathbf{b}^i),$$

kde $\mathbf{l}^i$ představuje aktivační funkci i-té dense vrstvy.

1.2.1 Regularizace

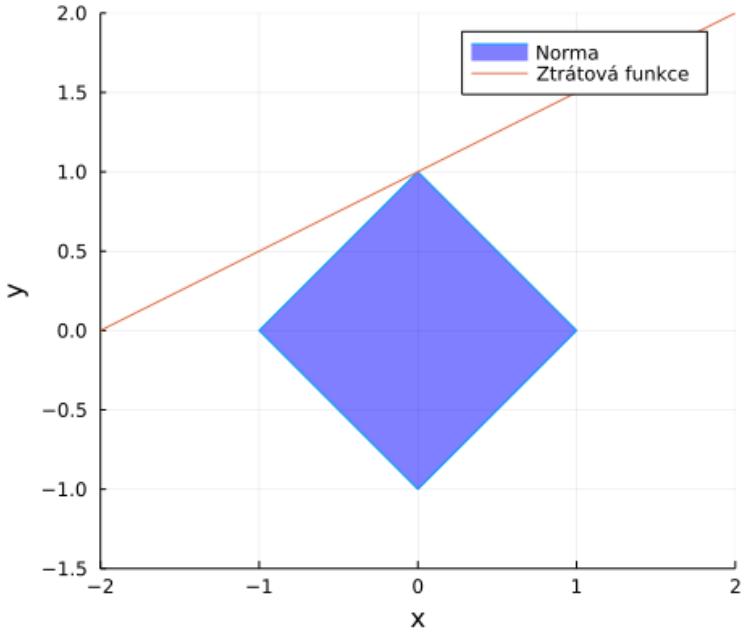
Regularizace neuronové sítě [11] je způsob zabránění jejího přeučení. Trénuje-li se neuronová síť s dostatkem volných parametrů dlouho na jednom setu dat, výsledný model pak může být přecitlivělý na chyby měření nebo na hodnoty neodpovídající datovému trendu. Získáme tedy příliš složitý model, který neodpovídá regresi. Takovýto výsledek pak dává špatné odhady pro nová testovací data. Při regularizaci pomocí normy l_1 nebo l_2 se snažíme minimalizovat účelovou funkci za dodatečné podmínky $\|\cdot\| \leq c$, kde konstanta c se určí tak, aby ztrátová funkce byla tečná této normě.

1.2.1.1 l_1 regularizace

Jedná se o typ regularizace, při které ke ztrátové funkci přidáme penalizační výraz odpovídající l_1 normě matice vah. To vzhledem k chování l_1 normy stlačuje koeficienty ztrátové funkce k nule.

$$f(\mathbf{w}) + \lambda\,\|\mathbf{w}\|_{l_1}.$$

Funkce f je zde zapsána se závislostí pouze na $\mathbf{w}$, jelikož $\mathbf{x}, y_1, \ldots, y_n$ i λ jsou předem známé. $\|\cdot\|_{l_1}$ představuje l_1 normu, čili absolutní hodnotu.



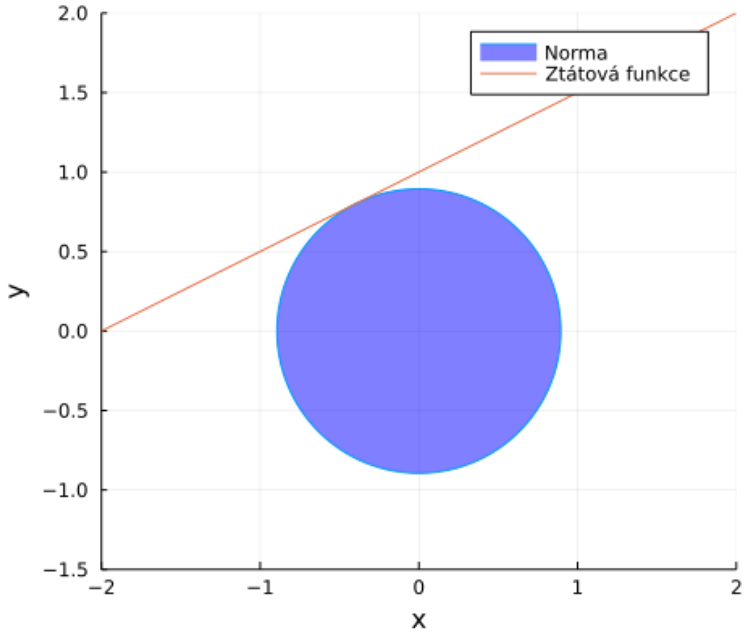
Obrázek 1.2: Vyobrazení l_1 normy společně se ztrátovou funkcí

1.2.1.2 l_2 regularizace

Podobně jako u l_1 regularizace přidáme ke ztrátové funkci výraz odpovídající l_2 normě matice vah.

$$f(\mathbf{w}) + \lambda \|\mathbf{w}\|_{l_2}.$$

$\|\cdot\|_{l_2}$ odpovídá Eukleidovské normě.



Obrázek 1.3: Vyobrazení l_2 normy společně se ztrátovou funkcí

Tento typ regularizace v konečném výsledků dává méně nulových koeficientů než l_1 regularizace, což je znázorněno na Obrázcích 1.3 a 4.1. l_1 norma má větší pravděpodobnost dotyku ztrátové funkce v bodě, který má jednu ze svých souřadnic nulovou.

1.3 Metoda největšího spádu

Pro účely naší práce budeme pracovat s metodami největšího spádu [14], které pro svůj chod počítají s gradientem. Jsou to iterativní metody prvního řádu, to znamená, že ke svému běhu využívají nejvýše derivací prvního řádu. Ve většině případů se optimalizací nedostaneme do exaktního bodu, ve kterém funkce dosahuje lokálního minima. Budeme se muset spokojit s tím, že pokles cenové funkce budeme muset zastavit splněním uspokojivého ukončovacího kritéria. Na druhou stranu jsou tyto metody velice robustní a lehké na implementaci. Budeme řešit úlohu

$$\begin{aligned} &\text{minimalizuj } f(\mathbf{w}) \\ &\text{vzhledem k } \mathbf{w} \in \mathbb{R}^d. \end{aligned}$$

Úkolem je tedy dle Věty 1.1.1 nalézt takový bod $\mathbf{w}^*$, pro který je $\nabla f(\mathbf{w}^*) = \mathbf{0}$. V obecném iteračním schématu hledáme následující krok ve tvaru

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \alpha_k \mathbf{d}_k \quad \forall k \in \mathbb{N},$$

kde α_k je skalár škálující délku kroku a $\mathbf{d}_k$ je hledaný směr poklesu. V metodě největšího spádu se jako hledaný směr $\mathbf{d}_k$ volí $-\nabla f(\mathbf{w}_k)$. Algoritmus pak vypadá takto:

Algorithm 1 Metoda největšího spádu

```
k ← 0
while není splněno ukončovací kritérium do
    w_k ← w_k − α_k ∇f(w_k)
    k ← k + 1
end while
```

Jako ukončovací kritérium bychom chtěli použít $\nabla f(\mathbf{w}_k) = \mathbf{0}$. Toho je však v aritmetice s konečnou přesností, kterou stroje využívají, většinou nemožné dosáhnout. Zastavit algoritmus tedy můžeme pomocí podmínky na maximální přípustnou hodnotu velikosti gradientu nebo případně na námi určený počet kroků.

Lemma 1.3.1. Mějme $\mathbf{w}^* \in \mathbb{R}^d$ a f spojitě diferencovatelnou na $B(\mathbf{w}^*, \epsilon)$. Je-li $\nabla f(\mathbf{w}^*) \neq \mathbf{0}$, potom $\exists \epsilon > 0$ takové, že $\forall \mathbf{w} \in B(\mathbf{w}^*, \epsilon)$ platí

$$f(\mathbf{w} - \epsilon \nabla f(\mathbf{w})) - f(\mathbf{w}) < -\frac{1}{2}\epsilon \|\nabla f(\mathbf{w}^*)\|^2 < 0.$$

Důkaz. Důkaz provedeme sporem. Nechť $\nabla f(\mathbf{w}^*) \neq \mathbf{0}$ a zároveň pro každé $\epsilon > 0$ existuje $\mathbf{w} \in B(\mathbf{w}^*, \epsilon)$ takové, že

$$f(\mathbf{w} - \epsilon \nabla f(\mathbf{w})) - f(\mathbf{w}) \geq -\frac{1}{2}\epsilon \|\nabla f(\mathbf{w}^*)\|^2.$$

Zvolíme $\epsilon = \frac{1}{n}$, pro které jistě existuje $\mathbf{w}_n \in B(\mathbf{w}^*, \frac{1}{n})$ neboli $\|\mathbf{w}_n - \mathbf{w}^*\| < \frac{1}{n}$ a

$$f(\mathbf{w}_n - \frac{1}{n}\nabla f(\mathbf{w}_n)) - f(\mathbf{w}_n) \geq -\frac{1}{2}\frac{1}{n}\|\nabla f(\mathbf{w}^*)\|^2. \tag{1.8}$$

Tímto dostáváme posloupnost $\{\mathbf{w}_n\}$. Využijeme věty o přírustku funkce. Dle [5] strana 59 existuje $\gamma_n \in (0, 1)$ takové, že

$$f(\mathbf{w}_n - \frac{1}{n}\nabla f(\mathbf{w}_n)) - f(\mathbf{w}_n) = -\frac{1}{n}\nabla f(\mathbf{w}_n)^T \nabla f(\mathbf{w}_n - \gamma_n \frac{1}{n}\nabla f(\mathbf{w}_n)). \tag{1.9}$$

Z (1.8) a (1.9) plyne

$$-\frac{1}{n}\nabla f(\mathbf{w}_n)^T \nabla f(\mathbf{w}_n - \gamma_n \frac{1}{n}\nabla f(\mathbf{w}_n)) \geq -\frac{1}{2}\frac{1}{n}\|\nabla f(\mathbf{w}^*)\|^2.$$

vynásobením kladným n dostaneme

$$-\nabla f(\mathbf{w}_n)^T \nabla f\Big(\mathbf{w}_n - \gamma_n \frac{1}{n}\nabla f(\mathbf{w}_n)\Big) \geq -\frac{1}{2}\|\nabla f(\mathbf{w}^*)\|^2. \tag{1.10}$$

Provedeme limitní přechod $n \to \infty$, z čehož $\mathbf{w}_n \to \mathbf{w}^*$, jelikož γ_n zůstane v intervalu $(0, 1)$, tudíž

$$-\|\nabla f(\mathbf{w}^*)\|^2 \geq -\frac{1}{2}\|\nabla f(\mathbf{w}^*)\|^2,$$

což je spor, jelikož $\nabla f(\mathbf{w}^*) \neq \mathbf{0}$. □

Nacházíme-li se tedy v bodě, ve kterém je gradient spojitě diferencovatelné funkce nenulový, pak vždy můžeme funkční hodnotu této funkce snížít pro všechny body v jeho epsilonovém okolí, pokud se vydáme v záporném směru tohoto gradientu. Lemma nám navíc velikost rozdílu funkční hodnoty odhaduje pomocí normy gradientu.

α_k z Algoritmu 1 musí splňovat Armijovo pravidlo:

$$f(\mathbf{w}_k - \alpha_k \nabla f(\mathbf{w}_k) \leq f(\mathbf{w}_k) - c\alpha_k \|\nabla f(\mathbf{w}_k)\|^2, \tag{1.11}$$

kde c je konstanta zvolená v intervalu $(0, 1)$. α_k splňující (1.11) můžeme nalézt tímto způsobem:

$$\begin{aligned} &1.\ \text{zvol}\quad \beta \in (0, 1) \\ &2.\ \alpha_k = \beta^q, \text{kde}\quad q = \min\{j \in \mathbb{N} : f(\mathbf{w}_k - \beta^j \nabla f(\mathbf{w}_k)) - f(\mathbf{w}_k) + \beta^{j+1}\|\nabla f(\mathbf{w}_k)\|^2 < 0\} \end{aligned} \tag{1.12}$$

Věta 1.3.1. Díky α_k zvolenému dle 1.12 máme zaručeno, že f klesne a hledané q najdeme po konečném počtu kroků.

Důkaz. Nejdříve ukážeme, že hodnota f klesne. Z lemmatu 1.3.1

$$f(\mathbf{w}_k - \beta^j \nabla f(\mathbf{w}_k)) - f(\mathbf{w}_k) < -\beta^j \frac{1}{2}\|\nabla f(\mathbf{w}_k)\|^2 < 0,$$

jelikož $\|\nabla f(\mathbf{w}_k)\|^2 > 0$, tudíž

$$f(\mathbf{w}_k - \beta^j \nabla f(\mathbf{w}_k)) < f(\mathbf{w}_k).$$

Při dokazování konečnosti hledání q vyjdeme z limity

$$\lim_{j \to \infty} \frac{f(\mathbf{w}_k - \beta^j \nabla f(\mathbf{w}_k)) - f(\mathbf{w}_k) + \beta^{j+1}\|\nabla f(\mathbf{w}_k)\|^2}{\beta^j} \tag{1.13}$$

Jelikož j se vyskytuje pouze u β, které je v intervalu $(0, 1)$, můžeme $j \to \infty$ zaměnit za $\alpha \to 0+$. Tudíž (1.13) můžeme zapsat následovně:

$$\lim_{\alpha \to 0+} \frac{f(\mathbf{w}_k - \alpha \nabla f(\mathbf{w}_k)) - f(\mathbf{w}_k)}{\alpha} + \beta\|\nabla f(\mathbf{w}_k)\|^2 = (\beta - 1)\|\nabla f(\mathbf{w}_k)\|^2 < 0, \tag{1.14}$$

kde bylo využito $\beta \in (0, 1)$. Z (1.14) vyplývá existence $j_0 \in \mathbb{N}$, takového, že pro všechna $j \geq j_0$ platí

$$f(\mathbf{w}_k - \beta^j \nabla f(\mathbf{w}_k)) - f(\mathbf{w}_k) + \beta^{j+1}\|\nabla f(\mathbf{w}_k)\|^2 < 0, \tag{1.15}$$

následně položíme $q = j_0$. Z (1.15) sice nemusíme dostat minimální q, bude avšak splňovat (1.12). □

V praxi se volí $j = 1$ a hodnota j se zvedá, dokud funkční hodnota f neklesne.

Věta 1.3.2. Mějme f spojitě diferencovatelnou na $\mathbb{R}^d$. Potom při libovolné volbě počátečního $\mathbf{w}_0$ v Algoritmu 1 je posloupnost $\{\mathbf{w}_k\}$ sestrojená tímto algoritmem buď konečná a její poslední člen je stacionárním bodem, nebo je nekonečná a každý její hromadný bod je stacionárním bodem.

Důkaz. Důkaz provedeme sporem. Nechť existuje hromadný bod $\mathbf{w}^* \in \mathbb{R}^d$, který není stacionárním bodem. Předpokládáme existenci konvergentní podposloupnosti $\{\mathbf{w}_{k_h}\}_{h\in\mathbb{N}} \subset \{\mathbf{w}_k\}_{k\in\mathbb{N}}$ takové, že $\mathbf{w}_{k_h} \to \mathbf{w}^*$ a zároveň $\nabla f(\mathbf{w}^*) \neq \mathbf{0}$. Dle Lemmatu 1.3.1 pro $\nabla f(\mathbf{w}^*) \neq \mathbf{0}$ existuje $\epsilon > 0$ a $h_0 \in \mathbb{N}$ takové, že pro všechna $h \geq h_0$ je $\mathbf{w}_{k_h} \in B(\mathbf{w}^*, \epsilon)$. Využijeme (1.12), podle kterého

$$f(\mathbf{w}_{k_h+1}) - f(\mathbf{w}_{k_h}) = \min\{f(\mathbf{w}_{k_h} - \alpha\nabla f(\mathbf{w}_{k_h})) : \alpha \geq 0\} - f(\mathbf{w}_{k_h}) \leq$$

$$\leq f(\mathbf{w}_{k_h} - \epsilon\nabla f(\mathbf{w}_{k_h})) - f(\mathbf{w}_{k_h}) < -\frac{1}{2}\epsilon\|\nabla f(\mathbf{w}^*)\|^2 := -\delta < 0,$$

jelikož z předpokladu $\nabla f(\mathbf{w}^*) \neq \mathbf{0}$ plyne $\|\nabla f(\mathbf{w}^*)\|^2 > 0$. Tedy $(\forall h \geq h_0)(f(\mathbf{w}_{k_h+1}) - f(\mathbf{w}_{k_h})) < -\delta < 0$, potřebujeme však $\mathbf{w}_{k_{h+1}}$ namísto $\mathbf{w}_{k_h+1}$. Jelikož $f(\mathbf{w}_k)$ je ostře klesající posloupnost a zároveň z definice posloupnosti indexů vybrané podposloupnosti $(\forall h \in \mathbb{N})(k_{h+1} \geq k_h + 1)$ platí:

$$f(\mathbf{w}_{k_{h+1}}) < f(\mathbf{w}_{k_h+1}) \text{ pak tedy } f(\mathbf{w}_{k_{h+1}}) - f(\mathbf{w}_{k_h}) < -\delta.$$

Posloupnost $a_h := f(\mathbf{w}_{k_h})$ má nekonečnou limitu, jelikož $(\forall h \geq h_0)(a_{h+1} - a_h < -\delta)$ pro pevné $h \geq h_0$ lze tento krok rekurentně aplikovat pro libovolné $n \in \mathbb{N}$, čili

$$a_{h+n} - a_h < -n\delta$$

limitním přechodem v n dostáváme

$$\lim_{n\to\infty} a_{h+n} \leq a_h - \lim_{n\to\infty} n\delta = -\infty$$

Což je spor, jelikož ze spojitosti f platí, že

$$\lim_{h\to\infty} f(\mathbf{w}_{k_h}) = f(\mathbf{w}^*) \in \mathbb{R}$$

ale zároveň

$$\lim_{h\to\infty} f(\mathbf{w}_{k_h}) = -\infty$$

□

Důsledkem předchozí věty je, že je-li $\{\mathbf{w}_k\}$ konvergentní, pak konverguje ke stacionárnímu bodu. Měli bychom však například funkci $f(\mathbf{w}) = -\mathbf{w}$, posloupnost bodů $\mathbf{w}_k$ generovaná metodou největšího spádu by byla divergentní. Pro zajištění existence právě jednoho minima bychom na funkci f museli ještě přidat požadavky spojitosti, konvexnosti a koercivnosti. Funkce $f : \mathbb{R}^d \to \mathbb{R} \cup \{-\infty, \infty\}$ je koercivní pokud platí $\lim_{\|\mathbf{w}\|\to\infty} f(\mathbf{w}) = \infty$.

1.4 ADAM

Metoda největšího spádu často selhává v oblastech, kde spád pro jednu složku parametru ztrátové funkce je mnohem větší než pro druhou, například v okolí sedlových bodů. Tímto problémem se již roku 1964 zabýval B. Polyak. Jeho „Heavy-ball“ metoda [13] vylepšuje metodu největšího spádu přidáním tzv. hybnostního členu, který zmenšuje oscilování při konvergenci.

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \alpha_k \nabla f(\mathbf{w}_k) + \underbrace{\theta(\mathbf{w}_k - \mathbf{w}_{k-1})}_{\text{hybnostní člen}}$$

Postupem času se začala objevovat široká škála nových hybnostních algoritmů. V dnešní době je jedním z nejpoužívanějších vylepšení metody největšího spádu ADAM [7]. Gradient se modifikuje výrazem, kterým zajistíme, že se budeme stále pohybovat správným směrem s větší rychlostí i v případě, kdy je gradient funkce malý. Navíc přidává střední hodnotu předešlých gradientů $\hat{\mathbf{m}}$, což bychom mohli chápat jako hybnost algoritmu, která se zvětšuje ve směru předešlých gradientů a zmenšuje v ostatních. Tím dosáhneme rychlejší konvergence a menších oscilací.

$$\mathbf{m}_k = \beta_1 \mathbf{m}_{k-1} + (1 - \beta_1) \nabla_{\mathbf{w}} f(\mathbf{w})$$
$$\mathbf{v}_k = \beta_2 \mathbf{v}_{k-1} + (1 - \beta_2) \nabla_{\mathbf{w}} f(\mathbf{w})^2$$

$\mathbf{m}_k$ a $\mathbf{v}_k$ jsou dva hybnostní členy, β_1 a β_2 jsou kladné pohybové parametry.

Jelikož $\mathbf{m}_k$ i $\mathbf{v}_k$ nazačátku inicializujeme jako nulové vektory, zůstávají v prvních krocích algoritmu blízké nule. K tomuto fenoménu také napomáhá, pokud jsou β_1 a β_2 blízké 1. Proto tyto vektory upravujeme následujícím způsobem.

$$\hat{\mathbf{m}}_k = \frac{\mathbf{m}_k}{1 - \beta_1^k}$$
$$\hat{\mathbf{v}}_k = \frac{\mathbf{v}_k}{1 - \beta_2^k}$$

Parametry funkce následně aktualizujeme takto:

$$\mathbf{w}_k = \mathbf{w}_{k-1} - \alpha_k \frac{\hat{\mathbf{m}}_k}{\sqrt{\hat{\mathbf{v}}_k} + \epsilon},$$

kde ϵ představuje malé kladné nenulové číslo, přidané, aby zabránilo dělení nulou. Jako další hojně využívané optimalizační metody bychom pouze zmínili RMSProp [16], či AdaGrad [2]

1.5 Stochastická metoda největšího spádu

Skloubíme-li vědomosti ze sekce 1.3 a obecnou minimalizační úlohu (1.7), můžeme za vhodných předpokladů hledat minimum ztrátové funkce pomocí metody největšího spádu následovně

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \alpha_k \frac{1}{n} \sum_{i=1}^{n} \nabla_{\mathbf{w}} L\left(predict(\mathbf{x}_i, \mathbf{w}_k), y_i\right) \nabla_{\mathbf{w}} predict(\mathbf{x}_i, \mathbf{w}_k)$$

Zde se využívá řetězového pravidla. Dolní index k označuje k-tý člen iterace. V případech, kdy máme velké množství dat, na kterých se neuronovou síť snažíme trénovat však tato iterační metoda může být časově nepřijatelná. Vrátíme se k úvodnímu Příkladu 1 zabývajícím se rozpoznáváním ručně psaných číslic. Uvažujme opět obrázky s rozlišením 28x28 pixelů na modelu stejné jednoduché neuronové sítě s 13 002 parametry. Dostáváme 13 002 derivací ztrátové funkce, do kterých bychom dávali naše data. Pracovali bychom s deseti tisíci ručně psaných obrázků s předpokladem, že ke správnému určení číslice potřebujeme alespoň jeden tisíc iterací. Celkově dostáváme řádově 10^{10} výpočtů. Metoda největšího spádu tudíž není nejvhodnější pokud pracujeme s velkým objemem dat.
Stochastická metoda největšího spádu [3] může pracovat s menším množstvím dat I náhodně vybraným z množiny $\{1, \ldots, n\}$. To jest

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \alpha_k \frac{1}{|I|} \sum_{i \in I} \nabla_{\mathbf{w}} L \left(predict(\mathbf{x}_i, \mathbf{w}_k), y_i \right) \nabla_{\mathbf{w}} predict(\mathbf{x}_i, \mathbf{w}_k),$$

kde $|I|$ označuje mohutnost mnnožiny I. Množině I se z angličtiny říká „minibatch“. Tato metoda je výrazně rychlejší než samostatná metoda největšího spádu. Během toho, co metoda největšího spádu jednou aktualizuje gradient ztrátové funkce během iterace, stochastická metoda to může udělat víckrát, avšak s menší přesností. Stochastická metoda největšího spádu využívající „minibatche“ je velice přínosná, jsou-li data ve shlucích nebo obsahují hodně přebytečných informací.

Kapitola 2

Neural Power Unit

Neural Power Unit [6], zkráceně NPU je aritmetická vrstva, která zlepšuje extrapolační vlastnosti neuronových sítí. Motivací pro její vznik bylo vytvoření vrstvy, která by aproximovala polynomy. Ty se skládají ze členů x^w, proto se budeme snažit aproximovat přímo je. NPU vychází z aritmetické vrstvy NALU [17].

Aplikuje-li se absolutní hodnota, logaritmus, exponenciála případně jakákoliv trigonometrická funkce na matici či vektor, míní se tím akce po složkách. Toto budeme zdůrazňovat pomocí znaménka $\odot$, které představuje Hadamardův součin. Součinem matice a vektoru se myslí klasický maticový součin.

Definice 2.0.1 (NALU)**.** [17] Neuronová vrstva NALU sestává z operací sčítání a násobení, které závisejí na maticích vah $\mathbf{W}$ a $\mathbf{M}$. Operace se definují následovně:

$$\begin{aligned} \text{sčítání: } \mathbf{a} &= \hat{\mathbf{W}}\mathbf{x}, \quad \text{kde } \hat{\mathbf{W}} = \tanh(\mathbf{W}) \odot \sigma(\mathbf{M}) \\ \text{násobení: } \mathbf{m} &= \exp(\hat{\mathbf{W}} \log(|\mathbf{x}| + \epsilon)) \\ \text{výstup: } \mathbf{y} &= \mathbf{a} \odot \mathbf{g} + \mathbf{m} \odot (1 - \mathbf{g}), \quad \text{kde } \mathbf{g} = \sigma(\mathbf{G}\mathbf{x}). \end{aligned} \tag{2.1}$$

$\mathbf{x}$ zde představuje vstup, $\mathbf{W}$, $\mathbf{M}$ a $\mathbf{G}$ jsou matice představující parametry.

Operace sčítání v (2.1) pochází z NAC [17], což je speciální druh lineární vrstvy. Matice $\hat{\mathbf{W}}$ sestává z čísel $\{-1, 0, 1\}$, což při aplikaci na vstupní vektor $\mathbf{x}$ umožňuje sčítání a odčítání. Vzhledem k učení neuronové sítě je však výhodnější udělat spojitou a diferencovatelnou parametrizaci $\hat{\mathbf{W}}$, což se vykazuje lepším fungováním při použití metody největšího spádu. Jak funkce sigmoid, tak hyperbolický tangenc mají své funkční hodnoty v intervalu $(-1, 1)$, tudíž i jejich součin bude v tomto rozmezí.

Operace násobení v (2.1) je schopná jako svůj výstup dávat polynomy, jelikož

$$x^w = \exp(w \log x), \text{ pro } x > 0.$$

Kvůli zabránění dělení nulou bylo ještě přidáno malé kladné reálné číslo ϵ. Avšak použití absolutní hodnoty v argumentu funkce logaritmus u operace násobení způsobovalo špatné výsledky pro negativní vstup $\mathbf{x}$. U výstupu (2.1) si můžeme všimnout brány $\mathbf{g}$, která je například $\mathbf{1}$ pokud chceme čistě sčítat/odčítat nebo $\mathbf{0}$, pokud chceme pouze násobit/dělit.

Špatné chování operace násobení v (2.1) bylo záminkou pro vznik NPU. NPU tento problém řeší pomocí komplexního logaritmu.

Máme-li nenulové komplexní číslo x, jako jeho komplexní logaritmus definujeme číslo u, které splňuje $e^u = x$, takové u budeme označovat jako $\log x$. Mějme x zapsané pomocí polárních souřadnic jako $x = re^{i\theta}$, kde $r = |x| > 0$ značí vzdálenost x od počátku souřadnic a θ je úhel, který svírá spojnice x s počátkem a reálnou osou. Potom za využití identity $e^{i\theta} = \cos(\theta) + i\sin(\theta)$ dostáváme:

$$x = re^{i\theta} = r(\cos(\theta) + i\sin(\theta)) = r\cos(\theta) + ir\sin(\theta)$$

Jelikož x je reálné, zbavíme se imaginární složky volbou $\theta = 0 + k\pi$. Dostáváme tedy

$$\log x = \log(re^{ik\pi}) = \log r + \log e^{ik\pi} = \log r + ik\pi \tag{2.2}$$

k volíme následovně:

$$k = \begin{cases} 0, & \text{pro } x > 0, \\ 1, & \text{pro } x < 0. \end{cases} \quad \text{čili} \quad \log x = \begin{cases} \log x, & \text{pro } x > 0, \\ \log|x| + i\pi, & \text{pro } x < 0, \\ \text{nedefinováno}, & \text{pro } x = 0. \end{cases}$$

2.1 Odvození NPU

Mějme komplexní matici vah $\mathbf{W}$, kterou můžeme napsat jako součet dvou reálných matic, její reálné a imaginární části $\mathbf{W} = \mathbf{W}^r + i\mathbf{W}^i$. Vyjdeme z rovnice pro násobení v Definici 2.0.1 a nahradíme reálný logaritmus komplexním viz. rovnice (2.2), epsilon zanedbáme. Připomínáme, že aplikace absolutní hodnoty, exponenciály/logaritmu/trigonometrické funkce na matici se provádí po složkách, ke zjednodušení zápisu nyní i součin dvou vektorů. Nechť tedy

$$\mathbf{r} = |\mathbf{x}|, \; k_j = \begin{cases} 1, & x_j < 0, \\ 0, & x_j > 0. \end{cases} \tag{2.3}$$

Poté

$$
\begin{aligned}
\mathbf{z} &= \exp(\mathbf{W}\log\mathbf{x}) \\
&= \exp((\mathbf{W}^r + i\mathbf{W}^i)(\log\mathbf{r} + i\pi\mathbf{k})) \\
&= \exp(\mathbf{W}^r\log\mathbf{r}) \odot \exp(i\pi\mathbf{W}^r\mathbf{k}) \odot \exp(i\mathbf{W}^i\log\mathbf{r}) \odot \exp(-\pi\mathbf{W}^i\mathbf{k}) \\
&= \exp(\mathbf{W}^r\log\mathbf{r} - \pi\mathbf{W}^i\mathbf{k}) \odot \exp(i\pi\mathbf{W}^r\mathbf{k}) \odot \exp(i\mathbf{W}^i\log\mathbf{r}).
\end{aligned} \tag{2.4}
$$

Nyní využijeme identity

$$\exp(i\theta) = \cos(\theta) + i\sin(\theta),$$

díky níž můžeme pokračovat v zápisu (2.4) jako

$$\mathbf{z} = \exp(\mathbf{W}^r\log\mathbf{r} - \pi\mathbf{W}^i\mathbf{k}) \odot (\cos(\pi\mathbf{W}^r\mathbf{k}) + i\sin(\pi\mathbf{W}^r\mathbf{k})) \odot (\cos(\mathbf{W}^i\log\mathbf{r}) + i\sin(\mathbf{W}^i\log\mathbf{r})). \tag{2.5}$$

S pomocí součtových vzorců

$$
\begin{aligned}
2\cos a\cos b &= \cos(a+b) + \cos(a-b) \\
2\sin a\sin b &= \cos(a+b) - \cos(a-b) \\
\sin(a+b) &= \cos a\sin b + \sin a\cos b
\end{aligned}
$$

již dostaneme konečnou podobu výstupu (2.5)

$$
\begin{aligned}
\mathbf{z} &= \exp(\mathbf{W}^r\log\mathbf{r} - \pi\mathbf{W}^i\mathbf{k}) \odot \left[\frac{1}{2}\cos(\pi\mathbf{W}^r\mathbf{k} + \mathbf{W}^i\log\mathbf{r}) + \frac{1}{2}\cos(\pi\mathbf{W}^r\mathbf{k} - \mathbf{W}^i\log\mathbf{r})\right. \\
&\qquad \left. + i\sin(\pi\mathbf{W}^r\mathbf{k} + \mathbf{W}^i\log\mathbf{r}) - \frac{1}{2}\cos(\pi\mathbf{W}^r\mathbf{k} - \mathbf{W}^i\log\mathbf{r}) + \frac{1}{2}\cos(\pi\mathbf{W}^r\mathbf{k} + \mathbf{W}^i\log\mathbf{r})\right] \\
&= \exp(\mathbf{W}^r\log\mathbf{r} - \pi\mathbf{W}^i\mathbf{k}) \odot \Big(\cos(\pi\mathbf{W}^r\mathbf{k} + \mathbf{W}^i\log\mathbf{r}) + i\sin(\pi\mathbf{W}^r\mathbf{k} + \mathbf{W}^i\log\mathbf{r})\Big) \\
&= \exp(\mathbf{W}^r\log\mathbf{r} - \pi\mathbf{W}^i\mathbf{k}) \odot \cos(\pi\mathbf{W}^r\mathbf{k} + \mathbf{W}^i\log\mathbf{r}) + i\exp(\mathbf{W}^r\log\mathbf{r} - \pi\mathbf{W}^i\mathbf{k}) \odot \sin(\pi\mathbf{W}^r\mathbf{k} + \mathbf{W}^i\log\mathbf{r}).
\end{aligned}
$$

NaiveNPU vychází z předchozí rovnice, kde uvažujeme pouze reálnou část, jelikož zadáváme pouze reálné vstupy a neočekáváme imaginární výstup.

Definice 2.1.1 (NaiveNPU)**.** [6] Naive Neural Power Unit s reálnými maticemi $\mathbf{W}^r$ a $\mathbf{W}^i$ definuje následovně:

$$\mathbf{z} = \exp(\mathbf{W}^r\log\mathbf{r} - \pi\mathbf{W}^i\mathbf{k}) \odot \cos(\pi\mathbf{W}^r\mathbf{k} + \mathbf{W}^i\log\mathbf{r}),$$

kde $\mathbf{r}$ a $\mathbf{k}$ jsou definované v (2.3).

Máme-li vstup $\mathbf{x}$, který má všechny složky kladné, výstup NaiveNPU nabude tvaru:

$$\mathbf{z} = \exp(\mathbf{W}^r\log\mathbf{r}) \odot \cos(\mathbf{W}^i\log\mathbf{r}) \tag{2.6}$$

Naopak pokud $\mathbf{x}$ má všechny složky záporné:

$$\mathbf{z} = \exp(\mathbf{W}^r \log \mathbf{r} - \pi \mathbf{W}^i \mathbf{1}) \odot \cos(\pi \mathbf{W}^r \mathbf{1} + \mathbf{W}^i \log \mathbf{r}) \tag{2.7}$$

Pro názornost si rozepíšeme vrstvu NaiveNPU pro m vstupů a 1 výstup. Nechť tedy

$$\mathbf{x} \in \mathbb{R}^{m\times 1},\ \mathbf{W} \in \mathbb{R}^{1\times m},\ \mathbf{r} = |\mathbf{x}|,\ k_j = \begin{cases} 1, & x_j < 0, \\ 0, & x_j > 0. \end{cases}$$

$$\mathbf{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_m \end{pmatrix}, \mathbf{W} = \mathbf{W}^r + i\mathbf{W}^i = (w_1^r \cdots w_m^r) + i\left(w_1^i \cdots w_m^i\right)$$

Výstup tedy bude mít tvar

$$\begin{aligned} z &= \exp(\mathbf{W}^r \log \mathbf{r} - \pi \mathbf{W}^i \mathbf{k}) \cos(\pi \mathbf{W}^r \mathbf{k} + \mathbf{W}^i \log \mathbf{r}) \\ &= \exp\Big(\sum_{s=1}^{m} w_s^r \log r_s\Big) \exp\Big(-\pi \sum_{s=1}^{m} w_s^i k_s\Big) \cos\Big(\pi \sum_{s=1}^{m} w_s^r k_s + \sum_{s=1}^{m} w_s^i \log r_s\Big) \\ &= \exp(w_1^r \log r_1) \ldots \exp(w_m^r \log r_m) \exp\Big(-\pi \sum_{s=1}^{m} w_s^i k_s\Big) \cos\Big(\pi \sum_{s=1}^{m} w_s^r k_s + \sum_{s=1}^{m} w_s^i \log r_s\Big) \\ &= r_1^{w_1^r} \ldots r_m^{w_m^r} \exp\Big(-\pi \sum_{s=1}^{m} w_s^i k_s\Big) \cos\Big(\pi \sum_{s=1}^{m} w_s^r k_s + \sum_{s=1}^{m} w_s^i \log r_s\Big). \end{aligned}$$

Pokud má vektor $\mathbf{x}$ všechny složky záporné, výstup bude mít tvar

$$z = x_1^{w_1^r} \ldots x_m^{w_m^r} \exp\Big(-\pi \sum_{s=1}^{m} w_s^i\Big) \cos\Big(\pi \sum_{s=1}^{m} w_s^r + \sum_{s=1}^{m} \log x_s^{w_s^i}\Big)$$

Naopak, pokud má vektor $\mathbf{x}$ všechny složky kladné, získáme

$$z = x_1^{w_1^r} \ldots x_m^{w_m^r} \cos\Big(\sum_{s=1}^{m} \log x_s^{w_s^i}\Big)$$

2.2 Vlastnosti NPU

V této sekci si odvodíme chování NaiveNPU v různých případech a popíšeme si vlastnosti této vrstvy.

Zavedeme nové značení. Chceme-li zapsat strukturu neuronové sítě, to jest jak na sebe různé vrstvy navazují, budeme je po řadě psát jako $f \to g \to h \to \ldots$, kde f, g, h představují neuronové vrstvy. Počet vstupů, výstupů případně použitou aktivační funkci budeme psát po řadě jako argumenty u konkrétních názvů neuronových vrstev.

2.2.1 Polynomiální aproximace

Podíváme se, jak NaiveNPU aproximuje polynomy.

Věta 2.2.1. Kombinací NaiveNPU a Dense vrstvy jsme schopni vyjádřit jakýkoliv polynom

$$p(x) = \sum_{l=0}^{m} a_l x^l \sim NaiveNPU(1, m) \to Dense(m, 1, identity).$$

Důkaz. Nyní uvažujeme 1 nenulový vstup a m výstupů vrstvy NaiveNPU, mocniny polynomu jsou přirozená čísla.

$$x \in \mathbb{R}^{1\times 1},\ \mathbf{W} \in \mathbb{R}^{m\times 1},\ r = |x|$$

Nechť má matice $\mathbf{W}$ tento tvar:

$$\mathbf{W} = \mathbf{W}^r + i\mathbf{W}^i = \begin{pmatrix} 1 \\ \vdots \\ m \end{pmatrix} + i \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}$$

Důkaz rozdělíme na dvě části podle hodnoty vstupu.

Pro kladný vstup x bude mít výstup NPU dle (2.6) tvar

$$\exp(\mathbf{W}^r \log \mathbf{r}) \odot \cos(\mathbf{W}^i \log \mathbf{r})$$

a výstup NaiveNPU takto

$$\mathbf{z} = \begin{pmatrix} x^1 \\ \vdots \\ x^m \end{pmatrix}$$

Matice vah $\mathbf{A} \in \mathbb{R}^{1\times m}$ vrstvy Dense pro výstup NaiveNPU pak udělá lineární kombinaci a přidá bias, neboli v tomto případě absolutní člen polynomu.

$$\mathbf{A} = \begin{pmatrix} a_1 & \cdots & a_m \end{pmatrix},\ b = a_0$$

Takto již získáme hledaný polynom $p(x) = \sum_{l=0}^{m} a_l x^l$.

Pro záporný vstup nabude výstup NaiveNPU dle (2.7) tvaru

$$\exp(\mathbf{W}^r \log \mathbf{r} - \pi\mathbf{W}^i) \odot \cos(\pi\mathbf{W}^r + \mathbf{W}^i \log \mathbf{r})$$

Matice $\mathbf{W}$ bude mít stejný tvar jako v předešlém případě a výstup tedy bude vypadat následovně

$$\mathbf{z} = \begin{pmatrix} (-x)^1 \\ (-x)^2 \\ (-x)^3 \\ \vdots \\ (-x)^m \end{pmatrix} \begin{pmatrix} -1 \\ 1 \\ -1 \\ \vdots \\ (-1)^m \end{pmatrix} = \begin{pmatrix} x^1 \\ x^2 \\ x^3 \\ \vdots \\ x^m \end{pmatrix}$$

nezávisle na lichosti, respektive sudosti m tedy znovu dostaneme stejný výstup jako pro v předchozím případě. Matice Dense vrstvy tedy bude mít stejný tvar jako pro kladný vstup

$$\mathbf{A} = \begin{pmatrix} a_1 & a_2 & a_3 & \cdots & a_m \end{pmatrix},\ b = a_0.$$

Z čehož tedy opět dostaneme hledaný polynom $p(x) = \sum_{l=0}^{m} a_l x^l$. □

Příklad 1. Polynom $2x^3+5x^2+5$ bychom dostali pomocí $NaiveNPU(1,2) \to Dense(2,1,identity)$ následovně $x \xrightarrow{NaiveNPU(1,2)} (x^3, x^2) \xrightarrow{Dense(2,1,identity)} 2x^3 + 5x^2 + 5$.

Kombinací více NaiveNPU a Dense vrstev jsme schopni vytvářet složitější zobrazení než polynomy, např. racionální lomené funkce.

Věta 2.2.2. Mějme

$$H(x) = \frac{p(x)}{q(x)}, \text{ kde } p(x) = \sum_{l=0}^{m} a_l x^l,\ q(x) = \sum_{l=0}^{p} b_l x^l \text{ nenulový.}$$

Pak $H(x)$ vznikne jako $NaiveNPU(1, m+p) \to Dense(m+p, 2, identity) \to NaiveNPU(2, 1)$

Důkaz. Nechť

$$x \in \mathbb{R}^{1\times 1},\ \mathbf{W}_1 \in \mathbb{Z}^{(m+p)\times 1},\ \mathbf{A} \in \mathbb{R}^{2\times(m+p)},\ \mathbf{W}_2 \in \mathbb{Z}^{1\times 2},\ r = |x|$$

postup bude obdobný jako v důkazu Věty 2.2.1. Nejprve rozebereme případ kladného vstupu x. Nechť má $\mathbf{W}_1$ tvar

$$\mathbf{W}_1 = \mathbf{W}_1^r + i\mathbf{W}_1^i = \begin{pmatrix} 1 \\ \vdots \\ m \\ 1 \\ \vdots \\ p \end{pmatrix} + i \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}$$

Výstup první NaiveNPU vrstvy bude mít dle (2.6) tvar

$$\mathbf{z}_1 = \exp(\mathbf{W}^r \log \mathbf{r}) \odot \cos(\mathbf{W}^i \log \mathbf{r}).$$

Výstup tedy bude mít tvar

$$\mathbf{z}_1 = \begin{pmatrix} x^1 \\ \vdots \\ x^m \\ x^1 \\ \vdots \\ x^p \end{pmatrix}$$

Pro kýžený výsledek musí matice vah vrstvy Dense nabýt tvaru

$$\mathbf{A} = \begin{pmatrix} a_1 & \cdots & a_m & 0 & \cdots & 0 \\ 0 & \cdots & 0 & b_1 & \cdots & b_p \end{pmatrix},\ b = \begin{pmatrix} a_0 \\ b_0 \end{pmatrix}$$

Působením Dense vrstvy na výstup první NaiveNPU vrstvy dostaneme

$$\mathbf{z}_2 = \begin{pmatrix} p(x) \\ q(x) \end{pmatrix}$$

Matice vah v poslední vrstvě bude mít tvar

$$\mathbf{W}_2 = \mathbf{W}_2^r + i\mathbf{W}_2^i = \begin{pmatrix} 1 & -1 \end{pmatrix} + i\begin{pmatrix} 0 & 0 \end{pmatrix}$$

Takto již získáme $H(x)$

tj. $x \xrightarrow{NPU(1,m+p)} (x_1, \ldots, x_{m+p}) \xrightarrow{Dense(m+p,2,identity)} (p(x), q(x)) \xrightarrow{NPU(2,1)} H(x)$.

Pro záporný vstup by byl postup dokazování obdobný. □

2.2.2 Nesprávné chování

Zde si ukážeme, jak může NaiveNPU v kombinaci s jinými vrstvami vyprodukovat špatný výsledek. Uvažujme nyní následující schéma:

$$NaiveNPU \to g \to MSE, \tag{2.8}$$

kde g označuje libovolnou vrstvu neuronové sítě, MSE označuje střední kvadratickou odchylku. Budeme zkoumat možnost špatného chování pro určité nastavení parametrů.
Nechť $\mathbf{x}_k$ jsou vstupní data do vrstvy NaiveNPU, která já závislá na parametrech $\mathbf{W}$, tuto vrstvu označíme $f(\mathbf{W}, \mathbf{x}_k)$ a nechť g je určena parametry $\mathbf{V}$, značme $g(\mathbf{V}, \cdot)$, $\mathbf{y}$ nechť je vektor předem známých dat. Řešíme tedy minimalizační problém tvaru:

$$\text{minimalizuj}_{\mathbf{W},\mathbf{V}} \ \frac{1}{2}\sum_{k=1}^{K} (g(\mathbf{V}, f(\mathbf{W}, \mathbf{x}_k)) - y_k)^2 \tag{2.9}$$

Pro jedoduchost uvažujme vstupní data $\mathbf{x}_k$ pouze kladná a matici vah bez imaginární složky, tj. $\mathbf{W} = \mathbf{W}^r$. V následující větě ukážeme, jak může NaiveNPU za určitých předpokladů poskytnout nesprávný výsledek.

Věta 2.2.3. Nechť $\mathbf{x}_k$ jsou kladná vstupní data schématu (2.8) a matice vah parametrizující NaiveNPU má nulovou imaginární složku, to jest $\mathbf{W} = \mathbf{W}^r$. Označme výstup NaiveNPU z (2.8) jako $\mathbf{z}$. Dále nechť existuje $\mathbf{V}$ tak, že

$$\nabla_{\mathbf{z}} g(\mathbf{V}, \mathbf{1}) = \mathbf{0} \tag{2.10}$$

a

$$\nabla_{\mathbf{V}} g(\mathbf{V}, \mathbf{1}) = \mathbf{0} \vee g(\mathbf{V}, \mathbf{1}) = \frac{1}{K}\sum_{k=1}^{K} y_k. \tag{2.11}$$

Pak $\mathbf{W} = \mathbf{0}$ společně s $\mathbf{V}$ tvoří stacionární bod (2.9).

Důkaz. Nejprve označme účelovou funkci (2.9) jako

$$M(\mathbf{W}, \mathbf{V}) = \frac{1}{2}\sum_{k=1}^{K} (g(\mathbf{V}, f(\mathbf{W}, \mathbf{x}_k)) - y_k)^2$$

Výstup NaiveNPU pak dle Definice 2.1.1 nabyde tvaru

$$\mathbf{z}_k = \exp(\mathbf{W} \log \mathbf{x}_k).$$

Schéma (2.8) bude pro názornost vypadat pro jeden určitý vzorek $\mathbf{x}$ takto:

$$\mathbf{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \xrightarrow{f(\mathbf{W},\cdot)} \exp(\mathbf{W}\log\mathbf{x}) = \exp\begin{pmatrix} w_{11}\log x_1 + w_{12}\log x_2 + \cdots + w_{1n}\log x_n \\ w_{21}\log x_1 + w_{22}\log x_2 + \cdots + w_{2n}\log x_n \\ \vdots \\ w_{m1}\log x_1 + w_{m2}\log x_2 + \cdots + w_{mn}\log x_n \end{pmatrix} =$$

$$= \exp\begin{pmatrix} \log(x_1^{w_{11}} x_2^{w_{12}} \cdots x_n^{w_{1n}}) \\ \log(x_1^{w_{21}} x_2^{w_{22}} \cdots x_n^{w_{2n}}) \\ \vdots \\ \log(x_1^{w_{m1}} x_2^{w_{m2}} \cdots x_n^{w_{mn}}) \end{pmatrix} = \begin{pmatrix} x_1^{w_{11}} x_2^{w_{12}} \cdots x_n^{w_{1n}} \\ x_1^{w_{21}} x_2^{w_{22}} \cdots x_n^{w_{2n}} \\ \vdots \\ x_1^{w_{m1}} x_2^{w_{m2}} \cdots x_n^{w_{mn}} \end{pmatrix} =: \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_m \end{pmatrix} = \mathbf{z} \xrightarrow{g(\mathbf{V},\cdot)} \hat{\mathbf{y}}$$

Platí

$$\nabla_{\mathbf{W}} M(\mathbf{W},\mathbf{V}) = \sum_{k=1}^{K} (g(\mathbf{V}, f(\mathbf{W},\mathbf{x}_k)) - y_k)\, \nabla_{\mathbf{z}} g(\mathbf{V},\mathbf{z})\, \nabla_{\mathbf{W}} f(\mathbf{W},\mathbf{x}_k),$$

$$\nabla_{\mathbf{V}} M(\mathbf{W},\mathbf{V}) = \sum_{k=1}^{K} (g(\mathbf{V}, f(\mathbf{W},\mathbf{x}_k)) - y_k)\, \nabla_{\mathbf{V}} g(\mathbf{V},\mathbf{z}).$$

Vidíme, že pokud $\mathbf{W} = \mathbf{0}$, pak $\mathbf{z} = \mathbf{1}$ nezávisle na vstupních datech. Dále

$$\nabla_{w_{ij}} f(\mathbf{W},\mathbf{x}) = \nabla_{w_{ij}} \exp(\mathbf{W}\log\mathbf{x}) = \nabla_{w_{ij}} \exp(w_{ij}\log x_j) = \exp(w_{ij}\log x_j)\log x_j = x_j^{w_{ij}}\log x_j$$

což se rovná $\log x_j$ kdykoliv je $w_{ij} = 0$, čili $\nabla_{\mathbf{W}} f(\mathbf{W},\mathbf{x}_k) = \log\mathbf{x}_k$. Z tohoto pozorování a předpokladů dostáváme

$$\nabla_{\mathbf{W}} M(\mathbf{0},\mathbf{V}) = \sum_{k=1}^{K} (g(\mathbf{V},\mathbf{1}) - y_k) \underbrace{\nabla_{\mathbf{z}} g(\mathbf{V},\mathbf{1})}_{=\mathbf{0}\text{ z (2.10)}} \log\mathbf{x}_k = \mathbf{0}, \tag{2.12}$$

$$\nabla_{\mathbf{V}} M(\mathbf{0},\mathbf{V}) = \sum_{k=1}^{K} (g(\mathbf{V},\mathbf{1}) - y_k) \nabla_{\mathbf{V}} g(\mathbf{V},\mathbf{1}) = \mathbf{0}. \tag{2.13}$$

Nulu v rovnici (2.13) dostaneme splněním alespoň jedné podmínky v (2.11). Tudíž $(\mathbf{0},\mathbf{V})$ je stacionárním bodem (2.9). □

Vezměme v potaz například lineární regresi a za vrstvu g v (2.8) dosaďme Dense vrstvu. Předpoklady Věty 2.2.3 by byly u Dense vrstvy splněny například pro nulové koeficienty v lineární kombinaci a bias tvaru $b = \frac{1}{K}\sum_{k=1}^{K} y_k$.

Uvažujme nyní, že data $\mathbf{x}$ budeme brát z intervalu [-1,1] a předem známý výsledek, který se budeme snažit odhadnout bude tvaru $y = 2x^2$, podle Věty 2.2.1 lze takový polynom aproximovat jako

$$y \sim NaiveNPU(1,1) \to Dense(1,1,identity),$$

stacionární bod neuronové sítě v tomto případě bude pro parametry NaiveNPU $\mathbf{W} = (2)$ a parametry Dense vrstvy $\mathbf{A} = (2), b = 0$. Zároveň ale podle Věty 2.2.3 bude mít tato neuronová

síť stacionární bod ve $\mathbf{W} = (0), \mathbf{A} = (0), b = \frac{1}{1-(-1)} \int_{-1}^{1} 2x^2\, dx = \frac{2}{3}$, to by znamenalo, že se kvadratickou funkci bude NaiveNPU snažit aproximovat pomocí konstantní funkce. Skutečně, po implementaci tohoto příkladu v jazyce Julia v1.6.2 pro definiční obor [-1,1] s krokem 0.00001 a inicializaci parametrů

$$\mathbf{W}^r = 0, \mathbf{W}^i = 0, \mathbf{A} = 0, b = 0.6666733 \text{ (střední hodnota funkce } 2x^2 \text{ na intervalu [-1,1])}$$

dostáváme hodnoty l_2 norem gradientů jednotlivých parametrů vrstev

$$||\nabla \mathbf{W}^r|| = 0.0, ||\nabla \mathbf{W}^i|| = 0.0, ||\nabla \mathbf{A}|| = 6.11022040e-8, || = 0.0, ||\nabla b|| = 6.11022040e-8.$$

Záleží tedy na počáteční inicializaci parametrů NaiveNPU, podle které pak bude metoda největšího spádu konvergovat buď k jednomu nebo druhému stacionárnímu bodu.

Kapitola 3

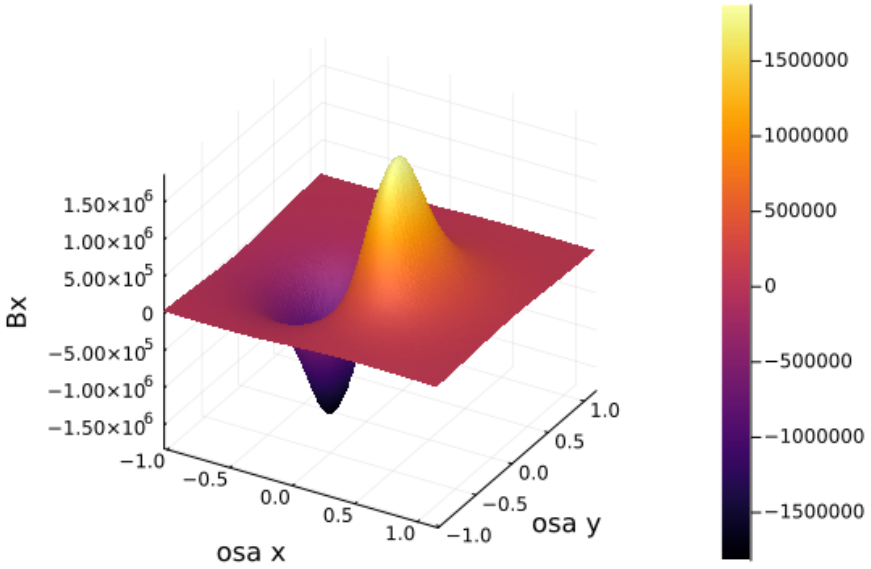
Data využívaná v praktické části

V této práci budeme dále využívat data pocházející z reálného experimentu (citace) zaměřeném na návrhu systému pro ovládání miniaturního robota.

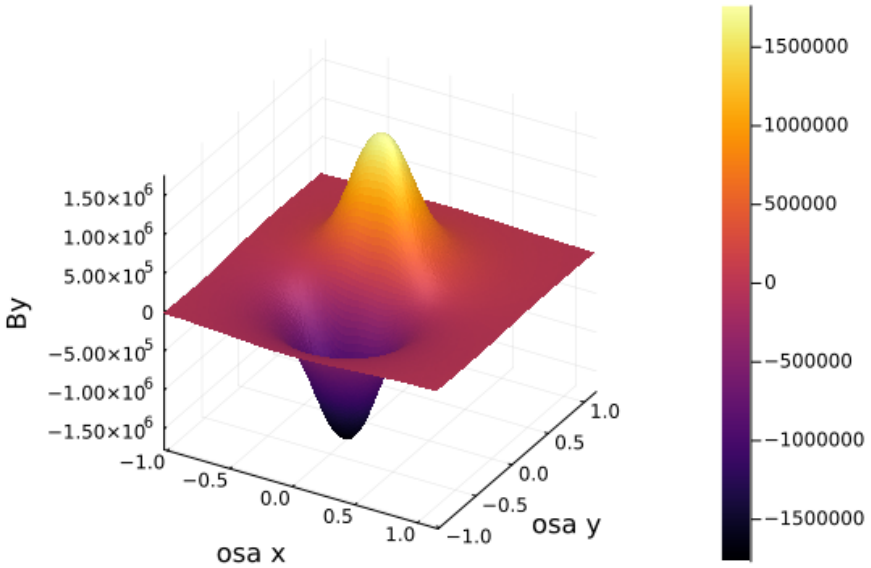
Robot byl umístěn na čtvercovém podkladu, pod kterým byly dva páry rovinných cívek svírající kolmý úhel. Každý z párů civek zajišťoval pohyb robota v jedné dimenzi. Pohybu bylo dosaženo vytvořením periodického magnetického pole, které vzniklo jako důsledek sekvenční aplikace kladného a záporného elektrického proudu.

Pro experiment bylo vytvořeno několik minirobotů. Tyto modely robotů se lišily především počtem magnetů. Jako nejvhodnější se ukázala verze modelu robota, která sestávala z pěti permanentních magnetek rozmístěných stejně jako na hrací kostce, kdy rohové magnety byly orientovány jižním pólem dolů a středový magnet byl orientován severním pólem dolů. Tento model byl vybrán, jelikož oproti ostatním variantám dosahoval při pokusu nejlepšímu poměru rychlosti a nosnosti.

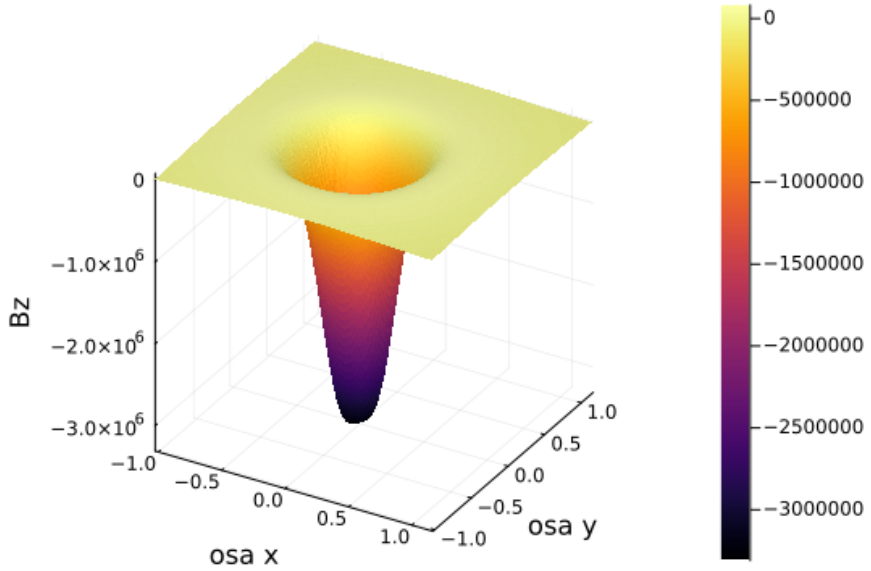
Data vyobrazená níže jsou pořízena pomocí magnetometru, zařízení pro měření magnetické indukce $\mathbf{B} = [Bx, By, Bz]$. Jedná se o dvourozměrné skeny magnetického pole miniaturního robota s využitím všech magnetometrů a jemného posunu s konečným rozlišením 220px x 220px.



Obrázek 3.1: Sken magnetického pole robota složky Bx



Obrázek 3.2: Sken magnetického pole robota složky By



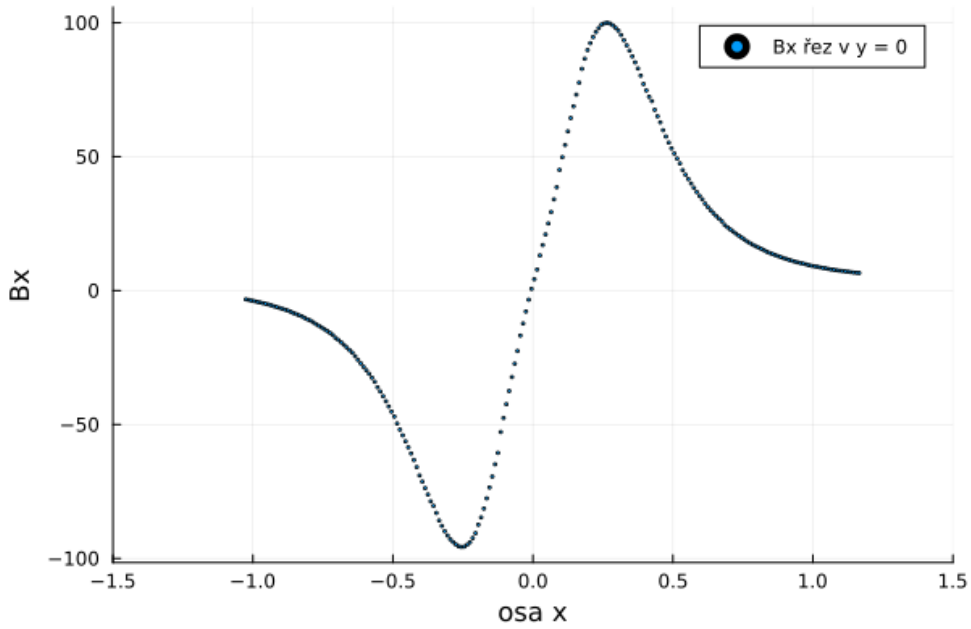
Obrázek 3.3: Sken magnetického pole robota složky Bz

Naměřená data mohou být zatížena chybami vyvstávajícími z fyzikální podstaty měření. Magnetické pole v okolí robota může být ovlivněno feromagnetickými předměty v okolí či vlivem externího magnetického pole.

Kapitola 4

NaiveNPU v 1D

V této kapitole se zaměříme na využití neuronové sítě sestávající z vrstev NaiveNPU a Dense pro aproximaci jednodimenzionálních normalizovaných dat. Pokusíme se vylepšit výsledek obdržený z bakalářské práce [15] a prodiskutujeme výhody a nevýhody tohoto postupu.



Obrázek 4.1: Řez přeškálovanými daty odpovídajícím složce Bx v y = 0

Vycházíme z popisu magnetického pole dle [12] pomocí rovnice pro magnetický dipól

$$\mathbf{B}(x, y, z, t) = \frac{\mu_0}{4\pi}\left[\frac{3(\mathbf{m}\cdot\mathbf{r})\mathbf{r} - \mathbf{m}r^2}{r^5}\right]e^{-i\omega t}, \tag{4.1}$$

kde μ_0 je permeabilita vakua, $\mathbf{r}$ jest vektor pozice, ve kterém hodnotu měříme vzhledem k počátku souřadnic, $|\mathbf{r}| = r$, $\mathbf{m}$ představuje magnetický moment cívky, ω je frekvence. t je čas.

V bakalářské práci [15] jsme nejpřesnější aproximace dat dosáhli pomocí racionální lomené funkce tvaru

$$Q(x) = \frac{f(x)}{g(x)},$$

kde

$$f(x) = p_1 \frac{x - p_2}{p_3} + p_4$$

a

$$g(x) = \left(\frac{x - p_5}{p_6}\right)^4 + p_7\left(\frac{x - p_8}{p_9}\right)^3 + p_{10}\left(\frac{x - p_{11}}{p_{12}}\right)^2 + p_{13}\frac{x - p_{14}}{p_{15}} + p_{16}.$$

$(p_1, ..., p_{16})$ = (25.2, 0.004, 0.116, 0.688, 0.014, 0.43, -1.22, -0.27, 1.25, 0.251, 0.6, 0.72, 1.17, -0.34, 1.38, 0.001).
Toto můžeme zjednodušit do podoby

$$Q(x) = \frac{a_1 x - a_2}{a_3 x^4 - a_4 x^3 + a_5 x^2 + a_6 x + a_7}, \tag{4.2}$$

kde $(a_1, \ldots, a_7) = (217.241, -1.557, 29.250, -2.263, 0.013, 0.130, 0.451)$
Následně se pokusíme tento polynom obdržet pomocí neuronové sítě obsahující vrstvy NaiveNPU a Dense, dle Věty 2.2.2 je toto možné. Konstrukční důkaz Věty 2.2.2 nám poslouží jako vodítko.

Příklad 1. Racionální lomený polynom $Q(x)$ ve tvaru 4.2 obdržíme z neuronové sítě sestavené dle následujících úvah. Hodnotu dat řezu v Bx složce známe celkem ve 220 bodech, proto bude mít vstup do neuronové sítě rozměry $\mathbf{x} \in \mathbb{R}^{1,220}$. V čitateli $Q(x)$ se proměnná x vyskytuje jednou, ve jmenovateli čtyřikrát, celkem tedy pětkrát v celém polynomu, proto reálná matice první NaiveNPU vrstvy bude mít rozměry $\mathbf{W}_1^r \in \mathbb{R}^{5,1}$ s koeficienty odpovídajícím mocninám u příslušných proměnných. Imaginární matice všech NaiveNPU vrstev budou nulové, jelikož neočekáváme výsledek s imaginární složkou.

$$\mathbf{x} = (x_1, \ldots, x_{220}), \mathbf{W}_1^r = \begin{pmatrix} 1 \\ 4 \\ 3 \\ 2 \\ 1 \end{pmatrix}, \mathbf{W}_1^i = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

Výstup $\mathbf{z}_1$ první vrstvy $NaiveNPU(1, 5)$ s maticemi $\mathbf{W}_1^r$ a $\mathbf{W}_1^i$ bude mít tvar

$$\mathbf{z}_1 = \begin{pmatrix} \mathbf{x} \\ \mathbf{x}^4 \\ \mathbf{x}^3 \\ \mathbf{x}^2 \\ \mathbf{x}, \end{pmatrix}.$$

zde se mocnění vektoru míní po složkách. Následně vytvoříme čitatele a jmenovatele polynomu $Q(x)$ pomocí vrstvy $Dense(5, 2)$ s maticemi $\mathbf{A} \in \mathbb{R}^{2,5}$ a $\mathbf{b} \in \mathbb{R}^{2,1}$

$$\mathbf{A} = \begin{pmatrix} a_1 & 0 & 0 & 0 & 0 \\ 0 & a_3 & a_4 & a_5 & a_6 \end{pmatrix}, \mathbf{b} = \begin{pmatrix} a_2 \\ a_7 \end{pmatrix}$$

Výsledek této vrstvy bude mít tvar

$$\mathbf{z}_2 = \begin{pmatrix} a_1\mathbf{x} + a_2 \\ a_3\mathbf{x}^4 + a_4\mathbf{x}^3 + a_5\mathbf{x}^2 + a_6\mathbf{x} + a_7 \end{pmatrix}$$

Konečná výsledek již obdržíme pomocí vrstvy $NaiveNPU(2,1)$, kdy druhou složku $\mathbf{z}_2$ umocníme na mínus první, tím z ní uděláme jmenovatel polynomu $Q(x)$.

$$\mathbf{W}_2^r = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \mathbf{W}_2^i = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}.$$

Neuronová síť tedy bude mít podobu $NaiveNPU(1,5) \rightarrow Dense(5,2) \rightarrow NaiveNPU(2,1)$ s maticemi uvedenými v postupu příkladu.

Závěr

Literatura

[1] Martin Anthony and Peter L Bartlett. *Neural network learning: Theoretical foundations*. cambridge university press, 2009.

[2] Hilal Asi, John Duchi, Alireza Fallah, Omid Javidbakht, and Kunal Talwar. Private adaptive gradient methods for convex optimization. In *International Conference on Machine Learning*, pages 383–392. PMLR, 2021.

[3] Léon Bottou. Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*, pages 421–436. Springer, 2012.

[4] Edwin KP Chong and Stanislaw H Zak. *An introduction to optimization*. John Wiley & Sons, 2004.

[5] Johannes Jisse Duistermaat and Johan AC Kolk. *Multidimensional real analysis I: differentiation*, volume 86. Cambridge University Press, 2004.

[6] Niklas Heim, Tomas Pevny, and Vasek Smidl. Neural power units. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 6573–6583. Curran Associates, Inc., 2020.

[7] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

[8] Sotiris B Kotsiantis, Ioannis D Zaharakis, and Panayiotis E Pintelas. Machine learning: a review of classification and combining techniques. *Artificial Intelligence Review*, 26(3):159–190, 2006.

[9] Dastan Maulud and Adnan M Abdulazeez. A review on linear regression comprehensive in machine learning. *Journal of Applied Science and Technology Trends*, 1(4):140–147, 2020.

[10] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[11] Ismoilov Nusrat and Sung-Bong Jang. A comparison of regularization techniques in deep neural networks. *Symmetry*, 10(11):648, 2018.

[12] Valter Pasku, Alessio De Angelis, Guido De Angelis, Darmindra D Arumugam, Marco Dionigi, Paolo Carbone, Antonio Moschitta, and David S Ricketts. Magnetic field-based positioning systems. *IEEE Communications Surveys & Tutorials*, 19(3):2003–2017, 2017.

[13] Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *Ussr computational mathematics and mathematical physics*, 4(5):1–17, 1964.

[14] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[15] Novotnỳ Stanislav. Lokalizace mikrorobota z magnetickỳch senzor. B.S. thesis, České vysoké učení technické v Praze. Vypočetní a informační centrum., 2020.

[16] Tijmen Tieleman, Geoffrey Hinton, et al. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[17] Andrew Trask, Felix Hill, Scott E Reed, Jack Rae, Chris Dyer, and Phil Blunsom. Neural arithmetic logic units. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.