VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ

FACULTY OF MECHANICAL ENGINEERING

ÚSTAV MECHANIKY TĚLES, MECHATRONIKY A BIOMECHANIKY

INSTITUTE OF SOLID MECHANICS, MECHATRONICS AND BIOMECHANICS

VYUŽITÍ RYCHLÉHO ALGORITMU KOSOÚHLÉHO ROVNÁNÍ K OPTIMALIZACI PROCESU ROVNÁNÍ POMOCÍ NEURONOVÝCH SÍTÍ

OPTIMIZATION OF THE STRAIGHTENING PROCESS BY USING NEURAL NETWORKS AND FAST ALGORITHM

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — **Bc. MILAN HLUŠKA**

AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. TOMÁŠ NÁVRAT, Ph.D.**

SUPERVISOR

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Ústav mechaniky těles, mechatroniky a biomechaniky
Student:	**Bc. Milan Hluška**
Studijní program:	Aplikované vědy v inženýrství
Studijní obor:	Inženýrská mechanika a biomechanika
Vedoucí práce:	**doc. Ing. Tomáš Návrat, Ph.D.**
Akademický rok:	2019/20

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:

Využití rychlého algoritmu kosoúhlého rovnání k optimalizaci procesu rovnání pomocí neuronových sítí

Stručná charakteristika problematiky úkolu:

Optimalizace procesu kosoúhlého rovnání představuje velmi složitý problém. Simulace procesu výpočtovým modelováním s využitím MKP vyžaduje formulovat úlohu nelineárně, což se výrazně projeví na výsledném výpočtovém času. Znalost vstupních hodnot je velmi omezená a zohlednění dalších podstatných vlivů (např. styk válce s tyčí) musí být značně zjednodušeno. Pro zmapování odhadu výstupních parametrů rovnání je nezbytné využít navržený rychlý algoritmus, protože celý proces vyžaduje opakované výpočty pro různé nastavení rovnačky. Získané výsledky můžeme využít optimalizaci celého procesu s využitím neuronových sítí.

Cíle diplomové práce:

1) Úprava výpočtového modelu rychlého algoritmu procesu rovnání tyčí kruhového průřezu pro jednoduchou volbu konfigurace rovnacího stroje.
2) Vytvoření programu pro automatizaci opakovaných výpočtů pro různé kombinace vstupních veličin.
3) Návrh postupu pro optimalizaci procesu rovnání pomocí neuronových sítí pro různé konfigurace rovnacího stroje.

Seznam doporučené literatury:

MARCINIAK, Z. Teorie tváření plechů. Praha: SNTL, 1964.

PETRUŠKA, J.; NÁVRAT, T.; ŠEBEK, F. A New Model for Fast Analysis of Leveling Process. Advanced Materials Research. 2012 (586) p. 389 - 393.

NASTRAN M., KUZMAN K.: Stabilisation of mechanical properties of the wire by roller straightening, J. Mat. Proc. Tech. 125-126 (2002) 711-719.

VONDRÁK, I. Umělá inteligence a neuronové sítě. 3. vyd. Ostrava: VŠB - Technická univerzita Ostrava, 2009. ISBN 978-80-248-1981-5.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2019/20

V Brně, dne

L. S.

prof. Ing. Jindřich Petruška, CSc.
ředitel ústavu

doc. Ing. Jaroslav Katolický, Ph.D.
děkan fakulty

Abstrakt
Diplomová práca sa zaoberá využitím rýchleho algoritmu kosouhlého rovnania tyčí a jeho úpravami pre možnosť zadávania veľkého množstva výpočtov a ľubovoľnú konfiguráciu valcovacieho stroja, a verifikáciou podľa pôvodného programu. Ďalej sa zaoberá využitím tohto algoritmu na optimalizáciu procesu rovnania pomocou metódy neurónových sietí.

Summary
Master thesis deals with a fast cross roll bar straightening algorithm and its modifications to allow for an automatic calculation of a large number of simulations and an arbitrary straightening machine configuration. Modified program is then verified using the original algorithm. It also deals with the algorithm's application to straightening process optimization using neural networks.

Klíčová slova
kosouhlé rovnanie, metóda konečných prvkov, MKP, neurónové siete

Keywords
cross roll straightening, finite element method, FEM, neural networks

HLUŠKA, M. *Využití rychlého algoritmu kosoúhlého rovnání k optimalizaci procesu rovnání pomocí neuronových sítí.* Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2020. 53 s. Vedoucí diplomové práce doc. Ing. Tomáš Návrat, PhD.

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne, iba pod odborným vedením doc. Ing. Tomáša Návrata, PhD a s použitím uvedenej literatúry.

Bc. Milan Hluška

Ďakujem doc. Ing. Tomášovi Návratovi, PhD za možnosť vypracovania tejto diplomovej práce pod jeho vedením, jeho ochotu a pomoc pri jej vypracovávaní.

Predovšetkým by som sa však chcel poďakovať celej svojej rodine, hlavne rodičom, za ich podporu počas môjho vysokoškolského štúdia.

Bc. Milan Hluška

Obsah

1. Úvod

Výpočtové modelovanie je v dnešnej dobe veľmi dôležitou súčasťou priemyselnej praxe. Pri návrhu a konštruovaní strojov ide o užitočný nástroj, ktorý pri správnom použití ľuďom pomáha získať náhľad do komplexných interakcií jeho súčastí medzi sebou, ako aj celku s jeho okolím. Na tento účel sa najčastejšie používajú komerčné balíky programov pre výpočty rôznych aspektov strojárskych konštrukcií využívajúce metódu konečných prvkov. Ich cieľom je ponúknuť prostredie, v ktorom je užívateľ schopný vytvoriť a simulovať ľubovoľné modelové objekty. Problematikou modelovania procesu rovnania tyčí kosouhlým valcovaním sa zaoberá rada prác a článkov [2], snažiacich sa o jeho čo najdetailnejší popis. S narastajúcim výkonom výpočtových zariadení sa pritom zvyšuje úroveň detailov, ktoré môžu byť v týchto analýzach obsiahnuté.

Počítanie rozsiahlych analýz počas návrhu rovnacieho stroja je ale vhodné až vo fáze, kedy sú známe hodnoty základných parametrov stroja, na základe ktorých sú určované podmienky simulácií. Ak však tieto hodnoty ešte nie sú známe, je výhodnejšie použitie prostriedkov, ktoré za oveľa kratšiu dobu výpočtu poskytnú výsledky zjednodušených úloh. Táto práca sa zaoberá úpravami rýchleho algoritmu rovnania tyčí [1, 3] pre praktické podmienky priemyslu. Tými sú hlavne automatizácia zadávania simulácií a možnosť nastavenia konfigurácie rovnačky, pri zachovaní krátkeho výpočtového času simulácie. Ďalej skúma možnosť využitia neurónových sietí, kedy by bolo možné vytvorenie programu poskytujúcom konečnému užívateľovi výsledné hodnoty vybraných veličín rovnacieho procesu v reálnom čase.

2. Formulácia problému

Problémová situácia je definovaná ako neštandardná situácia, od bežnej odlišná v tom, že jej vyriešenie vyžaduje použiť aj iné ako rutinné , tj. známe, respektíve algoritmizované činnosti [5]. Pritom je nutné riešenú entitu uvažovať z pohľadu celku zloženého zo súčastí, ktoré sú rôzne viazané medzi sebou, ako aj s okolím entity.

2.1. Analýza problémovej situácie

Návrh strojov je iteratívny proces, na začiatku ktorého je v prípade žiadnych predchádzajúcich skúseností potrebné zmapovať neznámy návrhový priestor parametrov stroja. Pre tento účel sa v súčasnosti často používa metóda konečných prvkov, ktorej úlohy ale musia byť kvôli charakteru procesu kosouhlého rovnania tyčí valcovaním formulované nelineárne, čo značne predlžuje dobu výpočtu. Ďalej je problematické získanie kvalitných vstupných veličín, ako rozloženie kontaktného tlaku medzi rovnacím valcom a tyčou a vstupná krivosť tyče, ktoré musia byť do značnej miery zjednodušené.

2.2. Formulácia problému

Implementácia výpočtového programu podľa rýchleho algoritmu riešiča MKP s možnosťou výpočtov viacerých konfigurácií rovnacieho stroja a vlastností tyče. Návrh postupu pre optimalizáciu rovnacieho procesu za použitia neurónových sietí.

2.3. Ciele riešenia problému

- Reimplementácia rýchleho algoritmu kosouhlého rovnania tyčí a jeho rozšírenie o možnosť nastavenia ľubovoľnej konfigurácie rovnacieho stroja a spôsob zadávania série výpočtov.
- Tvorba neurónových sietí a ich trénovanie na výstupoch reimplementovaného algoritmu.
- Využitie neurónových sietí pre optimalizáciu a vizualizáciu výstupov simulácií procesu rovnania.

2.4. Systém podstatných veličín

S0: prvky okolia

prázdna množina ($S0 = \varnothing$)

S1: topológia a geometria entity

- rovnací stroj
  - počet rovnacích valcov (číselná, deterministická, statická)
  - posuvy a natočenia rovnacích valcov (číselná, deterministická, statická)
  - vzdialenosť medzi jednotlivými rovnacími valcami (číselná, deterministická, statická)
  - opotrebovanie rovnacích valcov (číselná, stochastická, statická)
- tyč
  - priemer (číselná, stochastická, statická)
  - vstupná krivosť (číselná, stochastická, statická)
  - kvalita povrchu (číselná, stochstická, statická)
- neurónová sieť
  - topológia vrstiev neurónov (číselná, deterministická, statická)

S2: väzby a interakcie

- styk rovnacieho valca s tyčou (číselná, stochastická, dynamická)

S3: aktivácia entity

- tvrdé zaťažovnie tyče rovnacími valcami (číselná, stochastická, dynamická)

S4: ovplyvňovanie entity z okolia

prázdna množina ($S4 = \varnothing$)

S5: vlastnosti prvkov štruktúry

- výpočtový model MKP
  - model geometrie tyče (číselná, deterministicá, statická)
  - typ prvku MKP modelu tyče (číselná, deterministická, statická)
  - model materiálu tyče (číselná, deterministická, statická)
- neurónová sieť
  - trénovacie hyperparametre (číselná, deterministická, statická)

S6: procesy a stavy entity

- redistribúcia zbytkových napätí tyče (číselná, stochastická, dynamická)

S7: prejavy entity

- napätia (číselná, stochastická, dynamická)
- deformácie (číselná, stochastická, dynamická)
- reakčné sily pôsobiace na rovnacie valce (číselná, stochastická, dynamická)

S8: dôsledky prejavov entity

- zmena výstupnej krivosti tyče (číselná, stochastická, dynamická)

2.5. Výber metódy riešenia

Nakoľko neexistuje prakcticky používaná metóda na výpočet nelineárnych pevnostných úloh, ktorá je schopná pracovať so stochastickými veličinami spomenutými v časti 2.4, bude implementácia výpočtového programu využívať metódu konečných prvkov s formuláciou kontinua podľa Eulera. Veličiny budú považované za deterministické a statické aj preto, že sú kvôli ich pravdepodobnostnej a komplexnej povahe veľmi ťažko prakticky merateľné.

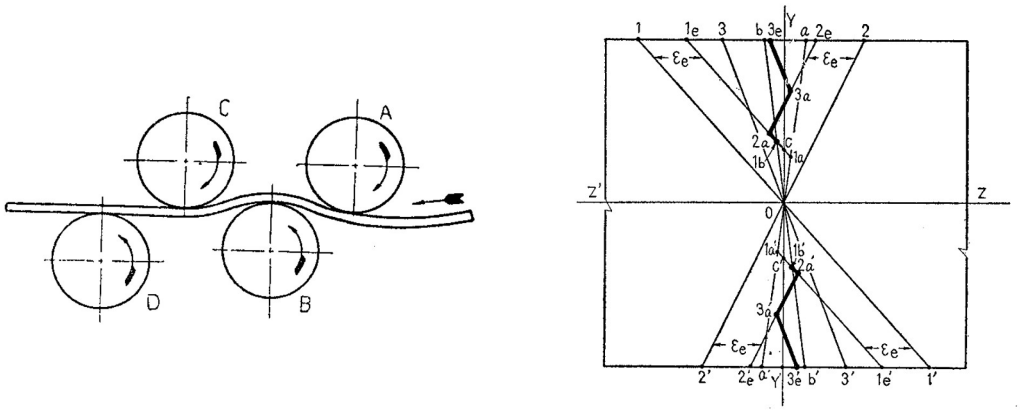
Druhá časť práce bude využívať neurónovú sieť, ktorá ako metóda výpočtového modelovania bude ako vstupy využívať výstupy z výpočtového programu prvej časti, kvôli čomu budú aj tieto veličiny uvažované ako číselné, deterministické a statické. Trénovanie siete je všeobecne proces stochastický, čo je v praxi z viacerých hľadísk užitočné.

Pre účely reprodukovateľnosti výsledkov práce však budú generátory pseudonáhodných čísel vytvárajúce náhodné prostredie manuálne nastavované, efektívne to tak bude proces deterministický.

3. Základný princíp rovnania valcovaním

Hutnícke polotovary ako plechy a rôzne profily obsahujú nerovnosti, ktoré vznikajú v samotnom procese výroby, napríklad chladnutím po valcovaní za tepla, alebo pri nevhodnej manipulácii a transporte. Tieto nedokonalosti sú pre ďalšie spracovanie často neprípustné, hlavne pokiaľ ide o vysokorýchlostné obrábanie. V takom prípade sa používa proces rovnania, ktorý ako technologický proces tvárnenia znižuje hodnotu krivosti výrobku na prijateľnú hodnotu.

Zmena krivosti sa dosahuje opakovaným ohýbaním polotovaru prechodom cez presadené valce (obrázok 3.1a), ktorými sa materiál zaťažuje za jeho medzu klzu, čo spôsobuje redistribúciu plastického napätia po priečnom priereze [6]. Vhodným nastavením presahov valcov je týmto spôsobom možné do materiálu vniesť plastické napätie, ktorého vonkajšia časť sa vplyvom nasledujúceho valca ohybom v opačnom smere vyruší (obrázok 3.1b). Postupným prechodom polotovaru popri všetkých valcoch rovnacieho stroja by sa tak mala jeho celková krivosť zmenšovať až pod prijateľnú hranicu. Zbytkové napätie by taktiež malo byť rovnomerné a aj keď nie nulové, nižšie ako pôvodne.



(a) Základná schéma valcovej rovnačky. (b) Diagram zbytkového napätia opakovaným ohýbaním.

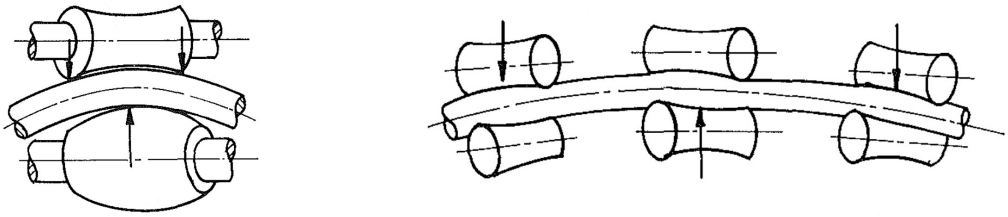
Obr. 3.1: Upravené z [6]

Existuje niekoľko konštrukčných prevedení valcovacích strojov, ktorých dizajn závisí predovšetkým od druhu rovnaného polotovaru. Na plechy a obecné profily sa používjú rovnačky s valcami uloženými kolmo na ich os, nastaviteľnými iba vo vertikálnom smere. Na tyče a rúry sa častejšie používajú kosouhlé rovnačky s valcami v tvare hyperboloidov.

3.1. Rovnanie tyčí kosouhlým valcovaním

Rovnačky tyčí s kruhovým prierezom môžu vďaka jeho osovej symetrii využívať valce, ktoré sú od axiálneho smeru prechádzajúcej tyče natočené, čím spôsobujú jej rotáciu okolo svojej osi. Výsledkom je preto šróbovicovitý pohyb polotovaru, čo umožňuje rovnanie vo všetkých jeho rovinách [1, 7].

Najjednoduchším typom kosouhlej rovnačky je dvojvalcová rovnačka. Pozostáva z dvoch valcov, ktorých osi sú mimobežné navzájom, ako aj voči osi rovnanej tyče. Jeden valec je konkávny, druhý konvexný, pričom obidva sú poháňané. Podľa nastavenia môže nastať kontakt medzi valcami a tyčou na okrajoch konkávneho a v strede konvexného valca, alebo môže nastať kontakt líniový. Líniový kontakt zaťažuje tyč rovnomernejšie a valce opotrebováva v menšej miere ako v prípade koncentrovaných pôsobísk reakčných síl. Tyč musí byť pri prechode rovnačkou pridržiavaná tzv. pravítkami, ktoré sa ale veľmi rýchlo opotrebovávajú, čím taktiež môžu spôsobiť poškodenie povrchu tyče. Výhodou tohto typu je schopnosť vyrovnania tyče po celej jej dĺžke, ale pri relatívne malej rýchlosti posuvu.



(a) Schéma dvojvalcovej rovnačky. (b) Schéma šesťvalcovej rovnačky.

Obr. 3.2: Upravené z [8]

Viacvalcová kosouhlá rovnačka využíva šesť a viac rovnacích valcov hyperboloidného tvaru, z ktorých môžu byť poháňané všetky, alebo iba niektoré. Ďalej sú rozlišovné výškovo nastaviteľné valce priehybové a tzv. prítlačné. Všetky valce pritom majú možnosť nastavenia uhlového natočenia, ktorá pomáha zaistiť líniový kontakt valcov s tyčou. Výhodou tohoto typu konštrukcie je pomerne vysoká rýchlosť a presnosť rovnania. Nevýhodou je ale neschopnosť narovnania koncov tyče, ktorých dĺžka závisí na vzájomných rozostupoch medzi valcami.

4. Rýchly algoritmus

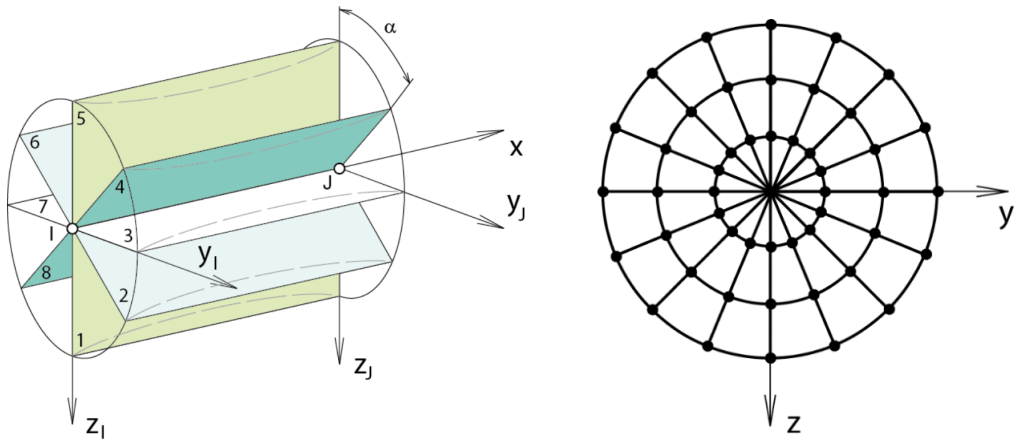
Táto kapitola sa zaoberá funkcionalitou rýchleho algoritmu [3, 4] určeného na vykonávanie analýzy kosouhlého rovnania tyčí pomocou metódy konečných prvkov, ktorý vytvorili docent Tomáš Návrat a profesor Jindřich Petruška.

Zámerom bolo vyvinúť jednoducho použiteľný nástroj na výpočet napäťovo-deformačných stavov tyče počas rovnania, ktorý by poskytoval podobné výsledky ako väčšie viacúčelové balíky MKP, avšak za oveľa kratšiu dobu. Významne by tak urýchlil hľadanie optimálneho nastavenia rovnačky pre tyče z rôznych materiálov a o rôznych rozmeroch.

Vývoj metódy konečných prvkov a výpočtového hardvéru v dnešnej dobe umožňuje pomerne komplexné analýzy s množstvom zahrnutých detailov. Kvôli silnej nelinearite rovnacieho procesu, a teda nutnosti iteratívneho riešenia, sú však tieto výpočty pomalé, čo pri finálnom dolaďovaní parametrov rovnačky nemusí byť významné. V prvotnej fáze, kedy je množstvo vstupných parametrov ešte neznámych alebo zatiaľ nepodstatných, je ale pre získanie odhadu výpočtový čas zásadný. Podľa [4] sa čas výpočtu skrátil z niekoľkých hodín až dní (pri použití výpočtového balíka ANSYS) na 1 až 3 minúty, pričom boli výsledky vrámci inžinierskej presnosti ekvivalentné.

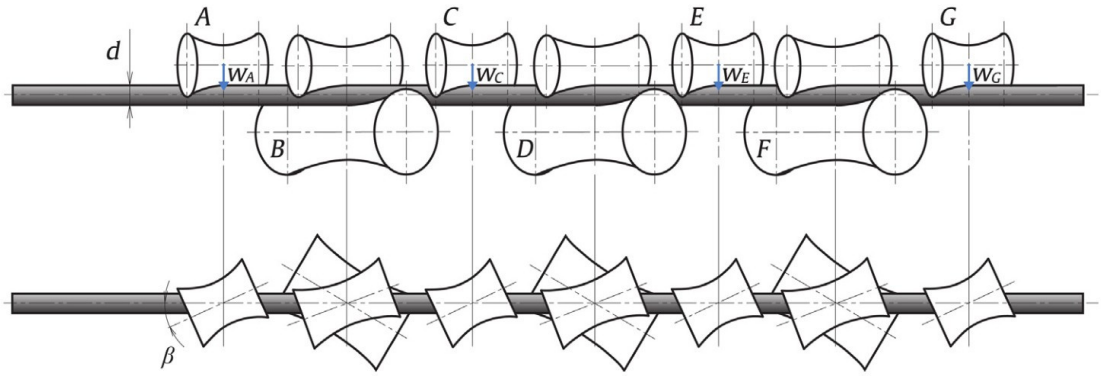
Algoritmus je implementovaný vo výpočtovom prostredí MATLAB, vyvíjanom firmou MathWorks. Keďže sa jedná o dynamický proces, ktorý vyžaduje uchovávanie histórie plastického pretvorenia, využíva program Eulerovu definíciu toku materiálu cez rovnačku. Použité sú priestorové prútové prvky, kvôli čomu od zadania úlohy vyžaduje splnenie prútových predpokladov. Vstupnými parametrami sú priemer rovnanej tyče d, Youngov modul pružnosti v ťahu E, medza klzu R_e, modul spevnenia E_T, vstupná krivosť k_0 v jednej rovine (obrázok 4.1a), posuvy v, w a natočenie β jednotlivých rovnacích valcov. Výstupmi sú potom priebehy posuvov, natočení a krivosti tyče, priebehy pôsobacich ohybových momentov, šmykových síl a reakčných síl valcov na tyč, ako aj priebehy celkových, elastických a plastických napätí a pretvorení prierezov v uzloch osi tyče. Najdôležitejšími výstupmi sú však výstupná krivosť a zbytkové napätie prierezu na konci rovnania.

Diskretizácia osi a prierezu závisí na uhle natočenia valcov voči osi tyče β. Dĺžka osových prvkov a vzdialenosť medzi uzlami prierezu na jeho obvode sú nastavované v takom pomere, aby čo najlepšie zodpovedali pomeru doprednej rýchlosti rovnania a súčasnej rotácií tyče okolo svojej osi. Program totiž pri každom posunutí tyče o jeden osový prvok pootočí o jeden prvok aj jej prierez. Rozdelenie prierezu po priemere je potom primerané rozdeleniu obvodovému (viď obrázok 4.1b).



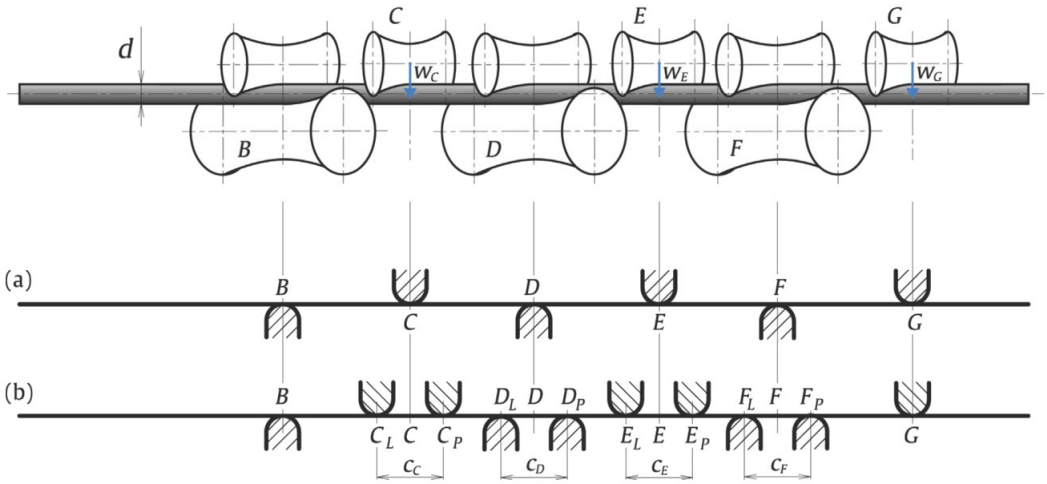
(a) Roviny krivosti tyče. (b) Diskretizácia priečneho prierezu.

Obr. 4.1: Diskretizácia rovnanej tyče. [1]



Obr. 4.2: Schématické znázornenie 9-valcovej rovnačky. [4]

Podporovaný je jeden typ rovnačky, konkrétne 9-valcová kosouhlá rovnačka (obrázok 4.2) so 4 priehybovými valcami, s pevne daným a rovnakým vzájomným rozostupom valcov. Deformačné okrajové podmienky reprezentujúce rovnacie valce sú zavádzané do jednotlivých uzlov osi tyče predpísaním zadaného posuvu (obrázok 4.3a). Druhou alternatívou, ktorá má bližšie aproximovať líniový kontakt medzi valcom a tyčou, je predpísanie posuvov do uzlov vzdialených od základného uzlu väzby o polovicu dĺžky kontaktu (obrázok 4.3b). Tento prístup pomáha s konvergenciou simulácií, nakoľko znižuje mieru koncentrácie zaťaženia tyče oproti jednobodovým okrajovým podmienkam.



Obr. 4.3: (a) Jednobodové okrajové podmienky. (b) Rozdvojené okrajové podmienky. [4]

Program používa bilineárny model materiálu, s možnosťou bez spevnenia, alebo so spevnením izotropickým alebo kinematickým.

4.1. Reimplementácia

Pre splnenie bodov 1 a 2 zadania práce bol rýchly algoritmus z časti 4 reimplementovaný v programovacom jazyku Python [17]. Oproti MATLABu, čo je komerčne licencovaný softvér, je Python voľne dostupný pod licenciou kompatibilnou s GNU General Public Licence (GPL).

Hlavné použité moduly sú NumPy [19] na matematické operácie, SciPy [20] a Scikit-learn [22] kvôli niekoľkým algoritmom na spracovanie počítaných veličín. Nakoniec využíva na vykresľovanie grafov priebehov výstupných veličín modul Matplotlib [21].

Tak ako originálny program, aj reimplemetácia využíva 2-uzlové prútové prvky s ôsmimi stupňami voľnosti (posuvy w a natočenia φ v dvoch osách kolmých na os prvku pre oba uzly). Oproti originálu je však rozšírený o možnosť nastavenia ľubovoľných rozostupov rovnacích valcov, pričom sa od seba môžu líšiť. Spolu s možnosťou voľby jednobodových alebo rozdvojených väzieb pre každý valec zvlášť je možné počítané úlohy nastaviť na obecnú konfiguráciu rovnačky. Predpisované posuvy v rozdvojených okrajových podmienkach musia byť korigované, nakoľko priehyb medzi zaväzbenými uzlami je väčší ako v nich, preto treba predpísať hodnotu nižšiu. Kvôli obecným rozostupom valcov sa hodnoty týchto korekcií odvodzujú analyticky iba veľmi ťažko. Pre získanie týchto hodnôt by bolo vhodné zistiť optimálne hodnoty počítaním priehybov tyče v statickej rovnováhe

iteratívnym spôsobom pred začatím samotného výpočtu rovnania, čo v súčasnej dobe nie je implementované. V prípade rozdielu polohy väzby od jej optimálneho stavu totiž kvôli charakteru rotačných väzieb sa môže stať, že niektorá z nich pôsobí v ťahu, čo neodpovedá realite, nakoľko rovnacie valce na tyč pôsobia iba v tlaku.

Ďalšou zmenou je pridanie funkcionality na jednoduchú serializáciu výpočtov pomocou ukladania zadaní v súboroch formátu `.xlsx` (obrázok 4.4), ktoré sú upravovateľné v programoch Microsoft Excel alebo LibreOffice Calc. Druhý typ vstupného súboru ukladá konfiguráciu rovnačky (obrázok 4.5), aby sa predišlo opakovaniu rovnakých parametrov v prvom súbore. Program po spustení postupne načítava vstupné parametre jednotlivých analýz, priebeh výpočtov zaznamenáva do samostatných textových súborov a výstupy ukladá v binárnom formáte, taktiež pre každé zadanie zvlášť.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1				SEC	MAT				BC		IN						
2	done	result_filename	config	diameter	Re	Em	ET	ductility	v	w	k0j	HM	step	max_iter	ftoler	mtoler	utoler
3	[-]	[-]	[-]	[mm]	[MPa]	[MPa]	[MPa]	[%]	[mm]	[mm]	[mm/m]	[-]	[-]	[-]	[%]	[%]	[%]
4		verification001	verconfig.xlsx	50	500	206000	10000	1	0,0,0,0,0,0,0	0,0,5,0,4,0,2	10	0	1	30	0.001	0.001	0.001
5		verification002	verconfig.xlsx	70	800	206000	10000	1	0,0,0,0,0,0,0	0,0,4,0,4,0,1	10	0	1	30	0.001	0.001	0.001
6		verification003	verconfig.xlsx	90	1100	206000	10000	1	0,0,0,0,0,0,0	0,0,3.5,0,3.5,0,1	5	0	1	30	0.001	0.001	0.001

Obr. 4.4: Štruktúra .xlsx súboru so vstupnými dátami.

Prvý riadok obrázku 4.4 nie je skriptom používaný, slúži na vizuálne rozdelenie typov zadávaných vstupov tabuľky, a to na priečny prierez (`SEC`), materiálové vlastnosti (`MAT`), okrajové podmienky (`BC`) a ostatné vstupné parametre (`IN`). Do stĺpca `done` sa po úspešnom vypočítaní zadania uloží názov súboru s danými výsledkami, za ktorého základ sa použije reťazec z `result_filename`. Stĺpec `config` obsahuje názov súboru s konfiguráciou rovnačky (obrázok 4.5). `diameter` predstavuje priemer rovnanej tyče d, `Re` medzu klzu materiálu tyče R_e, `Em` Youngov modul pružnosti v ťahu E, `ET` modul spevnenia E_T a `ductility` ťažnosť A. Zoznamy číselných hodnôt v stĺpcoch `v` a `w` určujú posuvy rovnacích valcov v osách y a z. `k0j` je vstupná krivosť tyče k_0, `HM` určuje typ spevňovania materiálu (0: bez spevnenia, 1: s izotropickým spevnením, 2: s kinematickým spevnením; podporovaná je zatiaľ iba hodnota 0). `step` určuje počet záťažných krokov riešenia, pričom počet podkrokov je definovaný stĺpcom `max_iter`. Hodnoty `ftoler`, `mtoler` a `utoler` predstavujú horné hranice tolerancií presnosti riešenia pre sily, momenty a posunutia.

	A	B	C	D	E	F	G
1	Lv	BCT	active		vs	rot_agn	reduction
2	[mm]	[-]	[-]		[m/min]	[°]	[-]
3	475	0	0		30	30	1.04167
4	475	0	1				
5	475	0	1				
6	475	0	1				
7	475	0	1				
8	475	0	1				
9		0	1				

Obr. 4.5: Štruktúra .xlsx súboru s konfiguráciou rovnačky.

StĺpecBCT (Boundary Condition Type) na obrázku 4.5 predstavuje pre hodnotu 0 jednobodovú väzbu, rozdvojenie tejto väzby nastáva pre riadky so zadanou hodnotou 1. Stĺpecactive potom obsahuje pravdivostné hodnoty indikujúce funkčnosť väzby. HodnotyLv určujú rozostup valca L_v na tom istom riadku od valca o riadok nižšie.vs určuje rýchlosť rovnania v_s,rot_agn je uhol natočenia rovnacieho valca β a reduction je hodnota redukcie rýchlosti otáčania tyče.

4.2. Verifikácia

V tejto časti bude zhodnotených niekoľko výsledkov výpočtov pre účely verifikovania reimplementovaného programu voči už validovanému programu z časti 4, spolu s niekoľkými jednoduchými ilustratívnymi zadaniami.

		Číslo zadania		
		1	2	3
d	[mm]	50	70	90
R_e	[MPa]	500	800	1100
w_C	[mm]	5	4	3.5
w_E	[mm]	4	4	3.5
w_G	[mm]	2	1	1
k_0	[mm/m]	10	10	5

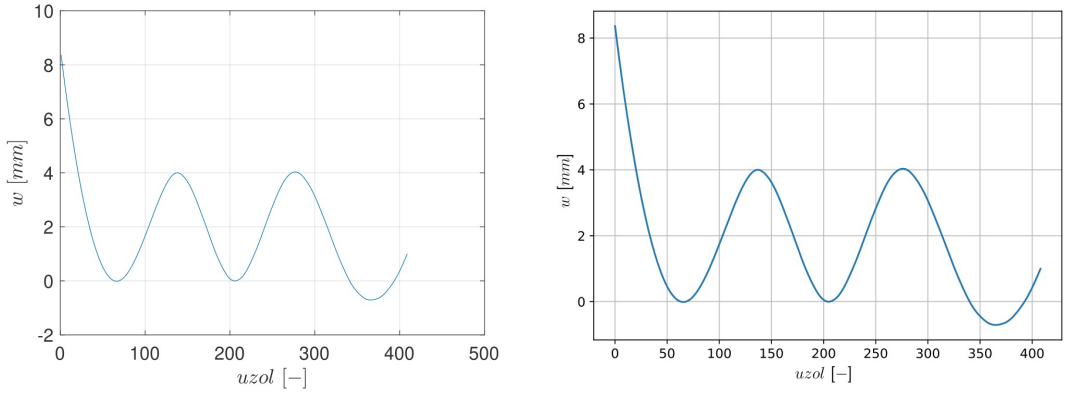
Tabuľka 4.1: Hodnoty vstupných parametrov jednotlivých verifikačných zadaní.

V obidvoch porovnávaných programoch sú vypočítané analýzy troch tyčí pri rôznych nastaveniach posuvov rovnacích valcov. Zvyšné parametre rovnačky budú konfigurované pre všetky zadania rovnako. Použité budú rotačné väzby jednobodové, bilineárny materiálový model bez spevnenia a konvergenčné kritériá nastavené na hranicu 0.001%. Konfigurácia rovnačky bude 9-valcová, používaná pôvodným algoritmom, pričom priehybový valec A bude deaktivovaný, podobne ako počas jeho validácie (viď schému na obrázku 4.3 a konfiguráciu na obrázku 4.5)

E	[GPa]	206
E_T	[GPa]	10
$v_{\mathrm{C,E,G}}$	[mm]	0
L_v	[mm]	475
v_s	[m/min]	30
β	[°]	30

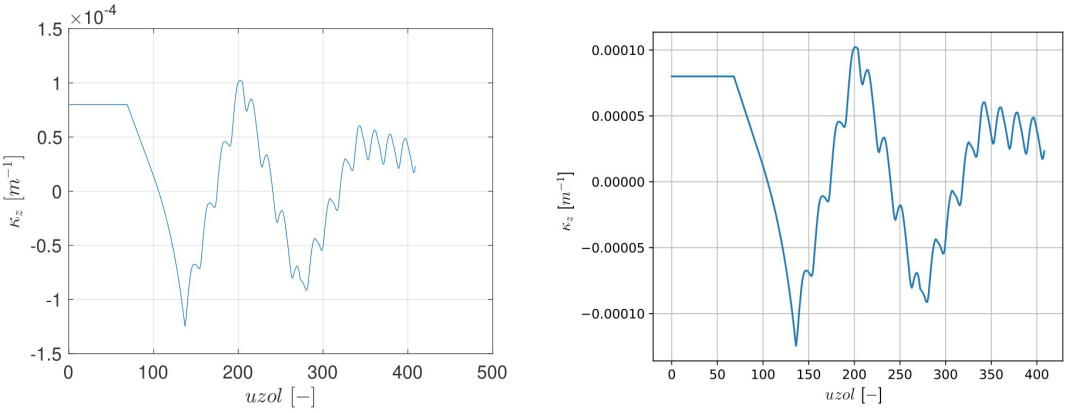
Tabuľka 4.2: Spoločné hodnoty vstupných parametrov verifikačných zadaní.

Grafy priebehov výstupných veličín budú vykresľované pre zadanie č. 2, v rovine 1.



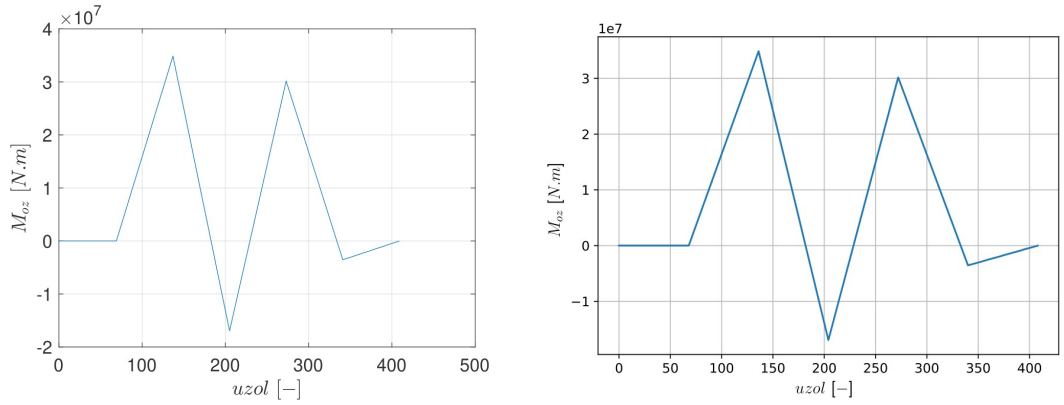
(a) MATLAB (b) Python

Obr. 4.6: Graf priebehov priehybov tyče.



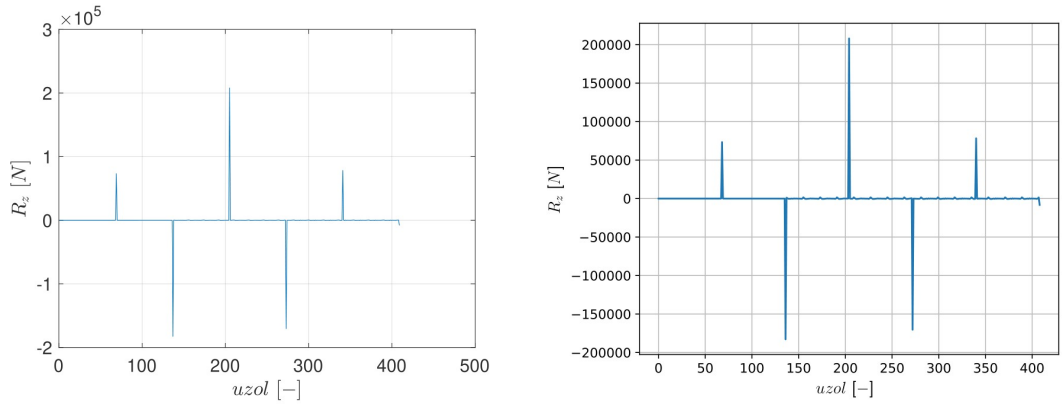
(a) MATLAB (b) Python

Obr. 4.7: Graf priebehov krivostí tyče.



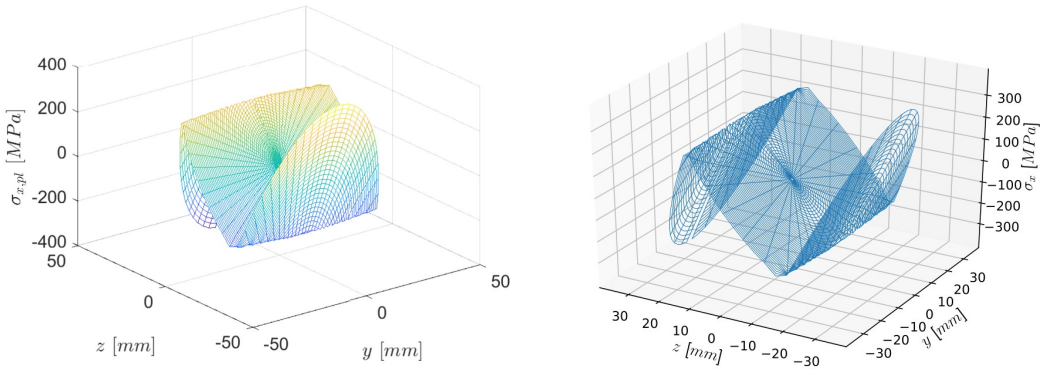
(a) MATLAB (b) Python

Obr. 4.8: Graf priebehov záťažných ohybových momentov tyče.



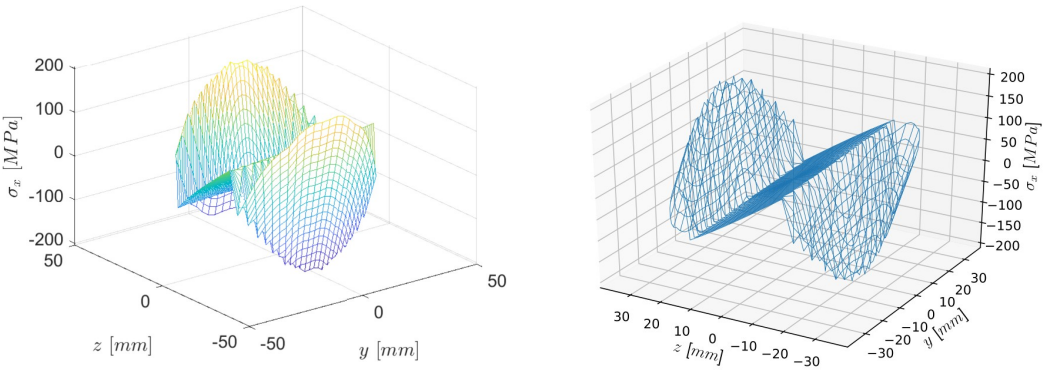
(a) MATLAB (b) Python

Obr. 4.9: Graf priebehov reakčných síl tyče.



(a) MATLAB (b) Python

Obr. 4.10: Graf zbytkových napätí v priereze tyče indukovaných vstupnou krivosťou.



(a) MATLAB (b) Python

Obr. 4.11: Graf napätí v priereze posledného uzlu tyče.

		Číslo zadania					
		1		2		3	
		m	py	m	py	m	py
$\sigma_{\text{rez,max}}$	[MPa]	146.0	146.1	219.1	219.1	288.8	288.8
R_{max}	[kN]	62.6	62.3	208.2	208.2	459.8	459.7
k	[mm/m]	1.16	1.16	3.96	3.96	3.84	3.84

Tabuľka 4.3: Výsledky verifikačných úloh.

Zadania používajú hodnoty vstupných veličín vypísané v tabuľke 4.4, zvyšné sú nastavené zhodne so zadaniami verifikačnými.

		Číslo zadania			
		4	5	6	7
d	[mm]	70	70	70	70
R_e	[MPa]	800	800	800	800
w_{C}	[mm]	0.05	0.05	0.05	0.05
w_{E}	[mm]	0.05	0.05	0.05	0.05
w_{G}	[mm]	0.05	10	20	30
k_0	[mm/m]	0	0	0	0

Tabuľka 4.4: Hodnoty vstupných parametrov jednotlivých ilustračných zadaní.

Rovnačka je v úlohe číslo 4 nastavená na veľmi malé posuvy jej rovnacích valcov, aby bol demonštrovaný prechod tyče strojom v elastickej oblasti, a teda bez zmeny krivosti. V ďalších úlohách sa postupne zvyšuje hodnota posuvu posledného rovnacieho valca s cieľom dosiahnutia plastizácie priečneho prierezu, čím sa v ňom indukuje reziduálne na-

pätie prejavujúce sa zvýšením pôvodne nulovej krivosti. S rastúcim priehybom by sa mala zvyšovať aj maximálna reakčná sila medzi tyčou a valcami.

		Číslo zadania			
		4	5	6	7
$\sigma_{\text{rez,max}}$	[MPa]	0	0	92.1	308.2
R_{max}	[kN]	2.3	83.8	164.8	206.1
k	[mm/m]	0	0	0.08	1.88

Tabuľka 4.5: Výsledky ilustračných úloh.

4.3. Porovnanie

Výsledky verifikačných príkladov sú zhrnuté v tabuľke 4.3 v predošlej časti. Hodnoty sa vzájomne mierne líšia, pravdepodobne kvôli rôznym implementáciám použitých interpolačných algoritmov. Grafy na obrázku 4.6–4.11 sú preto tiež prakticky totožné. Týmto je reimplementácia algoritmu považovaná za verifikovanú.

Verifikačné výpočty v prostredí MATLAB trvajú okolo 90 sekúnd, v Pythone okolo 100 sekúnd, záleží ale na počte iterácií. Kratší výpočtový čas MATLABu je spôsobený jeho schopnosťou paralelizácie výpočtu vďaka modulu Parallel Computing Toolbox. Súčasťou tohto modulu je knižnica LAPACK, ktorá v tomto prípade využíva viacero jadier procesora, výpočtovo náročné maticové operácie sú tak medzi ne rozdelené a vykonané rýchlejšie. Moduly umožňujúce paralelné výpočty na viacerých jadrách procesora sú implementované aj pre Python. Príkladom sú NumPy a SciPy, ktoré môžu byť pri manuálnej kompilácii zdrojového kódu nakonfigurované tak, aby využívali knižnice ako MKL, OpenBLAS alebo priamo LAPACK. Po ich inštalácii by sa výpočtový čas algoritmu v Pythone bližšie vyrovnal MATLABu.

Pre účely programovania neurónových sietí bol použitý modul umožňujúci vykonávanie maticových operácií na grafickej karte počítača. Kvôli väčšiemu počtu relatívne malých maticových rovníc algoritmu strávila grafická karta viac času sériovým kopírovaním dát z vyrovnávacej pamäte do internej pamäte a naspäť, čím sa výpočet oproti čisto procesorovej verzii ešte mierne predĺžil.

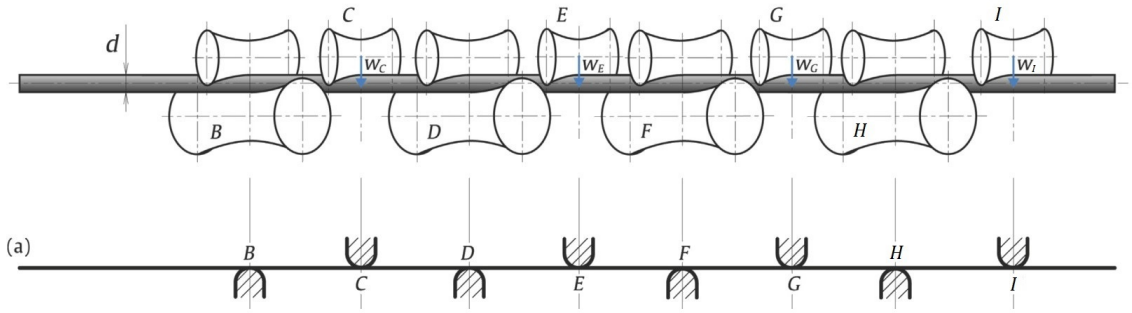
Prakticky najväčšie časové úspory by boli dosiahnuté paralelným výpočtom viacerých zadaní. Tento postup však znemožňuje GIL (Global Interpreter Lock), ktorý monitoruje všetky inštancie Pythonu a zabraňuje tomu, aby boli vykonávané súčasne. Naraz tak

môže byť spustených viacero výpočtov, interne sú však operačným systémom vykonávané po krátkych časových úsekoch sériovo.

Výsledky ilustračných úloh číslo 4 a 5 (z tabuľky 4.5) ukazujú, že tieto dva prípady sa pohybovali vrámci elastickej oblasti materiálu tyče, o čom svedčí nezmenená krivosť tyče a nulové reziduálne napätia jej prierezov. V úlohách číslo 6 a 7 už bola medza klzu presiahnutá, čo viedlo ku očakávanej zmene krivosti a napätia v tyči. Ďalej si môžeme všimnúť nelinearitu maximálnej reakčnej sily spôsobenú použitým bilineárnym modelom materiálu.

4.4. Zmena konfigurácie rovnacieho stroja

Pre účely ukážky funkcionality zadávania ľubovoľnej konfigurácie rovncieho stroja do reimplementovaného programu bude v tejto časti vypočítaná úloha s dvoma valcami navyše (označenými H a I, obrázok 4.12) oproti konfigurácii podporovanej v originálnom programe. Hodnoty vstupných parametrov sú definované v tabuľkách 4.6 a konkrétne vstupné súbory sú na obrázkoch 4.13 (vstupné hodnoty rozšírené v stĺpcoch `v` a `w` o príslušné hodnoty priehybov valcov) a 4.14 (konfigurácia rovnačky rozšírená o vzdialenosti medzi pridanými valcami a charaktermi väzieb). Rovnako ako počas verifikácie (časť 4.2) budú použité jednobodové väzby a bilineárny model materiálu bez spevnenia.



Obr. 4.12: Schéma rovnačky a väzieb (upravené z [4]).

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1				SEC	MAT				BC		IN						
2	done	result_filename	config	diameter	Re	Em	ET	ductility	v	w	k0j	HM	step	max_iter	ftoler	mtoler	utoler
3	[-]	[-]	[-]	[mm]	[MPa]	[MPa]	[MPa]	[%]	[mm]	[mm]	[mm/m]	[-]	[-]	[-]	[%]	[%]	[%]
4		alt0001	altconfig.xlsx	70	500	206000	10000	1	0,0,0,0,0,0,0,0,0	0,0,0,0,0.5,0,3,0,1.5	10	0	1	20	0.001	0.001	0.001
5		alt0002	altconfig.xlsx	70	800	206000	10000	1	0,0,0,0,0,0,0,0,0	0,0,0,0,5,0,3,0,2	10	0	1	20	0.001	0.001	0.001
6		alt0003	altconfig.xlsx	70	1200	206000	10000	1	0,0,0,0,0,0,0,0,0	0,0,0,0,7,0,7,0,3	5	0	1	20	0.001	0.001	0.001

Obr. 4.13: Súbor so vstupnými hodnotami.

	A	B	C	D	E	F	G
1	Lv	BCT	active		vs	rot_agn	reduction
2	[mm]	[-]	[-]		[m/min]	[°]	[-]
3	475	0	0		30	30	1.04167
4	475	0	1				
5	475	0	1				
6	475	0	1				
7	475	0	1				
8	475	0	1				
9	475	0	1				
10	475	0	1				
11		0	1				

Obr. 4.14: Súbor s rozšírenou konfiguráciou rovnačky.

d	[mm]	70
E	[GPa]	206
E_T	[GPa]	10
$v_{\mathrm{C,E,G,I}}$	[mm]	0
L_v	[mm]	475
v_s	[m/min]	30
β	[°]	30

		Číslo zadania		
		1	2	3
R_e	[MPa]	500	800	1200
w_{C}	[mm]	0	0	0
w_{E}	[mm]	0.5	5	7
w_{G}	[mm]	3	3	7
w_{I}	[mm]	1.5	2	3
k_0	[mm/m]	10	10	5

Tabuľka 4.6: Hodnoty vstupných parametrov zadania.

		Číslo zadania		
		1	2	3
$\sigma_{\mathrm{rez,max}}$	[MPa]	191.4	251.7	235.0
R_{max}	[kN]	308.7	616.1	787.0
k	[mm/m]	1.26	1.55	1.01

Tabuľka 4.7: Výsledky úlohy.

Výsledné hodnoty výstupných veličín úloh sú uvedené v tabuľke 4.7.

5. Strojové učenie

Táto kapitola sa zaoberá históriou a základnou teóriou vedeckej disciplíny strojového učenia. Ďalej bude predstavená základná teória neurónových sietí, ktoré sú jedným z algoritmov strojového učenia.

Strojové učenie je multi–oborová disciplína, čerpajúca z počítačových vied, matematiky, štatistiky, ekonómie, biológie, humanitných a spoločenských vied, psychológie a iných, a ako také nemá jednotnú definíciu. Môže byť definované ako súbor metód a algoritmov schopných automatickej analýzy dát a nachádzania vzorov, ktoré potom môžu byť použité na odhadnutie ďalších dát [10]. Alebo ako počítačový program, o ktorom hovoríme, že sa učí, ak zo skúseností s danou skupinou úloh a mierou úspešnosti sa jeho úspešnosť v týchto úlohách s nadobúdanými skúsenosťami zvyšuje [9]. Všeobecnejšie ide o snahu nahradenia klasicky štrukturovaných programov so striktne zadanými pravidlami za programy využívajúce štatistické metódy a teóriu pravdepodobnosti, ktorých správanie je určované naučenými dátami.

5.1. História

Výskum v oblasti strojového učenia je v dnešnej dobe veľmi intenzívny, s inovatívnymi teoretickými objavmi, ktoré sa veľmi rýchlo aplikujú v množstve aspektov každodenného života. Napriek tomuto modernému výzoru siahajú jeho základy až do 18. storočia, kedy Thomas Bayes a Richard Price položili základy teórie pravdepodobnosti formuláciou Bayesovej vety, ktorú neskôr prepracoval a uviedol do všeobecného povedomia Pierre Simon-Laplace. V tom istom čase vyvinul Adrien-Marie Legendre metódu najmenších štvorcov. V dvadsiatom storočí boli definované algoritmy ako Markovove reťazce, genetické algoritmy a prvé typy neurónových sietí. V roku 1951 postavil Marvin Minsky pravdepodobne prvý samostatne sa učiaci stroj zložený z navzájom prepojených elektrónok, pracujúci na princípe neurónových sietí, nazvaný SNARC. O 6 rokov neskôr vyvinul Frank Rosenblatt lineárny klasifikátor zvaný perceptrón, ktorý médiá a verejnosť považovali za algoritmus robotov budúcnosti. V knihe *Perceptrons* [13] však autori demonštrovali limitácie perceptrónu na niekoľkých jednoduchých príkladoch. Nasledoval prudký úpadok v záujme nielen o perceptróny, ale celý obor strojového učenia a umelej inteligencie, spolu s vážnymi dopadmi v ekonomickej sfére. Aby si verejnosť ľudí zaoberajúcich sa umelou inteligenciou prestala spájať s týmito udalosťami, dostal obor neurónových sietí nové meno, a to hlboké učenie (Deep Learning). Odvtedy boli vynájdené mnohé varianty pôvodného algoritmu neurónových sietí špecializujúce sa na čoraz ambicióznejšie technické problémy.

Príkladom sú rekurentné neurónové siete, schopné pracovať s časovo dynamickými dejmi alebo konvolučné neurónové siete, používané v oblasti rozpoznávania objektov z obrázkov.

Pokrok v úrovni algoritmov strojového učenia a umelej inteligencie sa dá ukázať vývojom programov na hranie hier. V päťdesiatych rokoch minulého storočia napísal Arthur Samuel program hrajúci dámy. Kvôli nízkym kapacitám pamäte vtedajšieho hardvéru ho aj po viacerých optimalizáciách bol schopný poraziť aj amatér. V rovnakom čase vyvinul Donald Michie algoritmus pre optimalizáciu stratégie pri hraní piškvoriek. Nepoužil však počítač, ale zápalkové krabičky a farebné guličky, reprezentujúce rôzne stavy hracej dosky a predikciu najlepšieho ďalšieho ťahu, ktorá sa menila na základe výsledku hry. V roku 1992 vytvoril Gerald Tesauro vo výzkumnom centre IBM program TD-Gammon na hranie backgammonu. Ten už je schopný hrania na vysokej úrovni konkurujúcej, ale neprevyšujúcej najlepších profesionálnych hráčov. To sa podarilo v roku 1997, kedy IBM Deep Blue porazil v šachu majstra sveta Garryho Kasparova. Firme IBM sa potom v 2011 podarilo ich počítačom Watson poraziť 2 šampiónov americkej kvízovej šou Jeopardy. Jedným z najnovších mediálne známych úspechov strojového učenia zahŕňa porážku profesionálneho hráča v hre Go programom AlphaGo, vyvíjanom spoločnosťou Google. Menej známa je séria výhier ich programu DeepMind nad profesionálnym hráčom v hre považovanej za jednu z najnáročnejších stratégií, Star Craft II. Ďalším príkladom je online hra pre 10 hráčov, Dota 2, pre ktorú výskumné laboratórium OpenAI vyvinulo program schopný vyhrať nad profesionálnymi hráčmi v móde jeden proti jednému, a neskôr aj plnom móde 5 proti 5. Na prekonanie týchto prekážok sa museli algoritmy zdokonaliť vo veľa aspektoch riešenia komlexných problémov od spracovávania prirodzeného jazyka a reči po zapamätávanie, plánovanie a spoluprácu v dynamických prostrediach.

5.2. Základná teória

Podľa prístupu a vstupných dát sa strojové učenie rozdeľuje na kontrolované, nekontrolované a menej používané spätnoväzbové. V prípade kontrolovaného učenia sú výstupy algoritmu porovnávané s hodnotami, ktoré sa má naučiť. Tento typ učenia sa ďalej rozdeľuje na klasifikáciu s konečným počtom tried (napr. rozpoznávanie objektov v obrázkoch), a regresiu, ktorej výstupná funkcia je spojitá (napr. predpovedanie cien na burze). Pri nekontrolovanom učení tieto správne hodnoty nie sú k dispozícii a program musí závery vyvodiť sám. Používa sa na úlohy zhlukovania a asociácií.

Cieľom kontrolovaného strojového učenia je s pomocou vstupného datasetu D naučiť program funkciu $h(\mathbf{x})$, ktorej tvar závisí od konkrétnej použitej metódy [11].

$$D = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots (\mathbf{x}_n, \mathbf{y}_n)\} \subseteq \mathbb{R}^d \times \mathcal{C} \tag{5.1}$$

V rovnici 5.1 predstavuje d rozmer vstupného priestoru, $\mathbf{x}_i$ je vektor vstupných hodnôt a $\mathbf{y}_i$ vektor správnych výstupných hodnôt, $\mathcal{C}$ je priestor výstupných hodnôt, ktorý môže obsahovať konečný počet hodnôt pre klasifikačné algoritmy alebo nekonečný počet hodnôt pre regresiu. Vzorky dát $(\mathbf{x}, \mathbf{y})$ sú vyberané z neznámeho rozdelenia $\mathcal{P}(X, Y)$. Pre funkciu h chceme aby platilo, že nový pár $(\mathbf{x}, \mathbf{y}) \sim \mathcal{P}$ a teda $h(\mathbf{x}) \approx \mathbf{y}$.

Pri výbere funkcie h musíme o probléme, na ktorý bude táto funkcia nasadená, učiniť isté predpoklady. Takzvaný *No Free Lunch Theorem* konštatuje, že každý užitočný algoritmus strojového učenia musí o probléme niečo predpokladať, čo znamená že neexistuje žiadny, ktorý by bol univerzálne aplikovateľný. Konkrétne znenia týchto predpokladov záležia na triede hypotéz $\mathcal{H}$, do ktorej h patrí.

5.3. Chybové funkcie

Proces učenia funkcie h typicky pozostáva z dvoch častí. Prvým krokom je výber vhodného algoritmu, čo definuje konkrétnu triedu hypotéz $\mathcal{H}$ predstvujúcu množinu všetkých naučiteľných funkcií. Druhým krokom je hľadanie funkcie $h \in \mathcal{H}$ najlepšie popisujúcej daný problém, prakticky takej, ktorá na trénovacom datasete robí najmenšie chyby a je čo najjednoduchšia (má minimálny počet parametrov). Práve tento krok je označovaný ako učenie a spravidla zahŕňa optimalizačný problém. Na vyhodnocovanie odlišnosti modelu od reality funkcií h sa používajú chybové funkcie $\mathcal{L}$. Čím vyššia chyba $\ell(h)$ pre $\ell \in \mathcal{L}$, tým je h horšia v popisovaní datasetu. Chybové funkcie musia byť nezáporné a nulovú hodnotu nadobúdajú iba v prípade, že sa predikcia zhoduje so žiadanou hodnotou. Základnými chybovými funkciami sú absolútna, logistická alebo kvadratická pre regresné modely a 0-1 alebo exponenciálna pre klasifikátory.

Pre vybranú chybovú funkciu ℓ, hľadaná funkcia h minimalizuje chybu:

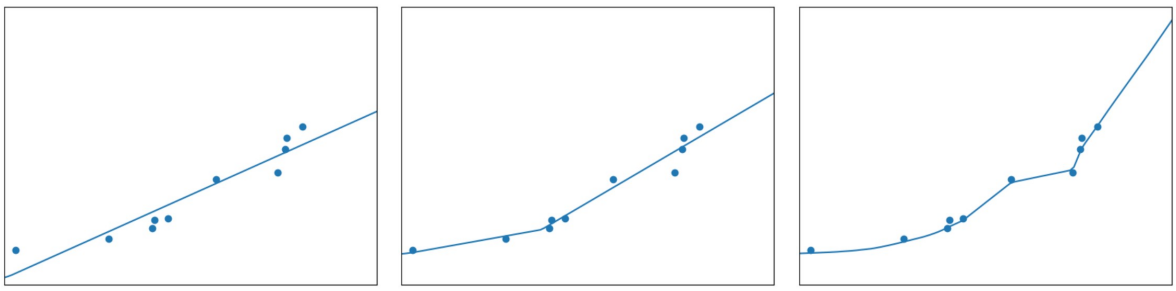
$$h = \text{argmin}_{h \in \mathcal{H}} \ell(h) \tag{5.2}$$

Veľká časť disciplíny strojového učenia sa zaoberá spôsobmi, akými sa dá táto minimalizácia účinne vykonávať. Malá hodnota chybovej funkcie h však automaticky neznamená, že je natrénovaný model prakticky použiteľný. Dôvodom je skutočnosť, že ak je model vstupným dátam vystavený príliš veľakrát, presne sa naučí trénovacie hodnoty, čo vedie k nulovej trénovacej chybe. Pre iné vstupné hodnoty však býva chyba veľká. Prakticky si tak iba zapamätá trénovacie dáta a mimo nich je nepoužiteľný.

Aby sa predišlo prehnanému zapamätaniu vstupných dát, rozdeľuje sa dataset na tri časti. Prvá, najväčšia, sa používa na trénovanie modelu. Periodicky sa vyhodnocuje chyba

aj na druhej, verifikačnej časti. Hotový model sa na konci ohodnotí s použitím poslednej, testovacej časti datasetu. Najčastejšie sa rozdeľuje v pomere 80% - 10% - 10%. Ak sú dáta časovo závislé, musia byť podľa času aj rozdelené. Ak nie, používa sa rovnomerné rozdelenie, kedy testovacia chyba najlepšie aproximuje skutočné chyby daného rozdelenia datasetu. Vyhodnocovanie pomeru trénovacej a validačnej chyby pomáha identifikovať základné nedostatky nastavenia procesu učenia. Ak je trénovacia chyba veľmi nízka a validačná vysoká, tj. pomer trénovacej ku validačnej chybe je veľmi nízky, ide o prípad už spomenutého zapamätávania vstupov. Tento prípad sa dá napraviť zjednodušením modelu, čiže znížením počtu trénovateľných parametrov alebo použitím väčšieho datasetu. Opačný je prípad, ak sú obe chyby vysoké a ich pomer je blízky jednej. Model vtedy nie je schopný dataset detailne postihnúť a produkovať tak presné predikcie. Náprava tohto stavu spočíva v použití komplexnejšieho modelu. Ideálny je pomer chýb približne rovný jednej pričom sú jednotlivé hodnoty chýb pod prijateľnou hranicou.

Chyba, ktorej sa model pri predikciách dopúšťa, sa dá rozdeliť na tri základné typy. Prvý zachytáva zmenu chyby pre rôzne inštancie učeného datasetu z rovnakého rozdelenia, inými slovami meria ako veľmi je model špecilizovaný na daný dataset. Druhý popisuje chybu, ktorú by mal model aj pri nekonečne veľkom datasete. Vzniká kvôli učineným predpokladom pri voľbe typu algoritmu strojového učenia a tvorbe modelu, napríklad lineárny model nie je schopný popísať obecné krivky a plochy, iba priamky a roviny. Rozsah tejto časti chyby je teda ovplyvniteľný použitým algoritmom a komplexnosťou modelu. Posledná časť chyby je spôsobená vnútorným šumom dát, nie je závislá na type algoritmu alebo komplexnosti modelu. Jediným spôsobom, ako túto časť chyby znížiť, je zvýšenie kvality vstupných dát.



(a) Jednoduchý model. (b) Primeraný model. (c) Zložitý model.

Obr. 5.1: Rôzne úrovne zložitosti modelov strojového učenia.

Na grafoch z obrázku 5.1 sú uvedené príklady komplexnosti predikčného modelu, trénované na zašumenú exponenciálnu funkciu. Model na obrázku 5.1a je príliš jednoduchý pre daný trénovací dataset a skresľuje priebeh aproximovaného priebehu exponenciály. Naopak 5.1c sa snaží dataset kopírovať čo najdetailnejšie, čo však pri testovaní spôsobí

chybu oveľa väčšiu ako pri tréningu, pretože naučená funkcia nemá žiadaný tvar exponenciály a šum to ešte zhorší. Medzistupňom týchto dvoch extrémov je varianta zobrazeá na obrázku 5.1b, ktorá sa snaží aproximovať priemerný priebeh funkcie bez toho, aby zachádala do kontraproduktívnych detailov.

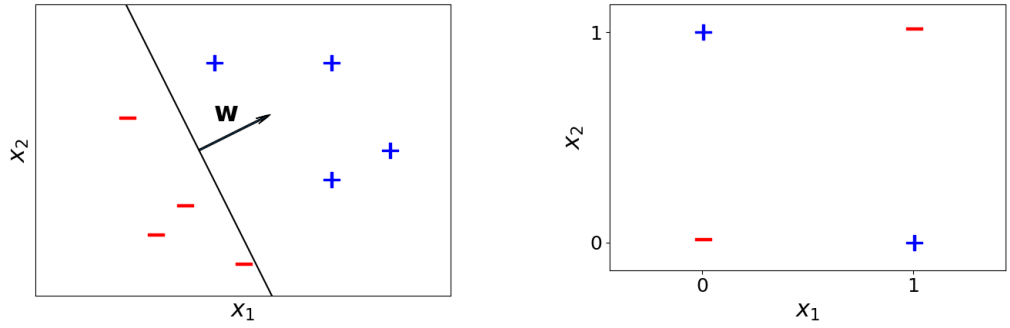
Pri používaní natrénovaných modelov na predikcie je potrebné dbať na to, že dotazované dátové body musia ležať vnútri definičného oboru datasetu, pretože extrapolované hodnoty z princípu nemali byť podľa čoho optimalizované a sú teda prakticky náhodné.

5.4. Neurónové siete

Princíp algoritmu neurónových sietí je založený na pozorovaniach biologických sústav a ich schopnosti samostatného učenia. Vo zvieratách a ľuďoch túto funkciu zastáva mozog pozostávajúci z veľkého množstva mozgových buniek – neurónov [14]. Tieto bunky sa navzájom spájajú takzvanými synapsiami, umožňujúcimi elektrochemickému prechodu elektrického potenciálu a teda prechodu informácií. Neuróny môžu mať viaceré synaptické spojenia slúžiace ako vstupy a ich výstup taktiež môže byť napojený na viacero ostatných neurónov. Neuróny fungujú ako nelineárna logická brána závislá práve na intenzitách a počte vstupných signálov. Neurobiologický proces učenia je vysvetľovaný ako zmena citlivostí jednotlivých synapsií a hraničných hodnôt intenzít signálov potrebných pre aktiváciu buniek, za účelom modifikácie správania organizmu. Významný podiel na objasňovaní mechanizmu učenia v ľuďoch má Donald O. Hebb, ktorý prišiel na to, že často vybudzované synapsie silnejú, čím sa komunikácia medzi danými dvoma neurónmi stáva efektívnejšou [15]. Tento základný princíp sa využíva v psychológii, neurobiológii a je na ňom založených niekoľko úspešných mechanizmov optimalizácie umelých neurónových sietí.

Snaha o detailné kopírovanie biologických sietí neurónov do programovej implementácie na všeobecnom výpočtovom hardvéri je však neefektívna. Jedným z najvýraznejších rozdielov medzi organickými a anorganickými procesormi je ich architektúra a z nej vyplývajúce vlastnosti. Ľudský mozog odhadom obsahuje okolo 10^{11} neurónov, s počtom synapsií vyšším ešte o niekoľko rádov, pracujúcich s frekvenciou okolo 100 Hz. Oproti tomu počítače využívajú centrálne procesorové jednotky s desiatkami jadier s taktovacou frekvenciou na úrovni 1-5 GHz. V poslednej dobe sa práve kvôli zvýšenej popularite strojového učenia začali vyrábať grafické karty schopné vykonávať obecné výpočty, ktoré so svojimi tisícami jadier značne zvyšujú možnú úroveň paralelizácie, voči mozgom však je to však stále zanedbateľné.

Priamym predchodcom moderných neurónových sietí je algoritmus známy ako perceptrón. Ide o lineárny binárny klasifikátor, čo znamená, že v prípadoch, kedy vstupný dataset obsahuje členy s dvoma ľubovoľnými hodnotami, často v tvare $\mathbf{y} \in \{-1, +1\}$, ktoré sú v danom priestore oddeliteľné lineárnou nadrovinou, túto nadrovinu nájde. Tento algoritmus má ako pravdepodobne prvý svojho druhu formálne zaručenú konvergenciu v konečnom počte krokov, samozrejme za predpokladu platnosti uvedených podmienok. Pre potrebný počet krokov do konvergencie sa dokonca dá analyticky vyhodnotiť horná hranica. Prvotná formulácia perceptrónu by však pri nesplnenej podmienke lineárnej separability iterovala donekonečna.



(a) Lineárne separabilný dataset. (b) Funkcia XOR.

Obr. 5.2: Jednoduché datasety.

Predpis funkcie $h(\mathbf{x})$ perceptrónu:

$$h(\mathbf{x}) = \text{sign}(\mathbf{w}^T\mathbf{x} + b) \tag{5.3}$$

Kvôli jednoduchšej manipulácii s touto funkciou sa člen b, slúžiaci na posunutie nadroviny mimo počiatok súradnicového systému, zapisuje spolu s vektorom váhových parametrov $\mathbf{w}$ do jednej premennej. Aby predpis funkcie naďalej platil, rozširuje sa aj vektor vstupných hodnôt $\mathbf{x}$ o konštantu 1:

$$\mathbf{x} \longmapsto \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} \tag{5.4}$$

$$\mathbf{w} \longmapsto \begin{bmatrix} \mathbf{w} \\ b \end{bmatrix} \tag{5.5}$$

Jednoduchou verifikáciou platnosti vzťahu

$$\begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}^T \begin{bmatrix} \mathbf{w} \\ b \end{bmatrix} = \mathbf{w}^T\mathbf{x} + b \tag{5.6}$$

potom môžeme zapísať funkciu h pre perceptrón v tvare:

$$h(\mathbf{x}) = \text{sign}(\mathbf{w}^T\mathbf{x}) \tag{5.7}$$

Súčin $\mathbf{w}^T\mathbf{x}$ je lineárna operácia, $h(\mathbf{x})$ je však aplikáciou funkcie sign nelineárnou, nadobúdajúcou tri hodnoty

$$h(\mathbf{x}) = \begin{cases} 1 & \text{pre predikciu zaradenia vzorky do 1. triedy} \\ 0 & \text{ak predikcia leží na deliacej nadrovine} \\ -1 & \text{pre predikciu zaradenia vzorky do 2. triedy} \end{cases} \tag{5.8}$$

Na obrázku 5.2a je znázornený lineárne separabilný binárny dataset, na ktorom vie perceptrón nájsť nadrovinu rozdeľujúcu vzorky podľa ich hodnôt. Znázornený je aj rozšírený vektor $\mathbf{w}$, definujúci túto nadrovinu. Ide však iba o jednu inštanciu z nekonečného množstva platných nadrovín, ktoré môžu byť pre daný dataset zvolené. Varianta perceptrónu, známa ako support-vector machine (SVM), maximalizuje najkratšiu vzdialenosť vzoriek z oboch tried ku deliacej nadrovine, čo je jej najrobustnejšia inštancia. Základný perceptrón túto optimalizáciu nevykonáva, jeho algoritmus učenia sa ukončuje akonáhle nájde ľubovoľnú zo všetkých možných deliacich nadrovín.

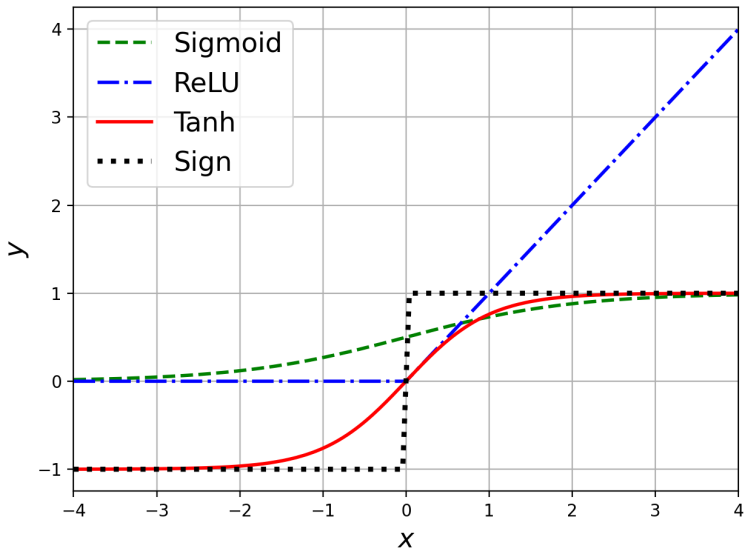
Perceptrón preto principiálne nie je schopný nájsť lineárnu deliacu nadrovinu v tak jednoduchých funkciách, akou je napríklad XOR (obrázok 5.2b). Práve túto funkciu použil Marvin Minsky pri svojej demonštrácii limitácií aplikovateľnosti tohto algoritmu. Riešením je použitie modelu s viacvrstvou konfiguráciou perceptrónov, pre ktorú ale v tej dobe nebol známy iný spôsob trénovania, ako náhodné zmeny jeho parametrov, čo niekoľko rokov predstavovalo prekážku v ďalšom výskume tejto technológie. Keď sa neskôr prišlo na vhodný algoritmus, tzv. backpropagation (BP), alebo metódu spätného šírenia, vyšlo najavo, že bol nezávisle objavený už skôr, nebol však aplikovaný na perceptróny.

Medzikrokom medzi perceptrónom a neurónovou sieťou je tzv. multilayer perceptron (MLP), čiže viacvrstvý perceptrón. Ide o sieť navzájom poprepájaných perceptrónov usporiadaných vo viacerých vrstvách, avšak namiesto funkcie sign v rovnici 5.7, ktorá nie je spojitá a jej derivácia (a teda aj gradient) je po častiach nulová, používajú funkcie s vhodnejšími vlastnosťami, najčastejšie hyperbolický tangens alebo tzv. sigmoid (rovnica 5.9). Vďaka nenulovým priebehom derivácií týchto funkcií vie algoritmus BP vypočítať gradienty chybovej funkcie vzhľadom k parametrom deliacej nadroviny $\mathbf{w}$, tiež označovaným ako váhy. Takto upravené perceptróny sú usporiadané do troch vrstiev, vstupnej, vnútornej a výstupnej, kde všetky perceptróny, až na tie nachádzajúce sa vo vstupnej vrstve, používajú aktivačnú funkciu. Pre regresné úlohy používajú neuróny vo výstupnej vrstve lineárnu aktivačnú funkciu, pretože výstupy musia v tomto prípade nadobúdať

neobmedzené hodnoty. Vstupná vrstva pozostáva z toľkých perceptrónov, koľko dimenzií má vektor vstupných veličín, počet výstupných perceptrónov potom zodpovedá počtu dimenzií výstupného vektoru. Perceptrónov vo vnútornej vrstve môže byť ľubovoľne veľa, prakticky sa používa najmenšie množstvo pri dosiahnutí želanej presnosti predikcií. Viacvrstvá konfigurácia umožňuje deliacej nadrovine byť nelineárnou, čím vie postihnúť viac problémov a ich datasetov.

$$\text{sigmoid}(x) = \frac{1}{1 + \exp(-x)} \tag{5.9}$$

$$\text{ReLU}(x) = \max(x, 0) \tag{5.10}$$



Obr. 5.3: Príklady niekoľkých aktivačných funkcií.

Pojem neurónová sieť dnes slúži na zastrešenie MLP ako jeho podmnožinu, na ktorej boli ostatné typy založené a upravené pre použitie v špecializovaných aspektoch strojového učenia. Tie sa líšia hlavne počtom vnútorných vrstiev, a často používajú komplexnejšie typy neurónov.

Pri matematickom odvodzovaní začneme s rovnicou lineárneho klasifikátora s rozšírenými parametrami $\mathbf{x}$ a $\mathbf{w}$ podľa rovníc 5.4 a 5.5 [12]:

$$h(\mathbf{x}) = \mathbf{w}^T\mathbf{x} \tag{5.11}$$

Tento výraz je však potrebné dostať do nelineárnej podoby, na čo sa používa tzv. kernelizácia. V oblasti počítačových vied ide o proces vytvorenia mapovania vstupov do priestoru s vyššou dimenziou, ktoré zodpovedajú vzájomným interakciám vstupov, čím vznikne li-

neárna nadrovina v priestore s vyššou dimenziou, čo sa v pôvodnom priestore prejaví jej nelinearzáciou. Toto mapovanie sa často označuje symbolom ϕ.

$$\mathbf{x} \mapsto \phi(\mathbf{x}) \tag{5.12}$$

Časť strojového učenia sa zaoberá návrhom týchto mapovaní, ich aplikáciou na lineárne klasifikátory a vyhodnocovaním ich použiteľnosti na rôzne druhy praktických problémov. Hlavným dôvodom používania takto upravených lineárnech klasifikátorov je konkávnosť chybových funkcií, čo pre neurónové siete neplatí. Neurónové siete na toto mapovanie používajú súbory nadrovín, kedy je každá z nich reprezentovaná neurónom vo vnútorných vrstvách:

$$\phi(\mathbf{x}) = \sigma(\mathbf{U}\mathbf{x} + \mathbf{c}) \tag{5.13}$$

$$\mathbf{U} \in \mathbb{R}^{h \times d} \tag{5.14}$$

$$\mathbf{c} \in \mathbb{R}^{h} \tag{5.15}$$

kde σ predstavuje aktivačnú funkciu (v dnešnej dobe prevažne ReLU), matica $\mathbf{U}$ a vektor $\mathbf{c}$ sú parametre lineárnych nadrovín, ktorých hodnoty sú trénovateľné a rozšíriteľné rovnakým spôsobom, ako váhy $\mathbf{w}$ (rovnice 5.4 a 5.5). h je počet neurónov vo vnútornej vrstve a d je počet vstupov. Dosadením rovnice 5.12, resp. 5.13 do 5.11 vzniknú

$$h(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) \tag{5.16}$$

$$h(\mathbf{x}) = \mathbf{w}^T \sigma(\mathbf{U}\mathbf{x} + \mathbf{c}) = \mathbf{w}^T \sigma(\mathbf{U}\mathbf{x}) \tag{5.17}$$

Rovnica 5.17 je matematickým predpisom neurónovej siete s jednou vnútornou vrstvou. Pre získanie predpisu neurónových sietí s ľubovoľným počtom vrstiev sa definuje mapovacia funkcia ϕ (z rovnice 5.13) pre každú vrstvu, ktoré sa do seba postupne vnárajú. Nasledujúci predpis je definovaný pre trojvrstvú sieť, princíp je ale aplikovateľný všeobecne:

$$\phi(\mathbf{x}) = \sigma(\mathbf{U}\, \phi'(\mathbf{x})) \tag{5.18}$$

$$\phi'(\mathbf{x}) = \sigma(\mathbf{U}'\, \phi''(\mathbf{x})) \tag{5.19}$$

$$\phi''(\mathbf{x}) = \sigma(\mathbf{U}''\mathbf{x}) \tag{5.20}$$

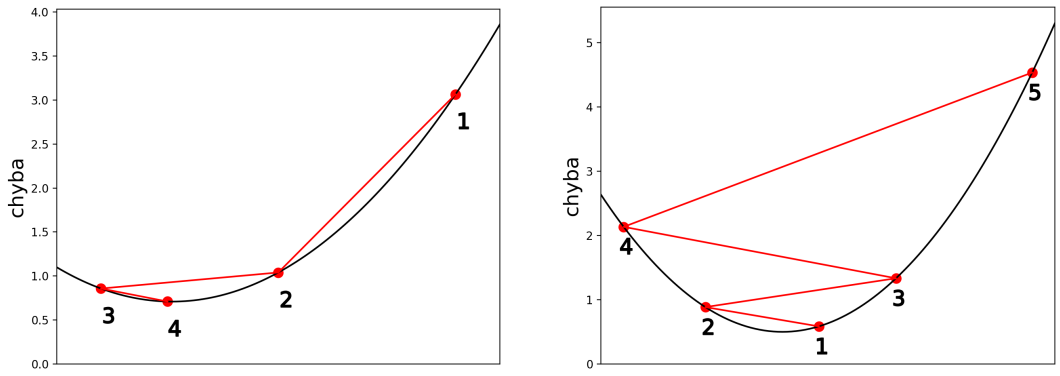
kde 5.18–5.20 popisujú prvú až tretiu vnútornú vrstvu. Rovnica 5.17 sa pre tri vnútorné vrstvy dá prepísať do tvaru

$$h(\mathbf{x}) = \mathbf{w}^T\, \sigma(\mathbf{U}\, \sigma(\mathbf{U}'\, \sigma(\mathbf{U}''\mathbf{x}))) \tag{5.21}$$

Algoritmus BP sa v procese trénovania siete používa na výpočet gradientov chybovej funkcie vzhľadom k váham neurónov. Postupuje od výstupnej vrstvy po vstupnú,

pričom využíva reťazové pravidlo. Gradienty pritom počíta pre celé vrstvy, čo je efektívnejšie a rýchlejšie ako vyhodnocovanie samostatných neurónov, postup používaný pred vynájdením BP. Zariadenia a softvérové knižnice navyše začínajú podporovať automatické derivovanie. Gradient chybovej funkcie siete sa používa pri hľadaní vhodných váh funkcie h podľa rovnice 5.2, kde na ňom pracuje vybraný optimalizačný algoritmus. Ten sa snaží zmapovať chybový priestor modelu, nájsť a dostať sa do oblasti minima. V prípade lineárneho perceptrónu mal chybový priestor tvar paraboloidu, ktorý má iba jeden extrém, a to globálne minimum. Nezáleží teda na počiatočnej pozícii v tomto priestore na začiatku trénovania, zakaždým je dosiahnuté to isté globálne minimum. Vrstvením neurónov sa však chybový priestor stáva nelineárnym s množstvom lokálnych miním. Rôzne počiatočné polohy môžu nájsť a zaseknúť sa v rôznych neoptimálnych lokálnych minimách. Váhy viacvrstvých sietí sa preto často inicializujú náhodne.

Najjednoduchším používaným optimalizačným algoritmom je gradient descent (GD). Využíva prvý stupeň gradientu $g(\mathbf{w}) = \nabla\ell(\mathbf{w})$ chybovej funkcie ℓ a predpokladá, že je v daných bodoch lineárna s predpisom $\ell(\mathbf{w})+g(\mathbf{w})^T\mathbf{s}$. Vektor $\mathbf{s}$ sa potom nastavuje podľa rovnice $\mathbf{s} = -\alpha g(\mathbf{w})$, kde α je takzvaná learning rate (LR), prekladaný ako koeficient učenia. α je jedným z hyperparametrov, čo znamená, že musí byť nastavená manuálne. To ale nie je triviálna úloha. Príliš malá hodnota síce zlepšuje šance na konvergenciu, celý proces učenia však môže byť zbytočne pomalý (obrázok 5.4a). Naopak príliš vysoká hodnota môže spôsobiť, že dostatočne úzke oblasti s minimami budú preskočené; v prípade, že takúto oblasť predsa len nájde, ju po niekoľkých iteráciách pravdepodobne opustí, alebo začne divergovať úplne (obrázok 5.4b). Problém s manuálnym nastavovaním koeficientu učenia rieši napríklad optimalizačný algoritmus Adagrad, skratka pre Adaptive Gradient algorithm. Tento názov dobre vystihuje rozdiel oproti klasickému GD, ktorým je automatické nastavovanie hodnôt α, dokonca pre každú váhu zvlášť.



(a) Dostatočne nízka α.

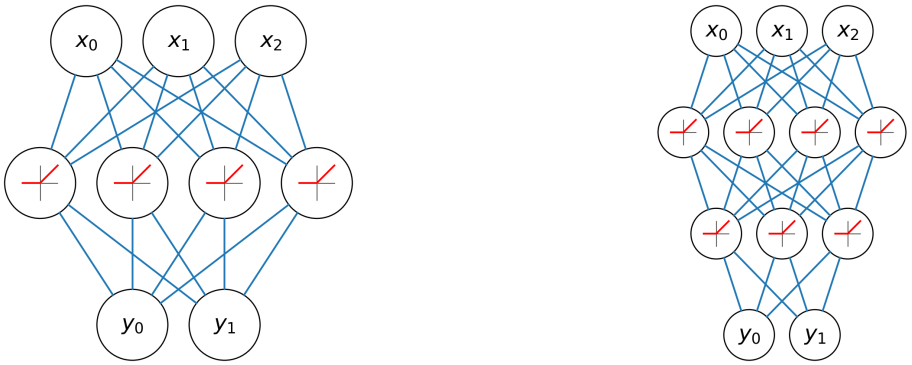
(b) Príliš vysoká α.

Obr. 5.4: Diagramy procesu učenia pre rôzne koeficienty učenia α.

Ďalšou nevýhodou klasického optimalizátoru GD je jeho citlivosť na šum vo vstupnom datasete, pretože upravuje váhy po každej iterácii vzorky vstupného datasetu. Riešenie tohto problému môže byť pomerne neintuitívne. Spočíva vo výpočte viacerých predikcií výstupu z náhodne vybraných vstupných vzoriek, na ktoré aplikuje agregačnú funkciu (najčastejšie max, mean alebo sum). Váhy potom upravuje až vzhľadom k takto získanej várke (batch). Tento prístup však zavádza nový hyperparameter – batch size, alebo veľkosť várky, ovplyvňujúci rýchlosť konvergencie učenia a mieru zovšeobecňovania vstupov. Takto modifikovaný algoritmus GD sa nazýva Stochastic Gradient Descent (SGD), čiže stochastický GD.

Pri príliš nízkych hodnotách α je optimalizačný algoritmus náchylný spadnúť do úzkeho a plytkého lokálneho minima a dokonvergovať v ňom. Na prevenciu týchto prípadov je zavádzané tzv. momentum, alebo hybnosť algoritmu počas učenia, čo je lineárna kombinácia priebežne ukladaných hodnôt zmien váh, ktorá spolu s aktuálnou hodnotou gradientu určuje veľkosť zmeny hodnôt váh pre danú iteráciu. Pri správnom nastavení tak napomáha urýchliť učenie modelu tým, že mu hybnosť pomáha pretlačiť sa cez dostatočne malé a plytké lokálne minimá. Ako obvykle to však prináša ďalší hyperparameter, tentokrát reprezentujúci tzv. decay factor, určujúci pomer vplyvov aktuálneho a predošlých gradientov na veľkosť zmeny váh.

Ako už bolo spomenuté, viacvrstvá konfigurácia neurónových sietí umožňuje ich deliacej nadrovine nadobúdať nelineárny tvar ľubovolnej zložitosti, limitovaný iba typom a počtom použitých neurónov. Vďaka tejto skutočnosti sa dajú neurónové siete používať aj na regresné úlohy, kedy hodnoty veličín zo vstupného datasetu patria do množiny reálnych čísel. Neurónová sieť pozostávajúca zo vstupnej, vnútornej a výstupnej vrstvy je schopná aproximovať ľubovoľnú funkciu na ľubovoľnú úroveň presnosti. Potrebný počet vnútorných neurónov však so zvyšujúcou sa úrovňou presnosti rastie exponenciálne, čo pre ich praktické používanie predstavuje veľké obmedzenie. Znížiť požiadavku na počet potrebných neurónov pri zachovaní presnosti sa dá dosiahnuť práve pridaním viacerých vnútorných vrstiev. V roku 2017 bolo dokázané, že neurónové siete s obmedzeným počtom neurónov vo vnútorných vrstvách sú univerzálnymi aproximátormi [16], ak nie je počet ich vnútorných vrstiev limitovaný. Konkrétne bolo dokázané, že siete o šírke d+4 (d je počet vstupných veličín siete), ktorej neuróny používajú aktivačnú funkciu ReLU (z rovnice 5.10) sú schopné aproximovať ľubovoľnú Lebesgueovsky integrovateľnú funkciu. Taktiež bolo dokázané, že pre aproximáciu spojitých funkcií postačuje šírka d+1. Tieto dôkazy síce zaručujú existenciu dostatočne blízkej aproximácie ľubovoľnej funkcie, nezaoberajú sa však obtiažnosťou ich naučenia.



(a) Jedna vnútorná vrstva. (b) Dve vnútorné vrstvy.

Obr. 5.5: Diagramy neurónových sietí s aktivačnou funkciou ReLU.

Obe tieto požiadavky na hĺbku siete sú na rozdiel od exponenciálnej požiadavky na jej šírku polynomické. To sa dá vysvetliť nasledovne: prvá vnútorná vrstva používa nemodifikované vstupy na konštrukciu nadroviny zloženej z primitívnych útvarov, ktoré potom slúžia ako vstupy pre neuróny v druhej vnútornej vrstve. Ich základnou stavebnou jednotkou sú útvary vytvorené predošlou vrstvou, ktoré už nie sú primitívne. Druhá vrstva svoje modifikácie posunie tretej vrstve, atď.

Vstupy pre čoraz hlbšie vrstvy preto postupne naberajú na komplexnosti. Týmto spôsobom vzniká množstvo kombinácií, akými sa môžu vrstvy ovplyvňovať a expresívna sila siete je preto, aj pri zachovaní pomerne malého počtu neurónov, veľká.

6. Optimalizácia

Táto kapitola sa bude zaoberať využitím reimplemenácie rýchleho algoritmu z časti 4.1 na vytvorenie skúšobných datasetov. Jedna skupina datasetov bude používať originálnu konfiguráciu rovnacieho stroja (ako počas verifikácie v časti 4.2), druhá časť bude rozšírená o dva rovnacie valce (časť 4.4). Na tieto datasety (ďalej označované ako std a alt) potom budú natrénované neurónové siete, ktoré budú použité v jednoduchom vizualizačnom a optimalizačnom programe.

6.1. Príprava datasetu

Prvým krokom pri tvorbe datasetu je vykonanie samotných konečnoprvkových výpočtov. Parametre sa zapíšu do vstupného súboru formátu `.xlsx` (viď obrázok 4.4). Pre prípady nutnosti vypočítania veľkých množstiev zadaní uľahčuje prácu skript `combinator.py`. Vstupom preň je súbor vo formáte `.json` s definovanými vstupnými parametrami zadaní.

```
[{
    "name": "th",
    "config": "stdconfig.xlsx",
    "diameter": [70],
    "Re": [70],
    "Em": [206000],
    "ET": [10000],
    "ductility": [1],
    "v": [[0],[0],[0],[0],[0],[0],[0]],
    "w": [[0], [0], [0,0.5,1,1.5,2,2.5],[0],
          [0,0.5,1,1.5,2,2.5],[0],[0,0.5,1]],
    "k0j": [0,2.5,5,7.5,10],
    "HM": [0],
    "step": [1],
    "max_iter": [20],
    "ftoler": [0.001],
    "mtoler": [0.001],
    "utoler": [0.001]
}]
```

Kód 6.1: Ukážka časti vstupného súboru pre `combinator.py`.

Každý objekt v hlavnom zozname súboru je kombinovaný zvlášť, príkladom je definícia jedného objektu so vstupnými dátami používanými v tejto časti práce (kód 6.1). Každý vygenerovaný riadok obsahuje jedinečný názov, ktorý vznikne spojením hodnoty parametra `name` a čísla indexu danej kombinácie. Parameter `config` obsahuje názov súboru, z ktorého má riešič načítavať konfiguráciu rovnačky. Nasledujú numerické hodnoty zvyšných veličín a parametrov úlohy. Tie sa definujú do zoznamov, pretože sa podľa nich generujú všetky vzájomné kombinácie hodnôt a to aj v prípade, že daný parameter nadobúda iba jednu hodnotu. Parametrom `HM` sa nastavuje žiadaný typ spevňovania materiálu (0 - bez spevnenia, 1 - izotropické, 2 - kinematické; možnosti 1 a 2 však oproti originálnemu rýchlemu algoritmu nie sú zatiaľ implementované).

Konkrétne hodnoty zadaní riešených úloh sú uvedené v tabuľkách nižšie. V 6.1 sú parametre, ktorých hodnoty pre všetky zadania rovnaké, tabuľka 6.2 potom obsahuje intervaly hodnôt parametrov, ktorých kombinácie, spolu s intervalom pre k_0 z predošlej tabuľky, poslúžia ako vstupy do výpočtov. Kombinovať sa potom bude pre každú hodnotu medze klzu zvlášť. Všetky úlohy budú počítané v jednom záťažnom kroku a použitý bude model bez spevnenia materiálu.

d	[mm]	70
E	[GPa]	206
E_T	[GPa]	10
A	[%]	1
$v_{A-(G/I)}$	[mm]	0
max_iter	[-]	20
ftoler	[%]	0.001
mtoler	[%]	0.001
utoler	[%]	0.001
k_0	[mm/m]	$\{0:2.5:10\}$

Tabuľka 6.1: Vstupné parametre identické pre všetky počítané úlohy.

	R_e [MPa]			
$w_{\mathrm{A-I}}$ [mm]	300	500	800	1200
$\mathrm{C,E_{alt}}$	$\{0:0.5:2.5\}$	$\{0:0.5:3\}$	$\{0:1:5\}$	$\{0:1:7\}$
$\mathrm{E,G_{alt}}$	$\{0:0.5:2.5\}$	$\{0:0.5:3\}$	$\{0:1:5\}$	$\{0:1:7\}$
$\mathrm{G,I_{alt}}$	$\{0:0.5:1\}$	$\{0:0.5:1.5\}$	$\{0:1:3\}$	$\{0:1:3\}$

Tabuľka 6.2: Intervaly vertikálnych priehybov rovnacích valcov pre rôzne medze klzu tyčí.

	A	B	C
1	Lv	BCT	active
2	[mm]	[-]	[-]
3	475	0	0
4	475	0	1
5	475	1	1
6	475	1	1
7	475	1	1
8	475	1	1
9		0	1

(a) Konfigurácia rovnačky.

	A	B	C
1	Lv	BCT	active
2	[mm]	[-]	[-]
3	475	0	0
4	475	0	1
5	475	1	1
6	475	1	1
7	475	1	1
8	475	1	1
9	475	1	1
10	475	1	1
11		0	1

(b) Rozšírená konfigurácia rovnačky.

Obr. 6.1: Konfigurácie rovnačky pre tvorbu datasetov.

Obe konfigurácie rovnačky (obrázky 6.1) budú využívať rozdvojenie väzieb v miestach styku tyče a rovnacích valcov. Zvyšok konfigurácií je identický s tou, ktorá bola použitá počas verifikácie (obrázok 4.5).

Kombináciami parametrov z predošlých tabuliek vzniklo 3520 zadaní pre každú konfiguráciu. Výpočet týchto úloh trval 135 (std) a 167 (alt) hodín. Využívané bolo jedno jadro procesora AMD Ryzen 5 1500X s taktovacou frekvenciou 3.50 GHz. Zvážené bolo aj použitie grafickej karty na vykonávanie maticových súčinov, ako už ale bolo spomenuté, počítané matice nie sú dostatočne veľké na to, aby sa čas strávený kopírovaním dát na kartu a naspäť vyvážil úsporou rýchlejším počítaním. Pri zhruba 20, resp. 25 sekundách na iteráciu úlohy vychádza priemerne 7 iterácií na zadanie.

Druhým krokom je zoskupenie výsledkových súborov. Jednotlivé výstupné súbory, obsahujúce hodnoty pre každé zadanie zvlášť, sa použitím skriptu `pickle_collector.py` transformujú do jednej zbierky, ktorá bude slúžiť ako zdroj dát pri rozdeľovaní do finálnych troch častí datasetu.

Posledným krokom je formátovanie vstupných dát a vytvorenie trénovacej, validačnej a testovacej časti datasetu rozdelením zbierky výsledkov z predošlého kroku. Na tento účel slúži skript `dataset_creator.py`, v ktorom sa definujú jednotlivé vstupné a výstupné parametre (veličiny, ich názvy a jednotky) pre neurónovú sieť a samotná štruktúra datasetu. Neurónové siete v tejto práci budú ako svoje vstupy používať maximálnu absolútnu hodnotu zbytkového napätia v priereze vyrovnanej tyče a reakčnú silu na najzaťaženejší rovnací valec (viď tabulku 6.3). Hodnoty maximálnych napätí sú v zbierke výsledkov uložené pre každú rovinu a reakčné sily pre každý rovnací valec, ako vo vertikálnom, tak v horizontálnom smere. Formátovanie týchto hodnôt je vykonané skriptom, čím sa získajú parametre vo forme vhodnej pre vstup do siete. Ďalšou úlohou skriptu je rozdelenie dát. To prebieha odfiltrovaním neskonvergovaných výsledkov zbierky, ich premiešaním a za-

radením do jednotlivých častí. Veľkostné pomery validačnej a testovacej časti ku celku sú od užívateľa požadované ako vstupné argumenty pri spúšťaní skriptu. Takto roztriedený dateset uloží do troch výstupných súborov (train, validation a test) s príponou `.ptd` (PyTorch Dataset). Vygeneruje taktiež kostru súboru pre dotazovanie sa predikcií natrénovanou sieťou (inference).

Vstupy		Výstupy	
d	[mm]	k	[mm/m]
R_e	[MPa]	$\sigma_{\mathrm{rez,max}}$	[MPa]
$w_{\mathrm{C,E_{alt}}}$	[mm]	R_{max}	[N]
$w_{\mathrm{E,G_{alt}}}$	[mm]		
$w_{\mathrm{G,I_{alt}}}$	[mm]		
k_0	[mm/m]		

Tabuľka 6.3: Parametre navrhovaných neurónových sietí.

Ako je vidieť v tabuľke 6.1, všetky zadania budú počítať s rovnakou hodnotou priemeru rovnanej tyče. Ten bol zvolený ako vstup do neurónových sietí, aby sa v prípade rozšírenia datasetu o iné hodnoty priemeru tyče pred trénovaním nemuseli prepracovávať. Preto je ale potrebné dbať na to, aby pracovali iba s touto naučenou hodnotou (v častiach 6.4 a 6.3). Z počítaných 3520 zadaní ich skonvergovalo 3070, resp. 3309. Rozdelenie do jednotlivých datasetov je uvedené v tabuľke 6.4.

		Počet vzoriek	
Časť datasetu	Veľkostný pomer	std	alt
trénovacia	0.8	2456	2647
validačná	0.1	307	331
testovacia	0.1	307	331
celkom	1.0	3070	3309

Tabuľka 6.4: Rozdelenie datasetu do častí.

6.2. Trénovanie neurónovej siete

Na účely vytvorenia a práce s neurónovými sieťami bol vytvorený skript, ktorý využíva modul PyTorch [18]. Vstupnými argumentami skriptu sú názvy súborov s trénovacou a validačnou časťou datasetu, hodnota semienka pre generátor náhodných čísel a názov, pod akým bude sieť uložená. Skript najskôr skontroluje prítomnosť grafickej karty a vyhodnotí, či ju je možné, z hľadiska nainštalovaných knižníc, použiť. To záleží na nainštalovanej ver-

zii modulu PyTorch, ktorého základná distribúcia podporuje iba procesory, existuje však aj verzia umožňujúca vykonávanie výpočtov na grafickej karte. Na tento účel modul používa výpočtovú platformu CUDA [23], vyvíjanú firmou NVIDIA pre svoje produkty, ktorej inštalácia je v prípade použitia tejto verzie PyTorch nutnou prerekvizitou. Skript potom nastaví generátory náhodných čísel na zadané semienko, aby učenie prebiehalo deterministicky, a to ako v prostredí procesora, tak na grafickej karte. Potom vytvorí štruktúry na postupné dávkovanie vstupov z trénovacej a validačnej časti datasetu počas učenia, ktoré automaticky pre každú epochu tieto vstupy náhodne premiešajú.

Nasleduje definícia modelu neurónovej siete. Počet vstupných a výstupných neurónov sa automaticky nastaví tak, aby zodpovedal počtom vstupných a výstupných parametrov datasetu. Počet vnútorných vrstiev a ich neurónov je nastaviteľný vstupnými argumentami pri inicializácii triedy. Vstupné hodnoty sú voliteľne individuálne škálovateľné. Využitie tejto funkcionality bude vysvetlené nižšie, v časti o navrhnutej chybovej funkcii. Nasleduje operácia lineárnej transformácie vstupu a váh prvej vnútornej vrstvy ($\mathbf{w}^T\mathbf{x}$). Ďalej bola zvolená aktivačná funkcia LeakyReLU, modifikácia funkcie ReLU, kde v zápornej oblasti nenadobúda nulovú hodnotu, ale so znižujúcim sa vstupom lineárne klesá (rovnica 6.1). Hodnota ε je ďalším hyperparametrom siete, v tomto prípade však bola ponechaná na predvolenej hodnote 0.01. Vnútorné vrstvy pozostávajúce z lineárnych transformácií a aktivačnej funkcie LeakyReLU sú definované dynamicky podľa zadaných hodnôt. Výstupná vrsta sa nakoniec dostane lineárnou transformáciou, ale keďže ide o regresnú sieť, nie je na ňu aplikovaná aktivačná funkcia. Výstupné hodnoty sú nakoniec spätne škálované do pôvodného merítka. Hodnoty váh jednotlivých vrstiev sú inicializované náhodne podľa rovnomerného rozdelenia $\mathcal{U}(-\sqrt{k}, \sqrt{k})$, kde k je rovné prevrátenej hodnote počtu vstupov vrstvy.

$$\text{LeakyReLU}(x) = \begin{cases} x & x \geq 0 \\ \varepsilon \cdot x & x < 0 \end{cases} \tag{6.1}$$

$$\text{MSELoss}(\mathbf{y}, \hat{\mathbf{y}}) = (\mathbf{y} - \hat{\mathbf{y}})^2 \tag{6.2}$$

Modifikácia LeakyReLU bola použitá kvôli javu označovanému ako "dying ReLU", alebo umierajúce ReLU. Môže nastať v prípade, keď sa pôsobením veľkého gradientu na neurón aktivovaný pomocou funkcie ReLU jeho váha posunie do záporných hodnôt. Tým pre všetky nasledovné vstupy bude dávať rovnaký výstup, a to nulu, čím prestane prispievať ostatným vrstvám a stáva sa nadbytočným. A keďže je derivácia konštanty rovná nule, nemá sa jeho váha ako zmeniť. LeakyReLU tento problém rieši práve malým, ale nenulovým gradientom aj v zápornej časti definičného oboru. Neurón tak má počas trénovania siete šancu "ožiť", čiže jeho váha môže opäť nadobudnúť kladnú hodnotu s väčším gradientom. Príklad tohto javu je zachytený na obrázku 6.3 na strane 42 v okolí 50. epochy,

kedy je prudký pokles predikčnej chyby danej siete spôsobený práve zotavením jedného neurónu.

V počiatočných fázach návrhu štruktúry neurónových sietí boli vyskúšané viaceré pokročilé typy vrstiev. Najsľubnejšou bola tzv. Dropout vrstva, ktorá pre jednotlivé iterácie trénovacích cyklov vynuluje časť vstupných hodnôt pre nasledujúcu vrstvu. Toto nulovanie prebieha náhodne a podiel ovplyvnených vstupov p je ďalším hyperparametrom. Účelom tejto modifikácie je zvýšenie robustnosti siete tým, že sa interakcie medzi neurónmi vrstiev počas tréningu neustále menia a sú tak na sebe menej závislé. Problém zmeny veľkostí výstupných hodnôt vrstvy je korigovaný automatickým škálovaním o hodnotu $\frac{1}{1-p}$. Počas vyhodnocovania validačných a testovacích chýb, a počas používania siete na tvorbu predikcií je $p = 0$, čím je jej výstup identický so vstupom. Výhody použitia tohto typu vrstvy sú však zreteľné až pre siete pracujúce s datasetmi mnohonásobne väčšími ako v tejto práci, preto nakoniec nebola použitá.

Druhým typom skúšanej vrstvy bola tzv. LayerNorm, definovaná rovnicou 6.3. Vrstva počas počítania trénovacích predikcií ukladá stredné hodnoty $\mathrm{E}[\mathbf{x}]$ a rozptyly $\mathrm{Var}[\mathbf{x}]$ vstupných hodnôt, ktoré potom využíva pri ich normalizácií. Cieľom je transformácia vstupov do podoby s nulovou strednou hodnotou a jednotkovým rozptylom.

$$\mathbf{y} = \frac{\mathbf{x} - \mathrm{E}[\mathbf{x}]}{\sqrt{\mathrm{Var}[\mathbf{x}] + \varepsilon}} \cdot \gamma + \beta \tag{6.3}$$

Parameter ε o malej hodnote (typicky 1e−5) je pridávaný ako prevencia delenia nulou, čím zvyšuje numerickú stabilitu. γ a β sú parametre váh a posunutí vrstvy, ktoré sú trénovateľné. Namiesto tohto typu vrstvy bolo však použité škálovanie vstupov a výstupov siete, vysvetlené nižšie, pretože umožňuje vyššiu mieru kontroly nad ich hodnotami a nezavádza dodatočné parametre, ktoré sa sieť musí naučiť.

Použitá chybová funkcia bola tzv. MSELoss (Mean Squared Error – ℓ^2 alebo Euklidovská norma, rovnica 6.2). Kvôli škálovaniu vstupných a výstupných hodnôt počas kalkulácie predikcií navrhnutou neurónovou sieťou však musela byť takto upravená aj chybová funkcia. Výstupné hodnoty neurónovej siete, taktiež nazývané predikcie $\hat{\mathbf{y}}$, a skutočné výstupné hodnoty $\mathbf{y}$, ktoré chceme sieť naučiť sú pred výpočtom chyby škálované o rovnakú hodnotu, akou sú spätne škálované výstupy siete. Následne sú jednotlivé chyby agregované vypočítaním ich súčtu alebo aritmetického priemeru, čím sa získa hodnota chyby danej predikcie modelu.

Vstupné argumenty neurónovej siete sa škálujú za účelom získania čo najrovnomernejších intervalov ich hodnôt, aspoň rádovo. To modifikuje chybový priestor jeho rozšírením v smere pôvodne najmenších veličín, čím efektívne zvyšuje citlivosť siete na tieto veličiny a znižuje pravdepodobnosť, že pri daných nastaveniach hyperparametrov sieť potenciálne

minimá preskočí. Výstupy siete sú potom škálované opačným smerom, opäť za účelom zvýšenia robustnosti.

Škálovanie počas výpočtu chyby predikcie má trochu inú úlohu. Umožňuje manuálne meniť dôraz, aký bude optimalizátor klásť na odchýlky jednotlivých veličín od ich žiadaných hodnôt. Príkladom sú hodnoty krivosti tyče, porovnávané s najvyššou reakčnou silou tyče na rovnací valec. Tie majú diametrálne odlišné jednotky a po ich umocňovaní počas výpočtu chyby je reakčná sila dominantným členom. Tým pádom sa bude algoritmus učenia snažiť minimalizovať takmer výlučne túto veličinu a krivosť, ktorá nadobúda hodnoty do 10 mm/m, začne byť braná do úvahy až keď bude odchýlka síl v jednotkách newtonov. Samozrejme záleží od konkrétnej voľby jednotiek, ktoré sa pri výpočte MKP alebo tvorbe datasetu dajú zmeniť tak, aby boli hodnoty v podobných intervaloch. Tento spôsob bol pôvodne použitý počas programovania skriptov použitých v tejto časti práce namiesto škálovania, výsledné hodnoty a vykresľované grafy tak ale mali neštandardné násobky jednotiek. Škálovanie tento efekt nemá, čo z neho v konečnom dôsledku robí užívateľsky vhodnejší prístup.

Vstupy	
d	10
Re	1
w_C	100
w_E	100
w_G	100
k_0	50

Výstupy	
k	50
$\sigma_{\mathrm{rez,max}}$	1
R_{max}	1e−3

Tabuľka 6.5: Použité škálovacie hodnoty parametrov sietí.

Za optimalizačný algoritmus bol zvolený tzv. Adam (Adaptive Moment Estimation), ktorý pred úpravou váh vrstvy popri aktuálnej hodnote gradientu chybovej funkcie berie do úvahy priemerované predošlé hodnoty gradientov a ich druhých momentov (zotrvačností). Na obrázku 6.4 na strane 42 je pri 30. epoche vidieť efekt zotrvačnosti. Po tom, ako sa sieť dostala na okraj plytkého lokálneho minima, ju zotrvačný člen optimalizátora z neho vyviedol, vďaka čomu hneď potom mohla nájsť ďalšie, výhodnejšie lokálne minimum.

Nasledujú cykly učenia siete. Ten pozostáva z načítania várky (o parametrizovanej veľkosti) vstupných dát z trénovacieho datasetu a vypočítaním predikcií pomocou spočiatku náhodne inicializovanej siete. Na základe veľkostí odchýlok týchto predikcií od predpísaných hodnôt sa vypočíta chyba várky, ktorá je priebežne ukladaná. Optimalizačný algoritmus potom vezme spočítaný gradient chybovej funkcie vzhľadom ku vstupným pa-

rametrom siete a upraví jej váhy v smere najstrmšieho poklesu (s ohľadom na ukladané hodnoty z minulých cyklov) o hodnoty proporcionálne nastavenému koeficientu učenia α. Tento cyklus pokračuje až do vyčerpania dát z trénovacej časti datasetu. Chyby trénovacích várok tejto epochy sa spriemerujú a uložia do zásobníka pre vykreslenie grafu priebehu chýb počas učenia. Nasleduje podobný cyklus, tentokrát sú však vstupné dáta brané z validačného datasetu a je počítaná iba chyba predikcií, gradient chybovej funkcie sa nepočíta a váhy siete sa nemenia. Tento cyklus slúži iba na vyhodnotenie chýb predikcií siete na vstupných hodnotách z rovnakého rozdelenia ako počas tréningu, tieto hodnoty sa však neučí. Priemerná validačná chyba epochy sa tak ako chyba trénovacia pridá do príslušného zásobnika na budúce spracovanie. Ako už bolo spomenuté v časti 5.3, porovnaním validačnej a trénovacej chyby sa dá o stave učenia modelu zistiť niekoľko dôležitých informácií, napr. miera zovšeobecňovania. Následne sa tento pomer trénovacej chyby ku chybe validačnej vyčísli. V prípade, že pre danú epochu nadobudla priemerná validačná chyba svoje historické minimum, sú parametre siete a optimalizátoru uložené. Tým je daná epocha ukončená.

Kritériá ukončenia učenia sa dajú ovládať niekoľkými parametrami skriptu. Prvým je zoznam koeficientov učenia `lrs`, ktorého príslušný element je pre každú skupinu epoch zo zoznamu `epochs` dosadený do optimalizátora.

```
epochs = [150, 150, 100, 50]
lrs = [1e-1, 1e-2, 1e-3, 1e-4]
minimum_lookahead = 10
tvu = 1.1
tvl = 0.8
```

Kód 6.2: Parametre ovplyvňujúce priebeh učenia neurónových sietí.

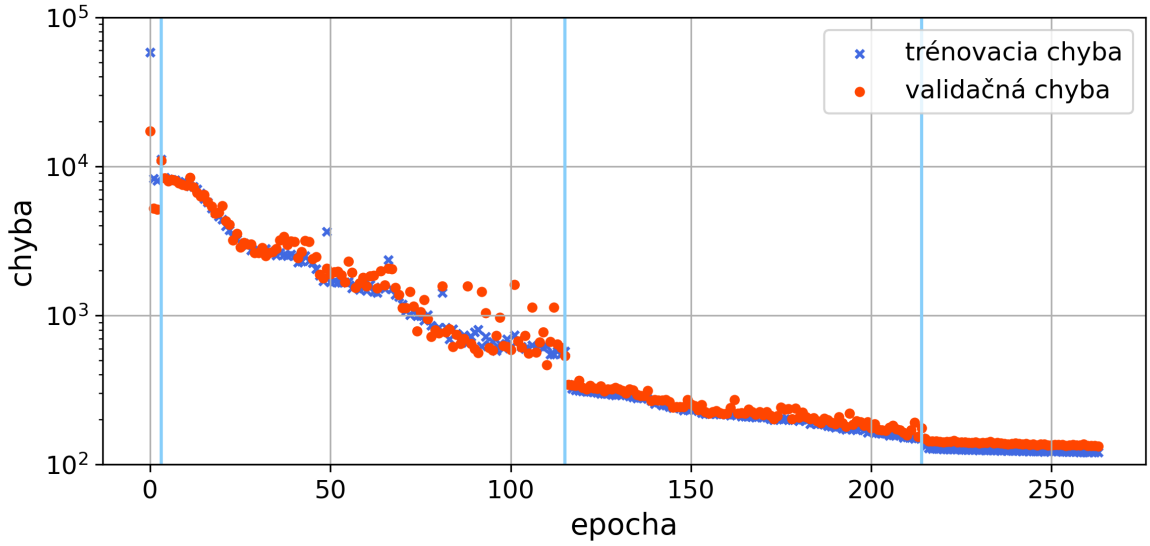
Podľa konfigurácie z kódu 6.2 bude prvých 150 epoch optimalizátor používať hodnotu koeficientu učenia $\alpha = 1e{-}1$, druhých 150 epoch bude $\alpha = 1e{-}2$, atď. Ak sa v priebehu niekoľkých epoch (určované premennou `minimum_lookahead`) nedosiahne minimálna hodnota validačnej chyby, načíta sa uložený stav neurónovej siete a optimalizátora z epochy s minimálnou validačnou chybou, zvyšné epochy danej skupiny sa preskočia. Pokračuje sa ďalšou skupinou epoch a príslušným koeficientom učenia.

Počas učenia nastávajú prípady, kedy je rozdiel trénovacej a validačnej chyby príliš veľký, čo znamená, že sa sieť špecializuje na jeden dataset. Schopnosť siete zovšeobecňovať klesá a chyba na druhom datasete preto narastá. Ak je pomer trénovacej chyby ku validačnej vysoký, sieť náhodou našla konfiguráciu špecializujúcu sa na validačný dataset. Tieto prípady sa dajú eliminovať použitím premennej `tvu` (Train-Validation ratio Upper limit),

ktorá zbráni uloženiu stavu ak pomer chýb prekročí zadanú hodnotu, a to aj v prípade, ak ide o historicky najnižšiu validačnú chybu. Premenná `tvl` (Train-Validation ratio Lower limit) funguje v opačnom smere a blokuje konfigurácie špecializované na trénovací dataset.

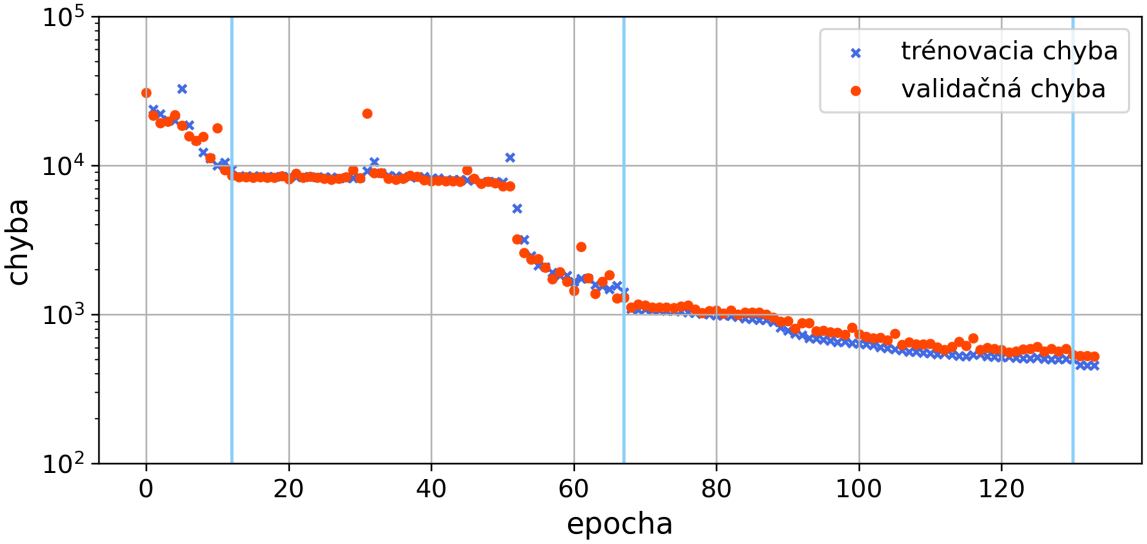
Pomocou možnosti nastavenia viacerých koeficientov učenia sa dá ilustrovať charakteristický jav učenia neuronových sietí. Ide o výrazný pokles a stabilizáciu chýb pri znížení hodnoty α, viditeľný na obrázku 6.2 v okolí 120. epochy. Svetlomodrou vertikálnou čiarou je znázornený prechod medzi skupinami epoch a teda miesto so skokovou zmenou hodnoty α.

Učenie sa končí prejdením všetkých skupín epoch. Natrénovaná neurónová sieť, optimalizátor a chybová funkcia sa spolu s niekoľkými ďalšími detailami uloží do súboru s názvom siete a koncovkou `.pt` (PyTorch). Do súboru s koncovkou `.ptf` a rovnakým názvom sa uložia hodnoty priebehu učenia pre účely vykreslenia grafov pomocou skriptu `nn_plot.py`.

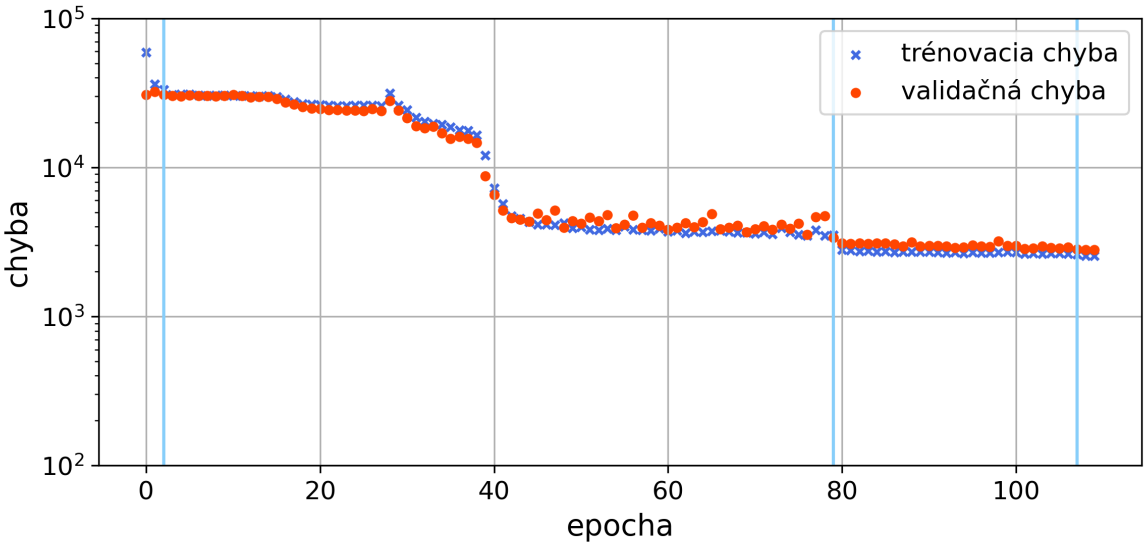


Obr. 6.2: Priebeh chýb počas trénovania siete (riadok 5 tabuľky 6.6).

Po ukončenom trénovaní siete sa musí objektívne overiť veľkosť chyby na doposiaľ nepoužitej testovacej časti datasetu. Skript `nn_test.py` načíta natrénovaný model uložený v `.pt` súbore a vypočíta priemernú testovaciu chybu siete podľa chybovej funkcie, ktorou bola trénovaná. Takto sa získa finálna hodnota na popis danej neurónovej siete. Je však dôležité testovať už hotovú sieť na neznámom datasete, aby sa predišlo prípadnej snahe tvorcov siete o informovanú optimalizáciu tejto chyby, pretože by prestala byť objektívnym merítkom schopnosti siete zovšeobecňovať naučený problém.



Obr. 6.3: Priebeh chýb počas trénovania siete (riadok 6 tabuľky 6.6).



Obr. 6.4: Priebeh chýb počas trénovania siete (riadok 13 tabuľky 6.6).

Posledným skriptom pre prácu s neurónovými sieťami je `nn_inference.py`. Vstupom preň je názov `.xlsx` súboru obsahujúci vstupné hodnoty pre sieť. Skript tieto hodnoty postupne načíta a do rovnakého súboru uloží vypočítané predikcie. Tento vstupný súbor sa štandardne vytvára spolu so súbormi obsahujúcimi rozdelenú kolekciu výsledkov výpočtov MKP skriptom `dataset_creator.py` z časti 6.1.

V tejto práci bolo trénovaných niekoľko topológií neurónových sietí, odlišnými počtom vnútorných vrstiev a ich neurónov, a použitými semienkami generátorov náhodných čísel počas tvorby datasetov a trénovania. Kritériom pri ich návrhu bol celkový počet vnútorných neurónov, v tomto prípade 100, so zohľadnením lepšej popisnej schopnosti

viacvrstvých sietí danou väčším vzájomným vplyvom neurónov na nadväzujúce vrstvy. Pre každú topológiu a dve rozdelenia zbierky výsledkov výpočtov MKP bolo natrénovaných 20 sietí (pre 8-vrstvé siete to bolo 40). Výsledné chyby dvoch najlepších sietí z každej kombinácie sú uvedené v tabuľke 6.6.

Počet		Veľkosť	Náhodné semienko		Chyba		
vrstiev	neurónov	várky	dataset	sieť	trénovacia	validačná	testovacia
1	100	10	20	2	814.8	981.3	773.2
				8	349.4	369.7	408.2
			21	2	798.6	836.2	881.5
				4	690.2	643.6	699.2
2	50	10	20	7	120.8	132.6	144.0
				13	455.0	525.4	506.7
			21	3	308.7	328.2	312.4
				7	231.7	256.9	295.7
5	15	15	20	10	493.8	501.5	520.9
				14	463.5	452.1	461.5
			21	9	517.9	562.0	589.5
				10	374.8	365.9	415.1
8	10	20	20	16	2543.3	2803.0	3113.9
				20	2610.8	2746.1	3032.2
			21	1	3206.6	3261.1	3519.8
				31	2457.7	2955.9	2775.0

Tabuľka 6.6: Chyby najlepších trénovaných sietí.

Najmenších chýb sa konzistentne dopúšťali siete s dvoma vnútornými vrstvami. Väčší počet vrstiev vedie na komplexnejšiu reprezentáciu učenej funkcie, ktorá môže byť presnejšia, prakticky sa ale kvôli vyššej citlivosti na zvolené hyperparametre ťažšie trénujú. Príkladom je veľkosť várky vstupných dát, ktorá významným spôsobom ovplyvňovala priebeh učenia, kedy pri veľmi malej hodnote neboli siete schopné konzistentne nachádzať minimá chybového priestoru. Väčšia veľkosť várky skresľuje chybový gradient a vedie sieť ku priemernej reprezentácií datasetu. Riešením by mohla byť meniteľná veľkosť várky počas trénovania siete, od čoho ale použitý modul PyTorch používateľov odrádza chybovou hláškou.

Sieť z piateho riadku tabuľky 6.6 vykazuje najnižšiu chybu zo všetkých navrhnutých sietí. Maximálna priemerná odchýlka predikcií δ pre jednotlivé stĺpce (index i), s uvážením

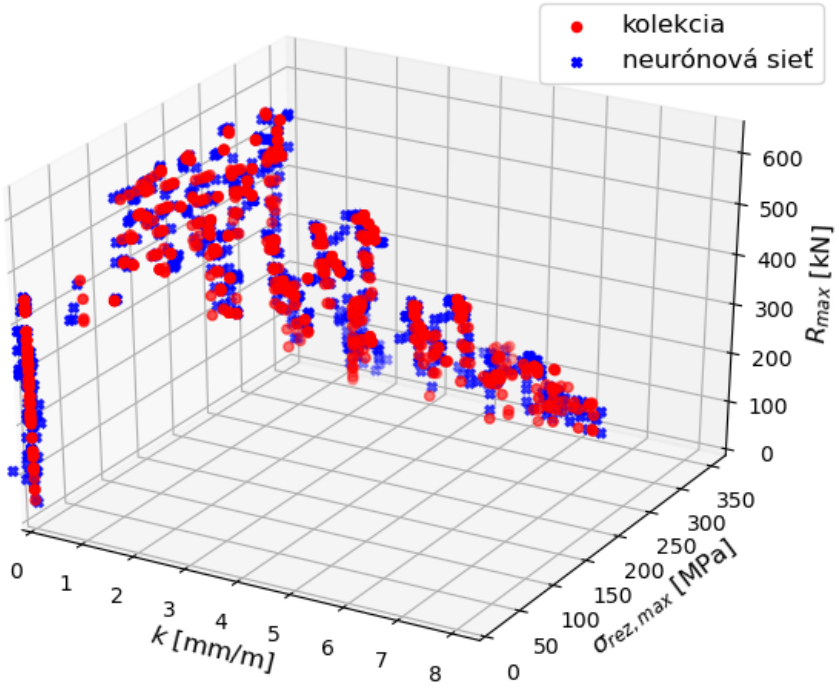
škálovacej hodnoty s_i a za predpokladu chybovej funkcie MSELoss, sa z chyby ℓ dá vyjadriť pomocou vzťahu

$$\delta_i = \frac{\sqrt{\ell}}{s_i} \tag{6.4}$$

s jednotkami danej výstupnej veličiny. V tabuľke 6.7 sú uvedené prepočty podľa 6.4 sietí s extrémnymi hodnotami testovacej chyby. Je vhodné podotknúť, že hodnotu maximálnej odchýlky môže daná veličina nadobudnúť iba v prípade, ak sú zvyšné dve zhodné s učenými. Prakticky sú preto odchýlky nižšie. História učenia sietí uvedených v tabuľke 6.6 sa spolu s ich grafmi nachádzajú v prílohe.

Riadok tab. 6.6	δ_k [mm/m]	δ_σ [MPa]	δ_R [kN]
5	0.24	12	12
15	1.19	59.32	59.32

Tabuľka 6.7: Priemerné maximálne odchýlky predikcií vybraných neurónových sietí.



Obr. 6.5: Hodnoty z kolekcie výsledkov a predikcií siete 5. riadku tab. 6.7 pre $R_e = 800$MPa.

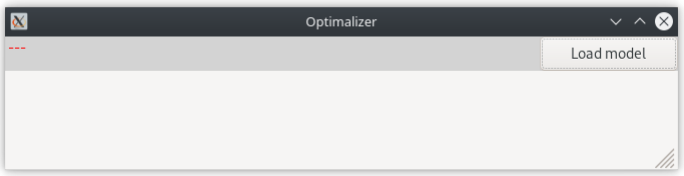
Počet vrstiev	Počet neurónov	Veľkosť várky	Náhodné semienko dataset	Náhodné semienko sieť	Chyba trénovacia	Chyba validačná	Chyba testovacia
2	50	10	30	3	247.8	231.2	276.1
				8	258.4	285.8	251.6
			31	1	137.8	149.8	124.2
				7	156.2	178.6	153.4
5	15	15	30	1	542.8	565.2	628.4
				6	882.0	924.5	740.0
			31	6	501.2	497.0	506.7
				17	415.0	425.6	396.4

Tabuľka 6.8: Chyby sietí rozšírenej konfigurácie rovnačky.

Rovnaký postup bol aplikovaný na datasety s rozšírenou konfiguráciou rovnačky. Hodnoty chýb niekoľkých takto natrénovaných neurónových sietí sú uvedené v tabuľke 6.8. História učenia trénovaných sietí sú v prílohe.

6.3. Optimalizačný skript

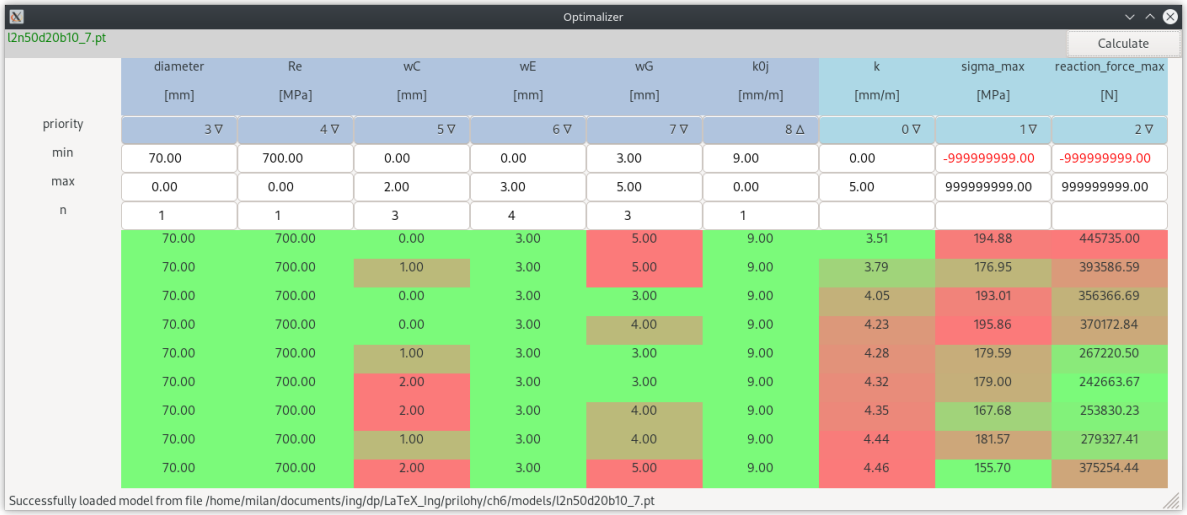
Táto časť sa zaoberá optimalizačným skriptom `optimalizer.pyw`. Grafické prostredie vytvára skript pomocou modulu wxPython, na prácu s neurónovými sieťami a počítanie predikcií využíva modul PyTorch.



Obr. 6.6: Optimalizačný skript po spustení.

Po spustení skriptu sa otvorí okno aplikácie s tlačidlom `Load model` (viď obrázok 6.7). Po kliknutí naň sa otvorí dialógové okno, v ktorom sa vyberie `.pt` súbor s natrénovanou neurónovou sieťou. Po načítaní súboru sa zobrazí hlavička tabuľky na zadávanie hodnôt vstupných veličín pre neurónovú sieť. Tmavomodrou farbou sú označené vstupné veličiny, svetlomodrou potom veličiny výstupné. Hodnoty v políčkach riadkov označených `min` a `max` určujú hraničné hodnoty intervalov jednotlivých veličín, `n` určuje počet vygenerovných hodnôt tohto intervalu. Tlačidlom `Calculate` sa spustí výpočet predikcií kombinácií zadaných intervalov vstupov, ktoré sa potom spolu s výstupmi zobrazia niž-

šie. Hodnoty sú farebne označené podľa ich pozície v intervaloch jednotlivých stĺpcov, kde je zelená použitá pre nízke hodnoty a červená pre hodnoty vysoké. Pre výstupné hodnoty siete fungujú vstupné políčka ako filtre. Napríklad v stĺpci pre k na obrázku 6.7 je určená podmienka, že sa daný riadok zobrazí iba ak jeho výstupná krivosť $0 \leq k \leq 5$ mm/m. Výsledky sú taktiež zoraďované, a to podľa hodnoty v riadku `priority`. Najvplyvnejší na výsledné poradie je stĺpec s nulovou prioritou, vyššie hodnoty postupne zoraďujú nerozhodné stavy. Obrázok 6.7 toto radenie ilustruje plynulým prechodom farby pozadia zo zelenej do červenej pre stĺpec k. Priority zoraďovania veličín sa menia klikaním na príslušné políčka pravým (zvýšenie) a ľavým (zníženie) tlačidlom myši. Stredným tlačidlom myši sa mení smer zoraďovania zo vzostupného na zostupné a naopak.

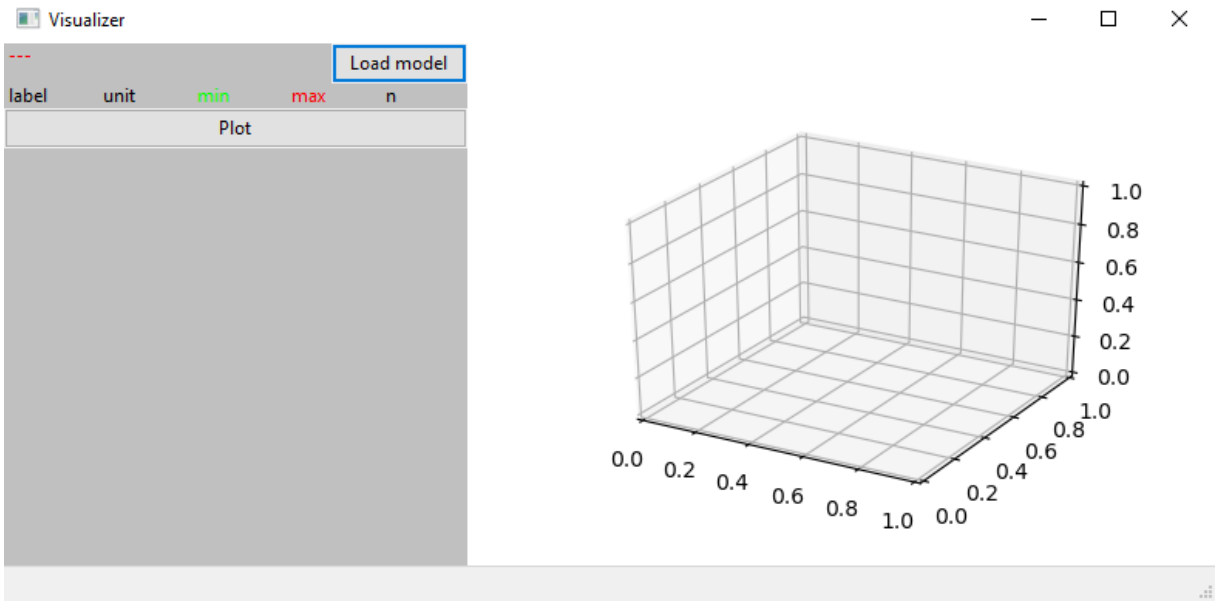


Obr. 6.7: Optimalizačný skript s načítanou neurónovou sieťou.

Optimalizačný skript umožňuje jednoduché počítanie veľkého množstva predikcií výstupných veličín procesu rovnania. Filtrovaním a zoraďovaním výsledkov tak môže poskytovať intuitívny prehľad o prejavoch zmien hodnôt vstupných parametrov na celý proces rovnania.

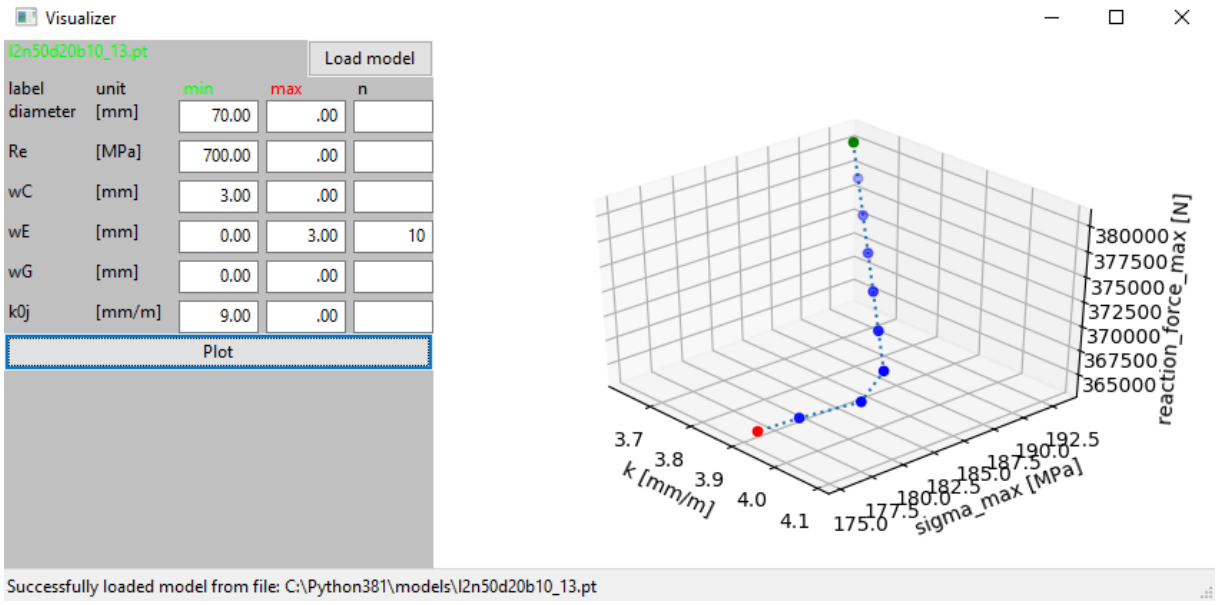
6.4. Vizualizačný skript

Táto časť sa zaoberá skriptom `visualizer.pyw` vytvoreným pre účely vizualizácie výstupov natrénovaných neurónových sietí. Na tvorbu grafického užívateľského rozhrania využíva skript popri module wxPython aj už spomenutý modul Matplotlib. O výpočet predikcií neurónových sietí sa opäť stará modul PyTorch. Na obrázku 6.8 je nasnímaný skript priamo po spustení.



Obr. 6.8: Vizualizačný skript po spustení.

Tlačidlo `Load model` otvorí prehliadač priečinkov počítača a umožní užívateľovi nájsť a označiť žiadaný `.pt` súbor s natrénovanou neurónovou sieťou. Po otvorení súboru sa jeho obsah načíta do vyrovnávacej pamäte. Spolu s natrénovanou neurónovou sieťou sú z otvoreného súboru načítané aj informácie o vstupných veličinách, ich počet, názvy a jednotky. Tie sa zobrazia na ľavej strane okna spolu s políčkami, pomocou ktorých užívateľ zadáva žiadané hodnoty pre neurónovú sieť. V prípade vyplnenia všetkých hodnôt v stĺpci `min` a ponechaní ostatných v pôvodnom stave sa po stlačení tlačidla `Plot` na pravej strane okna skriptu v grafe vykreslí bod predikcie siete. Počítanie predikcií pre jednotlivé body sa ale efektívnejšie vykonáva použitím skriptu `nn_inference.py` popísanom v časti 6.1. Zmyslom grafickej vizualizácie je získanie intuitívneho prehľadu zmeny výstupov procesu rovnania pri zadaní jedného vstupného parametra ako lineárny interval hodnôt, čo sa dosiahne doplnením hodnôt do zvyšných dvoch stĺpcov `max` a `n` v žiadanom riadku, pre doplnenie hornej hranice, respektíve počtu prvkov intervalu. Interval bodov je v grafe vykresľovaný od minimálnej hodnoty, znázornej bodkou zelenej farby, po maximálnu hodnotu červenou bodkou. Hodnoty medzi nimi sú vykreslené modrou farbou. Príklad zadania vstupov pre intervalový výpočet a vykreslenie grafu je na obrázku 6.9. Panel, v ktorom je graf vykreslený, je interaktívny, čo umožňuje užívateľovi ním rotovať. V tomto prípade sú v grafe vykreslené dve priamky, a to kvôli použitiu po častiach lineárnej aktivačnej funkcie LeakyReLU.



Obr. 6.9: Vizualizačný skript s načítanou neurónovou sieťou.

Použitie neurónových sietí pri návrhu rovnacieho stroja a optimalizácií procesu rovnania týmto spôsobom vyžaduje vypočítanie veľkého množstva zadaní na pokrytie návrhového priestoru. To je principiálne nutné vykonať ešte pred natrénovaním neurónovej siete, čo znamená, že pri používaní hotovej siete vo vizualizačnom programe je jediným potrebným výpočtom získanie predikcií, čo aj pri pomerne rozsiahlych sieťach trvá iba niekoľko desatín sekundy.

Princíp neurónových sietí by v praxi mohol byť aplikovaný aj v oblasti dynamického nastavovania parametrov rovnacích strojov, ako priehybov a natočení valcov v reálnom čase, kedy by sa po prvotnom natrénovaní na základnej zbierke výsledkov analýz MKP mohli učiť za pochodu z reálnych vstupných a výstupných hodnôt meraných veličín. Vhodným návrhom tejto siete by sa tak mohol do tvorby predikcií zahrnúť napríklad vplyv opotrebovania povrchu rovnacích valcov.

7. Záver

Hlavným cieľom tejto diplomovej práce bola úprava výpočtového modelu rýchleho algoritmu procesu rovnania tyčí kruhového prierezu [3] tak, aby bol jednoducho aplikovateľný na ľubovoľnú konfiguráciu kosouhlých valcových rovnacích strojov. V časti 4.2 bola táto reimplementácia úspešne verifikovaná porovnaním jej výsledkov s výstupmi pôvodného algoritmu.

Pre umožnenie tvorby väčšieho množstva zadaní výpočtov analýz MKP bol vytvorený pomocný skript pre ich jednoduché vytváranie a kombinovanie. Pôvodný algoritmus bol taktiež upravený tak, aby tieto zadania postupne načítaval a počítal sériovo, bez nutnosti ďalších vstupov od obsluhy.

Posledným cieľom bol návrh optimalizácie rovnacieho procesu. Na tento účel bol pomocou spomenutých programov vypočítaný skúšobný dataset, na ktorom bolo natrénovaných niekoľko rôznych neurónových sietí. Jednoduchý vizualizačný skript tieto siete využíval na rýchly výpočet predikcií zadaných užívateľom, ktoré vykresľuje do grafu v grafickom okne. Vytvorené skripty umožňujú natrénovanie tu použitých neurónových sietí na rozšírené alebo vlastné dáta, poprípade jednoduchými úpravami parametrov tvorbu nových sietí.

Počas vykonávania výpočtov MKP novovytvoreného programu a trénovania neurónových sietí bol kvôli obmedzeniam použitého programovacieho jazyka Python spúšťaný iba jeden proces, využívajúci jediné jadro procesora počítača. Snaha o urýchlenie týchto výpočtov paralelizáciou využitím grafickej karty nebola úspešná, nakoľko mierne predlžovala dobu riešenia úloh. Pri trénovaní a používaní neurónových sietí by nešlo o markantný rozdiel, no algoritmus rovnania by výrazne urýchlila možnosť využitia plného potenciálu viacjadrového procesora. V tomto ohľade by bol vhodnejšou voľbou programovací jazyk C/C++, ktorý štandardne podporuje viacero paralelne spustených procesov, v prípade nedostatočnej kapacity vyrovnávacej pamäte tiež využitie viacerých jadier jedným procesom.

Výsledné chyby trénovaných neurónových sietí vykazujú trend nárastu ich priemernej hodnoty s narastajúcim počtom vnútorných vrstiev. Teoreticky sa dá chyba akokoľvek rozsiahlej siete znížiť na žiadanú, ľubovoľne vybranú úroveň, čo však v praxi vyžaduje kvalitný a objemný dataset vstupných parametrov.

Literatúra

[1] NÁVRAT, T.: *Nový přístup k výpočtové simulaci procesu rovnání dlouhých vývalků.* Brno, 2015. Habilitační práce

[2] ŠČERBA, B.: *Vliv nastavení a konfigurace rovnačky na výsledky simulace kosoúhlého rovnání.* Brno, 2020. Dostupné z: https://www.vutbr.cz/studenti/zav-prace/detail/125207. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav mechaniky těles, mechatroniky a biomechaniky. Vedoucí práce Tomáš Návrat.

[3] PETRUŠKA, J., NÁVRAT, T., ŠEBEK, F.: *A New Model for Fast Analysis of Leveling Process.* Advanced Materials Research. 2012, s. 389 - 393

[4] PETRUŠKA, J., NÁVRAT, T., ŠEBEK, F.: *Novel Approach to Computational Simulation of Cross Roll Straightening of Bars.* Journal of Materials Processing Technology, 2016, 233. vyd., s. 53–67.

[5] JANÍČEK, P.: *Systémová metodologie: Brána do řešení problémů.* Brno: CERM, 2014. ISBN 978-80-7204-887-8

[6] TOKUNAGA, H.: *On the roller straightener (Report 1, Straightening of Sections)* Bulletin of JSME, 3. vyd., č. 12, s. 572–579

[7] TOKUNAGA, H.: *On the roller straightener (2nd Report, Straightening of Round Bars, Pipe and Tubings)* Bulletin of JSME, 4. vyd., č. 15, 1961, s. 605–611

[8] DAS TALUKDER, N. K., SINGH, A. N.: *Mechanics of Bar Straightening, Part 2: Straightening in Cross-Roll Straighteners.* Journal of Engineering for Industry, 113. vyd., 1991, s. 228–232

[9] MITCHELL, T. M.: *Machine Learning.* McGraw-Hill, 1997, ISBN 0070428077

[10] MURPHY, K. P.: *Machine Learning: A Probabilistic Perspective*, MIT Press, 2013, ISBN 978-0-262-01802-9

[11] WEINBERGER, K.: *Lecture 1 "Supervised Learning Setup" –Cornell CS4780 Machine Learning for Decision Making SP17* [online]. YouTube 9.7.2018 [28.3.2020]. Dostupné z: https://youtu.be/MrLPzBxG95I

[12] WEINBERGER, K.: *Machine Learning Lecture 35 "Neural Networks / Deep Learning" -Cornell CS4780 SP17* [online]. YouTube 11.7.2018 [28.3.2020]. Dostupné z: https://youtu.be/kPXxbmBsFxs

[13] MINSKY, M. L., PAPERT, S. A.: *Perceptrons: An introduction to computational geometry.* MIT Press, 1969. ISBN 9780262130431

[14] SAPOLSKY, R. M.: *Behave: The Biology of Humans at Our Best and Worst.* Vintage, 2018. ISBN 9780099575061

[15] HEBB, D. O.: *The Organization of Behavior: A Neuropsychological Theory.* Taylor & Francis Inc, 2002. ISBN 9780805843002

[16] LU, Z. et al.: *The Expressive Power of Neural Networks: A View from the Width.* Advances in Neural Information Processing Systems 30, Curran Associates, Inc., 2017, s. 6231–6239. Dostupné z: http://papers.nips.cc/paper/7203-the-expressive-power-of-neural-networks-a-view-from-the-width.pdf

[17] VAN ROSSUM, G.: *Python Tutorial.* Technical Report CS-R9526, Centrum voor Wiskunde en Informatica, Amsterdam, 1995

[18] PASZKE, A. et al.: *PyTorch: An Imperative Style, High-Performance Deep Learning Library.* Advances in Neural Information Processing Systems 32, 2019, s. 8024–8035. Dostupné z: https://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[19] VAN DER WALT, S., COLBERT, C., VAROQUAUX, G.: *The NumPy Array: A Structure for Efficient Numerical Computation.* Computing in Science & Engineering, 2011, 13. vyd., s. 22–30, DOI:10.1109/MCSE.2011.37

[20] VIRTANEN, P. et al.: *SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python.* Nature Methods, 2020, 17. vyd., s. 261–272. Dostupné z: https://rdcu.be/b08Wh. DOI:10.1038/s41592-019-0686-2

[21] HUNTER, J.D.: *Matplotlib: A 2D Graphics Environment.* Computing in Science & Engineering, 2007, 9. vyd., s. 90–95, DOI:10.1109/MCSE.2007.55

[22] PEDREGOSA, F. et al.: *Scikit-learn: Machine Learning in Python.* Journal of Machine Learning Research, 2011, 12. vyd., s. 2825–2830

[23] NICKOLLS, J. et al.: *Scalable Parallel Programming with CUDA.* ACM Queue, 2008, 6. vyd., č. 2, s. 40–53

8. Zoznam použitých skratiek a symbolov

d	Priemer tyče
E	Youngov modul pružnosti v ťahu
E_T	Modul spevnenia
R_e	Medza klzu
$v_{[A-G]}$	Posuv rovnacieho valca v horizontálnom smere kolmo na osu tyče
$w_{[A-G]}$	Posuv rovnacieho valca vo vertikálnom smere
β	Natočenie rovnacieho valca okolo vertikálnej osi voči osi tyče
k_0	Vstupná krivosť
k	Výstupná krivosť
A	Ťažnosť materiálu tyče
$[A-G]_L$	Ľavý zaväzbený uzol rozdvojenej okrajovej podmienky
$[A-G]_P$	Pravý zaväzbený uzol rozdvojenej okrajovej podmienky
L_v	Vzdialenosť dvoch rovnacích valcov
v_s	Rýchlosť rovnania
R	Reakčná sila rovnacieho valca na tyč
σ_{rez}	Reziduálne napätie materiálu tyče
ftoler	Silové tolerančné kritérium rýchleho algoritmu
mtoler	Momentové tolerančné kritérium rýchleho algoritmu
utoler	Priehybové tolerančné kritérium rýchleho algoritmu
D	Dataset
$\mathcal{P}$	Rozdelenie datasetu
$\mathbf{x}$	Vektor vstupov algoritmu strojového učenia
$\mathbf{y}$	Vektor výstupov algoritmu strojového učenia

$\mathcal{H}$	Trieda hypotéz algoritmov strojového učenia
h	Funkcia algoritmu strojového učenia
$\mathcal{L}$	Množina chybových funkcií
ℓ	Chybová funkcia
$\mathbf{w}$	Vektor váhových parametrov nadroviny perceptrónu
b	Člen posunutia nadroviny perceptrónu
ϕ	Kernelizačná mapovacia funkcia
$\mathbf{U}$	Matica váhových parametrov nadrovín neurónových sietí
$\mathbf{c}$	Vektor posunutí nadrovín neurónových sietí
p	Parameter pravdepodobnosti vynulovania vstupu vrstvy Dropout
γ	Parameter váh vrstvy LayerNorm
β	Parameter posunutia vrstvy LayerNorm
g	Gradient chybovej funkcie
$\mathbf{s}$	Vektor zmeny váhových parametrov nadrovín neurónových sietí
α	Koeficient učenia
ε	Koeficient záporného sklonu aktivačnej funkcie LeakyReLU
δ	Odchýlka predikcie neurónovej siete od skutočnej hodnoty
s_i	Hodnota škálovania výstupu neurónovej siete