VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
FACULTY OF MECHANICAL ENGINEERING

ÚSTAV MATEMATIKY
INSTITUTE OF MATHEMATICS

TVORBA UMĚLÉ NEURONOVÉ SÍTĚ PRO VÝPOČET TERMODYNAMICKÝCH VELIČIN

APPLICATION OF THE ARTIFICIAL NEURAL NETWORK TO CALCULATE THE THERMODYNAMIC PROPERTIES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Martin Groman**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Tomáš Mauder, Ph.D.**
SUPERVISOR

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Ústav matematiky
Student:	**Bc. Martin Groman**
Studijní program:	Aplikované vědy v inženýrství
Studijní obor:	Matematické inženýrství
Vedoucí práce:	**Ing. Tomáš Mauder, Ph.D.**
Akademický rok:	2018/19

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

Tvorba umělé neuronové sítě pro výpočet termodynamických veličin

Stručná charakteristika problematiky úkolu:

Umělá neuronová síť je jeden z výpočetních modelů používaných v umělé inteligenci, který je inspirován činností lidského mozku. Umělé neuronové sítě se skládají z neuronů propojených vazbami, které se ohodnocují příslušnou váhou. V dnešní době jsou umělé neuronové sítě používány v mnoha oblastech, jako jsou výrobní procesy, analýza obrazu, ekonomika a finance, expertní systémy, zdravotnictví, telekomunikace, atd. V této práci se bude diplomant zabývat tvorbou a možnostmi využití neuronové sítě v termodynamice. Tvorba modelu proběhne v programovacím jazyce Python za využití termodynamických knihoven. Výsledná umělá neuronová síť by potom měla po svém naučení vypočítat chybějící termodynamické stavové veličiny.

Cíle diplomové práce:

Cílem práce je vytvoření umělé neuronové sítě v programovacím jazyce Python (samostatně spustitelný kód) a její ověření na příkladech z oblasti termodynamiky. Podrobně by měl být rozebrán výběr topologie sítě, matematický popis neuronové sítě a matematický popis vybraných testovacích termodynamických úloh. Předpokládá se zde přístup strojového učení „učení s učitelem“, který by měl být do značné míry automatizován. Výstupem práce by mělo výt ověření sestavené neuronové sítě na vybrané tématice z oblasti termodynamiky a stavových veličin.

Seznam doporučené literatury:

TAYLOR, J. G. Neural Networks and Their Applications. Wiley, 1996. ISBN: 978-0-471-96282-3.

KULSHRESTHA, S. K. A textbook of applied thermodynamics, steam and thermal engineering. Vikas, 1983. ISBN: 978-0-706-92158-8.

CENGEL, Y. A. Introduction to Thermodynamics and Heat Transfer. McGraw-Hill, 2009. ISBN: 978--
-071-28773-9.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2018/19

V Brně, dne

L. S.

prof. RNDr. Josef Šlapal, CSc. ředitel ústavu	doc. Ing. Jaroslav Katolický, Ph.D. děkan fakulty

Abstrakt

Tato diplomová práce se zabývá tvorbou umělé neuronové sítě pro výpočet měrného objemu vodní páry. V práci je popsán výběr a konstrukce dané sítě. Výsledkem práce je spustitelná aplikace, která počítá měrný objem vodní páry pro zadaný tlak a teplotu, právě pomocí neuronových sítí.

Abstract

This master thesis is dealing with application of an artificial neural network for calculating specific volume of steam. There is described type and construction of the needed neural network. The main outcome of this work is an executable programme, which calculates specific volume of steam for given pressure and temperature, using neural nets.

klíčová slova

neuronová síť, tensorflow, optimalizace, vodní pára, výpočet termodynamických veličin

keywords

neural network, tensorflow, optimization, steam, calculation of thermodynamic properties

GROMAN, Martin. *Tvorba umělé neuronové sítě pro výpočet termodynamických veličin.* Brno, 2019. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/116370. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav matematiky. Vedoucí práce Tomáš Mauder.

Prohlašuji, že jsem diplomovou práci *Tvorba umělé neuronové sítě pro výpočet termodynamických veličin* vypracoval samostatně pod vedením Ing. Tomáše Maudera, Ph.D. s použitím materiálů uvedených v seznamu literatury.

V Brně dne ....

Martin Groman

Chtěl bych poděkovat panu Ing. Tomáši Mauderovi, Ph.D. za odborné vedení, konzultaci a cenné rady při psaní mé práce a za pomoc při konečných úpravách. Rovněž bych chtěl poděkovat panu Ing. Jiřímu Škorpíkovi, Ph.D. za odbornou konzultaci. V neposlední řadě, bych rád poděkoval své rodině, která mi byla oporou po celou dobu mých studií.

Martin Groman

Obsah

1 Úvod **3**

2 Umělé neuronové sítě **5**
2.1 Uvedení do problematiky . . . 5
2.2 Struktura neuronové sítě . . . 6
2.2.1 Model neuronu . . . 6
2.2.2 Hluboké dopředné neuronové sítě . . . 7
2.2.3 Univerzální aproximační věta . . . 8
2.3 Učení . . . 10
2.3.1 Overfitting a underfitting . . . 10
2.3.2 Předzpracování dat . . . 13
2.3.3 Optimalizace učení neuronových sítí . . . 14
2.3.4 Překážky při optimalizaci neuronových sítí . . . 23
2.4 Hyperparametry neuronových sítí . . . 27
2.4.1 Hyperparametry spojené s modelem . . . 27
2.4.2 Hyperparametry spojené s optimalizací . . . 31
2.4.3 Optimalizace hyperparametrů . . . 35

3 Termodynamika páry **37**
3.1 Základní popis par . . . 37
3.2 Výpočet termodynamických veličin páry . . . 38

4 Implementace **43**
4.1 Python, Tensorflow, XSteam . . . 43
4.2 Tvorba modelu . . . 44
4.2.1 Příprava dat . . . 44
4.2.2 Neuronová síť . . . 45

5 Výsledky **53**

6 Závěr **57**

Seznam použité literatury **59**

Seznam použitých zkratek a symbolů **63**

Seznam obrázků **65**

Seznam tabulek **67**

A Hessova matice **69**

B Koeficienty a exponenty fundamentálních rovnic **71**

1 Úvod

Fascinace lidským mozkem trvá už několik desítek let. Vědci i laici obdivují především jeho výpočetní schopnosti. Není proto divu, že se lidstvo snaží dovednosti mozku napodobit či simulovat počítačem. Vysněná představa sestrojit samostatně myslící stroj žene vývoj v tomto odvětví kupředu. Vědecký postup v neurovědě a matematice umožnil vznik prvních umělých neuronů a následně celých neuronových sítí. Potenciál neuronových sítí je obrovský a každoročně přibývá nespočet nově nalezených aplikací. Tato práce se bude věnovat jedné z možných aplikací, a to využití neuronové sítě pro výpočet termodynamických veličin páry.

V současné době existuje několik způsobů jak určit veličiny konkrétního stavu vodní páry. Ručně spočítat, najít hodnoty v tabulkách anebo využít specializovaného softwaru. Naším úkolem bude prozkoumat, zda se neuronová síť může vyrovnat existujicím softwarům ať už v přesnosti výpočtů nebo v době jejich provedení. Konkrétněji se v práci budeme zabývat výpočtem měrného objemu vodní páry pro zadaný tlak a teplotu.

Práce je strukturována do pěti kapitol. V následující, druhé, kapitole je rozebrána veškerá teorie potřebná k sestavení obecné neuronové sítě. Ve třetí kapitole jsou uvedeny již existující rovnice pro výpočty veličin páry odvozené Mezinárodní asociací vlastnosti vodní páry. Ve čtvrté kapitole bude sestavena neuronová síť a následně bude optimalizována dle popsané teorie. V poslední kapitole budou prezentovány výsledky.

Celkovým výstupem této diplomové práce bude spustitelná aplikace, která bude provádět výpočty pomocí neuronové sítě. Neuronová síť bude vytrénována na existujících datech ze specializovaných softwarů. Tato síť bude schopna po předložení tlaku a teploty vodní páry dopočítat její měrný objem.

2 Umělé neuronové sítě

2.1 Uvedení do problematiky

Ve své nejobecnější podobě, umělá neuronová síť neboli pouze neuronová síť, je stroj, který modeluje průběh, jakým lidský mozek provádí určitý úkol. Taková síť je nejčastěji zrealizována elektronickými součástkami anebo je simulována počítačovým softwarem. K dosažení dobrých výkonů, neuronové sítě využívají četných propojení mezi svými výpočetními buňkami, tzv. neurony. Neuronovou síť můžeme tedy definovat [1]:

Neuronová síť je paralelní procesor složený z jednoduchých výpočetních jednotek, které ukládají znalosti získané zkušenostmi a následně je zprostředkovávají k použití. Lidskému mozku se neuronová síť podobá ve 2 aspektech:

1. *Síť obdržuje znalosti ze svého prostředí skrze učení.*
2. *Síly propojení mezi neurony neboli synaptické váhy se používají k uložení nabytých znalostí.*

Proces, při němž dochází k učení, nazýváme učící algoritmus, v jehož průběhu se postupně upravují synaptické váhy k dosažení požadovaného cíle. Hlavní výpočetní silou neuronových sítí je tedy paralelní struktura, ale také schopnost zobecňování, neboli schopnost produkovat rozumné výstupy pro vstupy, se kterými se síť nesetkala v průběhu učení.

Podívejme se v rychlosti na neurologický aparát, kterým byly neuronové sítě inspirovány. Lidská nervová soustava se v podstatě skládá ze 3 základních stupňů. Hlavním komponentem je mozek, reprezentován neurální sítí, která nepřetržitě přijímá a zpracovává informace a následně provádí vhodná rozhodnutí. Dále receptory, které převádí podněty z prostředí na elektrické impulzy sloužící k dopravě informací do mozku. Efektory pak mění impulzy z mozku v příslušné reakce.

Rozdíly mezi neuronovou sítí umělou a lidskou jsou však značné. Elektronické operace dosahují rychlosti v řádech nanosekund oproti milisekundám u nervové soustavy. Nicméně tato skutečnost nestačí umělým neuronovým sítím, aby se těm lidským zcela vyrovnaly. Lidský mozek disponuje neskutečným množstvím neuronů a stejně tak obrovským počtem vzájemných propojení (synapsí) mezi nimi, které dohromady tvoří výpočetní sílu, jakou umělá neuronová síť nedokáže simulovat.

Předešlý odstavec už dává tušit problémy, se kterými se později v práci budeme zabývat. Je to například počet neuronů, jenž bude tvořit naši neuronovou síť. Počet a jejich uspořádání do vrstev, tzv. architektura sítě, definuje její vlastnosti a výpočetní možnosti. Dále jsou to vzájemná propojení, která jsou v umělém případě neuronové sítě v podobě synaptických vah. Hodnoty těchto vah, stejně jako jejich počet simulují sílu synapsí v mozku, důležitá je tedy volba vhodného učícího algoritmu, včetně jeho parametrů.

Výše uvedené vlastnosti neuronových sítí jsou součástí celku, který souhrnně nazýváme hyperparametry neuronové sítě, a právě nalezení optimálních hodnot těchto parametrů je klíčovým úkolem pro správné fungování sítě [1].

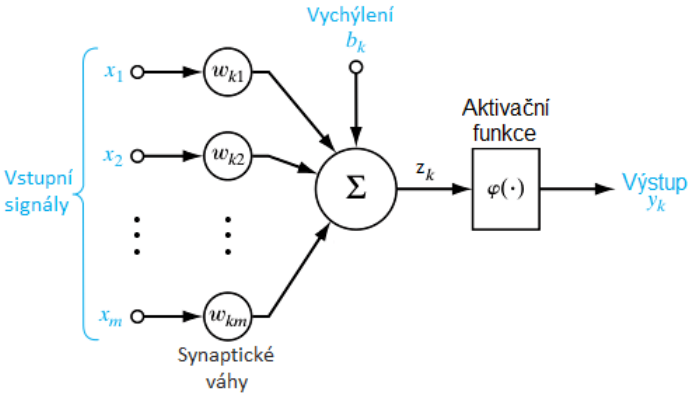
2.2 Struktura neuronové sítě

2.2.1 Model neuronu

Základní stavební jednotkou neuronové sítě je neuron. Matematický model neuronu je sice zjednodušením opravdového neuronu, přesto však vystihuje jeho funkci dostatečně dobře.

Než se dostaneme k samotnému modelu, věnujme se ještě dějům probíhajícím v opravdovém neuronu. Převod signálu mezi neurony je složitý chemický proces, proto se zaměříme pouze na jeho podstatu. Neuron vysílající signál uvolňuje specifické chemické látky, které mají za úkol zvýšit nebo snížit elektrický potenciál v přijímacím neuronu. Pokud tento potenciál dosáhne určitého prahu, dojde následně k převodu signálu mezi neurony [2].

Při sestavování matematického modelu je nezbytné tuto podstatu zachytit. Snažíme se převést soubor nějakých vstupů neboli signálu na odpovídající výstup. Dále musíme vystihnout vliv synaptických vah na sílu jednotlivých vstupů. V neposlední řadě pak musíme zvolit funkci, která bude realizovat zmíněné prahování.



Obrázek 1 – Model neuronu [1]

Samotný matematický model si lze představit jako na obrázku 1. Můžeme tedy k-tý neuron popsat rovnicemi

$$z_k = \sum_{j=1}^{m} w_{kj} x_j + b_k \tag{1}$$

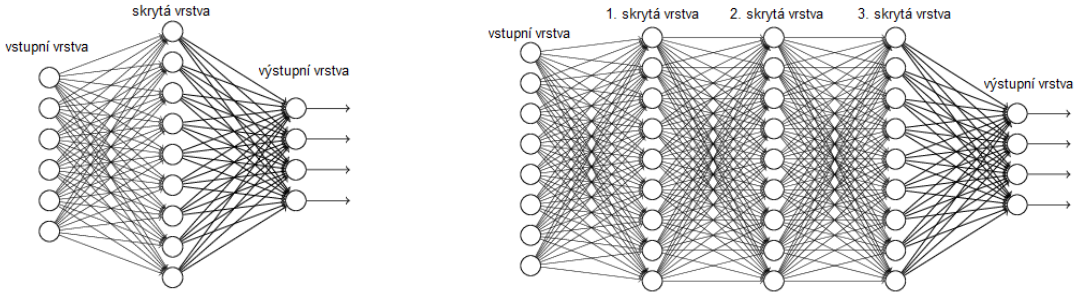
a

$$y_k = \varphi(z_k), \tag{2}$$

kde z_k nazveme aktivací k-tého neuronu, $x_1, x_2, ..., x_m$ jsou vstupní signály, $w_{k1}, w_{k2}, ..., w_{km}$ jsou synaptické váhy k-tého neuronu. A tedy, j-tý signál x_j přivedený do k-tého neuronu je násoben synaptickou váhou w_{kj} (první index vždy značí o jaký se jedná neuron a druhý se odkazuje na přivedený signál). Dále, b_k je tzv. bias, neboli vychýlení. Synaptické váhy a vychýlení budeme nazývat parametry neuronové sítě a značit vektorem $\boldsymbol{\theta}$. Funkce φ se nazývá aktivační funkce, jejíž účelem je omezení amplitudy výstupu daného neuronu. Výstupní signál y_k poté většinou leží v intervalu $\langle 0,1 \rangle$, popřípadě $\langle -1,1 \rangle$ [1].

2.2.2 Hluboké dopředné neuronové sítě

Hluboké dopředné neuronové sítě, také nazývány pouze dopředné neuronové sítě nebo vícevrstvé perceptrony, jsou samotnou podstatou tzv. deep learningu. Deep learning je termín užívaný pro algoritmy strojového učení pro architektury hlubokého typu. Nabízí se otázka, kdy je neuronová síť hluboká? Bohužel neexistuje přesná definice kolik vrstev daná síť musí mít, aby mohla být hluboká, ale obecně se považuje za hlubokou pokud má 2 a více skrytých vrstev. Ukázku mělké sítě můžeme vidět na obrázku 2a a hluboké na obrázku 2b.



(a) Mělká neuronová síť (b) Hluboká neuronová síť

Obrázek 2 – Porovnání mělké a hluboké neuronové sítě [3]

Cílem těchto sítí je aproximovat nějakou funkci f^*. Dopředné sítě definují zobrazení $\boldsymbol{y} = f(\boldsymbol{x}, \boldsymbol{\theta})$ a učí se hodnotu parametrů $\boldsymbol{\theta}$ vedoucí k nejlepší možné aproximaci funkce. Dopředné se nazývají proto, že informace proudí jen jedním směrem a to od vstupních dat $\boldsymbol{x}$ k výstupy $\boldsymbol{y}$ bez jakékoliv zpětné vazby, ve které by se výstupy modelu použily jako vstupy.

Dopředné sítě jsou reprezentovány složením několika funkcí. Mějme například 3 funkce $f^{(1)}, f^{(2)}$ a $f^{(3)}$ poskládané tak, že $f(\boldsymbol{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\boldsymbol{x})))$. V neuronových sítích jsou tyto řetězcové struktury nejčastěji používanými. V tomto případě, funkci $f^{(1)}$ nazveme první vrstva sítě, $f^{(2)}$ druhá vrstva, atd. Celkovou délku tohoto řetězce nazýváme hloubka neuronové sítě. Poslední vrstvu sítě nazýváme výstupní vrstva.

V průběhu učení je naší snahou dosáhnout shody $f(\boldsymbol{x})$ a $f^*(\boldsymbol{x})$. Trénovací data nám poskytnou přibližné příklady $f^*(\boldsymbol{x})$ vyhodnocené v různých trénovacích bodech. Ke každému příkladu $\boldsymbol{x}$ je přiložen jeho vzor $y \approx f^*(\boldsymbol{x})$. Trénovací příklady jasně specifikují, co se musí stát ve výstupní vrstvě v každém $\boldsymbol{x}$, a to dojít k hodnotě blízké k y. Učící algoritmus musí rozhodnout jak využít svých vrstev k dosáhnutí požadovaného výstupu. Jelikož trénovací data neukazují požadovaný výstup pro každou vrstvu, tak tyto vrstvy nazýváme skryté.

Poslední pojem, který bychom měli zavést, je šířka sítě. Neuronové sítě jsou inspirovány lidským mozkem. Každá vrstva tedy obsahuje několik neuronů, takovou vrstvu pak můžeme zapsat pomocí vektorového zápisu, kde vektory reprezentují jednotlivé vrstvy. Dimenze vektorů skrytých vrstev určuje šířku naší sítě. Důvod použití vektorové reprezentace je opět převzat z neurovědy. Volba funkcí $f^{(i)}(\boldsymbol{x})$ použitých k výpočtu těchto reprezantací je lehce inspirována vědeckým pozorováním chování biologických neuronů. Nicméně moderní sítě se už spíše řídí matematickým výzkumem a není obecnou snahou co nejlépe modelovat lidský mozek [4].

2.2.3 Univerzální aproximační věta

Připomeňme si nejdříve, jak zapisujeme jednotlivé neurony. Vektor parametrů $\boldsymbol{\theta}$ pro nás bude v drtivé většině případů představovat vektory vah $\boldsymbol{w}$ pro všechny vrstvy a odpovídající vektory vychýlení $\boldsymbol{b}$. Neurony ve skrytých vrstvách tedy obdrží vektor vstupů $\boldsymbol{x}$, proběhne v nich výpočet afinní transformace $\boldsymbol{z} = \boldsymbol{w}^\top \boldsymbol{x} + \boldsymbol{b}$ a pak po prvcích užijeme nelineární funkci $\varphi(\boldsymbol{z})$.

Jak již bylo zmíněno, neuronové sítě jsou organizovány do vrstev. Většina sítí pak tyto vrstvy řetězovitě propojuje, tak, že každá vrstva je funkcí vrstvy, která ji předcházela. V takové struktuře je první vrstva dána

$$\boldsymbol{y}^{(1)} = \varphi^{(1)}(\boldsymbol{w}^{(1)\top} \boldsymbol{x} + \boldsymbol{b}^{(1)}), \tag{3}$$

druhá vrstva je dána

$$\boldsymbol{y}^{(2)} = \varphi^{(2)}(\boldsymbol{w}^{(2)\top} \boldsymbol{h}^{(1)} + \boldsymbol{b}^{(2)}). \tag{4}$$

Zápis tedy můžeme zobecnit pro libovolnou k-tou vrstvu

$$\boldsymbol{y}^{(k)} = \varphi^{(k)}(\boldsymbol{w}^{(k)\top} \boldsymbol{h}^{(k-1)} + \boldsymbol{b}^{(k)}). \tag{5}$$

V těchto řetězitých architekturách jsou předmětem našeho zájmu především celková hloubka sítě a šířka jednotlivých vrstev. Hlubší sítě si obvykle vystačí s méně neurony ve vrstvách a s méně parametry než mělčí sítě, jsou však o to obtížněji optimalizovatelné.

Lineární model může dle definice reprezentovat pouze lineární funkce. Má výhodu ve snadném učení, protože spousta ztrátových funkcí aplikovaných na lineární modely vede na konvexní optimalizační problémy. Nicméně nás bude především zajímat učení se nelineárních funkcí.

Na první pohled by se mohlo zdát, že k učení se nelineárních funkcí budeme potřebovat speciální model pro konkrétní nelineární funkci. Naštěstí, dopředné neuronové sítě slouží jako univerzální aproximátor. Aproximační schopnosti sítě popisuje univerzální aproximační věta [5]:

Věta 2.1. *Nechť $\varphi : \mathbb{R} \to \mathbb{R}$ je nekonstantní, ohraničená a spojitá funkce. Nechť I_n je n-dimenzionální jednotková hyperkrychle $[0,1]^n$. Prostor reálných spojitých funkcí na I_n označíme $C(I_n)$. Pak pro libovolné $\epsilon > 0$ a libovolnou funkci $f \in C(I_n)$, existuje celé číslo N, reálné konstanty $v_i, b_i \in \mathbb{R}$ a reálné vektory $w_i \in \mathbb{R}^n$ pro $i = 1, ..., N$ takové, že můžeme definovat*

$$F(x) = \sum_{i=1}^{N} v_i \varphi(w_i^\top x + b_i), \tag{6}$$

jako aproximaci funkce f, tedy

$$|F(x) - f(x)| < \epsilon, \qquad \forall x \in I_n. \tag{7}$$

Neboli funkce tvaru $F(x)$ jsou husté v $C(I_n)$.

Důkaz je možno najít v [5].

Výše zmíněná věta nám v podstatě říká, že dopředná neuronová síť s alespoň jednou skrytou vrstvou s vhodnou aktivační funkcí φ dokáže aproximovat libovolnou funkci f.

Tato funkce musí být spojitá na nějaké uzavřené a ohraničené podmnožině $\mathbb{R}^n$. Aproximace proběhne s požadovanou nenulovou přesností za předpokladu, že neuronová síť má k dispozici dostatečný počet skrytých neuronů.

Univerzální aproximační věta tedy tvrdí, že nehledě na to, jakou funkci se snažíme naučit, víme, že dostatečně velká dopředná síť bude schopna reprezentovat tuto funkci. Nicméně nemáme zaručeno, že trénovací algoritmus se doopravdy danou funkci naučí. I za předpokladu, že neuronová síť dokáže reprezentovat danou funkci, učení může ztroskotat ze dvou důvodů.

Zaprvé, může se stát, že optimalizační algoritmus použitý pro trénování nebude schopný nalézt hodnoty parametrů odpovídající požadované funkci. A zadruhé, učící algoritmus může vybrat špatnou funkci kvůli overfittingu. Tedy dopředné sítě poskytují univerzální systém pro reprezentaci funkcí, ve smyslu, máme-li zadanou funkci, pak existuje dopředná síť, která tuto funkci aproximuje. Neexistuje však univerzální postup pro zkoumání trénovací sady příkladů a následně určení funkce zobecněné natolik, že bude platit i pro body, které nebyly zahrnuty v trénovací sadě dat.

Vraťme se opět k univerzální aproximační větě. V ní rovněž stojí, že s dostatečně velkou sítí s jednou skrytou vrstvou jsme schopni dosáhnout libovolné přesnosti. Síť s jednou skrytou vrstvou je tedy dostatečný předpoklad, avšak tato vrstva může narůst do enormní šířky a proces učení nemusí proběhnout vůbec úspěšně. V mnoha případech se tedy uchylujeme k přidání jedné nebo i více vrstev, neboť ty mohou zredukovat počet neuronů potřebných k aproximaci.

2.3 Učení

Učení neuronové sítě definujeme jako proces, při němž dochází ke změnám parametrů vlivem prostředí, ve kterém se síť nachází. Množinu předem stanovených pravidel pro řešení učícího problému nazveme učící algoritmus. Neexistuje však jeden unikátní algoritmus, který by vyhovoval každé síti. K dispozici máme různé algoritmy lišící se ve způsobu, jakým upravují synaptické váhy neuronové sítě.

Rozlišujeme dva druhy učení - s učitelem a bez učitele. Při prvně zmíněném učení, učitel má znalosti o prostředí, které jsou reprezentovány množinou vstupů a jim odpovídajícím výstupům. Nicméně, neuronová síť podobné znalosti nemá, avšak pokud jsou ji předloženy stejné vstupy, může porovnat svůj výstup s požadovaným výstupem, který ji poskytne učitel. Následně pak dojde k úpravě parametrů sítě.

Při učení bez učitele není k dispozici žádný dohled nad průběhem učení a síť samotná si musí vytvořit vlastní způsob klasifikace výstupů. Tento styl učení se v práci nadále rozebírat nebude.

Obecně existuje pět základních pravidel učení - učení na základě opravy chyby, na základě práce s pamětí, Hebbovo, kompetitivní a Boltzmannovo učení. Ve této práci budeme využívat první zmíněné pravidlo a proto se nadále budeme věnovat pouze tomu [1].

Umělé neurony učíme pomocí tzv. *opravy chyb* - spočítáme rozdíl mezi skutečným - požadovaným výstupem a výstupem našeho neuronu. Tento rozdíl nazýváme *chyba učení.* Pokud tedy výstup našeho neuronu je neuspokojivý, dojde ke změně jeho parametrů. Změna by měla být taková, abychom příště dosáhli lepšího výstupu při předložení stejných vstupů. Vektor parametrů k-tého neuronu tedy upravíme následovně

$$\boldsymbol{\theta}_k^{nové} = \boldsymbol{\theta}_k^{staré} + \Delta\boldsymbol{\theta}_k, \tag{8}$$

kde $\boldsymbol{\theta}^{nové}$ je nově obdržený vektor parametrů, $\boldsymbol{\theta}^{staré}$ původní vektor parametrů a $\Delta\boldsymbol{\theta}$ je vektor odhadnuté změny k lepšímu výstupu [6].

2.3.1 Overfitting a underfitting

Hlavní výzvou pro neuronové sítě, stejně jako pro ostatní algoritmy strojového učení, je schopnost zobecňování, neboli předvádění stejně dobrého výkonu na datech do té doby neviděných.

Obvykle máme k dispozici tréninkovou sadu dat, ze které obdržíme při učení tréninkovou chybu, kterou se snažíme minimalizovat. Abychom mohli posoudit schopnost zobecňování naší sítě, potřebujeme ještě testovací sadu dat a z ní příslušnou testovací chybu.

Nejdříve neuronové síti předložíme tréninkové data a vhodně upravíme její parametry, abychom dostatečně snížili tréninkovou chybu. Poté síti předložíme testovací data a spočítáme testovací chybu, která bude ve většině případů větší nebo rovna té tréninkové [4].

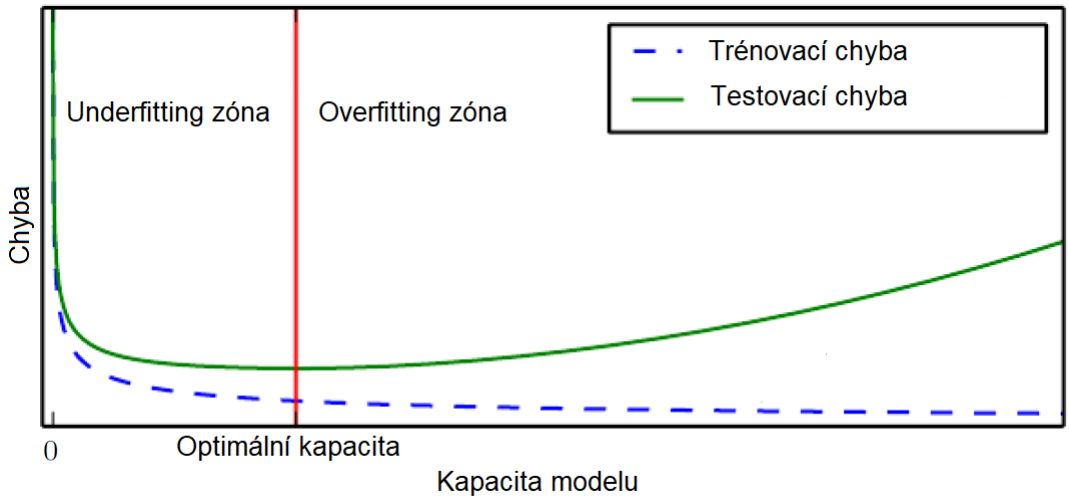
Následně se snažíme vyřešit dva problémy najednou, a to:

1. Zmenšit trénovací chybu.
2. Zmenšit rozdíl mezi trénovací a testovací chybou.

S těmito problémy souvisí pojmy underfitting a overfitting. Underfitting nastává, pokud náš model není schopen obdržet dostatečně malou trénovací chybu. Naopak overfitting znamená, že rozdíl mezi trénovací a testovací chybou je příliš velký.

K tomu, abychom se vyhnuli jednomu či druhému, nám slouží kapacita modelu. Modely s nízkou kapacitou nebudou schopny adekvátně popsat trénovací data a modely s vysokou kapacitou se mohou zaměřit na příliš konkrétní znaky trénovacích dat a stát se zbytečně komplexní.

I když jednodušší funkce pravděpodobněji zobecní daný problém lépe než komplexnější funkce (zmenší rozdíl mezi trénovací a testovací chybou), stále musíme vybrat dostatečně složitou funkci, abychom dosáhli malé trénovací chyby. Obecně platí, že trénovací chyba se zmenšuje, a následně se asymptoticky blíží k minimální možné chybě, zatímco kapacita modelu se zvětšuje. Testovací chyba je funkcí kapacity modelu a je konvexní křivkou. Vše je zachyceno na obrázku 3 [4].



Obrázek 3 – Srovnání trénovací a testovací chyby [4]

K porozumnění pojmů overfittingu a underfittingu věnujme ještě následující odstavce. Jelikož overfitting je daleko častější a závažnější problém, zaměřme se především na něj. Pro lepší vysvětlení overfittingu si uvedeme jednu zajímavost z historie USA:

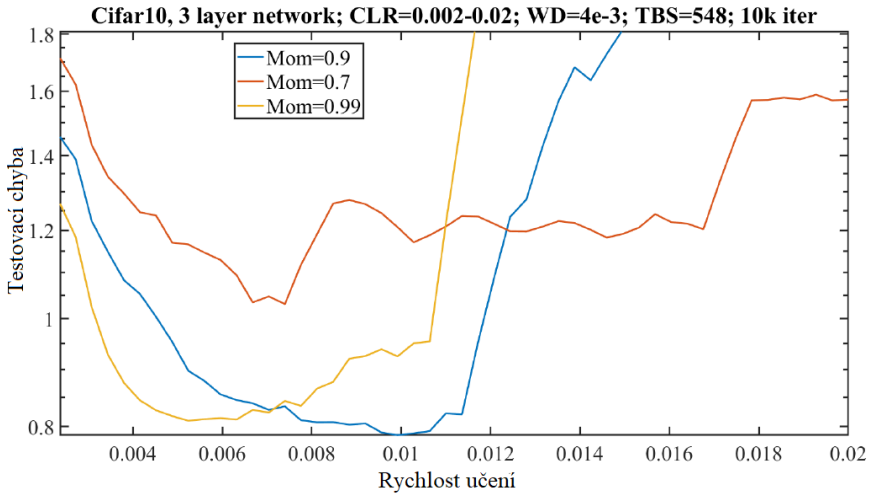
"Do roku 1996 nebyl žádný úřadující americký demokratický prezidentský kandidát, disponující nulovými válečnými zkušenostmi, poražen někým, jehož první jméno má větší hodnotu ve Scrabblu."(Dokud Bill neporazil Boba.)

Je zřejmé, že tento fakt je zcela irelevantní, co se týče prezidentské volby, avšak krásně ilustruje, jak je důležité vybrat ty správné faktory pro jakoukoliv předpověď. Rovněž ukazuje, že predikovat pomocí náhodných znaků není nejlepší nápad, neboť v tomto případě jde o prostou náhodu. V tomto leží podstata overfittingu, děláme předpovědi, které dokonale vyhovují našim současným datům, avšak nejsou dostatečně zobecněné pro širší datové sety.

Overfitting je tedy snaha zachytit šum (informace bez opravdového významu) v našich datech a donutit model, aby vyhovoval těmto malým odchylkám [7].

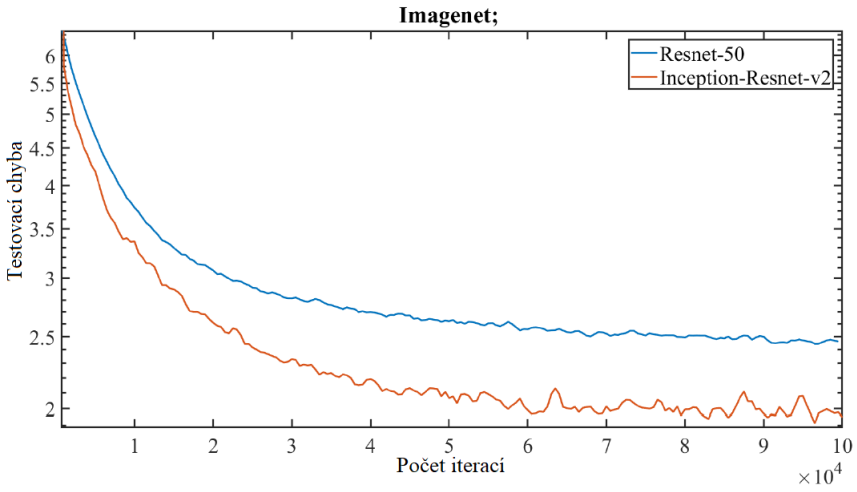
Ukažme si oba příklady na reálných datech. Správné podchycení overfittingu a underfittingu může vést k rychlejšímu určení optimálních hodnot hyperparametrů, o kterých bude řeč později.

Podívejme se nejdříve na obrázek 4. Prozatím se nezabývejme jednotlivými zkratkami a jejich významy, zajímá nás, že se jedná o graf testovací chyby v závislosti na nějakém hyperparametru sítě. Žlutá křivka vykazuje jasné známky overfittingu. Testovací chyba klesá jen pro malý rozsah daného parametru a následně stále roste. Ačkoliv modrá křivka má podobný průběh, nejedná se v tomto případě o overfitting. Ostrý nárust testovací chyby zde přičítáme nestabilitám při učení v důsledku vyšších hodnot daného parametru, který, jak se dozvíme později, značně ovlivňuje učení [8].



Obrázek 4 – Ukázka overfittingu [8]

Nyní přesuňme pozornost k obrázku 5. Underfitting lze pozorovat u obou vykreslených křivek. Projevuje se stálým poklesem testovací chyby. Ideálně chceme dospět do nějaké horizontální oblasti, ve které se testovací chyba už nebude nadále zmenšovat. Vidíme, že chyba se stále zmenšuje i po 100 000 iteracích, i když červená křivka se pomalu ustaluje a jen osciluje zhruba kolem hodnoty 2. Rozhodně můžeme konstatovat, že síť popsaná červenou křivkou vykazuje slabší známky underfittingu [8].



Obrázek 5 – Ukázka underfittingu [8]

2.3.2 Předzpracování dat

Existuje několik způsobů, jak učení urychlit. Následuje stručný přehled často používaných metod pro zefektivnění celého procesu.

Zamíchání příkladů

Síť se nejrychleji naučí z nejméně očekávaných vstupů. Doporučuje se vybírat v každé iteraci příklad, který je nejvíce odlišný od těch ostatních. Jak poznáme, zda vybraný příklad poskytl více nebo méně informací pro naši síť? Jednou z možností je sledování trénovací chyby. Pokud je relativně velká, potom víme, že jsme síti předložili příklad s větším množstvím informací než v případě, že by chyba byla malá. Je tedy logické vybírat podobné příklady často, abychom naši síť naučili co nejrůznorodější informace.

Ve většině případů se snažíme síti prezentovat příklady dosud neviděné, které mají největší potenciál pro další učení. Avšak můžeme taky narazit na problém. Pokud naše data obsahují tzv. *outliery* (hodnoty, které se výrazně liší od zbytku) neměli bychom se podobným přístupem řídit, neboť by se naše síť nenaučila ty podstatné informace [9].

Standardizace vstupů

Konvergence procesu učení je obvykle rychlejší, pokud průměr v trénovací sadě dat každé vstupní proměnné je blízký nule. Obecně každý posun průměru vstupů od nuly zapříčiní update synaptických vah v určitém směru, a tím pádem zpomalí učení. Uvažujme například případ, kdy všechny vstupy jsou kladné. Pokud jsou všechny složky vstupního vektoru kladné, potom složky updatovaného vektoru vah (viz rovnice (8)) budou všechny stejného znaménka v daném uzlu. Ve výsledku tyto složky mohou zároveň buď všechny růst nebo všechny klesat, což je velmi neefektivní.

Vyplatí se tedy posunout vstupy tak, aby jejich průměr byl blízký nule. Tuto heuristiku bychom měli aplikovat v každé vrstvě, protože každá vrstva slouží jako vstupní vrstva pro tu následující. S tímto problémem souvisí vhodný výběr aktivační funkce, což bude rozebráno později.

Dvě nejčastěji používané normalizační techniky jsou:

- Standardizované z-skóre
$$Z = \frac{X - \overline{X}}{\sigma(X)}, \tag{9}$$
- Min-max normalizace
$$Z = \frac{X - \min(X)}{\max(X) - \min(X)}, \tag{10}$$
kde X jsou hodnoty vstupní proměnné, $\overline{X}$ je jejich průměr, $\sigma(X)$ směrodatná odchylka a $min(X), max(X)$ jsou po řadě minimální a maximální prvek dané proměnné [10].

Kovariance

Za zmínku rovněž stojí význam kovariance. Konvergenci učení můžeme urychlit, pokud naše vstupy transformujeme tak, aby měly zhruba stejnou kovarianci C_i, kde

$$C_i = \frac{1}{n}\sum_{p=1}^{n}(x_i^{(p)})^2. \tag{11}$$

Zde n značí počet trénovacích příkladů, C_i je kovariance i-té vstupní proměnné a x_i^p je i-tá složka p-tého trénovacího příkladu. Tato transformace urychluje učení, protože vyrovnává rychlost napojení jednotlivých vah na vstupní uzly [9].

2.3.3 Optimalizace učení neuronových sítí

Spousta algoritmů strojového učení zahrnuje do jisté míry i optimalizaci. Snažíme se minimalizovat nebo maximalizovat nějakou vektorovou funkci $f(\boldsymbol{x})$, různými změnami $\boldsymbol{x}$. Funkci, kterou minimalizujeme nebo maximalizujeme, budeme nazývat chybovou funkcí nebo také ztrátovou funkcí. Následující odstavce věnujme učícím algoritmům, jejichž úkolem je zmíněnou optimalizaci provést.

Gradientní sestup
Začneme s metodou gradientního sestupu. Mějmě funkci $y = f(x)$, $x, y \in \mathbb{R}$. Derivaci této funkce značíme $f'(x) = \frac{dy}{dx}$. Říká nám, jakou změnu musíme udělat ve vstupu, abychom obdrželi odpovídají změnu na výstupu- $f(x + \eta) \approx f(x) + \eta f'(x)$.

Derivaci můžeme využít k minimalizaci funkce, protože nám říká, jak musíme změnit x abychom dosáhli menšího y. Například, víme, že $f(x - \eta \operatorname{sgn}(f'(x)))$ je menší než $f(x)$ pro dostatečně malé η. Můžeme tedy postupně zmenšovat $f(x)$, tím, že budeme po malých krocích posouvat x ve směru opačného znaménka derivace. Tuto metodu nazýváme gradientní sestup.

Pro naše účely je nejdůležitější minimalizace funkcí s více vstupy: $f : \mathbb{R}^n \to \mathbb{R}$. K tomuto úkolu využijeme parciální derivace. Parciální derivace $\frac{\partial}{\partial x_i} f(\boldsymbol{x})$ vyjadřuje, jak se f mění v závislosti na změně pouze proměnné x_i v daném bodě $\boldsymbol{x}$. Gradient funkce f je vektor obsahující všechny její parciální derivace, budeme jej značit $\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$. Prvek i gradientu je tedy parciální derivace f podle x_i.

Směrová derivace ve směru $\boldsymbol{u}$ (jednotkový vektor) je derivace funkce $f(\boldsymbol{x}+\alpha\boldsymbol{u})$ podle α pro $\alpha = 0$. Tedy $\frac{\partial}{\partial \alpha} f(\boldsymbol{x} + \alpha\boldsymbol{u}) = \boldsymbol{u}^\top \nabla_{\boldsymbol{x}} f(\boldsymbol{x})$, pro $\alpha = 0$. K minimalizaci funkce f,bychom rádi našli směr, ve kterém se f zmenšuje nejrychleji. K tomuto můžeme využít směrovou derivaci:

$$\min_{\boldsymbol{u}, \boldsymbol{u}^\top \boldsymbol{u}=1} \boldsymbol{u}^\top \nabla_{\boldsymbol{x}} f(x) = \min_{\boldsymbol{u}, \boldsymbol{u}^\top \boldsymbol{u}=1} ||\boldsymbol{u}||_2 ||\nabla_{\boldsymbol{x}} f(\boldsymbol{x})||_2 \cos\beta, \tag{12}$$

kde β je úhel mezi $\boldsymbol{u}$ a gradientem. Pokud dosadíme za $||\boldsymbol{u}||_2 = 1$ a vynecháme členy, které nezávisí na $\boldsymbol{u}$, tak se nám tento problém zjednoduší na

$$\min_{\boldsymbol{u}} \cos\beta, \tag{13}$$

což je minimalizováno pro vektor $\boldsymbol{u}$ směřující opačným směrem než gradient. Funkci f můžeme zmenšovat pohybem ve směru záporně vzatého gradientu. Tato metoda je známá taky jako metoda největšího spádu.

Metoda největšího spádu nám tedy nabídne nový bod $\boldsymbol{x}'$:

$$\boldsymbol{x}' = \boldsymbol{x} - \eta \nabla_{\boldsymbol{x}} f(\boldsymbol{x}), \tag{14}$$

kde η nazýváme rychlost učení, což je kladný skalár určující velikost kroku. Nejjednodušší přístup je zvolit η jako malou konstantu. Jednou z možností je vyhodnotit $f(\boldsymbol{x}-\eta\nabla_{\boldsymbol{x}} f(\boldsymbol{x}))$ pro několik různých η a vybrat takové, které vyústí v nejmenší hodnotu chybové funkce. Tuto metodu nazýváme line search [4].

Stochastický gradientní sestup

Stochastický gradientní sestup (stochastic gradient descent - SGD) je jedním z nejrozšířenějších algoritmů pro učení neuronových sítí. Jedná se o rozšíření metody popsané dříve.

Nejčastějším problémem strojového učení je nezbytnost velkých trénovacích datových sad, abychom dosáhli dobrého zobecnění. Nicméně zpracování takových datových sad je výpočetně náročnější.

Chybová funkce používaná v algoritmech strojového učení se dá rozepsat jako suma přes všechny trénovací příklady dané ztrátové funkce pro konkrétní příklad, tedy

$$E(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} L(\boldsymbol{y}^{(i)}, \boldsymbol{x}^{(i)}, \boldsymbol{\theta}), \tag{15}$$

kde funkce L představuje námi zvolenou ztrátovou funkci pro konkretní příklad. Pro takovou chybovou funkci je třeba spočítat gradient

$$\nabla_{\boldsymbol{\theta}} E(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{y}^{(i)}, \boldsymbol{x}^{(i)}, \boldsymbol{\theta}) \tag{16}$$

Uveďme si příklad nejčastěji používané ztrátové funkce, na které budeme ilustrovat náš postup. Pokud výstup naší sítě označíme $\hat{\boldsymbol{y}}^{(i)}$, tak ztrátovou funkci pro konkrétní i-tý příklad zapíšeme:

$$L(\boldsymbol{y}^{(i)}, \hat{\boldsymbol{y}}^{(i)}) = \frac{1}{2}(\boldsymbol{y}^{(i)} - \hat{\boldsymbol{y}}^{(i)})^2. \tag{17}$$

Pak stačí jen spočítat gradient chybové funkce

$$\nabla_{\boldsymbol{\theta}} E(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n} \nabla_{\boldsymbol{\theta}} (\boldsymbol{y}^{(i)} - \hat{\boldsymbol{y}}^{(i)})^2. \tag{18}$$

Pokusíme se lehce nástínit tento výpočet. Vektor parametrů $\boldsymbol{\theta}$ se skládá z vah $\boldsymbol{w}$ a z vychýlení $\boldsymbol{b}$. Naším úkolem je spočítat parciální derivace $\frac{\partial E}{\partial \boldsymbol{w}}$ a $\frac{\partial E}{\partial \boldsymbol{b}}$. Konkrétněji nás zajímá, jak se změní hodnota chybové funkce při změně váhy w^l_{jk} spojující neuron j v l-té vrstvě a neuron k ve vrstvě $l-1$. To samé platí pro vychýlení b^l_j neuronu j v l-té vrstvě. Tedy zajímá nás $\frac{\partial E}{\partial w^l_{jk}}$ a $\frac{\partial E}{\partial b^l_j}$ [4].

Výpočet gradientů chybové funkce se nazývá backpropagation. Síť upravuje své parametry na základě zpětně doručené chyby z výsledků. Algoritmus je postaven na čtyřech základních rovnicích [3]:

- Rovnice pro výpočet chyby ve výstupní vrstvě $\boldsymbol{\delta}^M$

$$\delta^M_j = \frac{\partial E}{\partial y^M_j}\varphi'(z^M_j), \tag{19}$$

  kde y^M_j je výstup j-tého neuronu ve výstupní vrstvě M, $z^M_j = \sum_k w^M_{jk} y^{M-1}_k + b^M_j$. A konkrétně pro naši ztrátovou funkci (vynecháme horní index značení příkladu pro přehlednost):

$$\delta^M_j = (y^M_j - y_j)\varphi'(z^M_j), \tag{20}$$

  kde zřejmě $\hat{\boldsymbol{y}} = \boldsymbol{y}^M$ jsme zavedli kvůli konzistenci následného značení.

Rovnici (20) můžeme přepsat maticově

$$\boldsymbol{\delta}^M = (\boldsymbol{y}^M - \boldsymbol{y}) \odot \varphi'(\boldsymbol{z}^M), \tag{21}$$

kde $\odot$ značí maticové násobení po složkách, tento zápis budeme užívat v následujících rovnicích.

- Rovnice pro chybu $\boldsymbol{\delta}^l$ vyjádřená pomocí chyby následující vrstvy $\boldsymbol{\delta}^{(l+1)}$

$$\boldsymbol{\delta}^l = ((\boldsymbol{w}^{l+1})^T \boldsymbol{\delta}^{l+1}) \odot \varphi'(\boldsymbol{z}^l), \tag{22}$$

kde $\boldsymbol{w}^{l+1}$ je matice vah pro $l+1$-ní vrstvu. Pomocí této a předchozí rovnice jsme schopni vyjádřit chybu $\boldsymbol{\delta}^l$ pro libovolnou vrstvu l sítě.

- Rovnice pro změnu chybové funkce podle vychýlení, nejdříve zápis po složkách

$$\frac{\partial E}{\partial b_j^l} = \delta_j^l, \tag{23}$$

kde se opět vyskytuje chyba δ_j^l j-tého neuronu pro l-tou vrstvu. Maticový zápis pak vypadá takto

$$\frac{\partial E}{\partial \boldsymbol{b}^l} = \boldsymbol{\delta}^l \tag{24}$$

- Rovnice pro změnu chybové funkce v závislosti na libovolné váze

$$\frac{\partial E}{\partial w_{jk}^l} = y_k^{l-1} \delta_j^l, \tag{25}$$

což můžeme přehledněji zapsat

$$\frac{\partial E}{\partial \boldsymbol{w}^{\boldsymbol{l}}} = \boldsymbol{y}^{\boldsymbol{l-1}} \boldsymbol{\delta}^{\boldsymbol{l}} \tag{26}$$

Díky výše zmíněným rovnicím můžeme podrobněji prozkoumat proces učení. Všimněme si třeba, že pokud v rovnici (26) výstup z předchozí vrstvy $\boldsymbol{y}^{l-1}$ je velmi malý, $\boldsymbol{y}^{l-1} \approx 0$, potom také gradient $\frac{\partial E}{\partial \boldsymbol{w}^{\boldsymbol{l}}}$ bude malý. V takovém případě hovoříme o pomalém učení, neboť nedochází ke větším změnám v hodnotách vah.

Podobným způsobem můžeme prozkoumat i další rovnice. Podívejme se například na rovnici (21), konkrétněji na výraz $\varphi'(\boldsymbol{z}^M)$. Pro lepší ilustraci si vybereme konkrétní aktivační funkci a to sigmoid. Tato funkce je téměř konstantní pro $\varphi(\boldsymbol{z}^M) \approx 0$ nebo $\varphi(\boldsymbol{z}^M) \approx 1$. Pokud tak nastane, obdržíme $\varphi'(\boldsymbol{z}^M) \approx 0$. A tedy váhy ve výstupní vrstvě se budou učit velmi pomalu, neboť došlo k tzv. saturaci výstupních neuronů. Podobné závěry platí i pro vychýlení výstupních neuronů.

Stejně tak můžeme odvodit, co se děje v předchozích vrstvách. Zaměřme se na výraz $\varphi'(\boldsymbol{z}^l)$ v rovnici (22). Pokud se neurony v l-té vrstvě blíží saturaci, pak $\boldsymbol{\delta}^l$ bude nabývat malých hodnot a tím pádem veškeré příchozí váhy do saturovaného neuronu se budou učit pomalu.

Zde vidíme, že saturace neuronů představuje problém pro učení. Snažíme se tedy vybrat takovou aktivační funkci, jejíž derivace je vždy kladná a není blízká nule. Volbou podobné aktivační funkce se vyhneme problémům saturace. Konkrétními aktivačními funkcemi se budeme zabývat v nadcházejících kapitolách [3].

Vraťme se nyní na úplný začátek této sekce. Ukázali jsme si, jak vypočítat gradient chybové funkce $\nabla_{\boldsymbol{\theta}} E(\boldsymbol{\theta})$, což potřebujeme pro algoritmus gradientního sestupu a následně stochastického gradientního sestupu.

Gradientní sestup pracuje na základě úpravy svých parametrů po projití všech trénovacích příkladů, index t značí časový krok, tedy

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \frac{1}{n} \sum_{i=1}^{n} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{y}^{(i)}, \boldsymbol{x}^{(i)}, \boldsymbol{\theta}) \tag{27}$$

Celkový počet operací je $\mathcal{O}(n)$. Pokud trénovací sada dat nabývá přes miliony příkladů, tak čas na výpočet jednoho gradientního kroku se stává poněkud zdlouhavý.

Proto většinou používáme stochastický gradientní sestup. Tento algoritmus v každém časovém kroku vybere náhodně jeden příklad, který využije k úpravě svých parametrů

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla_{\boldsymbol{\theta}} L(\boldsymbol{y}^{(p)}, \boldsymbol{x}^{(p)}, \boldsymbol{\theta}), \tag{28}$$

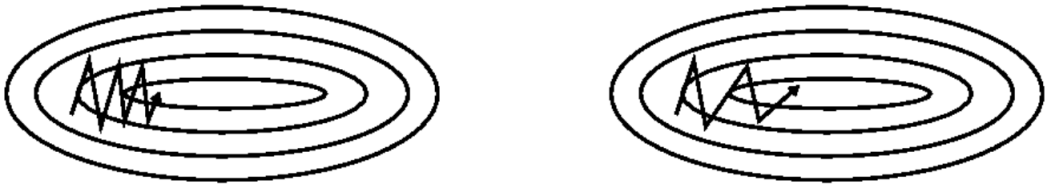
kde p značí náhodně vybraný příklad v každé iteraci.

Obecný gradientní sestup se obecně považuje za pomalý a nespolehlivý algoritmus. Nedokáže ani zaručit, že skončíme v lokálním minimu za rozumnou dobu, nicméně často dokáže najít dostatečně malou hodnotu chybové funkce pro to, aby byl užitečný.

Stochastický gradientní sestup má spoustu důležitých využití i mimo učení neuronových sítí. Je hlavním způsobem pro trénování velkých lineárních modelů na obrovských datových sadách. Pro model pevné velikosti, počet operací jednoho updatu SGD nezávisí na počtu trénovacích příkladů n. Počet iterací potřebných ke konvergenci algoritmu se obvykle zvyšuje s rostoucí sadou tréninkových dat. Nicméně pro n blížící se nekonečnu algoritmus bude konvergovat k nejlepší možné testovací chybě ještě předtím, než mu budou předloženy všechny příklady z trénovací sady. A tedy další zvyšování n nepovede ke zvyšení trénovací doby potřebné k dosáhnutí nejlepší možné testovací chyby [4].

Moment

SGD má problém s konvergencí v oblastech, kde plocha chybové funkce je daleko víc příkrá v jedné dimenzi než v ostatních, tedy v oblastech kolem lokálního optima. V takových situacích SGD osciluje napříč těmito oblastmi a jen velmi pomalu se posunuje k optimu, viz následující obrázek.



(a) SGD bez momentu (b) SGD s momentem

Obrázek 6 – Porovnání SGD s momentem a bez momentu [11]

Užitím momentu urychlíme algoritmus ve správném směru a oslabíme oscilace, jak lze vidět na obrázku výše. Zavedeme nový parametr - rychlost[1] a označíme ji $\boldsymbol{v}$. Pro algoritmus s momentem tedy můžeme zapsat následující update pravidla

$$\boldsymbol{v}_{t+1} = \alpha \boldsymbol{v}_t - \eta \nabla_{\boldsymbol{\theta}} E(\boldsymbol{\theta}) \tag{29}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \boldsymbol{v}_{t+1}, \tag{30}$$

kde $\alpha \in [0, 1)$, vyjadřuje jak rychle vymizí příspěvky předchozích gradientů. Obvykle volíme hodnotu kolem 0,9 [4].

Zjednodušeně si tuto metodu můžeme představit jako míč, jenž se kutálí z kopce. Míč nabírá moment setrvačnosti a kutálí se stále rychleji a rychleji. To samé lze pozorovat u první rovnice. Moment se zvětšuje pro dimenze, ve kterých gradienty směřují ve stejném směru a zmenšuje se pro dimenze, ve kterých gradienty mění směr. Ve výsledku obdržíme rychlejší konvergenci a nižší oscilace [11].

Nesterův moment

Předchozí algoritmus jsme přirovnali ke kutálejícímu se míči, avšak tento míč se jen slepě kutálí z kopce a netuší, co je před ním. Chceme tedy ,,chytřejší"míč, takový, který bude vědět, kdy zpomalit, protože po kopci dolů může následovat kopec nahoru.

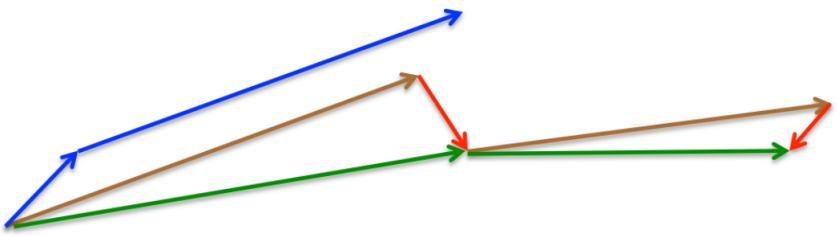
Podobnou myšlenku zachycuje algoritmus Nesterova zrychleného gradientu (NAG - Nesterov accelerated gradient). Opět využijeme momentového výrazu $\alpha \boldsymbol{v}_t$ z předchozího algoritmu, díky kterému pohybujeme s parametry $\boldsymbol{\theta}$. NAG však navíc pracuje s výrazem $\boldsymbol{\theta} - \alpha \boldsymbol{v}_t$, který nám udá příští přibližnou polohu parametrů $\boldsymbol{\theta}$.

Nyní nám stačí spočítat gradient chybové funkce, ne vzhledem k současné poloze parametrů, ale vzhledem k přibližné budoucí poloze, tedy

$$\boldsymbol{v}_{t+1} = \alpha \boldsymbol{v}_t - \eta \nabla_{\boldsymbol{\theta}} E(\boldsymbol{\theta} - \alpha \boldsymbol{v}_t), \tag{31}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \boldsymbol{v}_{t+1}. \tag{32}$$

Rozdíl mezi oběma momentovými algoritmy vidíme na obrázku 7. Obyčejný momentový algoritmus nejdříve spočítá současný gradient (malá modrá šipka) a pak provede velký skok ve směru nového gradientu (velká modrá šipka). Nesterův algoritmus nejdřív provede velký skok ve směru předchozího gradientu (velká hnědá šipka), změří nový gradient a provede mírnou korekci (zelená šipka). Tento algoritmus nám zabrání postupovat příliš rychle a také vede k lepší schopnosti reagovat na další vstupy [11].



Obrázek 7 – Porovnání momentových algoritmů [11]

[1]Můžeme jej rovněž označit jako moment, neboť moment počítáme jako hmotnost násobenou rychlostí a v tomtu algoritmu předpokládáme jednotkovou hmotnost [4].

AdaGrad
Tento algoritmus narozdíl od předchozího nepracuje s fixně danou rychlosti učení pro všechny parametry, ale průběžně ji mění, a to v závislosti na daném parametru θ_i a také v závislosti na iteračním kroku t. Pro přehlednost položme $g_{t,i}$ rovno gradientu chybové funkce E vzhledem k parametru θ_i v iteraci t:

$$g_{t,i} = \nabla_{\theta_t} E(\theta_{t,i}). \tag{33}$$

Update pravidla pro jednotlivé parametry θ_i v dané iteraci t pak můžeme zapsat jako

$$\theta_{t+1,i} = \theta_{t,i} - \eta g_{t,i}. \tag{34}$$

V tomto update pravidle, algoritmus AdaGrad upravuje obecnou rychlost učení v každém iteračním kroku t pro každý parametr θ_i na základě na minulých gradientů, které byly spočítany pro θ_i

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}}\, g_{t,i}, \tag{35}$$

kde $\boldsymbol{G}_t \in \mathbb{R}^{d\times d}$ je diagonální matice, jejíž prvky (i, i) jsou rovny součtu gradientů vzhledem k θ_i umocněných na druhou až do iterace t. Konstanta ϵ slouží jako vyhlazovací prvek, který zabraňuje dělení nulou (volíme například 10^{-8}).

Předchozí rovnici tedy můžeme přepsat pomocí vektorového zápisu

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\eta}{\sqrt{\boldsymbol{G}_t + \epsilon}} \odot \boldsymbol{g}_t, \tag{36}$$

kde $\odot$ značí násobení po složkách.

Hlavní výhodou AdaGradu je, že nemusíme ručně ladit rychlost učení, nejčastěji zvolíme výchozí hodnotu 0,01 a tu ponecháme.

Naopak nevýhodou je akumulace umocněných gradientů ve jmenovateli. Každý prvek, který přičteme je kladný, a tudíž součet v průběhu tréninku stále roste. To vyústí k infinitizemálnímu zmenšení rychlosti učení a tedy neschopnosti algoritmu získat nové znalosti [11].

AdaDelta
Tento algoritmus se snaží eliminovat hlavní nedostatek AdaGradu, a to monotónně klesající rychlost učení. Místo akumulace všech předchozích gradientů umocněných na druhou, AdaDelta se omezuje pouze na pevně dané rozmezí r naakumulovaných předchozích gradientů.

Namísto neefektivního ukládání r předchozích umocněných gradientů, je suma gradientu rekurzivně definována jako rozkládající se průměr všech minulých gradientů umocněných na druhou. Klouzavý průměr $p_{MA}[\boldsymbol{g}^2]_t$ v časovém kroku t pak pouze závisí na předchozím průměru a současném gradientu.

$$p_{MA}[\boldsymbol{g}^2]_t = \alpha p_{MA}[\boldsymbol{g}^2]_{t-1} + (1-\alpha)\boldsymbol{g}_t^2, \tag{37}$$

kde α je, podobně jako u momentové metody, konstanta rozkladu, nejčastěji volíme kolem 0,9.

Pro přehlednost ještě rozepíšeme naše update pravidla na dvě rovnice

$$\Delta\boldsymbol{\theta}_t = -\eta \boldsymbol{g}_t, \tag{38}$$
$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \Delta\boldsymbol{\theta}_t. \tag{39}$$

Následně se věnujme úpravě rovnice (38). Tato rovnice pro algoritmus AdaGrad vypadá následovně

$$\Delta\boldsymbol{\theta}_t = -\frac{\eta}{\sqrt{\boldsymbol{G}_t + \epsilon}} \odot \boldsymbol{g}_t. \tag{40}$$

Nyní stačí pouze nahradit diagonální matici $\boldsymbol{G}_t$ rozkládajícím se průměrem přes minulé gradienty $p_{MA}[\boldsymbol{g}^2]_t$

$$\Delta\boldsymbol{\theta}_t = -\frac{\eta}{\sqrt{p_{MA}[\boldsymbol{g}^2]_t + \epsilon}} \boldsymbol{g}_t. \tag{41}$$

Zde však vidíme, že jmenovatel je odmocnina střední kvadratické odchylky (RMS) gradientu, a proto

$$\Delta\boldsymbol{\theta}_t = -\frac{\eta}{RMS[\boldsymbol{g}]_t} \boldsymbol{g}_t. \tag{42}$$

Nicméně v tomto update pravidle si jednotky neodpovídají. Abychom to ukázali, musíme nejdříve definovat další exponenciálně se rozkládající průměr, tentokrát však přes jednotlivé updaty parametrů

$$p_{MA}[\Delta\boldsymbol{\theta}^2]_t = \alpha p_{MA}[\Delta\boldsymbol{\theta}^2]_{t-1} + (1-\alpha)\Delta\boldsymbol{\theta}_t^2. \tag{43}$$

Poté odmocnina střední kvadratické odchylka má tvar

$$RMS[\Delta\boldsymbol{\theta}]_t = \sqrt{p_{MA}[\Delta\boldsymbol{\theta}^2]_t + \epsilon}. \tag{44}$$

Jelikož $RMS[\Delta\boldsymbol{\theta}]_t$ neznáme, aproximujeme tuto hodnotu odmocninou kvadratické odchylky všech updatů parametrů do předchozího časového kroku. Konečně, nahrazením η v (42) výrazem $RMS[\Delta\boldsymbol{\theta}]_{t-1}$, dostane AdaDelta update pravidlo

$$\Delta\boldsymbol{\theta}_t = -\frac{RMS[\Delta\boldsymbol{\theta}]_{t-1}}{RMS[\boldsymbol{g}]_t} \boldsymbol{g}_t \tag{45}$$
$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \Delta\boldsymbol{\theta}_t \tag{46}$$

Vidíme, že při použití AdaDelta algoritmu nemusíme ani nastavovat rychlost učení, neboť byla eliminována z update pravidla [11],[12].

RMSprop

Algoritmy RMSprop a AdaDelta byly oba odvozeny ve stejné době nezávisle na sobě. Update pravidla tohoto algoritmu jsou identická rovnici (41) u algoritmu AdaDelta a tedy algoritmus RMSprop vypadá následovně [11]

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\eta}{\sqrt{p_{MA}[\boldsymbol{g}^2]_t + \epsilon}} \boldsymbol{g}_t. \tag{47}$$

Adam
Metoda odhadu adaptivního momentu je další metodou, která počítá adaptivní rychlosti učení pro jednotlivé parametry. Krom toho, že ukládá exponenciálně se rozkládající průměr předchozích umocněných gradientů $\boldsymbol{u}_t$, jako např. AdaDelta, Adam také ukládá exponenciálně se rozkládající průměr předchozích neumocněných gradientů, jako např. momentová metoda.

$$\boldsymbol{m}_t = \beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)\boldsymbol{g}_t, \tag{48}$$
$$\boldsymbol{u}_t = \beta_2 \boldsymbol{u}_{t-1} + (1-\beta_2)\boldsymbol{g}_t^2, \tag{49}$$

kde $\boldsymbol{m}_t$ a $\boldsymbol{u}_t$ jsou odhady prvního momentu (střední hodnoty), respektive druhého momentu (rozptylu) gradientů. Jelikož inicializujeme $\boldsymbol{m}_t$ a $\boldsymbol{u}_t$ jako nulové vektory, bylo zjištěno, že spočítáné odhady nejsou nestranné, obzvláště pří začátku učení a pokud hodnoty $\beta_{1,2}$ jsou malé.

Proto byla zavedeny následující korekce

$$\hat{\boldsymbol{m}}_t = \frac{m_t}{1-\beta_1^t}, \tag{50}$$
$$\hat{\boldsymbol{u}}_t = \frac{u_t}{1-\beta_2^t}. \tag{51}$$

Tyto korekce následně využijeme pro definování update pravidla pro Adam algoritmus

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\eta}{\sqrt{\hat{\boldsymbol{u}}_t} + \epsilon}\,\hat{\boldsymbol{m}}_t. \tag{52}$$

Autoři této metody doporučují nastavit výchozí hodnoty jako $\beta_1 = 0{,}9$, β_2=0,999, ϵ=10^{-8}. Rovněž ukázali, že tento algoritmus pracuje bez obtíží v praxi a jeho výkon v porovnání s ostatními algoritmy je více než uspokojivý [11],[13].

AdaMax
Jedná se o modifikaci algoritmu Adam. Člen $\boldsymbol{u}_t$ v update pravidla Adam algoritmu upravuje gradient nepřímo úměrně ℓ_2 normě předchozích gradientů (skrze výraz $\boldsymbol{u}_{t-1}$) a současného gradientu $|\boldsymbol{g}_t|^2$

$$\boldsymbol{u}_t = \beta_2 \boldsymbol{u}_{t-1} + (1-\beta_2)|\boldsymbol{g}_t|^2. \tag{53}$$

Výše zmíněné pravidlo můžeme zobecnit pomocí ℓ_p normy

$$\boldsymbol{u}_t = \beta_2^p \boldsymbol{u}_{t-1} + (1-\beta_2^p)|\boldsymbol{g}_t|^p. \tag{54}$$

Normy pro vysoké hodnoty p obecně bývají numericky nestabilní, proto se v praxi využívají hlavně ℓ_1 a ℓ_2 normy. Nicméně ℓ_∞ norma také obecně prokazuje stabilní chování. Z tohoto důvodu byl sestaven algoritmus AdaMax a jeho autoři ukázali, že $\boldsymbol{u}_t$ s ℓ_∞ normou konverguje k následující více stabilní hodnotě. Aby nedošlo ke zmatení s algoritmem Adam, zavedeme značení $\boldsymbol{\mu_t}$ pro $\boldsymbol{u}_t$ s ℓ_∞ normou

$$\boldsymbol{\mu}_k = \beta_2^\infty \boldsymbol{u}_{t-1} + (1-\beta_2^\infty)|\boldsymbol{u}_t|^\infty \tag{55}$$
$$= \max(\beta_2 \boldsymbol{u}_{t-1}, |\boldsymbol{g}_t|). \tag{56}$$

Nyní nám stačí v update pravidle pro Adam algoritmus, nahradit člen $\sqrt{\hat{\boldsymbol{u}}_t} + \epsilon$ výše vyjádřeným $\boldsymbol{\mu_t}$ a obdržíme update pravidlo pro AdaMax

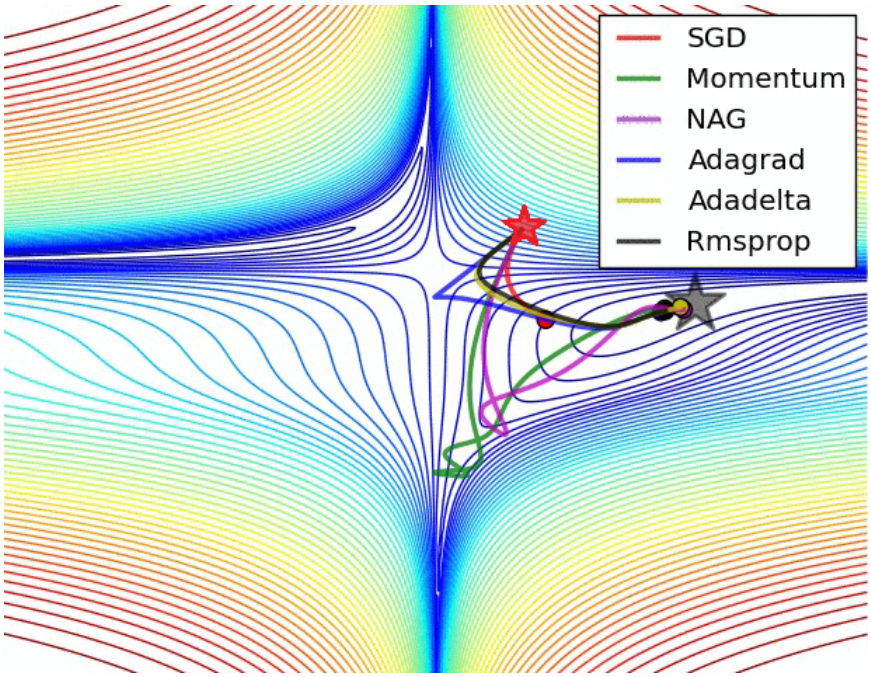
$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\eta}{\boldsymbol{\mu_t}} \hat{\boldsymbol{m}}_t \tag{57}$$

Doporučuje se volit výchozí hodnoty $\eta = 0,002$, $\beta_1 = 0,9$ a $\beta_2 = 0,999$ [13],[11].

Porovnání algoritmů učení

Chování jednotlivých algoritmů popisuje následující obrázek. Na obrázku 8 vidíme konturový graf tzv. Bealovy funkce. Jsou zde vykresleny cesty, jimiž se vydaly jednotlivé algoritmy ze stejného počátečního bodu (označen červenou hvězdičkou). Všimněme si, že AdaDelta, AdaGrad a RMSprop okamžitě zamířily správným směrem a konvergovaly zhruba stejně rychle do minima (znázorněno černou hvězdičkou). Na druhou stranu u obou momentových algoritmů můžeme pozorovat zmíněné chování kutálejícího se míče, s tím, že NAG algoritmus dosahuje minima dříve. Obyčejný SGD algoritmus nedosáhl minima a jeho rychlost konvergence je rovněž nejpomalejší.

Existuje však suverénně nejlepší algoritmus, který by se vyplatilo používat v každé situaci? Bohužel, nemůžeme tvrdit, že by existoval jasně dominující algoritmus. Ukazuje se, že algoritmy s adaptivní rychlosti učení předvádí často lepší výkony, avšak populární jsou stále i ostatní algoritmy. Ve výsledku tedy především záleží na uživatelově zkušenosti s daným algoritmem a na osobních preferencích [11], [4].



Obrázek 8 – Porovnání optimalizačních algoritmů [11]

2.3.4 Překážky při optimalizaci neuronových sítí

V této části se podíváme na možné problémy, na které lze narazit při optimalizaci naší neuronové sítě. Optimalizace ve strojovém učení je obtížný úkol, kterému se většinou je možno vyhnout vhodnou volbou chybové funkce. Ideálně bychom chtěli řešit konvexní optimalizační problém, avšak při trénování neuronových sítí se potýkáme obecně s nekonvexními problémy.

Špatná podmíněnost

Konkrétněji tedy špatně podmíněná Hessova matice $\boldsymbol{H}$ (více o Hessově matici v příloze A), bývá rozšířeným problémem v různých numerických optimalizacích, ať už konvexních nebo nekonvexních. Špatná podmíněnost Hessovy matice v SGD algoritmu může vést k tomu, že i velmi malé kroky budou zvyšovat chybovou funkci namísto zmenšování.

V příloze A nalezneme lehce upravenou rovnici (105) pro Taylorův rozvoj $f(\boldsymbol{x})$,

$$f(\boldsymbol{x}_0 - \eta \boldsymbol{g}) \approx f(\boldsymbol{x}_0) - \eta \boldsymbol{g}^\top \boldsymbol{g} + \frac{1}{2}\eta^2 \boldsymbol{g}^\top \boldsymbol{H} \boldsymbol{g}. \tag{58}$$

Zajímají nás především poslední dva členy na pravé straně aproximace. Špatná podmíněnost gradientu představuje problém, pokud

$$\frac{1}{2}\eta^2 \boldsymbol{g}^\top \boldsymbol{H} \boldsymbol{g} - \eta \boldsymbol{g}^\top \boldsymbol{g} > 0. \tag{59}$$

Abychom určili, zda špatná podmíněnost škodí neuronové síti, můžeme sledovat normy výrazů $\boldsymbol{g}^\top \boldsymbol{g}$ a $\boldsymbol{g}^\top \boldsymbol{H} \boldsymbol{g}$ umocněné na druhou. V mnoha případech se norma gradientu výrazně nezmenšuje v průběhu učení, avšak výraz $\boldsymbol{g}^\top \boldsymbol{H} \boldsymbol{g}$ roste o více než velikost řádu. Výsledkem pak je pomalé učení, navzdory přítomnosti velmi silného gradientu [4].

Jaké jsou možné příčiny vzniku špatně podmíněné Hessovy matice? Je jich hned několik, uvedeme si pár příkladů, včetně jejich možné nápravy [14]:

- Různé rozptyly ve vstupních proměnných. Tento problém spravíme vydělením každé vstupní proměnné její směrodatnou odchylkou.
- Nízká hodnota variačního koeficientu. Ke zbavení tohoto problému nám stačí odečíst střední hodnotu vstupní proměnné od všech jejich hodnot.
- Vysoká korelace mezi vstupními proměnnými. Tohoto problému se zbavíme ortonormalizací vstupních proměnných. Poznamenejme, že ortogonální prvky musí být standardizovány jako v prvním bodě.
- Nízké váhy mezi vstupní a skrytou vrstvou. Zde může pomoci zahrnout do sítě přímé spojení mezi vstupem a výstupem, abychom se postarali o možné lineární vztahy, zatímco se skryté vrstvy mohou zcela zaměřit na nelinearity.
- Nízké váhy mezi skrytou a výstupní vrstvou. Algoritmus se doporučuje začít s malým počtem skrytých neuronů a postupně je přidávat, až když jsou potřeba.
- Minima v nekonečnu. V průběhu učení se často stává, že váhy mezi skrytou a výstupní vrstvou se přibližují nekonečnu, zatímco váhy mezi vstupní a skrytou vrstvou se blíží k nule. Stačí však použít nějakou regularizační metodu, např. váhový rozklad, na váhy mezi výstupem a skrytou vrstvou.

Ačkoliv je důležité zvažovat špatnou podmíněnost při řešení problémů pomocí neuronových sítí, naším hlavním zájmem je přesnost výstupů (a celkové zobecnění), nikoliv však přesnost vah. Často je možné získat přesné výstupy bez přesných vah, neboť v podstatě špatná podmíněnost znamená, že velké změny ve váhách budou mít pouze malý dopad na chybovou funkci.

Lokální minima

Jedním z předních znaků konvexní optimalizace je fakt, že ji lze zjednodušit na hledání lokálního minima. Jakmile najdeme lokální minimum, tak víme, že se jedná i o globální minimum. Některé konvexní funkce mají místo jednoho lokálního minima konstantní plochou oblast, avšak jakýkoliv bod z této oblasti je akceptovatelné řešení.

V případě nekonvexní optimalizace, což jsou neuronové sítě, můžeme najít hned několik lokálních minim. V případě hlubokých sítí máme téměř zaručeno, že počet lokálních minim bude četný. Nicméně, ani tento fakt nemusí nutně představovat problém. Původem vzniku těchto lokálních minim je tzv. problém rozpoznatelnosti modelu. Řekneme, že model je rozpoznatelný, pokud dostatečně velká trénovací sada dat dokáže vyloučit všechny volby parametrů modelu krom jediného. Neuronové sítě a podobné modely však nejsou rozpoznatelné. Uvedeme příklad, uvažujme neuronovou síť, ve které upravíme první vrstvu následovně - příchozí vektor vah pro neuron i zaměníme s vektorem pro neuron j a to stejné provedeme i pro odchozí vektory. Pokud máme síť s m vrstvami a v každé n neuronů, pak existuje $n!^m$ způsobů jak uspořádat skryté neurony. Tento druh nerozpoznatelnosti je znám taky jako symetrie váhového prostoru.

Tento problém a jisté další znamenají, že v chybové funkci neuronové sítě může existovat nekonečně mnoho lokálních minim. Nicméně, všechna tato minima, pocházející z nerozpoznatelnosti modelu, jsou ekvivalentní, co se týče hodnoty chybové funkce. Tudíž tato lokální minima nejsou zcela nekonvexním problémem.

Lokální minima mohou představovat problém, pokud hodnota chybové funkce v těchto bodech je vysoká v porovnání s globálním minimem. Tato minima jsou problémem pro optimalizaci pomocí gradientního sestupu, obzvláště pokud jsou ve větším množství.

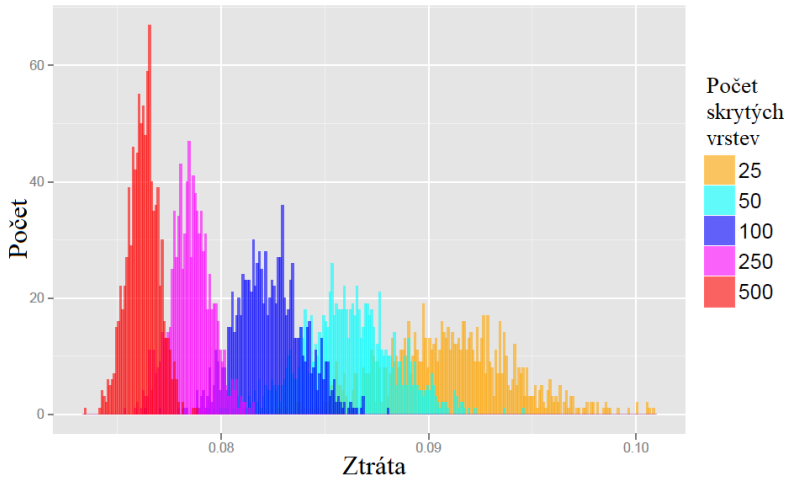
Na druhou stranu, zůstává stále otevřenou otázkou kolik doopravdy se takových minim ve většině sítí vyskytuje v praxi a zda-li na ně vůbec optimalizační algoritmy narazí. Dlouhou dobu se věřilo, že lokální minima jsou nejzákladnějším a nejčastějším problémem optimalizace neuronových sítí, avšak poslední dobou se ukazuje, že tomu tak nutně není [4]. Současná domněnka zní, že lokální minima, v dostatečně velkých neuronových sítích, mají malou hodnotu chybové funkce. Navíc není tendencí hledat opravdové globální minimum, ale spíš najít bod v prostoru parametrů s malou, ne nutně minimální, hodnotou chybové funkce.

Jednou z možností, jak přijít na to, jestli v naší síti lokální minima způsobují problémy, je vykreslit si průběh normy gradientu. Pokud se norma nezmenší na nevýraznou velikost, poté víme, že lokální minima problémem nejsou. Toto ověření může však být poněkud obtížné v prostorech vyšší dimenze [4].

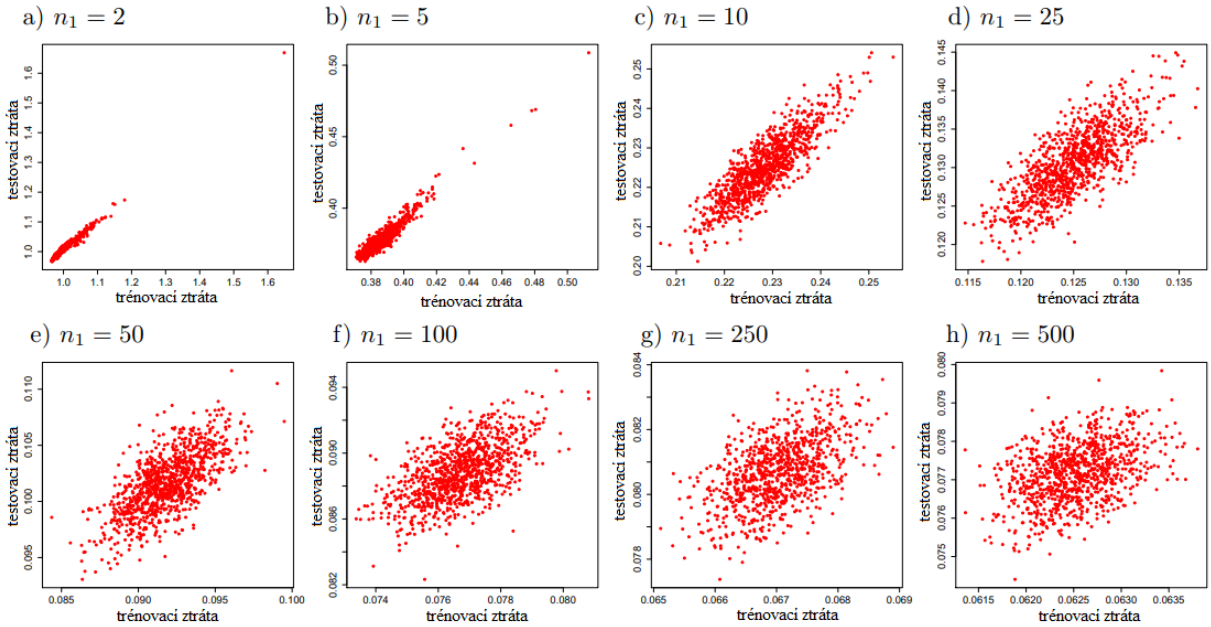
Zajímavá studie na toto téma byla publikována v roce 2015 [15]. Bylo zkoumáno rozložení hodnot chybové funkce pro model spinového skla a neuronové sítě napříč 1000 experimenty. Pokud se zaměříme na graf v obrázku 9, vidíme, že pro malý počet skrytých vrstev získáme lokální minima s relativně vyšší ztrátou, která jsou poněkud nekonzistentní napříč experimenty. Na druhou stranu, pro sítě s vysokým počtem skrytých vrstev vidíme velkou koncentraci lokálních minim kolem menší hodnoty ztrátové funkce, neboli rozptyl ztrát klesá s rostoucím počtem skrytých vrstev.

Toto zjištění naznačuje, že uvíznutí v lokálním minimu je opravdový problém pro malé sítě, který však s rostoucí velikosti sítě ztrácí na významnosti.

Výše zmíněné výsledky znamenají, že spočítaná řešení jsou s rostoucím počtem skrytých vrstev téměř ekvivalentní, co se týče tréninkové chyby. Zbývá otázka, jak se tento fakt promítne do testovací chyby? Aby to mohli posoudit, autoři spočítali korelaci mezi trénovací a testovací ztrátou pro každou velikost sítě, viz grafy na obrázku 10. Je patrné, že korelace těchto dvou chyb se zmenšuje s rostoucí velikostí sítě. Jedná se tedy o další náznak, že nalezení přesného globálního minimuma má jen omezený užitek v souvislosti se schopností sítě zobecňovat.



Obrázek 9 – Četnost ztrát v závislosti na počtu skrytých vrstev [15]



Obrázek 10 – Korelace mezi testovací a trénovací chybou [15]

Sedla a jiné ploché oblasti

Pro spoustu nekonvexních funkcí vyšších dimenzí, lokální minima (maxima) jsou poměrně vzácným úkazem ve srovnání s jiným typem bodů s nulovým gradientem - sedlem. Některé body v okolí sedla mají vyšší hodnotu chybové funkce, zatímco jiné nižší. V tomto bodu má Hessova matice kladné i záporné vlastní čísla, body ležící podél vlastních vektorů odpovídajících kladným vlastním číslům mají větší hodnotu chybové funkce a naopak.

U specifických funkcí můžeme pozorovat zvláštní chování: v prostorech nižší dimenze se lokální minima vyskytují běžně, zatímco ve vyšší dimenzi se častěji vyskytují sedlové body. Pro takovou funkci $f : \mathbb{R}^n \to \mathbb{R}$, poměr počtu sedlových bodů k počtu lokálních minim roste exponenciálně s n.

Pro tyto funkce dále platí, že vlastní čísla Hessovy matice budou pravděpodobněji kladná, když se dostaneme do oblasti s nízkou hodnotou chybové funkce. Z tohoto lze vyvodit, že lokální minima pravděpodobněji budou mít nižší hodnotu chybové funkce než vyšší. Pro sedlové body platí přesný opak.

Otázkou je, zda můžeme vidět podobné dění i v neuronových sítích? Několik studií v posledních desítkách let naznačuje, že ano [4]. Bylo teoreticky i experimentálně zjištěno, že se v neuronových sítích vyskytují chybové funkce s velkým počtem sedlových bodů. Proto nás zajímá, jaké implikace z tohoto plynou pro proces učení. Pro gradientní algoritmus je situace poněkud nejasná. Gradient může často nabývat malých hodnot v okolí sedlových bodů. Na druhou stranu, empiricky se ukazuje, že ve většině případů se algoritmus dokáže z těchto bodů dostat pryč. Rovněž bylo ukázáno, že časově spojitý gradientní sestup je dokonce analyticky odpuzován než přitahován okolním sedlovým bodem, avšak situace může být odlišná pro vyloženě praktické využití.

Srázy a explodující gradienty

Ve vícevrstvých neuoronových sítích se často vyskytují extrémně příkré oblasti, podobné srázům. Jsou výsledkem násobení několika vysokých váhových vektorů po sobě. Na této struktuře se krok gradientního algoritmu může „katapultovat" hodně daleko, obvykle zcela mimo srázovou strukturu.

Sráz může představovat problém, ať už se k němu přiblížíme zhora, anebo zespodu. Naštěstí se mu lze vyhnout pomocí jisté heuristiky. Gradientní algoritmus neurčuje optimální velikost kroku, nýbrž jen jeho směr v infinitezimální oblasti. Pokud algoritmus chce provést až přílíš velký krok, zasáhneme s heuristikou, která omezí tuto velikost [4].

2.4 Hyperparametry neuronových sítí

Neuronové sítě, podobně jako spousta dalších algoritmů strojového učení, vysoce zavisí na konkrétní počáteční inicializaci jejich tzv. hyperparametrů neboli specifických charakteristikách dané sítě, což je například konkrétní architektura sítě, rychlost učení, aktivační funkce, regulační parametry, optimalizační algoritmus a jiné. Ve spoustě algoritmů deep learningu počet hyperparametrů bez problémů překročí číslo 10. Výběr hodnot těchto parametrů je ekvivalentní otázce výběru modelu, mám-li množinu učících algoritmů, jak vybrat ten nejvhodnější?

Definujeme tedy hyperparametr učícího algoritmu jako proměnnou, jejíž hodnotu nastavíme před aplikací samotného algoritmu na zkoumané data, jedná se o proměnnou, kterou nenastavuje vybraný algoritmus. Můžeme se na hyperparametry dívat jako na kontrolní otočný knoflík naší sítě.

Učící algoritmy se skládají ze dvou prvků. První je model nebo trénovací kritérium, konkrétní chybová funkce, kterou chceme následně minimalizovat a druhý je postup pro optimalizaci tohoto kritéria. A tedy hyperparametry rozlišujeme, zaprvé jako hyperparametry spojené s optimalizací a zadruhé hyperparametry spojené se samotným modelem [16].

2.4.1 Hyperparametry spojené s modelem

Aktivační funkce

Než budou uvedeny samotné příklady aktivačních funkcí, zaměříme se na jejich vlastnosti. Úlohou aktivační funkce je vnést do neuronové sítě nelinearitu, a tím pádem jí umožnit řešit složitější problémy. Nepochybně proto musíme požadovat, aby tato funkce byla nelineární. Dalším požadavkem je spojitá diferencovatelnost, která zaručuje stabilnější průběh jistých optimalizačních algoritmů. Ze stejného důvodu rovněž vybíráme funkce, které jsou monotónně neklesající.

Dále je třeba rozlišovat, zda danou aktivační funkci je možno využít ve skryté vrstvě anebo ve výstupní vrstvě. Funkce ve skryté vrstvě většinou nemají velká omezení, zatímco ve výstupní vrstvě ano. Pokud řešíme klasifikační problém pomocí neuronových sítí, vyplatí se využívat funkce sigmoid a softmax, zatímco u regresních problémů nejčastěji využíváme identitu [17].

Podívejme se na krátký výčet možných funkcí [18],[4]:

- **Sigmoid**

$$\varphi(x) = \frac{1}{1 + e^{-x}} \tag{60}$$

  V minulosti se používal hlavně díky jasné interpretaci, kdy je daný neuron aktivní či neaktivní. Nicméně v dnešní době už se nevyužívá především ze dvou důvodů. Bez větších detailů jen zmiňme, že díky svým vlastnostem v oblasti, kde funkční hodnoty nabývají téměř 1 nebo 0, je nevhodný pro učení pomocí tzv. backpropagation. Rovněž výstupy této funkce nemají střední hodnotu rovnu 0, což opět negativně ovlivňuje proces učení.

- **Hyperbolický tangens**

$$\varphi(x) = \tanh(x) \tag{61}$$

Narozdíl od sigmoidu, výstupem této funkce jsou hodnoty v intervalu ⟨-1,1⟩. Nicméně se potýká s podobným problémem jako jeho předchůdce a tedy taky není ideálním kandidátem pro stejný styl učení.

- **ReLU**

$$\varphi(x) = \max(0, x) \tag{62}$$

Usměrněná lineární funkce se v posledních letech těší velké popularitě. Nepotýká se s problémy jako dvě předchozí funkce a navíc je méně náročná na výpočetní sílu, neboť neobsahuje žádné exponenciální funkce. Jednou z nevýhod však je fakt, že při učení sítě s touto aktivační funkcí mohou neurony ,,zemřít"- nebudou nikdy aktivovány, a tudíž nebudou přispívat k procesu učení.

- **Propustná ReLU**

$$\varphi(x) = \begin{cases} 1, & x < 0, \\ \alpha x + 1, & x \geq 0. \end{cases} \tag{63}$$

Modifikací předchozí funkce dostaneme propustnou usměrněnou lineární funkci, která má být řešením k problému usměrněné lineární funkce. V jistých případech je efektivnější volbou než obyčejná ReLU, ale bohužel se nejedná vždy o konsistentní výsledky.

- **Softmax**

$$\varphi(\mathbf{x}) = \text{softmax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{64}$$

Softmax funkce se nejčastěji používá ve výstupní vrstvě klasifikátorů s více než dvěmi třídami. Představuje distribuci pravděpodobnosti pro n různých tříd. Z hlediska neurovědy poskytuje tato funkce zajímavý pohled. Dá se totiž o ní uvažovat jako o vytvoření určité ,,soutěže"mezi neurony, neboť suma výstupu této funkce je vždy rovna 1 a tedy, pokud dojde ke zvýšení hodnoty u jednoho neuronu, nutně musí dojít ke snížení hodnoty u jiného neuronu.

- **Funkce identita**

$$\varphi(x) = x \tag{65}$$

Ačkoliv na začátku této sekce bylo uvedeno, že nás zajímají nelineární funkce, tak i lineární funkce mají svůj význam mezi aktivačními funkcemi. Využijeme ji především ve výstupní vrstvě u regresních problémů, kdy požadujeme, aby celkový výstup sítě dokázal aproximovat jakékoliv reálné číslo a nezajímají nás rozdělení pravděpodobnosti jako u předchozího příkladu.

Počet skrytých neuronů

Každé vrstvě v hluboké neuronové sítě můžeme určit libovolný počet skrytých neuronů. Nicméně, použití nedostatku skrytých neuronů povede k underfittingu, neboť tyto neurony nebudou schopny zachytit veškeré znaky v komplikované datové sadě. V důsledku brzkého zastavení učení (abychom předešli overfittingu) či jiných regularizačních technik (např. rozpad vah) je důležité zvolit tento počet dostatečně velký. Obecně, síť s větším než optimálním počtem skrytých neuronů nebude těžce postihnuta v rámci zobecňování, avšak prodlouží se výpočetní doba. Rovněž pokud jich použijeme až příliš, tak dojde k overfittingu, neboť síti poskytneme až moc velkou kapacitu pro zpracování informací.

Snažíme se najít kompromis mezi těmito extrémy. Bohužel, neexistují jasně daná pravidla, která by určovala přesná čísla. Musíme se úchylit k doporučeným konvencím [19]:

- Počet skrytých neuronů by se měl pohybovat mezi velikostí vstupní a výstupní vrstvy.
- Počet skrytých neuronů by měl být 2/3 velikosti vstupní vrstvy plus velikost výstupní vrstvy.
- Počet skrytých neuronů by měl být menší než dvojnásobek velikosti vstupní vrstvy.

Ani tato nepsaná pravidla ale neplatí vždy, proto pro určení optimální počtu skrytých neuronů je důležité pozorně prozkoumat naše data a zvážit obtížnost problému, který chceme vyřešit.

Za zmínku ještě stojí článek publikovaný v roce 2017 [20], který zkoumal aproximační schopnosti hlubokých sítí s ReLu aktivačními funkcemi. Zatímco univerzální aproximační věta vyšla z výzkumu vlivu šířky sítě bez pomoci hloubky, autoři tohoto článku se zaměřili na otázku, jaká je minimální šířka skrytých vrstev sítě, s libovolnou hloubkou, potřebná pro aproximaci libovolné funkce. V článku je dokázáno, že sítě s ReLu s šířkou skrytých vrstev $d + 2$ dokáží aproximovat spojitou funkci f s d proměnnými libovolně přesně, předpokládame-li neomezenou hloubku. Pro spojité a konvexní funkce je počet skrytých vrstev jen $d + 1$.

Jako i pro ostatní hyperparametry, máme možnost určit počet neuronů pro každou vrstvu zvlášť. Ukazuje se, že síť se stejným počem neuronů ve všech vrstvách obecně předvádí lepší nebo stejné výkony jako sítě s ubývajícím počtem neuronů (pyramidovitý tvar) nebo s rostoucím počtem neuronů (obrácená pyramida), nicméně vždy záleží na typu dat.

Zaměřme se ještě na počet skrytých vrstev, do kterých budeme neurony rozdělovat. Předmětem našeho zájmu bude především aproximace funkcí. Podle Věty 2.1 nám k aproximaci libovolné funkce stačí dopředná neuronová síť s jedinou skrytou vrstvou. Pro některé typy funkcí je opravdu jedna vrstva dostatečná. Avšak v roce 2017 byla publikována studie porovnávající sítě s jednou a dvěma skrytými vrstvami [21]. Autoři článku ukazáli, že pro 9 z 10 jimi vybraných data setů pro aproximaci funkce, síť se dvěmi skrytými vrstvami mírně předčila svým výkonem síť s jednou skrytou vrstvou. Zlepšení výkonu však bylo závislé na typu dat.

Rozpad vah

Jedná se o regularizační techniku zabraňující nárůstu hodnot vah do nechtěně vysokých hodnot. Vzpomeňme si na rovnici (28), kterou přepíšeme jen pro váhy, neboť vychýlení nevyžadují regularizaci

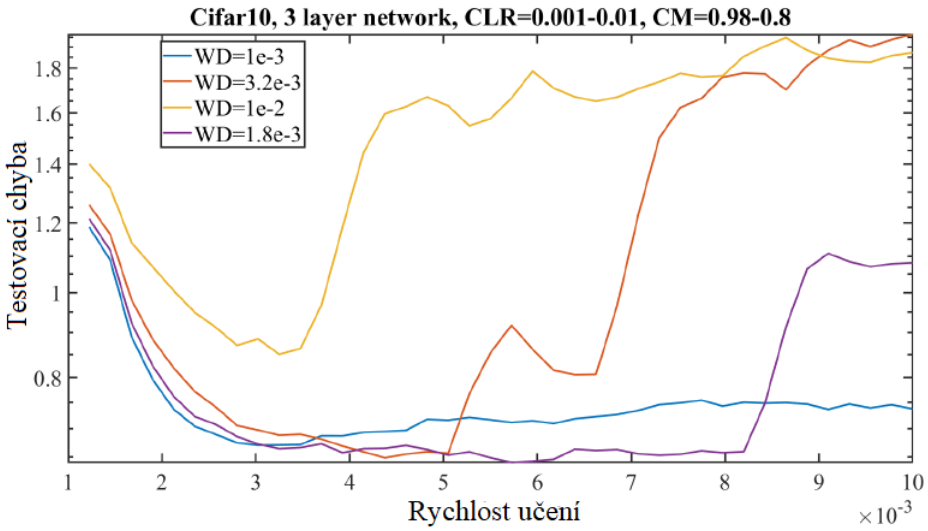
$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta \nabla_{\boldsymbol{\theta}} L(\boldsymbol{y}^{(p)}, \boldsymbol{x}^{(p)}, \boldsymbol{\theta}) - \lambda \boldsymbol{w}_t, \tag{66}$$

kde λ je rychlost rozpadu vah za jeden časový krok. Ukazuje se, že tento parametr je nejlepší ponechat konstantní po celou dobu učení. Jelikož výkon sítě je na tomto parametru závislý, je ideální použít tzv. mřížkové prohledávání pro jeho určení, neboť rozdíly ve výkonu půjdou vidět brzy v průběhu učení [22].

Hodnoty λ obvykle volíme $10^{-3}, 10^{-4}, 10^{-5}$ nebo 0. Ukazuje se, že pro menší datové sady a architektury je zapotřebí použít větších hodnot, zatímco pro větší neuronové sítě stačí hodnoty menší.

Je důležité mít na paměti, že regularizace sítě musí být vyvážená pro specifickou architekturu a data. Hodnota rozpadu vah je poměrně úzce propojena s rychlostí učení, o které bude řeč v další sekci.

Naší snahou je tedy ideálně vyladit oba parametry co nejoptimálněji. Často se tak děje najednou, čili zkoumáme testovací chybu naší sítě, zatímco měníme oba dva zmíněné parametry v určitých mezích, viz obrázek 11. Vidíme, že nejlepší volbou je fialová křivka, neboť zůstává stabilní pro největší rozsah rychlosti učení a zároveň si udržuje nejnižší testovací chybu [8].



Obrázek 11 – Testovací chyba pro různé hodnoty rozpadu vah - WD [8]

2.4.2 Hyperparametry spojené s optimalizací

Rychlost učení

Připomeňme si rovnici pro stochastický gradientní sestup

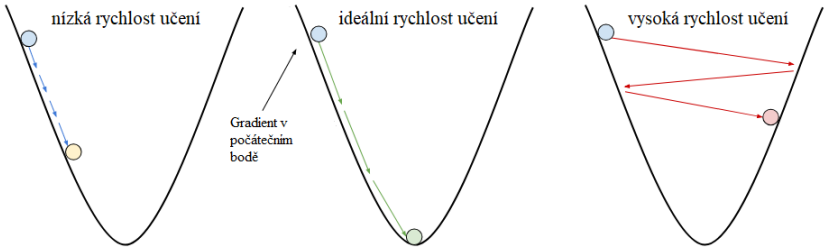
$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla_{\boldsymbol{\theta}} L(\boldsymbol{y}^{(p)}, \boldsymbol{x}^{(p)}, \boldsymbol{\theta}), \tag{67}$$

kde η je rychlost učení. Určuje o kolik se změní hodnoty parametrů v průběhu učení. Obvykle ji volíme jako malé kladné číslo, nejčastěji v intervalu $\eta \in (0, 1)$.

Pomocí tohoto hyperparametru kontrolujeme celkovou rychlost procesu učení. Obecně můžeme říct, že při zvolení vyšší hodnoty rychlosti učení se model bude učit rychleji, za cenu ne zcela optimálního výsledku. Naopak nižší hodnota rychlosti učení umožní modelu nalézt optimálnější řešení, avšak doba učení se může protáhnout.

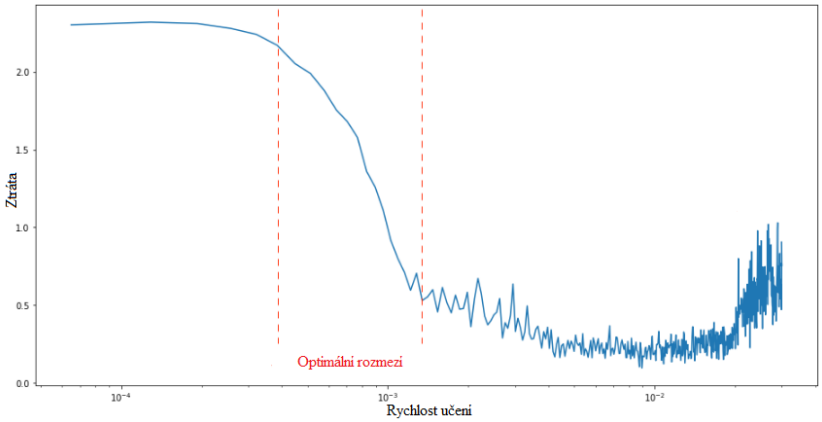
Kdybychom však vybrali hodnoty až příliš vysoké, učení vůbec nemusí být úspěšné, neboť gradientní algoritmus nebude snižovat trénovací chybu, ale zvyšovat ji. Pro hodnoty až příliš malé můžeme říct to stejné, protože algoritmus se může zaseknout s vysokou trénovací chybou a nebude schopen konvergence k optimálnímu řešení [16].

Zjednodušeně je situace popsána na obrázku 12. Vidíme graf nějaké chybové funkce a chování algoritmu, který se snaží najít její minimum.



Obrázek 12 – Rychlost učení [23]

Tedy, ideální volba rychlosti učení se promítne v prudký pokles hodnoty chybové funkce. Pro příliš malé rychlosti se ztráta bude snižovat jen velmi pomalu a pro příliš velké se může zcela odchýlit od minima. Vše je vidět na obrázku 13.



Obrázek 13 – Rychlost učení [24]

Očividně se jedná o nejdůležitější hyperparametr celé neuronové sítě. Pokud bychom měli čas nebo výpočetní kapacitu pro úpravu pouze jediného parametru, tak by to byl tento. Bohužel, nelze analyticky spočítat optimální hodnotu rychlosti učení, nýbrž určit ji metodou pokus omyl a sledovat jak se mění výkon naší sítě.

Alternativní metodou nalezení nejvhodnější hodnoty parametru, je použít prohledávání na mřížce. Díky této metodě zjistíme řád velikosti vhodných hodnot a rovněž uvidíme vztah mezi výkonem sítě a tímto parametrem. Obvykle prohledáváme hodnoty na logaritmické škále od 10^{-1} do 10^{-6}.

Jinou možností, jak přistoupit k rychlosti učení, je průběžně ji měnit v průběhu učení, namísto jediné fixně dané hodnoty. Nejjednodušeji tak provedeme lineárním snižováním hodnoty parametru z jeho původně vyšší hodnoty. Díky tomuto postupu dojde ke zrychlení na začátku učení, kdy si to můžeme dovolit. Naopak ke konci učení budeme pracovat s malými změnami u parametrů sítě, což nám umožní precizní nalezení optimálního řešení.

Výše zmíněnou rychlost učení můžeme popsat například takto

$$\eta_t = \frac{\eta_0 \tau}{\max(t, \tau)}. \tag{68}$$

Tento vztah můžeme interpretovat tak, že rychlost učení zůstane konstantní pro prvních τ časových kroků a následně se zmenšovat [16].

Další možností je použití tzv. adaptivní rychlosti učení. Některé učící algoritmy jsou schopny monitorovat výkon sítě během učení a v závislosti na něm tento parametr upravovat. Obtížnost správné volby rychlosti učení před samotným počátkem učení, je důvod, proč jsou adaptivní metody populární. Dobrý adaptivní algoritmus mnohdy konverguje rychleji než obyčejný algoritmus s ne zcela vhodně nastavenou rychlostí učení [25].

Poslední možností je využít tzv. cyklických rychlostí učení. U této metody musíme určit minimální a maximální meze rychlosti učení a velikost kroku. Tato velikost odpovídá nějakému počtu iterací. Jeden cyklus se pak skládá ze dvou takových kroků. V prvním kroku se hyperparametr lineárně zvětšuje z minima až po maximum a ve druhém se lineárně snižuje. Obvykle se nejdříve provede rozsahový test před samotným učením, tedy spustíme algoritmus a lineárně zvětšujeme rychlost učení. Tento test by nám měl napovědět, jak dobře jsme schopni naši síť vytrénovat na nějakém rozsahu rychlosti učení a jaké je její maximum [8].

Velikost dávky

Zlatou střední cestou mezi algoritmy gradientního sestupu a stochastického gradientního sestupu je stochastický gradientní sestup s mini-dávkou. Rovnici (28) upravíme následovně

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \frac{1}{B} \sum_{p=1}^{B} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{y}^{(p)}, \boldsymbol{x}^{(p)}, \boldsymbol{\theta}), \tag{69}$$

kde B značí velikost dávky.

Teoreticky vzato, volba tohoto hyperparametru ovlivňuje pouze výpočetní stránku našeho algoritmu. Pokud pracujeme s vysokou hodnotou B, updaty jednotlivých parametrů budou počítány efektivněji díky paralelním architekturám. Na druhou stranu, při volbě malé hodnoty B jsme schopni těchto updatů spočítat více. Jak již bylo řečeno, B neovlivňuje schopnost sítě zobecňovat a tedy můžeme ji optimalizovat odděleně od ostatních hyperparametrů [16] .

Velikost B nejčastěji volíme mezi jednou až několik stovek, záleží na velikosti datové sady. Klasicky ji volíme jako násobky dvou, které vyhovují paměťovým požadavkům GPU nebo CPU, takže se setkáme s hodnotami $32, 64, 128, 256$ a tak dále. Velmi oblíbenou volbou je především $B = 32$ [26].

Ideální volbu velikosti dávky pro naši síť nejčastěji určíme z grafu trénovací chyby v závislosti na výpočetní době, jakmile byly určeny ostatní hyperparametry.

Počet tréninkových iterací

Tento hyperparametr je velmi snadno optimalizovatelný, díky metodě dřívějšího zastavení. Stačí nám například sledovat trénovací chybu v průběhu učení a můžeme se rozhodnout, kolik více iterací našemu algoritmu povolíme. Jedná se o poměrně účinnou zbraň proti overfittingu, i v případě, že ostatní hyperparametru k tomu budou silně přispívat.

Na druhou stranu, ne vždy chceme využít dřívejšího zastavení učení naší sítě. Pokud nás zajímá vliv jednotlivých hyperparametrů, je vhodnější tuto metodu vůbec nepoužívat, abychom mohli pozorovat jejich opravdové vlivy na učení.

V praxi potom výběr tohoto hyperparametru probíhá následovně. Necháme algoritmus trénovat déle než vybraný počet $\hat{T}$ tréninkových iterací (v tomto bodě by měla být hodnota chybové funkce validační sady nejmenší), abychom se ujistili, že hodnota chybové funkce nebude dále klesat. Nadále byla zavedena heuristika tzv. parametr trpělivosti, což je minimální počet trénovacích příkladů po bodě $\hat{T}$, které ještě algoritmu předložíme, před samotným zastavením algoritmu. V průběhu učení pak získáváme nové kandidáty pro $\hat{T}$, přičemž zvyšujeme parametr trpělivosti pro každé nově zvolené $\hat{T}$. Pokud najdeme nové minimum v t, uložíme momentálně nejlepší model, updatujeme $\hat{T} \leftarrow t$ a zvýšíme parametr trpělivosti [16].

Chybová funkce

Chybová funkce je důležitou součástí neuronových sítí, která měří správnost našich výsledků, jež značíme $\hat{\boldsymbol{y}}$, s opravdovými výsledky $\boldsymbol{y}$. Rovněž ji nazýváme ztrátová funkce, tyto dva pojmy jsou pro naše účely zaměnitelné. Hodnoty této funkce pak můžeme nazývat ztrátou nebo chybou. Chybovou funkci vybíráme dle úkolu, který naši neuronové síti zadáme. Následuje stručný výčet nejčastěji používaných chybových funkcí [27],[28]:

- **Mean squared error**

$$E = \frac{1}{2n} \sum_{i=1}^{n} (\boldsymbol{y}^{(i)} - \hat{\boldsymbol{y}}^{(i)})^2 \tag{70}$$

  Nejčastěji tuto funkci využíváme pro regresní úlohy. Nicméně má jeden hlavní nedostatek, a to tzv. outliery. Tyto body jsou pak těžce penalizovány kvadratickou odchylkou, a proto je třeba data nejdřív odfiltrovat od outlierů a teprve potom použít tuto chybovou funkci.

- **Mean squared logarithmic error**

$$E = \frac{1}{n} \sum_{i=1}^{n} (\log(\boldsymbol{y}^{(i)} + 1) - \log(\hat{\boldsymbol{y}}^{(i)} + 1))^2 \tag{71}$$

  Rozdíl od předchozího funkce je ve využití logaritmů, díky kterým se změní měřený rozptyl. Tuto funkci využijeme, pokud nechceme penalizovat velké rozdíly mezi predikovanými a skutečnými hodnotami, když jsou obě tyto hodnoty vysoké.

- **Mean absolute error**

$$E = \frac{1}{n}\sum_{i=1}^{n}|\boldsymbol{y}^{(i)} - \hat{\boldsymbol{y}}^{(i)}| \tag{72}$$

MAE opět využijeme pro změření, jak blízko jsou naše predikce k opravdovým výstupům. Rozdíl mezi MSE a MAE je však ve složitosti výpočtu gradientu. Gradient MSE není moc obtížné spočítat, zatímco pro MAE je třeba využít např. lineárního programování. Na druhou stranu MAE je robustnější, co se týče přítomnosti outlierů.

- **Mean absolute percentage error**

$$E = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{\boldsymbol{y}^{(i)} - \hat{\boldsymbol{y}}^{(i)}}{\boldsymbol{y}^{(i)}}\right| \cdot 100 \tag{73}$$

Jedná se o modifikaci předchozí funkce. Ačkoliv na první pohled vypadá jednoduše, skrývá v sobě několik úskalí. Například, nelze použít pokud požadovaná data obsahují nulové hodnoty. Navíc pro předpovědi příliš vysoké prakticky neexistuje horní hranice procentuální chyby.

- **Kuhllback Leibler Divergence**

$$E = \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{y}^{(i)} \cdot \log(\boldsymbol{y}^{(i)})) - \sum_{i=1}^{n}(\boldsymbol{y}^{(i)} \cdot \log(\hat{\boldsymbol{y}}^{(i)})) \tag{74}$$

Rovněž známá jako relativní entropie, měří jak se jedno rozdělení pravděpodobnosti liší od druhého očekávaného rozdělení. Hodnota nula naznačuje, že můžeme očekávat podobné, ne-li stejné, chování dvou různých distribucí, zatímco hodnota jedna naznačuje, že tyto dvě distribuce se chovají zcela odlišně.

- **Cross entropy**

$$E = -\frac{1}{n}\sum_{i=1}^{n}\left[\boldsymbol{y}^{(i)} \cdot \log(\hat{\boldsymbol{y}}^{(i)}) + (1 - \boldsymbol{y}^{(i)})\log(1 - \hat{\boldsymbol{y}}^{(i)})\right] \tag{75}$$

Nejčastěji tuto funkci používáme pro binární klasifikaci, čili pokud výstupy mají hodnoty nula nebo jedna. Obdobně jako předchozí funkce měří podobnost dvou pravděpodobnostních rozdělení. Pro malé hodnoty této funkce můžeme prohlásit, že dvě rozdělení jsou si podobná a naopak.

- **Negative Logarithmic Likelihood**

$$E = -\frac{1}{n}\sum_{i=1}^{n}\log(\hat{\boldsymbol{y}}^{(i)}) \tag{76}$$

Široce využívaná funkce v neuronových sítích, používá se pro měření přesnosti klasifikačních úloh. Je tedy zapotřebí, aby výstupy našeho modelu byly pravděpodobnosti pro jednotlivé třídy.

2.4.3 Optimalizace hyperparametrů

Mějme libovolný učící algoritmus $\mathcal{A}$. Jeho úkolem je najít funkci f, která minimalizuje hodnotu chybové funkce $E(\boldsymbol{x}; f)$. Algoritmus $\mathcal{A}$ dospěje k funkci f díky optimalizaci zadaného tréninkového kritéria vzhledem k množině parametrů $\boldsymbol{\theta}$. Nicméně, samotný učící algoritmus má rovněž své parametry, které nazýváme hyperparametry λ a opravdový algoritmus obdržíme až po určení λ, což označíme $\mathcal{A}_\lambda$ a $f = \mathcal{A}_\lambda(\boldsymbol{x}^{(\text{trénink})})$ pro trénovací sadu dat $\boldsymbol{x}^{(\text{trénink})}$.

V praxi se snažíme najít takové hodnoty λ, které minimalizují chybu zobecnění $\mathbb{E}_{\boldsymbol{x}\sim\mathcal{G}_{\boldsymbol{x}}}[E(\boldsymbol{x}, \mathcal{A}_\lambda(\boldsymbol{x}^{(\text{trénink})}))]$, kde $\mathcal{G}_{\boldsymbol{x}}$ je nám neznámé rozdělení $\boldsymbol{x}$. Je zřejmé, že výpočet provedený algoritmem $\mathcal{A}$ sám o sobě obsahuje další optimalizační problém - optimalizace hyperparametrů λ. Můžeme to zapsat jako

$$\lambda^{(*)} = \arg\min_{\lambda\in\Lambda} \mathbb{E}_{\boldsymbol{x}\sim\mathcal{G}_{\boldsymbol{x}}} \left[E\left(\boldsymbol{x}, \mathcal{A}_\lambda(\boldsymbol{x}^{(\text{trénink})})\right)\right]. \tag{77}$$

Obecně však nemáme k dispozici takový algoritmus, který by provedl výše naznačenou optimalizaci. Navíc nejsme schopni určit ani střední hodnotu neznámého rozdělení. K dalšímu postupu tedy musíme využít dalších nástrojů, konkrétně - křížovou validaci. Tato technika využívá průměr tzv. validační sady dat $\boldsymbol{x}^{(validace)}$ následovně

$$\lambda^{(*)} \approx \arg\min_{\lambda\in\Lambda} \operatorname*{mean}_{\boldsymbol{x}\in\boldsymbol{x}^{(\text{validace})}} \left[E\left(\boldsymbol{x}, \mathcal{A}_\lambda(\boldsymbol{x}^{(\text{trénink})})\right)\right] \tag{78}$$

$$\equiv \arg\min_{\lambda\in\Lambda} \Psi(\lambda) \tag{79}$$

$$\approx \arg\min_{\lambda\in\{\lambda^{(1)},\ldots,\lambda^{(l)}\}} \Psi(\lambda) \equiv \hat{\lambda} \tag{80}$$

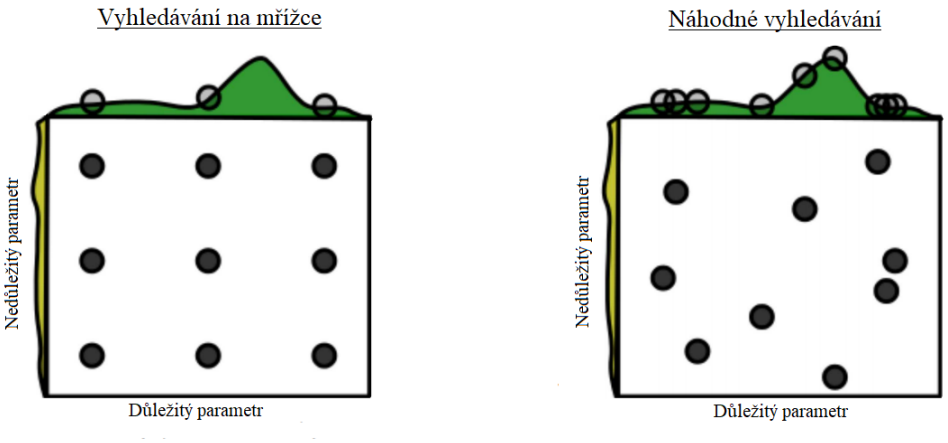
Optimalizace hyperparametrů je minimalizace funkce $\Psi(\lambda)$ přes $\lambda \in \Lambda$. Obecně o funkci Ψ nebo prohledávaném prostoru Λ nevíme skoro nic, proto se často uchylujeme k volbě l zkušebních bodů $\{\lambda^{(1)}, ..., \lambda^{(l)}\}$, vyhodnotíme pro všechny $\Psi(\lambda)$ a určíme $\lambda^{(i)}$, které mělo nejlepší výsledky jako $\hat{\lambda}$. Toto je popsáno poslední rovnicí z výše uvedené trojice.

Rozhodujícím krokem v optimalizaci hyperparametrů je volba množiny $\{\lambda^{(1)}, ..., \lambda^{(l)}\}$. Existuje několik postupů jak tuto množinu určit, uveďme si ty nejpoužívanější.

Nejpřirozenější metoda se nám bezpochyby jeví metoda pokus-omyl neboli ruční hledání. Vybereme hodnoty hyperparametrů λ a sledujeme jejich dopad na výkon sítě a následovně je intuitivně upravujeme dle potřeby. Ačkoliv tato metoda je nejméně efektivní, stále si nachází své uplatnění, především ze dvou důvodů. Prvním důvodem je nenáročná implementace, neboť se jedná o nejjednodušší možnou metodu. Druhým důvodem použití je za účelem získání prvotního náhledu na funkci Ψ. Avšak je zcela jasné, proč se manuální hledání nepoužívá jako hlavní metoda pro určení hyperparametrů. Je prakticky nemožné najít optimální hodnoty pomocí této metody.

Manuální vyhledávání se však často využívá s jinou, lehce komplikovanější metodou - vyhledávání na mřížce. Princip spočívá v určení množiny možných hodnot našich hyperparametrů a následně naučit síť se všemi možnými kombinacemi těchto hodnot. Výhody opět spočívají v jednoduché implementaci a možnosti paralelizovat výpočet. Hlavní nevýhoda pak spočívá v době výpočtu pro velký počet hyperparametrů. Máme-li například 10 hyperparametrů, každý s 10 možnými hodnotami, tak obdržíme 10^{10} různých vyhodnocení, což může být časově náročné.

Proto se často uchylujeme k další metodě - náhodnému prohledávání. Srovnání těchto dvou metod je názorně ukázáno na obrázku 14. Vidíme, že při prohledávání na mřížce máme rovnoměrné pokrytí původního prostoru, avšak při projekci na jednu či druhou osu dostaneme nedostatečné pokrytí daného podprostoru. Při náhodném prohledávání je situace zcela opačná. Hlavní síla náhodného prohledávání tkví ve faktu, že ne všechny hyperparametry jsou při optimalizaci stejně důležité. Naopak prohledávání na mřížce věnuje výpočetní sílu prohledávání dimenzí, které nejsou tolik důležité a poté ztrácí při prohledávání těch důležitějších [29].



Obrázek 14 – Porovnání náhodného hledání s hledáním na mřížce [29]

Poslední metodě, které se budeme věnovat je Bayesovská optimalizace. Mějme nějakou funkci $f(x)$, kterou se snažíme minimalizovat na omezené množině X. Podle této metody nejdříve sestavíme pravděpodobnostní model pro tuto funkci, a poté jej využijeme k rozhodnutí, kde v X provedeme další vyhodnocení funkce $f(x)$. Hlavní myšlenkou je využít veškerých dostupných informací z předchozích vyhodnocení $f(x)$.

Při práci s Bayesovskou optimalizací musíme učinit dvě hlavní rozhodnutí. Zaprvé, vybrat pravděpodobnostní rozdělení funkcí, které budou znázorňovat předpoklady o optimalizované funkci. Oblíbenou volbou je například Gaussovský proces, což je náhodný proces, jehož všechna konečně rozměrná rozdělení jsou normální. Zadruhé, musíme vybrat vhodnou tzv. akviziční funkci, díky které získáme další bod na vyhodnocení. Opět uvedeme jednu z nejpoužívanějších funkcí, a to očekávaného zlepšení [30]. Jednou z možností, jak ji zapsat je

$$g_{\min(x)} = \max(0, y_{\min} - y_{omin}), \tag{81}$$

kde y_{min} je minimum pozorovaných hodnot y a y_{omin} je nejnižší možná hodnota z intervalu spolehlivosti daných x, které obdržíme díky Gaussovským procesům [31].

3 Termodynamika páry

Hlavním cílem této kapitoly je stručně shrnout základní termodynamické vlastnosti vodní páry a veličiny jimiž jsou popsány. Rovněž ukážeme, jakým způsobem se tyto veličiny v současné době počítají.

3.1 Základní popis par

Než začneme s popisem páry jako takové, uveďme si nejdříve, co je to ideální plyn. Částice tohoto plynu považujeme za dokonale elastické hmotné body, s nulovým objemem, které na sebe nepůsobí přitažlivými ani odpudivými silami a jsou v neustálém a neuspořádaném pohybu. Jejich fyzikální vlastnosti jsou konstantní.

Pára však není ideálním plynem, nýbrž reálným. Vztahy mezi reálnými stavovými veličinami jsou složitější a jejich fyzikální vlastnosti se rovněž odlišují.

Je známo, že ideální plyny jsou popsány stavovou rovnicí pro ideální plyn

$$pv = rT \quad \text{neboli} \quad \frac{pv}{rT} = 1, \tag{82}$$

kde p je tlak, v měrný objem, r plynová konstanta a T termodynamická teplota. Nicméně, tento vztah neplatí pro reálné plyny, protože výše zmíněný zlomek může nabývat hodnot buď větších či menších než jedna

$$\frac{pv}{rT} \gtrless 1. \tag{83}$$

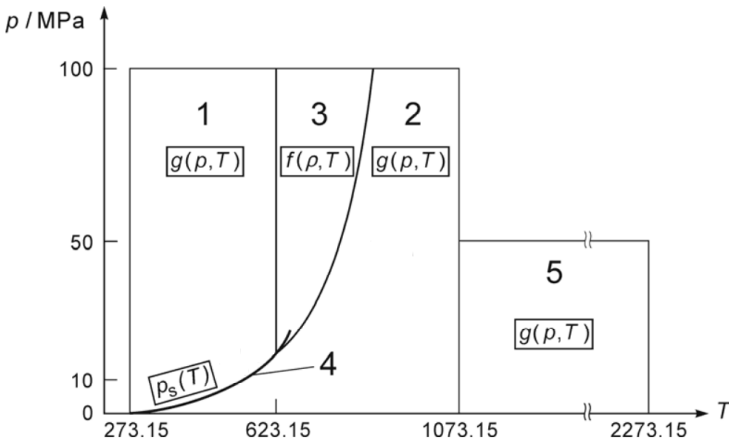
Pro reálné plyny existuje stavových rovnic hned několik. Uveďme si tu nejznámější rovnici, Van der Waalsovu rovnici, která byla odvozena ze stavové rovnice ideálního plynu

$$\left(p + \frac{a}{v^2}\right)(v - b) = rT, \tag{84}$$

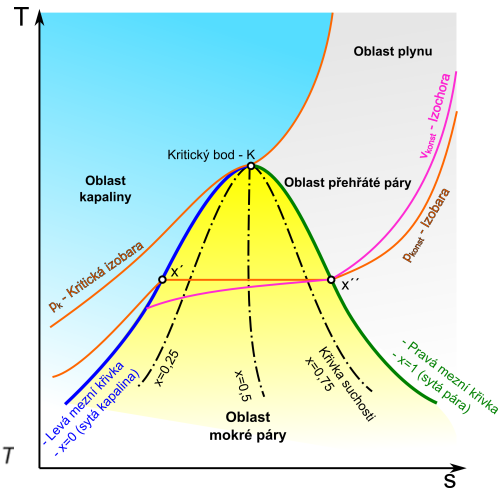
kde $\frac{a}{v^2}$ představuje tzv. kohezní tlak, jedná se o korekci související s existencí přitažlivých sil mezi molekulami. Člen b značí korekci vlastního objemu molekul, nazýváme jej kovolumen. Obě tyto konstanty závisí na daném plynu a obvykle jsou určeny experimentálně.

Stavy páry a závislost jejich stavových veličin zobrazujeme pomocí fázových diagramů. Fázových diagramů je několik druhů, využívají se například $p - v$, $p - T$, $T - s$, $h - s$ diagramy a další. Na konkrétním diagramu si popíšeme možné stavy páry.

Na obrázku 16 je znázorněn $T - s$ diagram vodní páry. Sledujme oranžovou křivku pojmenovanou pouze Izobara. Tato křivka znázorňuje ohřev vody za konstantního tlaku. V modré oblasti dochází k postupnému zvyšování teploty a měrné entropie vody. Jakmile teplota dosáhne hodnoty teploty varu (teplota odpovídající bodu x') při daném tlaku, tak se kapalina již dále neohřívá. O této kapalině hovoříme jako o syté kapalině. Při dalším ohřevu syté kapaliny však nedojde ke zvyšení teploty, dokud se veškerá kapalina nepřemění v sytou páru. V diagramu tento stav odpovídá bodu x''. Mezi oběma sytými stavy se nachází jejich směs, jež nazýváme mokrá pára. Při pokračujícím přívodu tepla syté páře, teplota páry opět roste a dojde k přechodu do přehřáté páry [32].



Obrázek 15 – $p-T$ diagram [33]



Obrázek 16 – $T-s$ diagram [34]

Věnujme ještě pozornost $p-T$ diagramu na obrázku 15. Tento diagram byl vydán Mezinárodní asociací pro vlastnosti vody a vodní páry (IAPWS). Pro každou z pěti oblastí pak existují tzv. základní rovnice, díky kterým je možno dopočítat termodynamické veličiny jako měrnou entalpii, měrnou entropii, měrný objem, atd. Oblast 1 představuje pevnou látku, oblast 2 přehřátou páru, oblast 3 kapalinu, oblast 4 je saturační křivka a oblast 5 je plyn [33].

V následující podkapitole se zaměříme jakým způsobem se vlastnosti páry počítají a uvedeme si zmíněné základní rovnice.

3.2 Výpočet termodynamických veličin páry

Jedním z hlavních cílů IAPWS je poskytnout mezinárodně uznávané formulace vlastností páry, vody a vybraných vodných roztoků pro vědecké a průmyslové aplikace. V roce 1997 byla vydána poslední a dodnes platící formulace IAPWS-IF97 pro průmyslové aplikace. Formulace je platná v rozmezí

$$\begin{aligned} 273.15K \leq T \leq 1073.15K, \quad & p \leq 100MPa \\ 1073.15K < T \leq 2273.15K, \quad & p \leq 50MPa. \end{aligned}$$

Formulací máme na mysli základní rovnice pro jednotlivé oblasti. Veškeré termodynamické vlastnosti pak z těchto rovnic můžeme odvodit jejich kombinacemi nebo derivacemi. Na obrázku 15 si rovněž můžeme všimnout, že oblasti 1,2 a 5 popisuje fundamentální rovnice pro měrnou Gibbsovu energii - $g(p,T)$, zatímco oblast 3 je popsána fundamentální rovnicí pro měrnou Helmholtzovou energií $f(\rho,T)$.

Podívejme se na znění rovnic v jednotlivých oblastech [33]:

Oblast 1

Tuto oblast můžeme popsat rovnicí

$$\frac{g(p,T)}{RT} = \gamma(\pi,\tau) = \sum_{i=1}^{34} n_i (7.1-\pi)^{I_i} (\tau - 1.222)^{J_i}, \tag{85}$$

kde $\pi = \frac{p}{p*}$, $\tau = \frac{T}{T*}$, $p^* = 16.53MPa$, $T^* = 1386K$, $R = 0.462kJ\ kg^{-1}K^{-1}$. Koeficienty n_i a exponenty I_i a J_i jsou uvedeny v příloze B.

K následnému určení termodynamických veličin jako vnitřní energie, měrná entalpie či měrná entropie, je třeba tuto rovnici upravit dle vztahů v tabulce 1. Pro přehlednost uvádím jen vztahy pro tyto tři veličiny, zbytek vztahů je možno najít v [33]. U každé z dalších oblastí budou rovněž uvedeny jen tyto tři výpočty.

Tabulka 1 – Výpočet veličin pro oblast 1

Veličina	Vztah pro výpočet
u	$\frac{u(\pi,\tau)}{RT} = \tau\gamma_\tau - \pi\gamma_\pi$
h	$\frac{h(\pi,\tau)}{RT} = \tau\gamma_\tau$
s	$\frac{s(\pi,\tau)}{R} = \tau\gamma_\tau - \gamma$

$\gamma_\pi = \left[\frac{\partial\gamma}{\partial\pi}\right]_\tau, \gamma_\tau = \left[\frac{\partial\gamma}{\partial\tau}\right]_\pi$

Oblast 2

Oblast 2 je popsána rovnicí, která je rozdělena na dvě části, a to část pro ideální plyn γ^o a reziduální část γ^r, tedy

$$\frac{g(p,T)}{RT} = \gamma(\pi,\tau) = \gamma^o(\pi,\tau) + \gamma^r(\pi,\tau), \tag{86}$$

kde význam proměnných π, τ je stejný jako v rovnici pro oblast 1. Část pro ideální plyn je popsána rovnicí

$$\gamma^o = \ln\pi + \sum_{i=1}^{9} n_i^o \tau^{J_i^o}, \tag{87}$$

reziduální část je pak popsána rovnicí

$$\gamma^r = \sum_{i=1}^{43} n_i \pi^{I_i} (\tau - 0.5)^{J_i}, \tag{88}$$

kde $\pi = \frac{p}{p*}$, $\tau = \frac{T*}{T}$, $p^* = 1MPa$, $T^* = 540K$. Koefiecienty n_i^o, n_i a exponenty J_i^o, J_i jsou uvedeny v příloze B. Veškeré dostupné veličiny opět můžeme dopočítat pomocí následujících vztahů v tabulce 2.

Tabulka 2 – Výpočet veličin pro oblast 2

Veličina	Vztah pro výpočet
u	$\frac{u(\pi,\tau)}{RT} = \tau(\gamma_\tau^o + \gamma_\tau^r) - \pi(\gamma_\pi^o + \gamma_\pi^r)$
h	$\frac{h(\pi,\tau)}{RT} = \tau(\gamma_\tau^o + \gamma_\tau^r)$
s	$\frac{s(\pi,\tau)}{R} = \tau(\gamma_\tau^o + \gamma_\tau^r) - (\gamma^o + \gamma^r)$

$\gamma_\pi^r = \left[\frac{\partial\gamma^r}{\partial\pi}\right]_\tau, \gamma_\tau^r = \left[\frac{\partial\gamma^r}{\partial\tau}\right]_\pi, \gamma_\pi^o = \left[\frac{\partial\gamma^o}{\partial\pi}\right]_\tau, \gamma_\tau^o = \left[\frac{\partial\gamma^o}{\partial\tau}\right]_\pi$

Oblast 3
Tato oblast je popsána fundamentální rovnicí pro měrnou Helmholtzovu energii, tedy

$$\frac{f(\rho,T)}{RT} = \phi(\delta,\tau) = n_1 \ln\delta + \sum_{i=2}^{40} n_i \delta^{I_i} \tau^{J_i}, \tag{89}$$

kde $\delta = \frac{\rho}{\rho*}$, $\tau = \frac{T*}{T}$ s $\rho^* = \rho_c = 322kg\,m^{-3}$, $T^* = T_c = 647.096K$. Koefiecienty n_i a exponenty I_i, J_i jsou uvedeny v příloze B. Termodynamické veličiny určíme z tabulky 3.

Tabulka 3 – Výpočet veličin pro oblast 3

Veličina	Vztah pro výpočet
p	$\frac{p(\delta,\tau)}{\rho RT} = \delta\phi_\delta$
h	$\frac{h(\delta,\tau)}{RT} = \tau\phi_\tau + \delta\phi_d elta$
s	$\frac{s(\delta,\tau)}{R} = \tau\phi_\delta - \phi$

$$\phi_\delta = \left[\frac{\partial\phi}{\partial\delta}\right]_\tau, \phi_\tau = \left[\frac{\partial\phi}{\partial\tau}\right]_\delta$$

Oblast 4
Oblast 4 se vztahuje pouze na saturační křivku. Je popsána kvadratickou rovnicí, která může být přímo vyřešena pro saturační tlak p_s a saturační teplotu T_s

$$\beta^2\vartheta^2 + n_1\beta^2\vartheta + n_2\beta^2 + n_3\beta\vartheta^2 + n_4\beta\vartheta + n_5\beta + n_6\vartheta^2 + n_7\vartheta + n_8 = 0, \tag{90}$$

kde

$$\beta = \left(\frac{p_s}{p^*}\right)^{\frac{1}{4}}, \quad \vartheta = \frac{T_s}{T^*} + \frac{n_9}{\frac{T_s}{T^*} - n_{10}}, \tag{91}$$

pro $p^* = 1Mpa$ a $T^* = 1K$, koeficienty n_i je možno nalézt v příloze B. Řešení této rovnice pro saturační tlak je

$$\frac{p_s}{p^*} = \left[\frac{2C}{-B + (B^2 - 4AC)^{1/2}}\right]^4, \tag{92}$$

kde $p* = 1MPa$ a

$$A = \vartheta^2 + n_1\vartheta + n_2, \quad B = n_3\vartheta^2 + n_4\vartheta + n_5, \quad C = n_6\vartheta^2 + n_7\vartheta + n_8, \tag{93}$$

kde ϑ má stejný význam jako v předchozí rovnici, podobně jako koeficienty n_i. Uveďme ještě řešení pro saturační teplotu

$$\frac{T_s}{T^*} = \frac{n_{10} + D - \left[(n_{10}+D)^2 - 4(n_9 + n_{10}D)\right]^{1/2}}{2}, \tag{94}$$

kde $T^* = 1K$ a

$$D = \frac{2G}{-F - (F^2 - 4EG)^{1/2}}, \tag{95}$$

kde

$$E = \beta^2 + n_3 + n_6, \quad F = n_1\beta^2 + n_4\beta + n_7, \quad G = n_2\beta^2 + n_5\beta + n_8. \tag{96}$$

Oblast 5

Poslední oblast je popsána rovnicí, která je opět rozdělena na 2 části, stejně jako rovnice pro oblast 2, tedy

$$\frac{g(p,T)}{RT} = \gamma(\pi,\tau) = \gamma^o(\pi,\tau) + \gamma^r(\pi,\tau), \tag{97}$$

kde význam proměnných π, τ je stejný jako pro oblast 2. Liší se až rovnice pro konkrétní části, nejdříve rovnice pro ideální plyn

$$\gamma^o = \ln \pi + \sum_{i=1}^{6} n_i^o \tau^{J_i^o}, \tag{98}$$

a rovnice pro reziduální část

$$\gamma^r = \sum_{i=1}^{6} n_i \pi^{I_i} \tau^{J_i}, \tag{99}$$

kde $\pi = \frac{p}{p*}$, $\tau = \frac{T*}{T}$, $p^* = 1MPa$, $T^* = 1000K$. Koefiecienty n_i^o, n_i a exponenty I_i, J_i^o, J_i jsou uvedeny v příloze B. Veškeré dostupné veličiny opět můžeme dopočítat pomocí vztahů uvedených v tabulce 2 pro oblast 2.

4 Implementace

V této kapitole se budeme zabývat stavbou samotné neuronové sítě. Jak jsme si mohli všimnout, tak v kapitole 3 je uvedeno několik veličin, které se dají vypočítat z fundamentálních rovnic. Naším úkolem je doplnit výpočet měrného objemu pro jednotlivé oblasti právě pomocí neuronové sítě. Velkou pomocí by pro nás měla být data z již existujících rovnic pro ostatní veličiny. Celkovým výstupem poté bude spustitelná aplikace, která spočítá hodnotu měrného objemu vodní páry pro zadaný tlak a teplotu.

Před samotnou tvorbou sítě se budeme věnovat krátkému popisu programovacího jazyka Python a knihovně Tensorflow, pomocí nichž budeme modely tvořit. Dále ve stručnosti popíšeme termodynamickou knihovnu pro MATLAB - XSteam, která nám poslouží jako zdroj dat.

4.1 Python, Tensorflow, XSteam

Veškerý kód, pomocí něhož budeme budovat neuronové sítě, bude psán v Pythonu, konkrétně verzi 3.6. Motivace k výběru tohoto programovacího jazyka byla především široká podpora různých knihoven, které buď pomáhájí se strojovým učením anebo usnadňují obecně práci s daty. Jedná se zejména o knihovny NumPy, pandas, Keras, matplotlib, scikit-learn nebo Tensorflow. Poslední zmíněné knihovně se v následujícím odstavci budeme věnovat o něco podrobněji, neboť tvoří základní stavební kámen celého kódu. Krom podpory těchto knihoven je Python vhodnou volbou, protože nabízí jednoduchý a přehledný syntax, možnost objektově orientovaného programování či podporu všech větších operačních systémů.

Zaměřme se nyní na nejpodstatnější knihovnu, kterou budeme využívat, kterou je Tensorflow. Jedná se o open source knihovnu, která byla vyvinuta Google Brain týmem v rámci výzkumu strojového učení a hlubokých neuronových sítí. Dá se spustit téměř na čemkoliv - GPU, CPU - včetně mobilních a vestavěných platforem, TPU, což je specializované hardware pro tenzorovou matematiku. Tensorflow pracuje na principu vybudování výpočetního grafu a jeho následném spuštění. Tento graf představuje datovou strukturu, která plně popisuje výpočet, který se snažíme provést. Tyto grafy lze vizualizovat pomocí nástroje Tensorboard, což je sada webových aplikací sloužící k tomuto účelu [35].

Další knihovnu, kterou budeme využívat je XSteam. Jedná se o implementaci rovnic uvedených v dokumentu IAPWS IF97 v MATLABu. XSteam obsahuje dostupné výpočty pro termodynamické veličiny v závislosti na jednom či dvou jiných veličinách. Tato knihovna nám poslouží jako zdroj dat, pomocí kterých budeme neuronovou síť trénovat, a později nám bude sloužit pro ověření správnosti našich výsledků.

4.2 Tvorba modelu

Nyní se zaměříme na stavbu neouronové sítě. V následujících odstavcích podrobně okomentujeme výběr architektury sítě, jejích hyperparametrů a průběh učení.

Nabízejí se nám dvě možné cesty, kterými se můžeme vydat. Sestavit jednu velkou neuronovou síť pro všechny oblasti 1-5 (viz. kapitola 3) anebo sestavit několik menších sítí. Kdybychom se pokusili sestavit jednu neuronovou síť, tak můžeme mít problémy se spojitostí funce, kterou se snažíme aproximovat. Navíc velká neuronová síť nebude zcela tak přesná, jako několik menších sítí vytvořených pro jednotlivé oblasti.

Začneme s neuronovou sítí pro jednu libovolnou oblast, na které si ukážeme veškeré optimalizační postupy a celkovou stavbu modelu samotného. Prvně vybraná oblast nám poslouží jako ilustrační příklad. Jakmile budeme spokojeni s výsledkem této sítě, přesuneme se na tvorbu sítí pro další oblasti. Podrobný popis tvorby modelu bude rozebrán především pro první oblast, pro další oblasti se budeme snažit popis zestručnit nebo zcela omezit. Jak již bylo zmíněno, tak budeme modelovat závislost měrného objemu na tlaku a teplotě, $v = v(p, T)$.

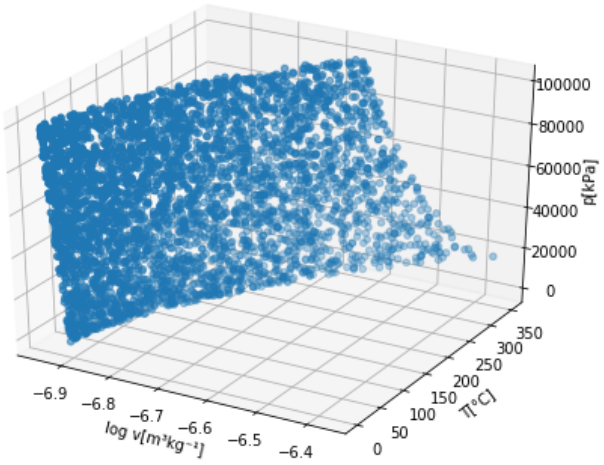
4.2.1 Příprava dat

Pro veškerou generaci dat, se kterou naše síť bude pracovat, budeme používat knihovnu XSteam. Pro jednoduchý export těchto dat budeme používat distribuci této knihovny upravené pro Microsoft Excel. Pro tvorbu našich dat budeme využívat implementované funkce **h_pT()** a **v_ph()**.

Začneme s oblastí číslo 1. Data budeme generovat v rozmezí

$$0°C \leq T \leq 350°C, \quad p_s(T) \leq p \leq 100MPa, \tag{100}$$

kde $p_s(T)$ značí hodnoty na saturační křivce. Na obrázku 17 je vykresleno pro představu 3500 různých bodů v p-v-T diagramu.



Obrázek 17 – Vygenerovaná data

Díky knihovně XSteam nejsme omezeni počtem dat, pokud budeme potřebovat další data, není problém si je vygenerovat. Předchozí obrázek slouží tedy pouze pro vizualizaci oblasti, kterou budeme aproximovat, nejedná se o veškerá dostupná data.

Nyní se můžeme zaměřit na transformaci dat. Jednu transformaci už jsme provedli a to konkrétně logaritmickou transformaci měrného objemu. Díky této transformaci získáme lepší povědomí o oblasti, kterou modelujeme. Bylo ukázáno, že podobné nelineární transformace nemají negativní účinky na učení, spíše naopak [10].

Nadále je potřeba provést standardizaci dat. Pro naši síť zvolíme obyčejnou standardizaci pro nulovou střední hodnotu a jednotkový rozptyl. Jak již bylo uvedeno v kapitole 2, budeme standardizovat vstupní veličiny, tlak a teplotu, zatímco měrný objem ponecháme beze změny. Od této úpravy dat si slibujeme zrychlené učení díky lepší inicializace parametrů sítě.

Dalším krokem je náhodné promíchaní všech dat. Snažíme se omezit předkládání příliš podobných vstupů za sebou v průběhu učení. Důvodem je opět zkvalitnění a zrychlení samotného učení.

Poslední úpravu, kterou musíme provést je rozdělení celé datové sady do třech menších specifických skupin. Jedná se o trénovací, validační a testovací skupiny dat. Každá z těchto skupin má svůj vlastní účel:

- Trénovací data - tato nejpočetnější skupina dat slouží k vytrénování neboli naučení neuronové sítě
- Validační data - výkon již naučené sítě na validační sadě dat nám slouží k úpravě hyperparametrů a celkové modifikace neuronové sítě
- Testovací data - slouží čistě ke zhodnocení celkového výkonu sítě, pomocí těchto dat neupravujeme žádné hyperparametry ani nezasahujeme do učícího algoritmu

Máme několik možností v jakém poměru naše data do těchto skupin rozdělit. Obecně záleží na počtu hyperparametrů v našem modelu. Obecně platí, že nejméně polovina, ideálně až 3/4 dat jsou vyhrazeny pro trénink a zbytek se rozdělí mezi validační a testovací sadu. Pro naše účely tento poměr zvolme například na 75% dat pro trénink, 15% pro validaci a zbylých 10% využijeme k testování.

Nyní už máme vše připraveno pro to, abychom mohli začít s konstrukcí samotné neuronové sítě a následně s jejím učením.

4.2.2 Neuronová síť

V této části se pokusíme sestavit samotnou neuronovou síť a budeme se snažit najít optimální hodnoty jejich hyperparametrů. Jak již bylo uvedeno v kapitole 2, neexistují žádné univerzální hodnoty těchto parametrů, neboť každá úloha si vyžaduje individuální přístup. V našem případě pro určení těchto hodnot využijeme převážně náhodného prohledávání a Bayesovské optimalizace.

Než se však pustíme do stavby sítě, musíme si nejdřív určit nějakou metriku, která bude vyhodnocovat výkon sestavené sítě. Častým zvykem je pozorovat ztrátu a přesnost neuronové sítě. Ztrátou rozumíme hodnotu ztrátové (chybové) funkce. Obecně platí, že čím nižší ztráta, tím kvalitnější model. V naší síti budeme využívat mean squared error funkci, tedy

$$E = \frac{1}{2n} \sum_{i=1}^{n} (\boldsymbol{y}^{(i)} - \hat{\boldsymbol{y}}^{(i)})^2, \tag{101}$$

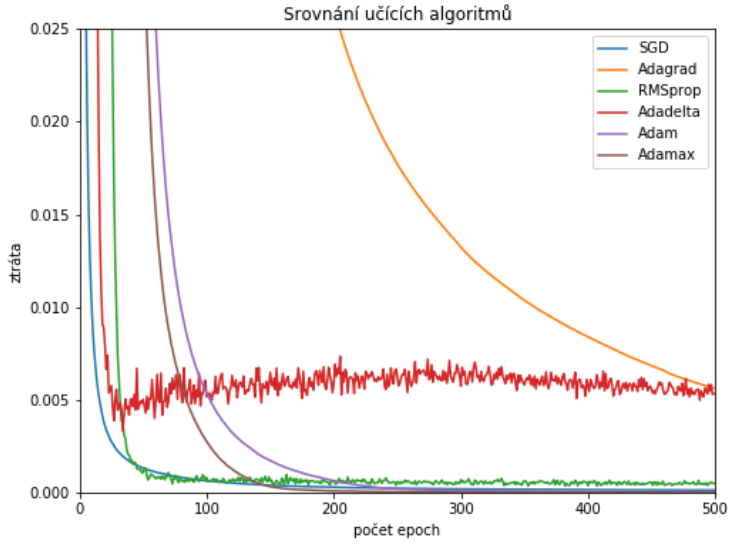
kde n je počet vstupních příkladů, $\boldsymbol{y}$ je vektor požadovaných výstupů a $\hat{\boldsymbol{y}}$ je vektor výstupů z sítě.

Přesnost nám poslouží jako pomocný údaj ke ztrátě. S metrikou pro přesnost je to trochu složitější, protože přesnost aproximace funkce není tak jednoduše změřitelná jako přesnost obyčejných klasifikačních úloh. Vhodnou volbou je například mean absolute percentage error, kterou drobně upravíme

$$A = \left[1 - \frac{1}{n}\sum_{i=1}^{n}\left|\frac{\boldsymbol{y}^{(i)} - \hat{\boldsymbol{y}}^{(i)}}{\boldsymbol{y}^{(i)}}\right|\right] \cdot 100\%. \tag{102}$$

Nyní se dostáváme ke konstrukci neuronové sítě. Nastiňme si nejdříve, jakým způsobem budeme postupovat. Budeme pracovat s několika hyperparametry, nicméně u některých se spokojíme s doporučovanými hodnotami a nebudeme je nadále optimalizovat. Ještě zmiňme množství dat, se kterým budou všechny následující výpočty probíhat. Konstrukci sítě zahájíme s celkovým množstvím 5000 různých datových bodů, čili 3750 pro trénink, 750 pro validaci a 500 pro testování.

Ze všeho nejdříve zvolíme algoritmus učení. Pravděpodobně nejlepší volbou bude jeden z modernějších adaptivních algoritmů. Pro definitivní výběr sestrojíme jednoduchou síť a budeme sledovat výkon této zkušební sítě, zatímco ponecháme ostatní hyperparametry beze změny. Graf závislosti validační ztráty na počtu epoch na obrázku 18 zobrazuje srovnání některých algoritmů uvedených v kapitole 2. Křivky vyjadřují průměr přes deset různých běhů pro každý algoritmus. Všechny algoritmy byly spuštěny s doporučenými hodnotami inicializace. Po 500 iteracích lze pozorovat, že algoritmy SGD a RMSprop předvádějí velice rychlou konvergenci. Jejich dosažené minimum po ukončení učení je však 2-3krát vyšší než u adaptivních algoritmů Adam a Adamax. Jelikož rozdíl v rychlosti konvergence není příliš výrazný, vybereme jeden z těchto dvou algoritmů. Vezměme například algoritmus Adam s hodnotami $\beta_1 = 0{,}9$, β_2=0,999 a $\epsilon = 10^{-8}$.



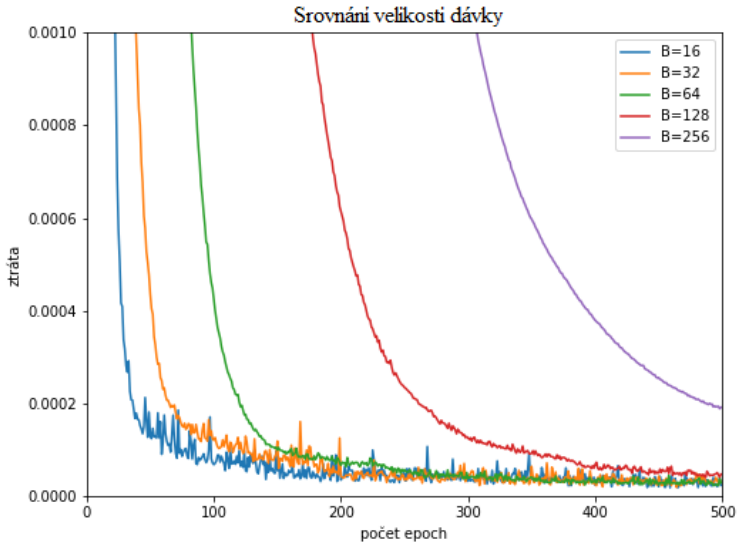
Obrázek 18 – Srovnání algoritmů

Volbou adaptivního algoritmu učení jsme si rovněž ušetřili práci s dalším hyperparametrem - rychlosti učení. Námi zvolený algoritmus totiž rychlost učení upravuje sám v každé iteraci, a proto stačí jen nastavit počáteční hodnotu tohoto hyperparametru.

Než se dostaneme k samotné architektuře sítě, podíváme se ještě na velikost dávky, neboť tento hyperparametr výrazně ovlivňuje dobu průběhu učení. Opět budeme zkoušet doporučené hodnoty uvedené v teoretické části.

Trénink sítě budeme tedy již provádět s algoritmem Adam. K výběru nejlepší možné velikosti dávky nám opět poslouží graf závislosti validační ztráty na počtu epoch. Obrázek 19 zachycuje srovnání učení sítě s různými velikostmi dávky. Pro omezení vlivu náhody je na grafu vykreslen průměr přes deset různých běhů pro každou hodnotu B. Vidíme, že rozdíly mezi jednotlivými velikostmi nejsou až tak markantní, skoro všechny pokusy skončily se ztrátou menší než 0.0001.

Další aspekt, který musíme brát v potaz při výběru velikosti dávky, je celková doba učení. Ačkoliv B=16 vykazuje nejlepší výsledky, doba výpočtu byla mnohonásobně vyšší než u ostatních hodnot, a proto tuto hodnotu nadále uvažovat nebudeme. Rovněž vyloučíme možnost B=256, neboť pro tuto hodnotu učení konverguje nejpomaleji. Mezi hodnotami B=32, B=64 a B=128 už máme volnější výběr. Nejlépe se jeví hodnota B=64, pokud vezmeme v úvahu její větší stabilitu ve srovnání s B=32 a rychlejší konvergenci ve srovnání s B=128. Nadále v práci budeme pracovat s velikostí dávky B=64.



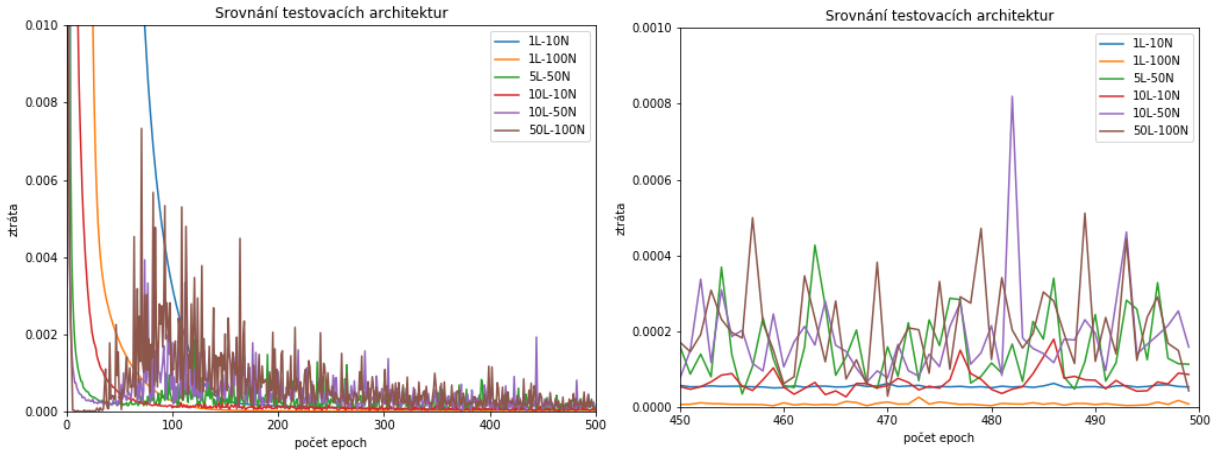
Obrázek 19 – Srovnání velikosti dávky

V následujících odstavcích se budeme věnovat samotné architektuře sítě, tedy budeme se snažit určit počet skrytých vrstvev a neuronů v nich obsažených. Zde však narazíme, protože neexistují doporučená nastavení jako u předchozích dvou hyperparametrů. K určení architektury budeme muset využít nějakou optimalizační metodu pro hledání hyperparametrů.

Ze všeho nejdříve se však vyplatí věnovat pár okamžiků obyčejné metodě pokus omyl. Rádi bychom pomocí této metody získali alespoň hrubou představu o tom, jak rozsáhlou síť vlastně budujeme. Zkusíme tedy sestavit několik zkušebních sítí s diametrálně odlišnými velikostmi vrstev a neuronů. Následně porovnáme jejich výkony.

Na obrázku 20 vidíme porovnání několika náhodných architektur neuronové sítě. Jedná se opět o grafy závislosti validační ztráty na počtu iterací. Všechny křivky byly vykresleny jako průměr přes deset různých běhů trénovacího algoritmu. Zaměřme se nejdřív na obrázek 20a. Z tohoto obrázku můžeme vyčíst především nevhodné kandidáty pro naši architekturu. Všimněme si, že křivky hnědé a fialové barvy vykazují známky overfittingu. Tyto dvě architektury jsou rovněž nekonzistentní, co se týče jejich výkonu. Z tohoto můžeme vyvodit, že pro naše účely nám postačí méně rozsáhlá síť. Neměli bychom potřebovat více než 10 skrytých vrstev.

Nyní se věnujme obrázku 20b. Jedná se o přiblížení grafu vlevo pro rozmezí 450 až 500 epoch. Lze pozorovat, že nejstabilnější výkony předvádí sítě reprezentované modrou a žlutou křivkou. Rovněž červená křivka vykazuje poněkud uspokojivé výsledky. Chceme-li dosáhnout co nejmenší validační ztráty, musíme zvolit síť o nízkém počtu skrytých vrstev a vysokém počtu skrytých neuronů anebo naopak.



(a) Výkony testovacích sítí

(b) Výkony testovacích sítí - přiblíženo

Obrázek 20 – Porovnání vybraných architektur

Manuální prohledávání nám tedy poskytlo přibližnou představu o architektuře, nicméně pro její konkrétnější podobu budeme muset využít jiných optimalizačních metod. Uvažujme pro začátek prostor architektur s maximálním počtem 10 skrytých vstev a maximálním počtem 100 skrytých neuronů. Zamysleme se nyní nad tím, co by znamenalo využití prohledávání na mřížce, konkrétněji prohledávání na mřížce s křížovou validací. Řekněme, že bychom rádi zmíněný prostor prohledali touto metodou. Pro 3-násobnou křížovou validaci se jedná celkem o 2673 různých vyhodnocení učení sítě. Doba výpočtu, řekněme pro 250 iterací v každém učení, zabere několik dnů na obyčejném 4-jádrovém procesoru. Vzhledem k tomu, že se nejedná o jedinou síť, kterou budeme budovat, je tento přístup nepraktický.

Jednodušším a efektivnějším přístupem bude například využití náhodného prohledávání. Použijeme tedy náhodné prohledávání s 3-násobnou křížovou validací. Algoritmus necháme pracovat po 200 iterací, což znamená 600 různých vyhodnocení učení sítě. Celková doba běhu prohledávacího algoritmu byla asi 20 hodin. Tabulka 4 ukazuje nejlepších 10 výsledků. Hodnoty jsou seřazeny od nejmenší hodnoty ztráty na testovacích datech po největší po 500 epochách učícího algoritmu. Pro křížovou validaci byly použity trénovací a validační data, tedy dohromady 4500 dat. Bavíme-li se o testovacích datech v rámci křížové validace, máme na mysli jinou skupinu dat než původně odložených 500 dat pro závěrečné testování.

Je zřejmé, že nejvíce preferována architektura je s jednou skrytou vrstvou a vysokým počtem skrytých neuronů. Rozdíly mezi všemi 10 hodnotami, co se týče průměrné ztráty na testovacích datech, jsou téměř zanedbatelné. Za zmínku ovšem stojí rozdílné výpočetní doby celého učení. Je logické, že s rostoucí velikostí sítě roste i doba potřebná k jejímu naučení. Nicméně, můžeme si všimnout, že tomu tak vždy není. Dalším faktorem je rozdělení dat na trénovací a testovací skupiny v rámci křížové validace. Proto lze například pozorovat skoro 4-krát kratší dobu učení u sítě s 92 neurony než u sítě s 67 neurony.

Tabulka 4 – Výsledky náhodného prohledávání

Pořadí	Prům. doba výpočtu (v sek.)	Prům. ztráta test. dat	Počet skryt. vrstev	Počet skryt. neuronů
1	272.59	$3.46 \cdot 10^{-6}$	1	99
2	119.36	$5.69 \cdot 10^{-6}$	1	65
3	269.08	$6.89 \cdot 10^{-6}$	1	67
4	182.80	$6.98 \cdot 10^{-6}$	1	71
5	32.73	$6.99 \cdot 10^{-6}$	1	39
6	68.39	$7.08 \cdot 10^{-6}$	1	92
7	92.14	$7.32 \cdot 10^{-6}$	1	80
8	210.64	$7.48 \cdot 10^{-6}$	1	90
9	195.98	$7.51 \cdot 10^{-6}$	1	70
10	60.64	$9.84 \cdot 10^{-6}$	2	26

Náhodné prohledávání nám tedy poskytlo velmi slušné kandidáty pro konečnou architekturu sítě, ale stále věnovat skoro celý den k nalezení optimální architektury se nám může zdát jako zbytečně dlouhá doba. Proto se v následujících odstavcích budeme věnovat ještě Bayesovské optimalizaci.

Bayesovský algoritmus jsme nastavili na stejné hodnoty jako náhodné prohledávání, tedy 200 iterací algoritmu, 3-násobnou křížovou validaci, 500 epoch učení a prostor s maximálně 10 vrstvami a 100 neurony. V tomto případě jsme však ani zdaleka 200 iterací nepotřebovali. Ukázalo se, že k najití optimální hodnoty nám stačilo pouhých 12 iterací. Výsledky jsou zachyceny v tabulce 5. Pořadí v této tabulce ukazuje prohledávané možnosti, jak šly za sebou. Vidíme, že nám v podstatě vyšla stejná architektura, jako u náhodného prohledávání. Avšak, rozdíl je v celkové době najití tohoto optima. Zatímco v předchozím případě jsme museli čekat zhruba 20 hodin, nyní jsme ke stejnému výsledku došli zhruba za 30 minut. Ukazuje se tedy, že se nám vyplatí nadále pracovat pouze s Bayesovskou optimalizací hyperparametrů.

U obou algoritmů si lze všimnout, že nejlepší výsledky poskytovaly sítě s malým počtem skrytých vrstev a vysokým počtem skrytých neuronů. Rozdílné hodnoty ztrát u skoro stejných architektur si můžeme vysvětlit různým rozdělením dat při křížové validaci. Hlavním bodem je však optimální architektura, tedy architektura s absolutně nejnižší ztrátou. Očividně nejlepší volbou je síť s jednou skrytou vrstvou a 100 skrytými neurony.

Nabízí se otázka, proč nezvolit více než 100 skrytých neuronů. Dosáhli jsme velmi nízké hodnoty ztráty v řádu 10^{-6}, je velice nepravděpodobné, že bychom dokázali tuto hodnotu snížit o další řád pouze volbou architektury. Jednalo by se spíše o marginální zlepšení, proto budeme nadále pracovat s touto konkrétní architekturou.

Tabulka 5 – Výsledky Bayesovské optimalizace

Pořadí	Prům. doba výpočtu (v sek.)	Prům. ztráta test. dat	Počet skryt. vrstev	Počet skryt. neuronů
1	46.16	$5.93{\cdot}10^{-4}$	5	67
2	63.52	$8.97{\cdot}10^{-4}$	9	51
3	46.90	$1.77{\cdot}10^{-4}$	5	54
4	43.75	$2.61{\cdot}10^{-4}$	3	83
5	33.93	$3.26{\cdot}10^{-5}$	3	28
6	37.40	$1.50{\cdot}10^{-4}$	4	24
7	34.25	$3.27{\cdot}10^{-5}$	3	12
8	51.77	$6.33{\cdot}10^{-5}$	5	51
9	57.86	$5.61{\cdot}10^{-5}$	7	34
10	81.99	$1.82{\cdot}10^{-5}$	9	58
11	132.26	$1.88{\cdot}10^{-4}$	10	100
12	43.50	$3.44{\cdot}10^{-6}$	1	100

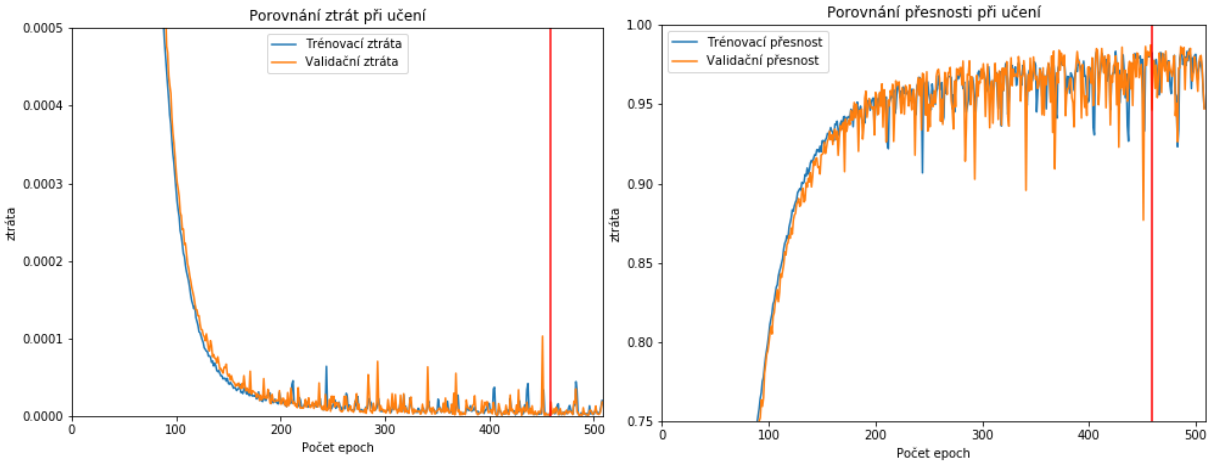
Momentálně už máme skoro vše připraveno k tomu abychom mohli naši síť plnohodnotně vytrénovat. Poslední hyperparametr, o jehož zapojení se musíme rozhodnout je rozpad vah, tedy obecněji o zapojení nějakých regularizačních technik. Avšak v průběhu učení naší neuronové sítě jsme se nesetkali s overfittingem. Pro námi zvolenou architekturu je skoro nemožné overfittingu dosáhnout, neboť není dostatečně komplexní. Pro tuto konkrétní síť není zapotřebí využít rozpadu vah, a proto jej nebudeme aplikovat.

Posledním krokem je učení sítě. S učením také souvisí poslední hyperparametr, který musíme upravit, a to počet tréninkových iterací. Zde využijeme metodu dřívějšího zastavení. Prvotně nastavíme například 5000 tréninkových epoch, avšak dodáme ještě parametry pro dřívější zastavení. Zadejme toto kritérium - zastavíme učení, pokud se každých 50 iterací validační ztráta nesníží alespoň o 10^{-8}.

Na obrázku 21 vidíme průběh tréninkové a validační ztráty a přesnosti. Pokus jsme nechali běžet 10-krát a vybrali nejlepší možný průběh. Červená křivka značí iteraci, ve které jsme učení zastavili, v našem případě to bylo v iteraci číslo 459. Lze pozorovat, že učení pak probíhalo dalších 50 iterací, avšak během této doby nedošlo k požadovanému zlepšení a tak algoritmus skončil. Podívejme se na jednotlivé grafy detailněji.

V grafu 21a vidíme očekáváný klesající průběh obou křivek. Obě ztráty dosahují na konci učení hodnot v řádu 10^{-6}, konkrétněji trénovací ztráta dosáhla hodnoty $3.03 \cdot 10^{-6}$ a validační ztráta hodnoty $1.20 \cdot 10^{-6}$. Když se podíváme na vedlejší graf, tak vidíme poněkud divokou oscilaci obou metrik okolo hodnoty 95%. Na konci učení byla tréninková přesnost rovna 98.04% a validační 98.74%. To jsou ovšem data v průběhu učení, nás spíše zajímají konečné přesnosti po naučení sítě pro jednotlivé sady. Pro tréninkovou sadu vyšla průměrná přesnost 99.92% a pro validační sadu 99.91%.

Tímto je konstrukce neuronové sítě pro oblast 1 zcela kompletní. Máme nyní k dispozici vytrénovanou síť, na kterou můžeme aplikovat dosud neviděná data. Naše síť by měla být schopná s poměrně velkou přesností vrátit požadované hodnoty. Naším dalším úkolem je vytvořit ještě sítě pro zbylé oblasti, jak bylo zmíněno na začátku této kapitoly. Nebudeme celý proces již podrobně rozepisovat. Celkové shrnutí je k dispozici v tabulce 6.



(a) Ztráty při učení

(b) Přesnosti při učení

Obrázek 21 – Průběh učení neuronové sítě

K níže uvedenému shrnutí ještě doplňme, že všechny sítě měly k dispozice stejné množství dat a poměr tréninkové:validační:testovací data je rovněž pro všechny stejný. Pro všechny níže uvedené sítě jsme použili ReLU aktivační funkci ve skrytých vrstvách. Druhý a třetí sloupec popisuje rozmezí hodnot, pro které je daná síť sestavena. Hodnota p_s značí saturační tlak při dané teplotě a hodnota t_{23} popisuje body na křivce mezi oblastmi 2 a 3. Pro oblast 4 není možné sestavit neuronovou síť pouze na základě znalosti objemu a tlaku.

Tabulka 6 – Shrnutí pro jednotlivé oblasti

Oblast	Tlak[MPa]	Teplota[°C]	Alg.učení	Velikost dávky	Počet skryt. vrstev	Počet skryt. neuronů
1	$(p_s, 100\rangle$	$(0,350\rangle$	Adam	64	1	100
2a	$\langle 16.529,100)$	$(350,800)$	Adam	32	3	100
2b	$\langle 2.5, 16.529)$	$(350,800)$	Adam	32	2	100
2c	$\langle 0.01,2.5)$	$(350,800)$	Adam	32	3	100
2d	$\langle 1, 16.529)$	$(180,350)$	Adam	32	2	75
2e	$\langle 0.01,1)$	$(180,350)$	Adam	32	2	100
3a	$\langle 40,100)$	$(350,t_{23})$	Adamax	32	2	50
3b	$\langle 25,40)$	$(350,t_{23})$	Adamax	16	3	50
3c	$\langle 16.259,25)$	$(350,t_{23})$	Adamax	32	5	50
4	-	-	-	-	-	-
5a	$\langle 10,50)$	$(800,2000)$	Adam	64	2	100
5b	$\langle 0.01,10)$	$(800,2000)$	Adamax	16	2	50

5 Výsledky

V této kapitole budeme sledovat výkony námi sestavených sítí. Pro všechny oblasti jsme si vyhradili speciální skupiny testovacích dat, které jsme doposud síti nepředložili. Můžeme tedy pomocí nich zhodnotit, jak dobrý výkon naše sítě předvádějí. Celkovým výsledkem práce poté bude spustitelná aplikace, která těchto sítí bude využívat a pomocí nich počítat měrný objem na základě předloženého tlaku a teploty.

K sestavení konečné aplikace tedy budeme potřebovat vytvořené modely pro jednotlivé oblasti. Stejně tak si musíme uložit střední hodnoty a rozptyly trénovacích dat. Tyto hodnoty jsou důležité, protože jsou nezbytné pro standardizaci testovacích dat. Při spojení všech sítí dohromady budeme muset mít na paměti, že data musíme standardizovat podle příslušných oblastí.

Výsledky jsou prezentovány v níže uvedených tabulkách. Pro každou oblast bylo vybráno 10 náhodných bodů, u kterých jsme uvedli konkrétní hodnoty měrného objemu. Dále jsme zahrnuli stručné shrnutí na konci každé tabulky pro celkovou sadu 500 testovacích dat. Toto shrnutí obsahuje některé popisné statistické údaje.

Oblast 1

Tabulka 7 – Výsledky pro oblast 1

Tlak[kPa]	Teplota[°C]	Měrný objem[m^3kg^{-1}] (software)	Měrný objem[m^3kg^{-1}] (neuronová síť)	Přesnost (MAPE)
9060.45	283.67	0.00133761	0.00133677	99.937%
2030.21	116.83	0.00105647	0.00105467	99.830%
58878.11	201.47	0.00111027	0.00110942	99.923%
5562.50	189.53	0.00113682	0.00113552	99.885%
11087.55	185.81	0.00112684	0.00112517	99.852%
8552.72	82.24	0.00102654	0.00102597	99.945%
14492.79	203.76	0.00114958	0.00114914	99.961%
41491.60	212.90	0.00113853	0.00113887	99.970%
8358.39	52.57	0.00100967	0.00100984	99.983%
42470.35	249.35	0.00119454	0.00119264	99.841%
500 různých testovacích dat			průměr	99.918%
			minimum	99.517%
			maximum	99.999%
			směrodatná odchylka	0.073%

Oblast 2

Tabulka 8 – Výsledky pro oblast 2

Tlak[kPa]	Teplota[°C]	Měrný objem[m^3kg^{-1}] (software)	Měrný objem[m^3kg^{-1}] (neuronová síť)	Přesnost (MAPE)
77830.91	552.67	0.00287858	0.00286748	99.614%
92158.64	737.89	0.0042024	0.00420166	99.982%
40910.06	712.17	0.00988415	0.00979553	99.103%
44998.88	606.07	0.00709861	0.00706776	99.565%
21447.46	435.65	0.01084789	0.01076501	99.236%
5021.20	516.13	0.06993691	0.0697668	99.757%
2965.95	630.36	0.13890883	0.13895543	99.966%
5374.99	453.66	0.05902437	0.05900836	99.973%
4878.98	466.62	0.06678904	0.06665596	99.801%
6900.03	372.41	0.03807707	0.03805941	99.954%
500 různých testovacích dat - oblast 2a			průměr	99.668%
			minimum	97.631%
			maximum	99.999%
			směrodatná odchylka	0.298%
500 různých testovacích dat - oblast 2b			průměr	99.479%
			minimum	92.083%
			maximum	99.998%
			směrodatná odchylka	0.703%
500 různých testovacích dat - oblast 2c			průměr	99.853%
			minimum	99.174%
			maximum	99.999%
			směrodatná odchylka	0.123%
500 různých testovacích dat - oblast 2d			průměr	99.786%
			minimum	98.601%
			maximum	99.999%
			směrodatná odchylka	0.188%
500 různých testovacích dat - oblast 2e			průměr	99.833%
			minimum	99.091%
			maximum	99.999%
			směrodatná odchylka	0.149%

Oblast 3

Tabulka 9 – Výsledky pro oblast 3

Tlak[kPa]	Teplota[°C]	Měrný objem[m^3kg^{-1}] (software)	Měrný objem[m^3kg^{-1}] (neuronová síť)	Přesnost (MAPE)
82310.02	485.75	0.00201351	0.002013	99.975%
87489.13	407.29	0.00151264	0.00151157	99.929%
64593.20	464.85	0.00215552	0.00215197	99.835%
47082.68	440.48	0.0024375	0.00244213	99.810%
72994.49	520.59	0.00263985	0.00264227	99.908%
36707.11	390.75	0.00185877	0.00186631	99.594%
39045.97	427.75	0.00277089	0.0027826	99.577%
35771.46	362.08	0.00158669	0.00158369	99.811%
37732.24	385.24	0.00176673	0.00176956	99.840%
36286.38	404.79	0.00216121	0.00216718	99.724%
500 různých testovacích dat - oblast 3a			průměr	99.835%
			minimum	99.141%
			maximum	99.999%
			směrodatná odchylka	0.129%
500 různých testovacích dat - oblast 3b			průměr	99.789%
			minimum	96.813%
			maximum	99.999%
			směrodatná odchylka	0.211%
500 různých testovacích dat - oblast 3c			průměr	99.007%
			minimum	76.967%
			maximum	99.996%
			směrodatná odchylka	2.181%

Oblast 5

Tabulka 10 – Výsledky pro oblast 5

Tlak[kPa]	Teplota[°C]	Měrný objem[m^3kg^{-1}] (software)	Měrný objem[m^3kg^{-1}] (neuronová síť)	Přesnost (MAPE)
46746.16	1637.29	0.01919820	0.01922638	99.853%
15479.85	1952.99	0.06682303	0.06677491	99.928%
19737.92	1489.15	0.04143572	0.04125315	99.559%
49247.22	1741.09	0.01926413	0.01924169	99.884%
22450.53	1610.02	0.03901937	0.03901334	99.985%
9405.89	885.35	0.05620718	0.05590836	99.468%
412.24	1205.23	1.65507655	1.6434947	99.300%
976.79	1143.61	0.66927342	0.6671294	99.680%
598.46	1875.99	1.65781179	1.6502323	99.543%
7570.75	1571.82	0.11274723	0.11237288	99.668%
500 různých testovacích dat - oblast 5a			průměr	99.875%
			minimum	99.118%
			maximum	99.999%
			směrodatná odchylka	0.114%
500 různých testovacích dat - oblast 5b			průměr	99.650%
			minimum	96.078%
			maximum	99.998%
			směrodatná odchylka	0.452%

6 Závěr

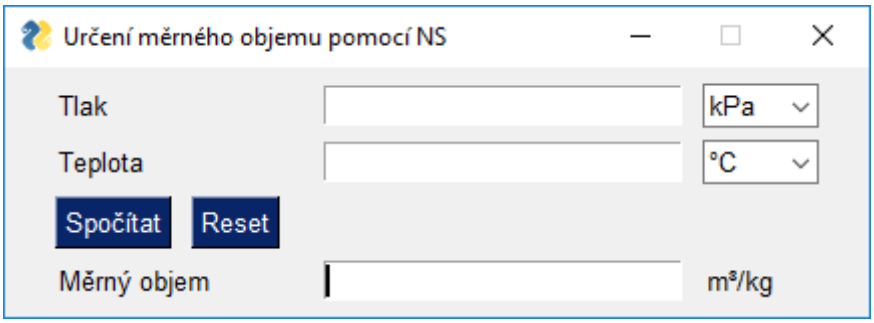
Tato diplomová práce se zabývala tvorbou neuronové sítě pro výpočet měrného objemu vodní páry. Neuronové sítě a jejich vlastnosti jsme rozebrali důkladně v kapitole 2. Zabývali jsme se typem učení, jejich strukturou a hyperparametry. Těchto teoretických poznatků jsme později využili v implementační fázi.

Následující 3. kapitola popisovala vodní páru z termodynamického hlediska. Uvedli jsme její základní vlastnosti a některé její veličiny, které se dají vypočítat pomocí fundamentálních rovnic.

Ve 4. kapitole jsme popsali tvorbu neuronové sítě pro jednu konkrétní oblast. Tento postup jsme zopakovali i pro další oblasti a celkem vytvořili jedenáct různých neuronových sítí pro specifické oblasti v prostoru stavových veličin p, v, T. Zmíněné rozdělení jsme provedli kvůli naplnění předpokladů pro tvorbu neuronových sítí a také kvůli celkovým přesnějším výsledkům.

V 5. kapitole byly stručně shrnuty výsledky v tabulkách pro všechny námi vytvořené oblasti.

Konečným výstupem této diplomové práce je spustitelná aplikace, která vypočítá měrný objem vodní páry pro zadaný tlak a teplotu pomocí neuronových sítí. Aplikace se skládá z několika neuronových sítí, které jsou spojeny v jeden funkční celek. Její uživatelské rozhraní je zobrazeno na obrázku 22. Aplikace nabízí volbu jednotek tlaku v kPa, MPa nebo barech. Teplotu je možno zadat ve stupních Celsia nebo Kelvinech. Po zadání obou hodnot stačí stisknout „Spočítat" a bude vypočtena hodnota pro měrný objem. Stisknutím tlačítka „Reset" dojde k vymazání všech hodnot.



Obrázek 22 – Aplikace pro výpočet měrného objemu

Aplikace je schopna počítat hodnoty měrného objemu v rozmezí platnosti formulace IAPWS-IF97 pro průmyslové aplikace, tedy

$$273.15K \leq T \leq 1073.15K, \quad p \leq 100MPa$$
$$1073.15K < T \leq 2273.15K, \quad p \leq 50MPa.$$

Přesnost výpočtů záleží na zvolené oblasti. Obecně můžeme očekávat průměrnou přesnost uvedenou v kapitole 5, což u všech oblastí byla hodnota přesahující 99% dle námi zvoleného kritéria přesnosti.

Lze konstatovat, že neuronové sítě dokázaly dostatečně dobře aproximovat vybraný prostor stavových veličin, o čem vypovídají získané hodnoty přesnosti. Je třeba však být na pozoru, neboť pro některé hodnoty naše aplikace nebude dosahovat tak vysoké přesnosti. Přestože formulace IAPWS-IF97 je platná pro všechny hodnoty tlaku pod 100MPa, tak naše aplikace u hodnot tlaku pod 0.01MPa bude méně přesná než pro vyšší hodnoty. Tato skutečnost je zapříčiněna nedokonalou aproximací pro tyto velmi nízké hodnoty. Pro spolehlivé výsledky bychom doporučili zadávat tlak vyšší než 0.1MPa.

Dalším nedostatkem je okolí bodu pro $p = 16.529MPa$ a $t = 350°C$. V tomto bodě se nám střetnou hned čtyři neuronové sítě, a jelikož je u všech sítí okrajovou hodnotou, tak přesnost aproximace v okolí tohoto bodu může být opět nižší. Je to cena, kterou platíme za rozdělení prostoru stavových veličin do tolika různých sítí. Avšak při menším dělení by průměrná přesnost jednotlivých oblastí byla nepochybně nižší.

Seznam použité literatury

[1] HAYKIN, Simon S. *Neural networks: a comprehensive foundation.* 2nd ed. Upper Saddle River, N.J.: Prentice Hall, 1999. ISBN 01-327-3350-1.

[2] HERTZ, John, Anders KROGH a Richard G. PALMER. *Introduction to the theory of neural computation: a comprehensive foundation*. 2nd ed. Redwood City, Calif.: Addison-Wesley Pub. Co., 1991. ISBN 02-015-1560-1.

[3] NIELSEN, Michael. Neural Networks and Deep Learning [online]. 2018 [cit. 2018-09-29]. Dostupné z: http://neuralnetworksanddeeplearning.com/index.html

[4] GOODFELLOW, Ian, Yoshua BENGIO a Aaron COURVILLE. *Deep learning*. Cambridge, Massachusetts: The MIT Press, 2016. ISBN 978-026-2035-613.

[5] CYBENKO, George. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems* [online]. 1989, **2**(4), 303-314 [cit. 2019-03-12]. Dostupné z: https://pdfs.semanticscholar.org/05ce/b32839c26c8d2cb38d5529cf7720a68c3fab.pdf

[6] MAGOULAS, George D. a Michael N. VRAHATIS. Adaptive algorithms for neural network supervised learning: A deterministic optimization approach. International Journal of Bifurcation and Chaos [online]. 2006, 16(07), 1929-1950 [cit. 2019-03-12]. DOI: 10.1142/S0218127406015805. ISSN 0218-1274. Dostupné z: http://www.worldscientific.com/doi/abs/10.1142/S0218127406015805

[7] ZOCCA, Valentino, Gianmario SPACAGNA, Daniel SLATER a Peter ROELANTS. *Python Deep Learning*. 1. Birmingham B3 2PB, UK.: Packt Publishing, 2017. ISBN 978-1-78646-445-3.

[8] SMITH, Leslie N. *A disciplined approach to neural network hyper-parameters: Part 1 - Learning rate, batch size, momentum and weight decay* [online]. 2018, , 1-21 [cit. 2018-11-19]. Dostupné z: https://arxiv.org/abs/1803.09820

[9] ORR, Genevieve a Klaus-Robert MULLER. *Neural networks: tricks of the trade*. New York: Springer, 1998. ISBN 35-406-5311-2.

[10] SARLE, Warren S. *AI Neural nets FAQ* [online]. USA, 2002 [cit. 2019-03-12]. Dostupné z: http://www.faqs.org/faqs/ai-faq/neural-nets/part2/

[11] RUDER, Sebastian. *An overview of gradient descent optimization algorithms* [online]. , 1-10 [cit. 2018-09-12]. Dostupné z: https://arxiv.org/pdf/1609.04747.pdf

[12] ZEILER, Matthew D. *Adadelta: An adaptive learning rate method* [online]. , 1- [cit. 2018-09-17]. Dostupné z: https://arxiv.org/pdf/1212.5701.pdf

[13] KINGMA, Diederik P. a Jimmy Lei BA. *Adam: A method for stochastic optimization* [online]. , 1-6 [cit. 2018-09-15]. Dostupné z: https://arxiv.org/pdf/1412.6980.pdf

[14] SARLE, Warren S. *Ill-Conditioning in Neural Networks* [online]. Cary, NC, USA, 1999 [cit. 2018-10-13]. Dostupné z: ftp://ftp.sas.com/pub/neural/illcond/illcond.html

[15] CHOROMANSKA, Anna, Mikael HENAFF, Michael MATHIEU, Gérard Ben AROUS a Yann LECUN. *The Loss Surfaces of Multilayer Networks* [online]. San Diego, CA, USA, 2015, , 1-13 [cit. 2018-11-01]. Dostupné z: http://proceedings.mlr.press/v38/choromanska15.pdf

[16] BENGIO, Yoshua. *Practical Recommendations for Gradient-Based Training of Deep Architectures* [online]. 2012, , 1-19 [cit. 2018-10-03]. Dostupné z: https://arxiv.org/pdf/1206.5533.pdf

[17] FARHADI, Farnoush. *Learning activation functions in deep neural networks* [online]. Montréal, 2017 [cit. 2018-06-02]. Dostupné z: https://publications.polymtl.ca/2945/1/2017_FarnoushFarhadi.pdf. Disertace. Université de Montréal.

[18] KARPATHY, Andrej. *CS231n: Convolutional Neural Networks for Visual Recognition* [online]. Stanford University [cit. 2018-05-16]. Dostupné z: http://cs231n.github.io/neural-networks-1/

[19] HEATON, Jeff. I*ntroduction to neural networks with Java*. 2nd ed. St. Louis: Heaton Research, 2008. ISBN 978-160-4390-087.

[20] HANIN, Boris. *Universal function approximation by deep neural nets with bounded width and ReLU activation* [online]. , 1-4 [cit. 2018-09-25]. Dostupné z: https://arxiv.org/pdf/1708.02691.pdf

[21] THOMAS, Alan J., Miltos PETRIDIS, Simon D. WALTERS, Saeed Malekshahi GHEYTASSI a Robert E. MORGAN. Two Hidden Layers are Usually Better than One. *Engineering Applications of Neural Networks* [online]. Cham: Springer International Publishing, 2017, 2017-08-02, , 279-290 [cit. 2019-03-13]. Communications in Computer and Information Science. DOI: 10.1007/978-3-319-65172-9_24. ISBN 978-3-319-65171-2. Dostupné z: http://link.springer.com/10.1007/978-3-319-65172-9_24

[22] LOSHCHILOV, Ilya a Frank HUTTER. *Decoupled weight decay regularization* [online]. Freiburg, Germany, 2019, , 1-3 [cit. 2018-11-25]. Dostupné z: https://arxiv.org/pdf/1711.05101.pdf

[23] Artificial Neural Networks. Imperial College C395 Machine Learning - Neural Networks [online]. 2019 [cit. 2019-01-18]. Dostupné z: https://www.doc.ic.ac.uk/ nuric/teaching/imperial-college-c395-machine-learning-neural-networks.html

[24] JORDAN, Jeremy. Setting the learning rate of your neural network. *Jeremy Jordan* [online]. 2018 [cit. 2018-12-05]. Dostupné z: https://www.jeremyjordan.me/nn-learning-rate/

[25] BROWNLEE, Jason. How to Configure the Learning Rate Hyperparameter When Training Deep Learning Neural Networks. *Machine learning mastery* [online]. 2019 [cit. 2018-11-08]. Dostupné z: https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/

[26] BROWNLEE, Jason. A Gentle Introduction to Mini-Batch Gradient Descent and How to Configure Batch Size. *Machine learning mastery* [online]. 2017 [cit. 2018-12-14]. Dostupné z: https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/

[27] CHANGHAU, Isaac. *Loss Functions in Neural Networks* [online]. 2017 [cit. 2018-07-19]. Dostupné z: https://isaacchanghau.github.io/post/loss_functions/

[28] KRAUSE, Andreas. *Advanced Topics in Machine Learning: Nonparametric learning and Gaussian processes* [online]. 2010 [cit. 2018-12-21]. Dostupné z: http://courses.cms.caltech.edu/cs253/slides/cs253-14-GPs.pdf

[29] BERGSTRA, James a Joshua BENGIO. Random search for hyper-parameter optimization. *The Journal of Machine Learning Research* [online]. 2012, 13(1), 1-10 [cit. 2018-12-13]. Dostupné z: http://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf

[30] SNOEK, Jasper, Hugo LAROCHELLE a Ryan P. ADAMS. *Practical Bayesian Optimization of Machine Learning Algorithms* [online]. 2012, , 1-12 [cit. 2019-01-10]. Dostupné z: https://arxiv.org/pdf/1206.2944.pdf

[31] GARNETT, Roman. *Bayesian Methods in Machine Learning: lecture notes [online]. 2019* [cit. 2019-03-13]. Dostupné z: https://www.cse.wustl.edu/ garnett/cse515t/spring_2019/files/lecture_notes/12.pdf

[32] PAVELEK, Milan. *Termomechanika*. Brno: Akademické nakladatelství CERM, 2011. ISBN 978-80-214-4300-6.

[33] IAPWS R7-97: *Revised release on the IAPWS industrial formulation 1997 for the thermodynamic properties of water and steam*. 2007 [cit. 2018-09-03]. Dostupné z: http://www.iapws.org

[34] KOVAŘÍK, Petr. Termomechanika [online]. [cit. 2019-03-13]. Dostupné z: http://home.zcu.cz/ kovarikp/termomechanika.html

[35] UNRUH, Amy. What is the TensorFlow machine intelligence platform?. *Opensource.com* [online]. 2017 [cit. 2018-11-25]. Dostupné z: https://opensource.com/article/17/11/intro-tensorflow

Seznam použitých zkratek a symbolů

$\mathbf{w}$	vektor synaptických vah
$\mathbf{b}$	vektor vychýlení
φ	aktivační funkce
z	aktivace neuronu
$\boldsymbol{\theta}$	vektor parametrů neuronové sítě
C	kovariance
η	rychlost učení
E	chybová funkce
$\boldsymbol{H}$	Hessova matice
B	velikost dávky
p	tlak [kPa]
p_s	saturační tlak [kPa]
v	měrný objem$[m^3 kg^{-1}]$
T	termodynamická teplota[K]
R	plynová konstanta $[J\ kg^{-1} \cdot K^{-1}]$
s	měrná entropie$[J\ kg^{-1} K^{-1}]$
h	měrná entalpie$[J\ kg^{-1}]$
u	měrná vnitřní energie$[J\ kg^{-1}]$

Seznam obrázků

1 Model neuronu 6
2 Porovnání mělké a hluboké neuronové sítě 7
3 Srovnání trénovací a testovací chyby 11
4 Ukázka overfittingu 12
5 Ukázka underfittingu 12
6 Porovnání SGD s momentem a bez momentu 17
7 Porovnání momentových algoritmů 18
8 Porovnání optimalizačních algoritmů 22
9 Četnost ztrát v závislosti na počtu skrytých vrstev 25
10 Korelace mezi testovací a trénovací chybou 25
11 Testovací chyba pro různé hodnoty rozpadu vah - WD 30
12 Rychlost učení 31
13 Rychlost učení 31
14 Porovnání náhodného hledání s hledáním na mřížce 36
15 $p - T$ diagram 38
16 $T - s$ diagram 38
17 Vygenerovaná data 44
18 Srovnání algoritmů 46
19 Srovnání velikosti dávky 47
20 Porovnání vybraných architektur 48
21 Průběh učení neuronové sítě 51
22 Aplikace pro výpočet měrného objemu 57

Seznam tabulek

1 Výpočet veličin pro oblast 1 39
2 Výpočet veličin pro oblast 2 39
3 Výpočet veličin pro oblast 3 40
4 Výsledky náhodného prohledávání 49
5 Výsledky Bayesovské optimalizace 50
6 Shrnutí pro jednotlivé oblasti 51
7 Výsledky pro oblast 1 53
8 Výsledky pro oblast 2 54
9 Výsledky pro oblast 3 55
10 Výsledky pro oblast 5 56
11 Hodnoty pro oblast 1 71
12 Hodnoty pro oblast 2 - část pro ideální plyn 72
13 Hodnoty pro oblast 2 - reziduální část 72
14 Hodnoty pro oblast 3 73
15 Hodnoty pro oblast 4 73
16 Hodnoty pro oblast 5 - část pro ideální plyn 74
17 Hodnoty pro oblast 5 - reziduální část 74

A Hessova matice

V této příloze si připomeneme pojmy Jacobiho a Hessovy matice. Mějme vektorovou funkci $\boldsymbol{f} : \mathbb{R}^m \to \mathbb{R}^n$, pak Jacobiho matice $\boldsymbol{J} \in \mathbb{R}^{n \times m}$ funkce $\boldsymbol{f}$ je definována jako $J_{i,j} = \frac{\partial}{\partial x_j} f(\boldsymbol{x})_i$.

Krom prvních derivací nás rovněž zajímají druhé derivace, kterou v jedné dimenzi značíme $f''(x) = \frac{d^2}{dx^2} f(x)$. Zajímají nás z toho důvodu, neboť nám mohou říct, jestli gradientní posun způsobí takové zlepšení, jaké očekáváme, pokud uvažujeme samotný gradient. Často se o druhé derivaci uvažuje jako o zakřivení.

Pokud naše funkce má vícedimenzionální vstup, tak počet všech možných druhých derivací může být značný. Tyto derivace můžeme přehledně zapsat pomocí Hessovy matice. Hessova matice $\boldsymbol{H}(f)(\boldsymbol{x})$ je definována jako

$$\boldsymbol{H}(f)(\boldsymbol{x})_{i,j} = \frac{\partial^2}{\partial x_i \partial x_j} f(\boldsymbol{x}). \tag{103}$$

Připomeňme také, že Hessova matice je symetrická ve všech bodech, kde jsou druhé derivace funkce f spojité, tedy $H_{i,j} = H_{j,i}$. Většina funkcí, se kterými se setkáme v kontextu neuronových sítí mají symetrické Hessovy matice skoro všude.

Protože Hessova matice je reálná a symetrická, můžeme ji rozložit na množinu reálných vlastních čísel a ortogonální bázi vlastních vektorů. Druhá derivace ve specifickém směru reprezentovaný jednotkovým vektorem $\boldsymbol{d}$ je dána $\boldsymbol{d}^\top \boldsymbol{H} \boldsymbol{d}$. Pokud je $\boldsymbol{d}$ vlastní vektor matice $\boldsymbol{H}$, druhá derivace v tomto směru je dána odpovídajícím vlastním číslem. Největší vlastní číslo určuje největší druhou derivaci a naopak.

Druhá (směrová) derivace nám říká, jak dobrý posun můžeme očekávat pomocí gradientního sestupu. Můžeme využít aproximace pomocí Taylorova rozvoje funkce $f(\boldsymbol{x})$ v okolí bodu $\boldsymbol{x}_0$

$$f(\boldsymbol{x}) \approx f(\boldsymbol{x}_0) + (\boldsymbol{x} - \boldsymbol{x}_0)^\top \boldsymbol{g} + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}_0)^\top \boldsymbol{H}(\boldsymbol{x} - \boldsymbol{x}_0), \tag{104}$$

kde $\boldsymbol{g}$ je gradient a $\boldsymbol{H}$ je Hessova matice v $\boldsymbol{x}_0$. Pokud použijeme rychlost učení η, pak nový bod $\boldsymbol{x}$ bude dán $\boldsymbol{x} = \boldsymbol{x}_0 - \eta \boldsymbol{g}$. Pokud to dosadíme do předchozí rovnice, obdržíme

$$f(\boldsymbol{x}_0 - \eta \boldsymbol{g}) \approx f(\boldsymbol{x}_0) - \eta \boldsymbol{g}^\top \boldsymbol{g} + \frac{1}{2}\eta^2 \boldsymbol{g}^\top \boldsymbol{H} \boldsymbol{g}. \tag{105}$$

V této rovnici vidíme tři různé členy - původní hodnotu funkce, očekávané zlepšení díky trendu funkce a korekci, kterou musíme aplikovat kvůli zakřivení funkce.

Pokud výraz $\boldsymbol{g}^\top \boldsymbol{H} \boldsymbol{g}$ je nula nebo záporné, tak aproximace podle Taylorova rozvoje naznačuje, že pokud budeme pořád zvětšovat η, tak díky tomu budeme pořád zmenšovat f. Ve skutečnosti Taylorův rozvoj velice pravděpodobně zůstane stále přesný pro vysoké η, takže se v takovém případě musíme uchýlit k heuristickým volbám η.

Pokud výraz $\boldsymbol{g}^\top \boldsymbol{H} \boldsymbol{g}$ je kladné, pak řešení pro optimální velikost kroku, který zmenší Taylorovu aproximaci nejvíce je dáno

$$\eta^* = \frac{\boldsymbol{g}^\top \boldsymbol{g}}{\boldsymbol{g}^\top \boldsymbol{H} \boldsymbol{g}} \tag{106}$$

V nejhorším případě se může stát, že $\boldsymbol{g}$ splývá s vlastním vektorem $\boldsymbol{H}$, který odpovídá největšímu vlastnímu číslu λ_{max}. Pak optimální velikost kroku je dána $\frac{1}{\lambda_{max}}$.

Další význam Hessovy matice je v určování extrémů. V bodě, kde $\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = 0$, můžeme na základně vlastních čísel rozhodnout, zda se jedná o lokální minimum či maximum nebo sedlo. Pokud tedy je Hessova matice pozitivně definitní, jedná se o lokální minimum. Pokud je Hessova matice negativní definitní, jedná se o lokální maximum. Dále pokud alespoň jedno vlastní číslo je kladné a alespoň jedno vlastní číslo je záporné, jedná se o sedlo. Pokud však všechny nenulové vlastní čísla mají stejné znaménko a zároveň alespoň jedno vlastní číslo je nulové, pak nedokážeme o vlastnosti tohoto bodu rozhodnout.

Ve více dimenzích lze pro daný bod určit různé druhé derivace pro každý směr. Číslo podmíněnosti Hessovy matice v tomto bodě měří, jak moc se jednotlivé druhé derivace navzájem od sebe liší. Špatně podmíněná Hessova matice znamená, že i gradientní sestup bude fungovat hůře. Děje se tak proto, že v jednom směru se derivace zvětšuje rychleji, zatímco v jiném se zvětšuje jen pomalu. Gradientní sestup tuto změnu v derivaci nezaznamená, a tudíž neví, že by se radši měl vydat ve směru, kde derivace zůstane záporná po delší dobu. V takovém případě je také obtížnější vybrat vhodnou velikost kroku [4].

B Koeficienty a exponenty fundamentálních rovnic

Oblast 1

Tabulka 11 – Hodnoty pro oblast 1

i	I_i	J_i	n_i	i	I_i	J_i	n_i
1	0	-2	0.146 329 712 131 67	18	2	3	$-0.441\,418\,453\,308\,46 \cdot 10^{-5}$
2	0	-1	–0.845 481 871 691 14	19	2	17	$-0.726\,949\,962\,975\,94 \cdot 10^{-15}$
3	0	0	$-0.375\,636\,036\,720\,40 \cdot 10^{1}$	20	3	-4	$-0.316\,796\,448\,450\,54 \cdot 10^{-4}$
4	0	1	$0.338\,551\,691\,683\,85 \cdot 10^{1}$	21	3	0	$-0.282\,707\,979\,853\,12 \cdot 10^{-5}$
5	0	2	–0.957 919 633 878 72	22	3	6	$-0.852\,051\,281\,201\,03 \cdot 10^{-9}$
6	0	3	0.157 720 385 132 28	23	4	-5	$-0.224\,252\,819\,080\,00 \cdot 10^{-5}$
7	0	4	$-0.166\,164\,171\,995\,01 \cdot 10^{-1}$	24	4	-2	$-0.651\,712\,228\,956\,01 \cdot 10^{-6}$
8	0	5	$0.812\,146\,299\,835\,68 \cdot 10^{-3}$	25	4	10	$-0.143\,417\,299\,379\,24 \cdot 10^{-12}$
9	1	-9	$0.283\,190\,801\,238\,04 \cdot 10^{-3}$	26	5	-8	$-0.405\,169\,968\,601\,17 \cdot 10^{-6}$
10	1	-7	$-0.607\,063\,015\,658\,74 \cdot 10^{-3}$	27	8	-11	$-0.127\,343\,017\,416\,41 \cdot 10^{-8}$
11	1	-1	$-0.189\,900\,682\,184\,19 \cdot 10^{-1}$	28	8	-6	$-0.174\,248\,712\,306\,34 \cdot 10^{-9}$
12	1	0	$-0.325\,297\,487\,705\,05 \cdot 10^{-1}$	29	21	-29	$0.687\,621\,312\,955\,31 \cdot 10^{-18}$
13	1	1	$-0.218\,417\,171\,754\,14 \cdot 10^{-1}$	30	23	-31	$0.144\,783\,078\,285\,21 \cdot 10^{-19}$
14	1	3	$-0.528\,383\,579\,699\,30 \cdot 10^{-4}$	31	29	-38	$0.263\,357\,816\,627\,95 \cdot 10^{-22}$
15	2	-3	$-0.471\,843\,210\,732\,67 \cdot 10^{-3}$	32	30	-39	$-0.119\,476\,226\,400\,71 \cdot 10^{-22}$
16	2	0	$-0.300\,017\,807\,930\,26 \cdot 10^{-3}$	33	31	-40	$0.182\,280\,945\,814\,04 \cdot 10^{-23}$
17	2	1	$0.476\,613\,939\,069\,87 \cdot 10^{-4}$	34	32	-41	$-0.935\,370\,872\,924\,58 \cdot 10^{-25}$

Oblast 2

Tabulka 12 – Hodnoty pro oblast 2 - část pro ideální plyn

i^0	J_i^0	n_i^0	i^0	J_i^0	n_i^0
1	0	–0.969 276 865 002 17·10^{1}	6	-2	0.142 408 191 714 44·10^{1}
2	1	0.100 866 559 680 18·10^{2}	7	-1	–0.438 395 113 194 50 ·10^{1}
3	-5	–0.560 879 112 830 20 ·10^{-2}	8	2	–0.284 086 324 607 72
4	-4	0.714 527 380 814 55·10^{-1}	9	3	0.212 684 637 533 07·10^{-1}
5	-3	–0.407 104 982 239 28			

Tabulka 13 – Hodnoty pro oblast 2 - reziduální část

i	I_i	J_i	n_i	i	I_i	J_i	n_i
1	1	0	–0.177 317 424 732 13 ·10^{-2}	23	7	0	–0.590 595 643 242 70 ·10^{-17}
2	1	1	–0.178 348 622 923 58 ·10^{-1}	24	7	11	–0.126 218 088 991 01·10^{-5}
3	1	2	–0.459 960 136 963 65·10^{-1}	25	7	25	–0.389 468 424 357 39·10^{-1}
4	1	3	-0.575 812 590 834 32·10^{-1}	26	8	8	0.112 562 113 604 59·10^{-10}
5	1	6	–0.503 252 787 279 30·10^{-1}	27	8	36	–0.823 113 408 979 98 ·10^{1}
6	2	1	–0.330 326 416 702 03·10^{-4}	28	9	13	0.198 097 128 020 88·10^{-7}
7	2	2	–0.189 489 875 163 15·10^{-3}	29	10	4	0.104 069 652 101 74·10^{-18}
8	2	4	–0.393 927 772 433 55·10^{-2}	30	10	10	-0.102 347 470 959 29·10^{-23}
9	2	7	–0.437 972 956 505 73·10^{-1}	31	10	14	–0.100 181 793 795 11·10^{-8}
10	2	36	–0.266 745 479 140 87·10^{-4}	32	16	29	–0.808 829 086 469 85·10^{-10}
11	3	0	0.204 817 376 923 09·10^{-7}	33	16	50	0.106 930 318 794 09
12	3	1	0.438 706 672 844 35·10^{-6}	34	18	57	– 0.336 622 505 741 71
13	3	3	–0.322 776 772 385 70·10^{-4}	35	20	20	0.891 858 453 554 21 ·10^{-24}
14	3	6	–0.150 339 245 421 48·10^{-2}	36	20	35	0.306 293 168 762 32·10^{-12}
15	3	35	–0.406 682 535 626 49·10^{-1}	37	20	48	–0.420 024 676 982 08 ·10^{-5}
16	4	1	–0.788 473 095 593 67·10^{-9}	38	21	21	–0.590 560 296 856 39·10^{-25}
17	4	2	0.127 907 178 522 85·10^{-7}	39	22	53	0.378 269 476 134 57 ·10^{-5}
18	4	3	0.482 253 727 185 07·10^{-6}	40	23	59	–0.127 686 089 346 81 ·10^{-14}
19	5	7	0.229 220 763 376 61·10^{-5}	41	24	26	0.730 876 105 950 61·10^{-28}
20	6	3	–0.167 147 664 510 61·10^{-10}	42	24	40	0.554 147 153 507 78·10^{-16}
21	6	16	–0.211 714 723 213 55·10^{-2}	43	24	58	–0.943 697 072 412 10·10^{-6}
22	6	35	– 0.238 957 419 341 04·10^{2}				

Oblast 3

Tabulka 14 – Hodnoty pro oblast 3

i	I_i	J_i	n_i	i	I_i	J_i	n_i
1	-	-	$0.106\,580\,700\,285\,13 \cdot 10^{1}$	21	3	4	$-0.201\,899\,150\,235\,70 \cdot 10^{1}$
2	0	0	$-0.157\,328\,452\,902\,39 \cdot 10^{2}$	22	3	16	$-0.821\,476\,371\,739\,63 \cdot 10^{-2}$
3	0	1	$0.209\,443\,969\,743\,07 \cdot 10^{2}$	23	3	26	$-0.475\,960\,357\,349\,23$
4	0	2	$-0.768\,677\,078\,787\,16 \cdot 10^{1}$	24	4	0	$0.439\,840\,744\,735\,00 \cdot 10^{1}$
5	0	7	$0.261\,859\,477\,879\,54 \cdot 10^{1}$	25	4	2	$-0.444\,764\,354\,287\,39$
6	0	10	$-0.280\,807\,811\,486\,20 \cdot 10^{1}$	26	4	4	$0.905\,720\,707\,197\,33$
7	0	12	$0.120\,533\,696\,965\,17 \cdot 10^{1}$	27	4	26	$0.705\,224\,500\,879\,67$
8	0	23	$-0.845\,668\,128\,125\,02 \cdot 10^{-2}$	28	5	1	$0.107\,705\,126\,263\,32$
9	1	2	$-0.126\,543\,154\,777\,14 \cdot 10^{1}$	29	5	3	$-0.329\,136\,232\,589\,54$
10	1	6	$-0.115\,244\,078\,066\,81 \cdot 10^{1}$	30	5	26	$-0.508\,710\,620\,411\,58$
11	1	15	$0.885\,210\,439\,843\,18$	31	6	0	$-0.221\,754\,008\,730\,96 \cdot 10^{-1}$
12	1	17	$-0.642\,077\,651\,816\,07$	32	6	2	$0.942\,607\,516\,650\,92 \cdot 10^{-1}$
13	2	0	$0.384\,934\,601\,866\,71$	33	6	26	$0.164\,362\,784\,479\,61$
14	2	2	$-0.852\,147\,088\,242\,06$	34	7	2	$-0.135\,033\,722\,413\,48 \cdot 10^{-1}$
15	2	6	$0.489\,722\,815\,418\,77 \cdot 10^{1}$	35	8	26	$-0.148\,343\,453\,524\,72 \cdot 10^{-1}$
16	2	7	$-0.305\,026\,172\,569\,65 \cdot 10^{1}$	36	9	2	$0.579\,229\,536\,280\,84 \cdot 10^{-3}$
17	2	22	$0.394\,205\,368\,791\,54 \cdot 10^{-1}$	37	9	26	$0.323\,089\,047\,037\,11 \cdot 10^{-2}$
18	2	26	$0.125\,584\,084\,243\,08$	38	10	0	$0.809\,648\,029\,962\,15 \cdot 10^{-4}$
19	3	0	$-0.279\,993\,296\,987\,10$	39	10	1	$-0.165\,576\,797\,950\,37 \cdot 10^{-3}$
20	3	2	$0.138\,997\,995\,694\,60 \cdot 10^{1}$	40	11	26	$-0.449\,238\,990\,618\,15 \cdot 10^{-4}$

Oblast 4

Tabulka 15 – Hodnoty pro oblast 4

i	n_i	i	n_i
1	$0.116\,705\,214\,527\,67 \cdot 10^{4}$	6	$0.149\,151\,086\,135\,30 \cdot 10^{2}$
2	$-0.724\,213\,167\,032\,06 \cdot 10^{6}$	7	$-0.482\,326\,573\,615\,91 \cdot 10^{4}$
3	$-0.170\,738\,469\,400\,92 \cdot 10^{2}$	8	$0.405\,113\,405\,420\,57 \cdot 10^{6}$
4	$0.120\,208\,247\,024\,70 \cdot 10^{5}$	9	$-0.238\,555\,575\,678\,49$
5	$-0.323\,255\,503\,223\,33 \cdot 10^{7}$	10	$0.650\,175\,348\,447\,98 \cdot 10^{3}$

Oblast 5

Tabulka 16 – Hodnoty pro oblast 5 - část pro ideální plyn

i^0	J_i^0	n_i^0	i^0	J_i^0	n_i^0
1	0	–0.131 799 836 742 01$\cdot 10^{2}$	4	-2	0.369 015 349 803 33
2	1	0.685 408 416 344 34$\cdot 10^{1}$	5	-1	–0.311 613 182 139 25 $\cdot 10^{1}$
3	-3	–0.248 051 489 334 66 $\cdot 10^{-1}$	6	2	–0.329 616 265 389 17

Tabulka 17 – Hodnoty pro oblast 5 - reziduální část

i	I_i	J_i	n_i	i	I_i	J_i	n_i
1	1	1	0.157 364 048 552 59 $\cdot 10^{-2}$	2	3	0	0.224 400 374 094 85 $\cdot 10^{-5}$
2	1	2	0.901 537 616 739 44 $\cdot 10^{-3}$	2	9	11	-0.411 632 754 534 71$\cdot 10^{-5}$
3	1	3	-0.502 700 776 776 48$\cdot 10^{-2}$	3	7	25	0.379 194 548 229 55$\cdot 10^{-7}$