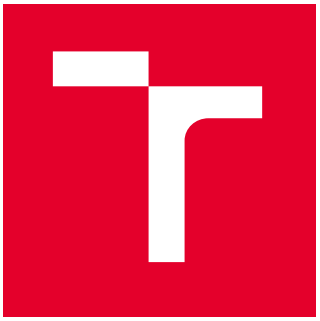


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STAVEBNÍ

FACULTY OF CIVIL ENGINEERING

ÚSTAV GEOTECHNIKY

INSTITUTE OF GEOTECHNICS

METAMODEL GEOTECHNICKÉ KONSTRUKCE

METAMODEL OF GEOTECHNICAL STRUCTURE

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — **Lucie Maslowská**

AUTHOR

VEDOUCÍ PRÁCE — **Ing. Pavel Koudela**

SUPERVISOR

BRNO 2025

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STAVEBNÍ

Zadání bakalářské práce

Ústav:	Ústav geotechniky
Studentka:	**Lucie Maslowská**
Vedoucí práce:	**Ing. Pavel Koudela**
Akademický rok:	2024/25
Studijní program:	B0732A260005 Stavební inženýrství
Studijní obor:	Konstrukce a dopravní stavby

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

Metamodel geotechnické konstrukce

Stručná charakteristika problematiky úkolu:

V dnešní době jsou nejpokročilejším způsobem navrhování a analýzy geotechnickcých konstrukcí metody založené na konečných prvcích a konečných diferencích. Za využití vhodných materiálových modelů jsou tyto nástroje silným prostředkem jak získat velmi věrohodné výsledky. U rozsáhlých modelů a za využití pokročilých materiálových modelů však narůstá časová náročnost řešení problémů. V moderním stavitelství se čím dál více uplatňují různé optimalizace úloh. Materiálové, tvarové, topologické. Ty však vyžadují řádově desítky až stovky opakování výpočtu konstrukce, kde je časová náročnost již extrémní. Tato negativa je možné řešit vytvořením tzv. metamodelu, který je náhradou modelu vytvořeného například pomocí MKP. Metamodely jsou obyvkle vytvářeny pomocí deterministických analytických metod. Po vytvoření metamodelu je výhodou rychlost řešení, která je až o několik řádů nižší než u plných MKP modelů. Nevýhodou je, že meta-model není plnou náhradou původního numerického modelu, ale je schopen vystihnout pouze změny parametrů numerického modelu, pro které je sestaven. Stěžejní ve výzkumu použití metamodelů je problematika nalezení metod vytváření metamodelů a volba parametrů, které jsou využívány metodami pro vytváření metamodelů.

Cíle a výstupy bakalářské práce:

Cílem je vytvoření metamodelu geotechnické konstrukce s přijatelnou chybou vůči řešení této konstrukce, poskytnuném pomocí softwaru na bázi metody konečných prvků.

Seznam doporučené literatury a podklady:

Buljak, V., & Maier, G. (2011). Proper Orthogonal Decomposition and Radial Basis Functions in material characterization based on instrumented indentation. Engineering Structures, 33(2), 492-501. https://doi.org/10.1016/j.engstruct.2010.11.006

Khaledi, K., Miro, S., König, M., & Schanz, T. (2014). Robust and reliable metamodels for mechanized tunnel simulations. Computers and Geotechnics, 61, 1-12. https://doi.org/10.1016/j.compgeo.2014.04.005

Zhao, D., Xue, D. A multi-surrogate approximation method for metamodeling. Engineering with Computers 27, 139–153 (2011). https://doi.org/10.1007/s00366-009-0173-y

Shahzadi G, Soulaïmani A. Deep Neural Network and Polynomial Chaos Expansion-Based Surrogate Models for Sensitivity and Uncertainty Propagation: An Application to a Rockfill Dam. Water. 2021; 13(13):1830. https://doi.org/10.3390/w13131830

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku.

V Brně, dne 25. 11. 2024

L. S.

doc. Ing. Lumír Miča, Ph.D. vedoucí ústavu	Ing. Pavel Koudela vedoucí práce

prof. Ing. Rostislav Drochytka, CSc., MBA, dr. h. c.
děkan

ABSTRAKT

Práce se zabývá využitím metamodelování v geotechnice, a to především při aproximaci MKP modelu metamodelem. Metamodel je model modelu a jeho použití umožňuje snížit výpočetní čas a výkon u mnohokrát opakovaných výpočtu, např. při optimalizaci úloh. První část práce se soustředí na vysvětlení základního principu 5 vybraných metamodelovacích metod: Polynomiální regrese, Moving Least Squares, Proper Orthogonal Decomposition v kombinaci s radiální bázovou funkcí, Rozvoj polynomiálním chaosem a nejvíce pozornosti bylo věnováno neuronovým sítím založeným na strojovém učení. V práci jsou popsány jednotlivé parametry neuronových sítí a možnosti jejich nastavení, přičemž tyto poznatky pak byly uplatněny v části druhé. V té je popsáno vytvoření neuronové sítě aproximující výpočet deformací matematického modelu pažící konstrukce. Proběhlo testování různých nastavení neuronové sítě vytvořené v programovacím jazyku Python za účelem nalezení kombinace parametrů, při jejichž nastavení se neuronová síť nejvíce přiblíží výstupním hodnotám z původního MKP modelu. Výsledkem práce je vytrénovaná síť předpovídající deformace matematického modelu konstrukce na základě materiálových parametrů zeminy s chybou okolo 2,5 %. Pro použité nastavení neuronové sítě pak existuje teoretický předpoklad, že bude s podobnou přesností předpovídat chování matematických modelů i jiných konstrukcí. Tématem vyžadující další zkoumání se však ukázala problematika kvality učících dat při využití metamodelovacích technik.

KLÍČOVÁ SLOVA

metamodel, náhradní model, neuronové sítě, strojové učení, metoda konečných prvků, geotechnická konstrukce

ABSTRACT

This thesis focuses on the application of metamodeling in geotechnical engineering, especially on approximating finite element models using metamodels. A metamodel is model of a model and is used to reduce computation time for tasks that involve many repeated analyses, such as optimalization. In the first part of the thesis, five metamodeling approaches are presented: Quadratic polynomial regression, Moving Least Squares, Proper Orthogonal Decomposition with radial basis function and artificial neural networks based on machine learning. Great emphasis is placed on neural networks, as they are the main topic of the second part of the thesis. In the second part is described creation process of a neural network in Python. The neural network is designed to approximate the calculation of deformations in a mathematical model of a retaining structure. Various configurations of the neural network's key parameters were tested to identify parameter combination for which the neural network most accurately approximates the results of the original FEM model. The outcome of the thesis is a trained neural network that predicts deformations of the mathematical model based on soil material parameters with 2,5 % error. There is a theoretical assumption that the chosen configuration of the neural network can be with similar accuracy apllied to predict a behaviour of other models. However, the issue of training data quality has proven to be a subject requiring further investigation.

KEYWORDS

matamodel, surrogate model, neural networks, machine learning, finite element method, geotechnical structure

BIBLIOGRAFICKÁ CITACE

Citace práce v tištěné podobě:

MASLOWSKÁ, Lucie. *Metamodel geotechnické konstrukce*. Bakalářská práce. Pavel KOUDELA (vedoucí práce). Brno: Vysoké učení technické v Brně, Fakulta stavební, 2025.

Citace elektronického zdroje:

MASLOWSKÁ, Lucie. *Metamodel geotechnické konstrukce*. Online, bakalářská práce. Pavel KOUDELA (vedoucí práce). Brno: Vysoké učení technické v Brně, Fakulta stavební, 2025. Dostupné z: https://www.vut.cz/studenti/zav-prace/detail/166896.

PODĚKOVÁNÍ

Tímto bych ráda poděkovala vedoucímu bakalářské práce Ing. Pavlu Koudelovi za odborné vedení, podnětné připomínky a zároveň za návrh zajímavého tématu práce. Mé rodině pak patří poděkování za podporu ve studiu.

PROHLÁŠENÍ AUTORA O SHODĚ TIŠTĚNÉ A ELEKTRONICKÉ FORMY ZÁVĚREČNÉ PRÁCE

Prohlašuji, že elektronická forma odevzdané diplomové práce s názvem Metamodel geotechnické konstrukce je shodná s odevzdanou listinnou formou.

V Brně dne 29. 5. 2025

Lucie Maslowská

Autor práce

PROHLÁŠENÍ AUTORA O PŮVODNOSTI ZÁVĚREČNÉ PRÁCE

Prohlašuji, že jsem bakalářskou práci s názvem Metamodel geotechnické konstrukce zpracovala samostatně a že jsem uvedla všechny použité informační zdroje.

V Brně dne 29. 5. 2025

Lucie Maslowská

Autor práce

OBSAH

1 ÚVOD

V současné době se k řešení stavebních konstrukcí používají výpočetní modely založené na numerických metodách, např. v praxi nejvíce využívaná metoda konečných prvků (MKP). Pomocí MKP jsme teoreticky schopni vyřešit jakoukoli konstrukci a výsledkem analýzy je kromě numerického řešení i názorná grafická vizualizace. Značnou nevýhodou je však nutnost použití nákladného softwaru vyžadujícího velký výpočetní výkon a časová náročnost výpočtů. Potřebná je také znalost uživatele daného softwaru. Tyto negativa stojí za snahou aproximovat aktuálně používané výpočetní modely.

Metamodeling je proces vytváření modelu již existujícího a zaběhnutého modelu. Tento model modelu pak označujeme jako metamodel. Principem jejich vytváření je hledání vztahu reprezentujícího spojitost mezi vstupem a výstupem. Tímto vztahem může být vztah matematický, např. polynomiální regrese nebo aproximace polynomiálním chaosem, nebo složitější modely jako umělé neuronové sítě. Právě na umělé neuronové síti bude založen metamodel testovaný v rámci této práce [1, 2, 3].

V porovnáni s MKP přináší využití metamodelu několik výhod. Cílem modelování v softwaru MKP totiž není pouze samotný posudek, kdy získáme hodnoty vybraných veličin, ale i optimalizace parametrů konstrukce. Ta umožňuje co nejúspornější řešení, ať už z hlediska ekonomického nebo stále více skloňovaného hlediska ekologického. Aproximace vztahu mezi vstupními a výstupními parametry na vztah výrazně jednodušší přináší lepší pochopení modelu, pochopení vztahu pak umožňuje efektivnější optimalizaci [2].

Použití jednoduchých matematických metamodelů (např. polynomiální regrese) je však obvykle výrazně limitováno předpoklady, které musí řešený model splňovat. To znamená, že jednoduchý matematický metamodel bude vykazovat využitelnou přesnost pouze pro úzkou skupinu jednoduchých problémů. Pro složitější případy pak tyto nástroje obvykle nepřináší relevantní výsledky [4].

U složitějších metamodelů (např. umělé neuronové sítě) není vztah mezi vstupy a výstupy tak jednoduše odvoditelný, ale využití při optimalizaci je i tak velmi výhodné. Hledání nejvýhodnějších parametrů numerickými metodami znamená stovky opakování výpočtu, což povede k velké výpočetní zátěži, a především dlouhému výpočetnímu času. Při využití metamodelu lze tento čas výrazně zkrátit, a to řádově na zlomky vteřin [5].

Možné uplatnění metamodelovácích technik bylo dále v popsáno při provádění hybridních simulací v reálném čase, kdy je žádoucí, aby výpočet jednotlivých kroků byl co nejkratší (popsáno v [1]) nebo při aproximaci nákladných laboratorních zkoušek (popsáno v [6]).

Původní modely (např. model vytvořený metodou MKP) jsou však i při využití metamodelů v prvotní fázi potřebné, a to pro získání dat pro učení. Samotný proces metamodelování se skládá ze tří hlavních fází:

1. Získání počátečních dat pro tvorbu metamodelu – např. pomocí simulací s využitím softwaru na principu metody konečných prvků. Data se skládají ze vstupních a příslušných výstupních dat. Příkladem může být metamodel z praktické části této bakalářské práce, vytvořený pro výpočet modelu pažící konstrukce. Vstupními parametry jsou pro tento metamodel vytvořený metodou neuronových sítí čtyři charakteristiky zeminového prostředí, které je v příkladu popsáno elastoplastickým materiálovým modelem Hardening Soil: soudržnost zeminy c, úhel vnitřního tření φ, sečnový modul tuhosti v primárním zatěžování E_{50}^{ref} a poměr tuhosti v odtížení a přitížení ku tuhosti v primárním zatěžování $E_{50}^{ref} vs E_{ur}^{ref}$. Výstupními parametry metamodelu jsou pak tři zvolené deformace modelu pažící konstrukce.
2. Tvorba metamodelu a jeho učení (trénování)
3. Validace metamodelu – od počátečních dat se oddělí část, která se nevyužije ke tvorbě, ale až k porovnání s výstupy metamodelu [7].

2 METODY VYUŽÍVANÉ PRO VYTVÁŘENÍ METAMODELŮ

2.1 Polynomiální regrese

Nejjednodušší nástroj metamodelování je polynomiální regrese. Velmi rozšířená je jednoduchá regrese, kdy se snažíme aproximovat závislost predikované proměnné (výstupu) na jedné vstupní proměnné. U složitějších problému, kdy výstup nezávisí pouze na jedné vstupní veličině, se pak uplatňuje vícenásobná regrese [6].

Cílem jednoduché polynomiální regrese je tréninková data ve formě uspořádaných dvojic $[x_i, y_i]$ proložit křivkou (polynomem) $f(x)$, která data co nejlépe kopíruje. Pro hledané $f(x)$ platí:

$$S = \sum_i [y_i - f(x_i)]^2 = min, \tag{2.1}$$

kde hledáme polynom $f(x)$, pro který je součet S druhých mocnin vzdáleností bodů y_i od funkčních hodnot $f(x_i)$ minimální [6].

Vícenásobná regrese umožňuje předpověď hodnoty výstupní veličiny na základě více vstupních proměnných (např. více materiálových parametrech zeminy) na obdobném principu. Výsledkem analýzy je pak následující rovnice reprezentující aproximované vztahy mezi vstupními a výstupními veličinami

$$Y = a_0 + a_1 x_{i1} + a_2 x_{i2} + a_3 x_{i3} + \cdots + a_k x_{ik}, \tag{2.2}$$

kde Y je závislá proměnná (výstup metamodelu), X_i nezávislé proměnné (vstupy metamodelu) a a_k regresní koeficienty [6].

2.2 Moving Least Squares

Tato metoda metamodelování vznikla původně jako metoda interpolace a vyhlazování bodových mračen v oblasti počítačové grafiky a geometrického modelování – pomocí metody nejmenších pohyblivých čtverců se ze známých bodů povrchu vygenerují souřadnice zbylých bodů povrchu [8].

Až později se tato metoda začala využívat pro vytváření aproximací napodobující chování původního modelu. Původní uplatnění v počítačové grafice však může sloužit k lepšímu pochopení, místo skutečného povrchu se však dopočítávají souřadnice funkce metamodelu v prostoru.

Metoda nejmenších pohyblivých čtverců je založená na již zmiňované regresi. V klasické regresi popsané v předchozí podkapitole se vytváří jedna globální regrese pro celý soubor dat. V metodě Moving Least Squares se však namísto jedné globální regrese v každém bodě prostoru vytváří lokální model na základě dat v předem stanoveném okruhu daného bodu [8].

Predikci hodnoty v novém bodě x získáme přes váženou regresi metodou nejmenších čtverců hodnot okolních bodů. Váhy jednotlivých vstupních dat závisí na vzdálenosti bodů od predikovaného bodu x, přičemž čím je bod blíže, tím větší má váhu [8].

Při aproximaci modelu $f(x)$ metodou Moving Least Squares hledáme takový polynom $p(x)$, který minimalizuje váženou chybu

$$J(p) = \sum_i \theta(\|\bar{x} - x_i\|)\, \|f(x_i) - p(x_i)\|^2. \tag{2.3}$$

V rovnici (2.3) je $\theta(\|\bar{x} - x_i\|)$ funkční hodnota váhové funkce pro bod x_i v Euklidovské vzdálenosti $\|\bar{x} - x_i\|$ od aproximovaného bodu $\bar{x}$, x_i známý bod, a $\|p(x_i) - y_i\|^2$ druhá mocnina vzdálenosti funkční hodnoty $p(x_i)$ od funkční hodnoty $f(x_i)$ [9, 10].

K určení hodnot vah bodů na základě vzdálenosti se nejčastěji používá Gaussova váhová funkce

$$\theta(d) = e^{-\frac{d^2}{h^2}}, \tag{2.4}$$

kde $d = \|\bar{x} - x_i\|$ a h je vyhlazovací parametr, který ovlivňuje citlivost modelu na lokální změny vstupních dat [9, 10].

Větší hodnota h vede k hladší aproximaci, která může lépe zobecňovat, ale zároveň přehlédnout drobné detaily. Menší hodnota h umožňuje metamodelu zachytit jemné detaily, ale také může dojít k overfittingu – metamodel se příliš přizpůsobuje detailům dat, které jsou ve skutečnosti pouze náhodné šumy a chyby.

Díky použití Gaussovy váhové funkce se jedná o spojitou diferencovatelnou aproximaci (nemá ostré rohy a skoky). Tato vlastnost je velmi výhodná pro usnadnění optimalizace za využití derivací [9, 10].

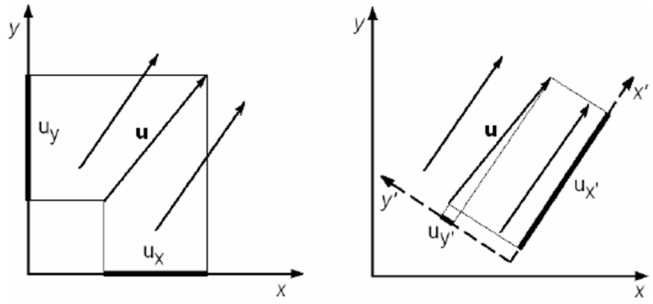
2.3 Proper orthogonal decomposition a radiální bázová funkce

Tato metoda byla popsána v [11] a skládá se ze dvou hlavních částí. Nejprve je provedena dekompozice matice označované jako „snapshot matrix“ do nového souřadnicového systému a v tomto souřadnicovém systému jsou pak aproximovány výsledky původního modelu zachycené v snapshot matici[12].

2.3.1 Proper orthogonal decomposition

Snapshot matice je tvořena výstupy z původního modelu. Jedná se o n_p výsledků pro různá nastavení (kombinace parametrů) pro m sledovaných veličin. Výslednou matici U lze interpretovat jako sadu o m, n_p-rozměrných vektorů.

Na takto vytvořenou snapshot matrix U se aplikuje metoda Proper orthogonal decomposition (POD). Cílem je transformace vektorů matice U do takových souřadnic, aby bylo dosaženo maxima součtu projekcí původních vektorů na novou osu $x´$. Obdobným způsobem se volí zbylé souřadnicové osy nového pravoúhlého souřadnicového systému. Volba tohoto nového souřadnicového je pak znázorněna na obrázku 2.1 [11, 13].



Obrázek 2.1: Volba nového souřadnicového systému v metodě Proper orthogonal decomposition [11].

Vztah pro výpočet POD vektorů je popsán rovnicí

$$\varphi^i = U \cdot v^i \cdot \lambda_i^{-\frac{1}{2}}, \tag{2.5}$$

založené na vlastních číslech λ_i a vlastních vektorech v^i matice $D = U^T \cdot U$. Tyto vektory tvoří matici POD báze Φ rozměru $n_p \times m$, ve které jsou vektory φ^i uspořádaný dle odpovídajícího vlastního čísla λ_i od největšího po nejmenší. Výsledkem tohoto uspořádání vektorů φ^i v matici Φ je seřazení směrů podle výstižnosti aproximace snapshot matice v tomto směru. To umožňuje jednoduše vybrat prvních k řádků pro nejpřesnější možnou aproximaci na základě k členů [11].

Pomocí matice Φ se vyjádří snapshot matice v novém souřadnicovém systému lineární transformací

$$U = \Phi \cdot \mathrm{A}, \tag{2.6}$$

kde A je matice amplitud obsahující projekční koeficienty [11].

2.3.2 Interpolace amplitud pomocí radiální bázové funkce

V tomto kroku je cílem nalézt spojitý aproximační vztah mezi trénovacími body v prostoru zachycenými snapshot maticí. Předchozí úprava pomocí POD umožní s velkou přesností aproximovat pouze na základě k členů ve směrech danými vektory φ^i s největšími vlastními čísly λ_i [11].

Cílem je aproximovat redukovanou matici amplitud $\overline{\mathrm{A}}$. Jednotlivé projekční koeficienty $\bar{a}_k$ matice $\overline{\mathrm{A}}$ se vyjádří nikoliv jako konstanty, ale jako nelineární funkce vstupních parametrů modelu. Toho je docíleno vyjádřením složek vektorů amplitud $\bar{a}_k$ rovnicí

$$\bar{a}_k(z) = \sum_{i=1}^{m} b_k^i \, g_i(z), \tag{2.7}$$

kde b_k^i jsou hledané interpolační koeficienty, k je zvolený počet řádků redukované matice amplitud $\overline{\mathrm{A}}$ ($k < n_p$) a $g_i()$ je libovolná radiální bázová funkce se středem v bodě z_i [11].

Rovnice (2.7) lze rozepsat jako $k \times m$ rovnic o stejném počtu neznámých, kdy m je počet vzorků (trénovacích dat). Řešením této soustavy rovnic dostaneme matici B tvořenou interpolačními koeficienty [11].

Předpověď výsledků původního modelu je pak vyjádřena rovnicí

$$u(z) \approx \overline{\Phi} \cdot B \cdot g(z), \tag{2.8}$$

kde $u(z)$ je odhadovaná funkční hodnota v bodě z, $\overline{\Phi}$ matice transformovaných bází, B matice obsahující vypočtené interpolační koeficienty a $g(z)$ funkční hodnota zvolené radiální bázové funkce v bodě z [11, 13].

2.4 Rozvoj polynomiálním chaosem

Rozvoj polynomiálním chaosem (Polynomial Chaos Expansion – PCE) je metoda, která výstup modelu aproximuje rozvojem (součtem) polynomů. Využívá se k metamodelování, citlivostní analýze a kvantifikaci nejistot matematických modelů [14, 15].

Základní princip této metody je popsán následující rovnicí

$$Y \approx \sum_{\alpha \in \mathbb{N}^d} y_\alpha \psi_\alpha(X), \tag{2.9}$$

kde Y je výstup modelu, d počet vstupních veličin X, y_α neznáme deterministické koeficienty a $\psi_\alpha()$ jsou bázové funkce [16].

2.4.1 Bázové funkce

Bázové funkce jsou tvořeny ortogonálními (vzájemně kolmými, nezávislými) polynomy o více proměnných, přičemž proměnnými jsou nezávislé vstupní parametry X_i $(i = 1, \dots, d)$. Díky ortogonalitě polynomů lze snadno vypočítat koeficienty y_α a také oddělit vliv jednotlivých proměnných [16].

Bázové funkce se volí na základě rozdělení vstupních veličin. Sady ortogonálních polynomů jsou definovány pro standartní rozdělení, Legendreovy polynomy pro rovnoměrné rozdělení $\mathcal{U}(-1,1)$ a Hermiteovy polynomy pro normální (Gaussovo) rozdělení $\mathcal{N}(0,1)$. Pro jiné rozdělení vstupních veličin lze využít isopravděpodobnostní transformaci nebo numericky určené polynomy [15, 16].

Pro praktické využití se z důvodu výpočetní náročnosti omezuje množství bázových funkcí na konečný počet P, a to podle vztahu

$$P = \frac{(M+p)!}{M!\,p!}, \tag{2.10}$$

kde n je počet vstupních proměnných a p stanovený maximální stupeň polynomu [16].

Při popisování složitějších funkcí, kdy je potřeba použít polynomy vyšších stupňů, se tento počet bázových funkcí redukuje na řidší sadu. K výběru nejdůležitějších bázových funkcí se pak používají metody jako metoda nejmenších úhlů nebo hyperbolické oříznutí [15].

2.4.2 Stanovení deterministických koeficientů

Ke stanovení deterministických koeficientů se používají dva typy přístupů – intruzivní a neintruzivní. Nevýhodou intruzivních metod je nutnost zasáhnout do vnitřní struktury původního modelu, což je ale v některých případech obtížně aplikovatelné, např. když je původní model vytvořen ve výpočetním softwaru [16].

Naopak neintruzivní metody, obdobně jako ostatní metamodelovací techniky, chápou původní model jako tzv. černou skříňku a pracují pouze se vstupy a jim odpovídajícími výstupy modelu. Využívanou metodou je např. metoda nejmenších čtverců, která je popsána rovnicí (2.1) [16].

2.5 Neuronové sítě

Poslední metamodelovací technika popsaná v teoretické části této práce jsou neuronové sítě. Ty se díky své univerzálnosti staly běžným nástrojem v mnoha oblastech. Možné využití ve stavební praxi je právě k aproximaci používaných modelů.

Počítače jsou schopné provádět složité numerické výpočty, pokud je problém formulován pomocí formálních matematických pravidel – algoritmů. Některé problémy ale nelze, nebo jen velmi obtížně, pomocí algoritmů popsat. A to zpravidla tehdy, pokud záleží na mnoha faktorech. Právě k těmto problémům je zapotřebí využít komplexnějších nástrojů [17].

Umělé neuronové sítě vznikly jako snaha napodobit fungování lidského mozku. Ten se skládá z velkého množství jednoduchých, paralelně uspořádaných neuronů. Neurony jsou propojeny pomocí elektrických signálů, přičemž různá spojení mají různé váhy. V neuronu jsou pak příchozí signály sečteny, chemickou reakcí upraveny a výsledný signál poslán dále. V umělých neuronových sítích jsou pak elektrické signály nahrazeny čísly, které si neurony mezi sebou předávají a dále s nimi pracují pomocí matematických operací a funkcí. Při průchodu umělou neuronovou sítí se tedy z čísel reprezentující vstupy stanou čísla reprezentující výstupy [18].

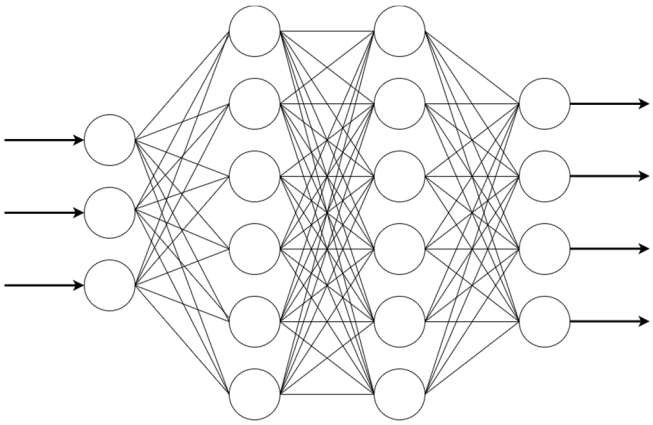
Umělé neuronové sítě (dále jen neuronové sítě) jsou na rozdíl od obvyklých algoritmů schopny řešit problémy, aniž by jim programátor formálně specifikoval všechny informace popisující danou problematiku. Algoritmus neuronové sítě zvládne „pochopit“ vztahy mezi vstupy a výstupy sám, a to na základě zkušenosti. Zkušenost si můžeme představit jako soubor vstupů a jim příslušných řešení [14].

Schopnost počítačů něco chápat bývá označovaná jako umělá inteligence (artificial intelligence – AI). Neuronové sítě jsou modely založené na strojovém učení, jenž leží na pomezí uměle inteligence. V současné době existuje velké množství definic umělé inteligence, které se liší právě i v tom, jestli lze strojové učení do AI zahrnout, či nikoliv. Spíš než o otázku technického typu se ale jedná o otázku filozofickou, a nalézt na ni odpověď, není předmětem této bakalářské práce.

Faktem zůstává, ať už neuronové sítě za AI považujeme nebo ne, že po úspěšném tréninku jsou neuronové sítě v rámci stejné třídy schopné řešit i problémy, na kterých nebyly specificky trénovány. Pod třídou problému si lze v rámci stavebnictví a geotechniky představit konkrétní stavební konstrukci, např. pažící konstrukci daných rozměrů, a jednotlivými problémy jsou pak situace kdy je konstrukce obklopena zeminami různých parametrů.

2.5.1 Struktura neuronových sítí

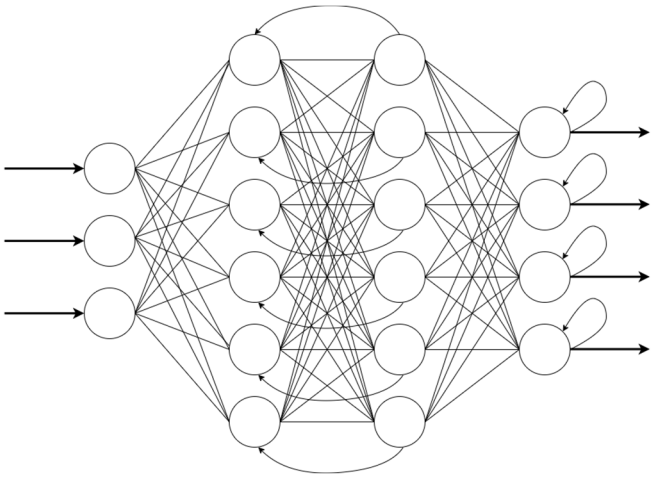
Základní stavební jednotkou neuronových sítí jsou jednoduché neurony. Ty jsou uspořádány do jednotlivých vrstev, přičemž každý neuron je spojen se všemi neurony předchozí i následující vrstvy. Tyto spojení reprezentují přenos informací mezi jednotlivými neurony a každé takové spojení má přirazenou svou číselnou váhu. Přenášenou informací je výstup z předešlého neuronu, ten je pak při průchodu spojením vynásoben konkrétní váhou [4].



Obrázek 2.2: Dopředná neuronová síť o dvou skrytých vrstvách.

Neuronové sítě popsaná v předchozím odstavci a znázorněná na obrázku 2.2 je označována jako fully connected-feedforward. Dopředné neuronové sítě se využívají pro úlohy jako rozpoznávání obrázků a také regresní úlohy [18].

Druhým typem jsou rekurentní neuronové sítě, kde na rozdíl od dopředných neproudí informace pouze jedním směrem, ale data mohou cirkulovat zpět. Také si neuron při průchodu dat částečně uchovává předchozí stav. To umožňuje i časově závislé analýzy jako např. převod řeči na text. Rekurentní neuronová síť je znázorněna na obrázku 2.3 [18].



Obrázek 2.3: Rekurentní neuronová síť se znázorněnou zpětnou cirkulací dat.

Rekurentní neuronové sítě jsou však velmi náročné na trénování a pro regresní úlohy v rámci geotechniky, na které je tato bakalářská práce zaměřena, je jejich využití nevhodné. Z tohoto důvodu jim už nebude věnováno více pozornosti.

Neuronové sítě, především ty dopředné, obsahují tři typy vrstev: vstupní vrstvu (input layer), skrytou vrstvu/vrstvy (hidden layers) a výstupní vrstvu (output layer).

Počtem skrytých vrstev a množstvím neuronů v jednotlivých vrstvách se zabývá topologie neuronových sítí, konkrétní uspořádání dané sítě pak bývá označováno jako architektura neuronové sítě [4, 18].

Počet neuronů v první (vstupní) vrstvě odpovídá počtu vstupních parametrů neuronové sítě. Neurony této vrstvy mají každý pouze jeden vstup – každý neuron má pevně přirazen jiný parametr ze souboru vstupních dat [4].

Obdobně platí, že počet neuronů ve výstupní vrstvě se rovná počtu očekávaných výstupů neuronové sítě. Každý z těchto neuronů má pak jeden výstup odpovídající konkrétnímu výstupu metamodelu [4].

2.5.2 Neuron

Vstupními daty pro neurony u feedforward neuronové sítě jsou výstupy neuronů předešlé vrstvy (neplatí pouze u vrstvy vstupní).

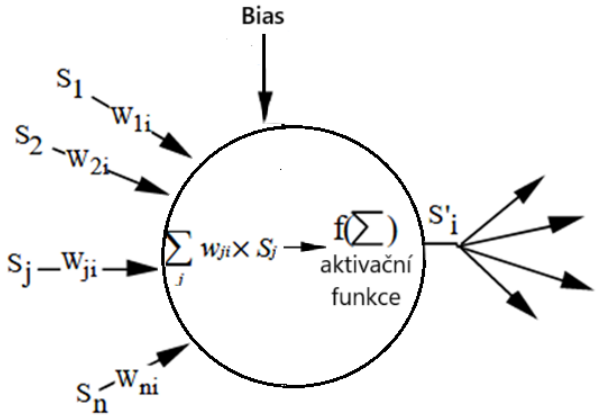
Základem neuronu je propagační funkce (propagation function), která transformuje výstupy předešlých neuronů do vstupu daného neuronu. Obvykle se jedná o váženou sumu, jak bude znázorněno na obrázku. Vstupní signály jsou tedy vynásobeny příslušnými váhami daných neuronových spojení a tyto hodnoty jsou sečteny [4, 18].

Kromě propagační funkce je neuron tvořen funkcí aktivační, někdy nazývané také přenosová. Tato matematická funkce určuje reakci neuronu na vstupní hodnoty – jestli, popřípadě v jaké míře, bude neuron aktivován. Důležitá je prahová hodnota, tedy hodnota, při které je neuron aktivován [18].

Aktivační funkce je obvykle definována globálně pro skupinu (vrstvu) neuronů. Při tvorbě neuronových sítí se používají různé typy aktivačních funkcí, přičemž vybraným typům se věnuje samostatná podkapitola 2.5.3. Typ aktivační funkce je obvykle pevně daný a není předmětem trénování učícími algoritmy. Aby tedy mohla být při trénování ovlivněna alespoň prahová hodnota vstupů jednotlivých neuronů, byl přidán parametr neuronu bias. Ten se sečte spolu se vstupy z neuronů předešlé vrstvy v rámci propagační funkce [4, 18].

Z důvodu snadnější implementace biasu do neuronové sítě a umožnění změny jejich parametrů v průběhu tréninkového procesu, jsou biasy v neuronové síti vytvořené jako samostatné neurony. Tyto biasové neurony pak mají pouze jedno spojení směrem k příslušnému neuronu v neuronové síti. Výstupem biasového neuronu je vždy hodnota 1, přičemž hodnoty biasu je dosaženo definováním váhy připojení tohoto neuronu na hodnotu, která se rovná hodnotě biasu. Hodnota biasu tedy přichází do neuronu spolu s výstupy neuronů z předešlé vrstvy. Díky tomuto uspořádání pak mohou být hodnoty biasů trénovány stejným způsobem jako váhy spojení klasických neuronů [18].

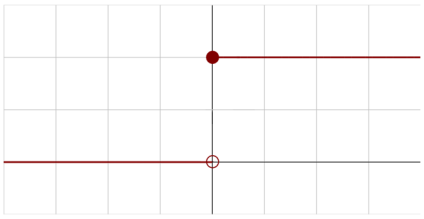
Procesy probíhající v rámci neuronu popsány v této podkapitole jsou schematicky znázorněny na obrázku 2.4.



Obrázek 2.4: Zpracování dat v neuronu sečtením příchozích signálů a vyhodnocení aktivační funkcí $f(\sum)$, [4] upraveno.

2.5.3 Typy aktivačních funkcí

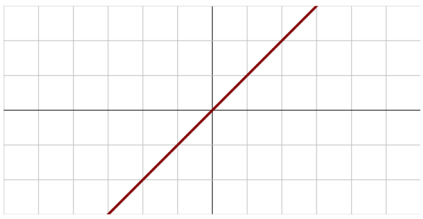
Prahová aktivační funkce – nejjednodušší aktivační funkce, závisí pouze na prahové hodnotě. Pokud je vstup aktivační funkce větší než prahová hodnota, neuron je plně aktivován. V opačném případě je neuron deaktivován. V zápisu funkce je x vstup do neuronu a y výstup [17, 18].



Obrázek 2.5: Graf prahové aktivační funkce, [19]uraveno.

$$y = \begin{cases} 0 & if \;\; x < 0 \\ 1 & if \;\; x \geq 0 \end{cases} \qquad (2.11)$$

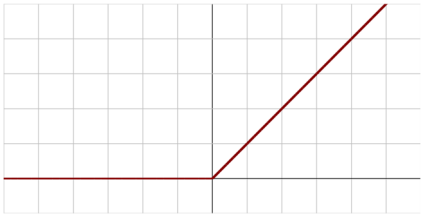
Lineární aktivační funkce – definována přímkou $y = x$. Tato aktivační funkce tedy hodnoty v rámci neuronové sítě nijak neovlivní. Použití lineární funkce v rámci celé neuronové sítě je velmi nevýhodné, a to z důvodu, že aktivační funkce jsou jediné, které do neuronové sítě přináší prvek nelinearity. Při použití lineární funkce tedy bude výstup neuronové sítě jen lineární kombinací vstupních dat, a to pro jakýkoli počet neuronových vrstev. Skutečnost, že výstup bude vždy proporcionálně úměrný vstupům, znemožňuje neuronové síti simulovat složitější vztahy. Problémem je také to, že derivací lineární funkce je konstanta, která nezávisí na vstupu x. Na základě změny derivace funkce pro vstupy x však pracují učící algoritmy popsány v podkapitole 2.5.5. Algoritmy tedy nejsou schopny odhalit, jak jednotlivé váhy vstupů do neuronu ovlivňují výstup a na základě toho je upravit. Možné uplatnění lineární aktivační funkce je v kombinaci s nelineární aktivační funkcí, přičemž obvykle se používá pro výstupní vrstvu při řešení regresních úloh [17].



Obrázek 2.6: Graf lineární aktivačn funkce [19].

$$y = x \qquad (2.12)$$

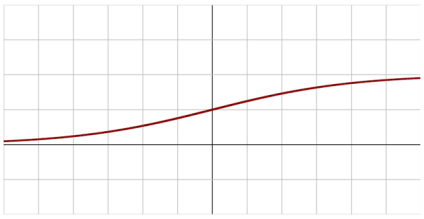
ReLU (rectified linear unit) – funkce je matematicky popsaná rovnicí (2.13). Při překročení prahové hodnoty dojde k aktivaci neuronu, přičemž jeho výstup se bude rovnat vstupu. Tato aktivační funkce je velmi využívaná – je nenáročná na výpočet, ale zároveň se už jedná o funkci nelineární [17].



Obrázek 2.7: Graf aktivační funkce ReLu [19].

$$y = \begin{cases} 0 & if \quad x < 0 \\ x & if \quad x \geq 0 \end{cases} \tag{2.13}$$

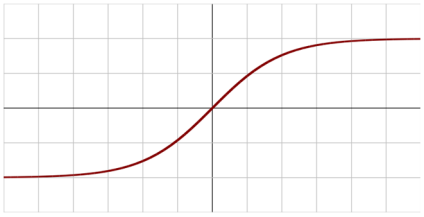
Sigmoidní funkce – matematická funkce ve tvaru *S* s oborem hodnot *(0,1)*. Na rozdíl od předešlých aktivačních funkcí je neuron vždy aktivní, což způsobuje větší výpočetní zátěž při trénování neuronové sítě. Využívá se hlavně v klasifikačních úlohách, ale také pro zahlazení výstupů. Problémem u této funkce je derivace, která má v některých místech funkce velmi malý sklon. Po této tečně se však mění parametry neuronové sítě při jejím trénování, což vede k velmi pomalé optimalizaci [18].



Obrázek 2.8: Graf Sigmoidní aktivační funkce [19].

$$y = \frac{1}{1 + e^{-x}} \tag{2.14}$$

Hyperbolický tangens – aktivační funkce velmi podobná předešlé, rozdíl je v rozsahu výstupů (oboru hodnot), který je pro hyperbolický tangens roven intervalu *(-1,1)*. Stejně jako pro sigmoidní funkci je zde problém s malým sklonem funkce. Zde je však výhodou vycentrování na nulu, kdy negativním vstupům odpovídají negativní výstupy a obdobně pozitivním vstupům odpovídají pozitivním výstupy. Hyperbolický tangens je z tohoto důvodu, umožňujícího lepší trénování, preferován před sigmoidní funkcí pro skryté neuronové vrstvy [18].



Obrázek 2.9: Graf aktivační funkce Hyperbolický tangens [19].

$$y = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.15}$$

2.5.4 Strojové učení

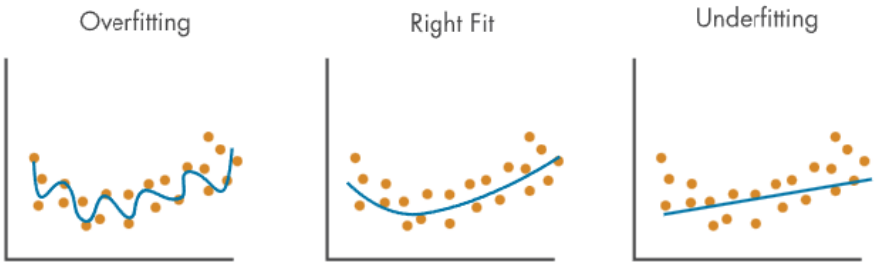
Jak už bylo zmíněno dříve, neuronová síť je metamodel založený na strojovém učení. „Učení“, označované většinou jako trénování neuronové sítě, je u strojového učení založeno na algoritmech. Často se také používá označení z angličtiny „optimizers“. Tyto učící algoritmy definují, jak se během tréninkového procesu dynamicky mění parametry neuronové sítě, a to za účelem dosažení co největší shody s trénovacími daty [18].

Metoda učení neuronových sítí se nazývá error backpropagation, do češtiny pak bývá překládána jako metoda zpětného šíření chyby. Při učení neuronová síť předpoví pro daný vstup odpovídající výstup a následně porovná tuto předpověď s výstupem původního modelu (tedy výstupu, kterému se snaží co nejvíce přiblížit). Ze souboru těchto hodnot je stanovena chybová funkce (loss function), na jejímž základě učící algoritmus upraví parametry neuronové sítě tak, aby tuto chybu minimalizoval. Tento proces je opakován v rámci každé učící epochy [17, 18].

V rámci strojového učení jsou obvykle hodnoty chybové funkce $f(x)$ stanoveny jako střední kvadratická chyba MSE (mean squared error) nebo RMSE (root mean squared error).

Kromě klasické chybové funkce, která se počítá z dat, na kterých se neuronová síť trénuje, se v průběhu tréninkového procesu vyhodnocuje také tzv. validační ztrátová funkce (validation loss). Ta se vyhodnocuje na základě souboru hodnot vstupů a odpovídajících výstupů z původního modelu, na kterých neuronová síť nebyla trénována [18].

Sledování validační ztráty je důležité z toho důvodu, že během učícího procesu může dojít k přetrénování (overlearning, overfitting). To znamená, že metamodel se až moc přizpůsobil datům určených k učení a díky tomu nepostihuje problém globálně. Toto zkreslení pak ovlivňuje schopnost metamodelu generalizovat poznatky o problému na nové data, které nebyly součásti učícího procesu. Přeučení může nastat hlavně u neuronových sítí s velkým počtem neuronů. K lepšímu pochopení principu přeučení může pomoct obrázek 2.10 [20].



Obrázek 2.10: Přetrénování (overfitting) a underfitting neuronové sítě [21].

Overfitting není problém jen neuronových sítí, ale i jiných metamodelů. Zmíněn byl již například v podkapitole 2.2 v souvislosti s metodou nejmenších pohyblivých čtverců.

2.5.5 Učící algoritmy

Základní učící algoritmus je Gradient descent optimizer, který pomocí derivace hledá globální minimum chybové funkce. Gradient je tečna chybové funkce ve směru x, jednoho z původních parametrů neuronové sítě, kterou získáme derivací chybové funkce. Nový parametr získáme pohybem po této tečně směrem k minimu chybové funkce. Jak daleko se parametr x posune, udává stanovená rychlost adaptace (learning rate). Výpočet upraveného algoritmu je popsán rovnicí

$$x_{i+1} = x_i - \alpha * f'(x), \tag{2.16}$$

kde x_i je původní parametr neuronové sítě (např. váha jednoho z neuronových spojení), x_{i+1} nová hodnota téhož parametru, α rychlost adaptace a $f'(x)$ gradient [17].

Gradient descent optimizer (SGD) funguje dobře hlavně pro konvexní funkce, u ostatních nemusí dosáhnout takové přesnosti, pokud je lokální minimum nebo sedlo zaměněno za minimum globální. Nevýhodou je také časová náročnost výpočtu gradientu pro celý soubor dat v jednom kroku. Z těchto důvodu se obvykle používají komplexnější učící algoritmy [18].

V této podkapitole jsou dále popsány vybrané učící algoritmy, které budou zároveň použity v rámci praktické části této bakalářské práce.

Gradient descent with momentum optimizer – nevýhodou klasického SGD je skutečnost, že v plochých místech chybové funkce je gradient velmi malý, a změna parametrů probíhá velmi pomalu. Momentum bylo založena na metafoře z fyziky a zákona zachování hybnosti, kdy na současný pohyb částice má vliv i zrychlení v předchozích krocích. Momentum (hybnost) tedy pomocí klouzavého průměru gradientů upravuje gradient aktuální. Díky tomu nedochází k tak značné změně gradientů v čase. Výpočet rychlosti učení (upravený gradient) popisuje rovnice (2.17)

$$m_{t+1} = \beta m_t + (1 - \beta) g(t), \tag{2.17}$$

kde m_t je průměr předchozích gradientů, β faktor hybnosti (hodnota mezi 0 a 1, typicky 0,9) a $g(t)$ aktuální gradient [18].

RMSprop (Root Mean Square Propagation) – tento učící algoritmus stejně jako SGD with momentum zohledňuje velikost předchozích gradientů pro výpočet aktuálního. Využívá k tomu klouzavý průměr druhých mocnin gradientů. Výhodou RMSprop je také, že tento učící algoritmus počítá pro každý parametr gradienty zvlášť. V rovnici

$$x_{i+1} = x_i - \frac{\alpha}{\sqrt{v_t + \varepsilon}} * g(t), \tag{2.18}$$

je x_i je původní parametr neuronové sítě x_{i+1} nová hodnota téhož parametru, α rychlost adaptace, $g(t)$ gradient v čase t, v_t klouzavý průměr druhých mocnin gradientů a ε malá konstanta zabraňující dělení nulou [17].

Adam (Adaptive Moment Estimation) – Adam kombinuje předchozí dva učící algoritmy: SGD a RMSprop. Díky tomuto spojení Adam tyto učící algoritmy překonal jak v čase učení, tak v přesnosti. Výpočet nových parametrů v rámci trénování neuronové sítě je založen na následující rovnici

$$x_{i+1} = x_i - \frac{m_t}{\sqrt{v_t + \varepsilon}} \alpha * g(t), \tag{2.19}$$

kde α je rychlost adaptace, m_t průměr předchozích gradientů stanovený jako m_{t+1} podle rovnice (2.17) a v_t klouzavý průměr druhých mocnin [17].

Nadam (Nestor Adaptive Moment Estimation) – tento učící algoritmus je velmi podobný Adamovi, jediný rozdíl je v použití Nestorovy urychlené hybnosti gradientu. Toto momentum (hybnost) nebere v potaz jen předchozí gradienty, jak popisuje rovnice (2.19), ale snaží se předpovídat i gradienty budoucí [22].

3 METAMODEL GEOTECHNICKÉ ÚLOHY

3.1 Matematický model geotechnické úlohy

Matematický model, jenž bude pomocí neuronové sítě aproximován, byl vytvořen v softwaru Plaxis 2D založeném na metodě konečných prvků. Nejedná se o model skutečné konstrukce, ale o model vytvořený za účely otestování aproximace matematického modelu neuronovou sítí. V tomto případě nejsou konkrétní vlastnosti matematického modelu zásadní, ale pro úplnost jsou v této podkapitole alespoň základní uvedeny.

Model popisuje chování dvakrát kotvené pažící stěny výšky 10 m a obsahuje 6 jednotlivých fází výstavby:

1. Fáze počátečních podmínek
2. Aktivace podzemní stěny včetně kontaktních prvků na materiálovém rozhraní, výkop na úroveň první etáže
3. Aktivace první kotvy a její předepnutí na 150 kN
4. Výkop na úroveň druhé etáže
5. Aktivace druhé kotvy a její předepnutí na 150 kN
6. Výkop na konečnou úroveň

Pro modelování zeminového prostředí bylo použito odvodněné chování a elastoplastický materiálový model Hardening Soil (HS). V tomto modelu jsou na rozdíl od Mohr-Coulombova (MC) celkové přetvoření vypočtena pomocí tuhosti závislé na napětí a tuhost je v HS modelu rozdílná pro prvotní zatěžování a pro odtížení/opětovné přitížení. To umožňuje přesnější popis reálného chování zemin. Další předností HS modelu je, že díky definování druhé, objemové, plochy plasticity lépe popisuje chování zeminy při izotropním zatěžování [23].

3.1.1 Vstupní parametry matematického modelu

Celkový počet vstupních parametrů matematického modelu může v být řádu desítek. Obecně se jedná např. o parametry materiálových modelů (velký počet parametrů je značnou nevýhodou pokročilých konstitučních modelů jako právě HS model), velikost předpětí v kotvách, zatížení konstrukce od vnějšího zatížení, ale i parametry geometrické.

Tento velký počet vstupních parametrů, jenž ovlivňují výsledky výpočtu, zatěžuje uživatele, a to i v případech, kdy dané vstupní parametry nejsou předmětem zájmu. Výhodou metamodelování je, že za vstupní parametry metamodelu je možné si zvolit jen část vstupních parametrů matematického modelu (i pouze 1). Zbylé vstupní parametry matematického modelu jsou pro metamodel považovány za konstanty a vliv těchto konstant je během učení zahrnut ve vnitřní struktuře neuronové sítě (vahách neuronových spojení). Nebo naopak, metamodel vytvořit s širokou škálou vstupních parametrů (materiálové, geometrické, …) a využít tak metamodel jako plnou náhradu původního matematického modelu.

Pro metamodel vytvořený v této bakalářské práci byly jako vstupní parametry zvoleny tyto čtyři parametry materiálového modelu HS: soudržnost zeminy c, úhel vnitřního tření φ, sečnový modul tuhosti v primárním zatěžování E_{50}^{ref} a poměr tuhosti v odtížení a přitížení ku tuhosti v primárním zatěžování $E_{50}^{ref} vs E_{ur}^{ref}$ (nepřímé určení materiálového parametru E_{ur}^{ref}).

Tyto vstupní parametry byly zvoleny na základě jejich největšího vlivu na výsledné deformace konstrukce. Tato citlivostní analýza byla převzata z předchozí interní studie.

Pro ostatní základní parametry MPK modelu bylo použito nastavení uvedené v tabulce 3.1.

Tabulka 3.1: Základní parametry MKP modelu.

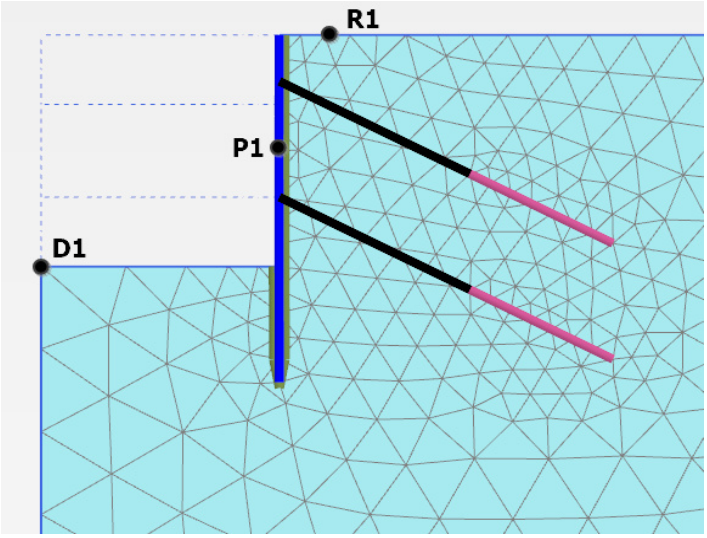
$E_{oed} =$	20000	kN/m^2
$\psi =$	0	°
$\gamma =$	16	kN/m^3
$\gamma_{sat} =$	20	kN/m^3
η =	0,2	
Rinter =	0,65	
m =	0,65	
Drainage type =	' *Drained* '	

3.1.2 Výstupní parametry matematického modelu

Obdobně jako se redukuje počet vstupních parametrů při tvorbě metamodelu, i u výstupních parametrů se volí jen několik zájmových veličin, jenž jsou výstupem matematického modelu, které bude metamodel aproximovat. Výstupními parametry může být např. posunutí konkrétního bodu sítě konečných prvků, výsledná vnitřní síla v konstrukci, hlavní napětí v zemině, pórové tlaky v zemině, stupeň stability, …

Pro vytvoření matamodelu v tomto konkrétním případě byly jako výstupní parametry neuronové sítě vybrány tři deformace pažící konstrukce (posunutí bodů znázorněných na obrázku číslo 3.1): $\mathrm{D1}(\mathrm{u}_x), \mathrm{P1}(\mathrm{u}_y)$ a $\mathrm{R1}(\mathrm{u}_x)$.

Z výše uvedeného vyplývá, že účelem není aproximovat celý matematický model, ale pouze výpočet zvolených deformaci pro různé hodnoty c, φ, E_{50}^{ref} a $E_{50}^{ref}\, vs E_{ur}^{ref}$.



Obrázek 3.1: Matematický model konstrukce v programu Plaxis 2D.

3.2 Data pro trénování a testování metamodelu

Data pro učení a testování matamodelu uvnitř prostoru návrhu byla vybrána na základě metody LHS (Latin hypercube sampling), popsané v podkapitole 3.2.1.

Na základě této metody byly určeny kombinace čtyř vstupních parametrů. S touto sadou vstupních parametrů byly pomocí softwaru Plaxis 2D a naprogramovaného matematického modelu pažící konstrukce provedeny simulace pro každou kombinaci vstupních parametrů. Po každé simulaci byly získány výsledky zvolených výstupních parametrů (3 deformace modelu konstrukce) a uloženy do jednoho společného datasetu obsahující vstupní a výstupní parametry.

Počet vzorků v trénovací sadě byl určen na základě následujícího vztahu převzatého z [12]:

$$n = 3l(s+1)(s+2) \tag{3.1}$$

kde l je škálovací parametr v intervalu $\langle 0{,}5; 2\rangle$ a s počet vstupních parametrů. Dosazením $s = 4$ dostaneme pro okrajové hodnoty l tyto hodnoty:

$$n = 3 \cdot 0{,}5 \cdot (4+1) \cdot (4+2) = 45$$
$$n = 3 \cdot 2 \cdot (4+1) \cdot (4+2) = 180$$

Na základě těchto hodnot byla stanovena velikost základní trénovací sady pro metamodel na 100 vzorků. Pro testování vlivu počtu vzorků na přesnost metamodelu byly dále použity trénovací sady o 50 a 200 vzorcích. Pro každou trénovací sadu byla vytvořena i sada testovací s 20% počtem vzorků oproti trénovací sadě.

Základní trénovací sada se tedy skládá ze 100 vzorků a základní testovací sada z 20. Každý tento vzorek je tvořen 4 hodnotami vstupních parametrů a jim odpovídajícími 3 hodnotami výstupních parametrů.

Vzorky trénovacích dat byly generovány pro rozmezí hodnot jednotlivých vstupních parametrů uvedených v tabulce číslo 3.2.

Tabulka 3.2: Rozmezí hodnot vstupních parametrů metamodelu

c	0-10	kN/m^2
φ	20-30	°
E_{50}^{ref}	8 000-20 000	kN/m^2
$E_{50}^{ref} vs E_{ur}^{ref}$	3,0-5,0	

Ze stejných intervalů jsou i vstupní parametry testovacích vzorků. Pro vstupní parametry mimo meze, na kterých byl natrénován, metamodel testován nebyl.

3.2.1 Latin Hypercube Sampling

Tato metoda spočívá v rozdělení vertikální osy grafu distribuční funkce vstupní proměnné (v tomto případě přímka $y = x$) do předem určeného počtu nepřekrývajících se intervalů stejné délky. Tyto intervaly jsou pak promítnuty na vertikální osu, přičemž pravděpodobnost výskytu náhodné proměnné je ve všech intervalech stejná. Z každého takto vzniklého intervalu je náhodně vybrána přesně jedna hodnota.

Stejný postup je opakován pro ostatní vstupní parametry a následně jsou vytvořeny náhodné kombinace zvolených reprezentativních vzorků jednotlivých vstupních parametrů.

3.3 Tvorba metamodelu geotechnické úlohy

Metamodel byl vytvořen v programovacím jazyce Python za pomocí knihoven Keras a TenserFlow. Keras je open-source knihovna, která poskytuje uživatelsky snadno pochopitelné rozhraní pro práci s neuronovými sítěmi. Backend knihovny Keras je pak tvořen knihovnou TensorFlow.

V této podkapitole jsou popsány stěžejní části kódu, kompletní kód je pak součásti bakalářské práce jako příloha.

1) Načtení a rozdělení dat

```
dataset,snapshot =
read_dataset_snapshot_data("output_dataset_training.txt")
dataset_test,snapshot_test =
read_dataset_snapshot_data("output_dataset _test.txt")
```

Nejprve jsou načteny trénovací a testovací datasety z textových souborů, kdy každý obsahuje určitý počet uspořádaných dvojic vstupů a jim odpovídajících výstupů z MKP výpočtu.

Předem definovaná funkce tyto data na základě očekávaného počtu vstupů a výstupů zároveň rozdělí do dvou listů. Jeden bude tvořen pouze vstupními daty a na odpovídajícím místě v listu druhém pak budou data výstupní.

2) Normování dat

```
scaler_X = MinMaxScaler()
scaler_y = MinMaxScaler()

X_train = scaler_X.fit_transform(dataset)
X_test = scaler_X.transform(dataset_test)

y_train = scaler_y.fit_transform(snapshot)
y_test = scaler_y.transform(snapshot_test)

joblib.dump(scaler_X, 'scaler_x1.pkl')
joblib.dump(scaler_y, 'scaler_y1.pkl')
```

Před počátkem tréninku neuronové sítě na datech je potřeba data (především vstupy) znormovat do rozsahu 0 až 1. Vstupní proměnné mají neznormované velmi rozdílné rozsahy, což by mohlo způsobit, že proměnné s vyššími číselnými hodnotami budou dominovat výpočtu a neuronová síť nebude dostatečně zohledňovat změny ostatních proměnných.

Nejprve jsou vytvořeny dva scalery (zvlášť pro vstupní a výstupní hodnoty) pomocí funkce MinMaxScaler. Následně jsou upraveny podle hodnot trénovacích dat a aplikovány jak na data trénovací, tak data testovací.

Uložení těchto scalerů je důležité pro budoucí použití neuronové sítě, protože při jakémkoli výpočtu je potřeba nejprve vstupní parametry transformovat pomocí scaler_X a předpovědi neuronové sítě inverzně transformovat pomocí scaler_y do požadovaných výsledků.

3) Návrh architektury neuronové sítě

```
model = keras.Sequential([
    layers.Input(4),
    layers.Dense(20, activation="tanh"),
    layers.Dense(10, activation="tanh"),
    layers.Dense(3)
])
```

Model neuronové sítě je vytvořen jako sekvenční, což umožňuje snadno skládat jednotlivé vrstvy sítě v pořadí od vstupu do výstupu.

První vrstva neuronové sítě, jak už bylo popsáno v teoretické části, má vždy počet neuronů odpovídající počtu vstupních parametrů. V tomto případě tedy 4. Následně jsou přidány dvě vrstvy skryté, o 20 a 10 neuronech, obě s tangenciální aktivační funkcí. Poslední, výstupní vrstva, je tvořena 3 neurony, což odpovídá počtu výstupních parametrů.

4) Kompilace modelu

```
optimizer = keras.optimizers.RMSprop(learning_rate=0.01)
model.compile(optimizer, loss=nrmse_khaledi_tf)
```

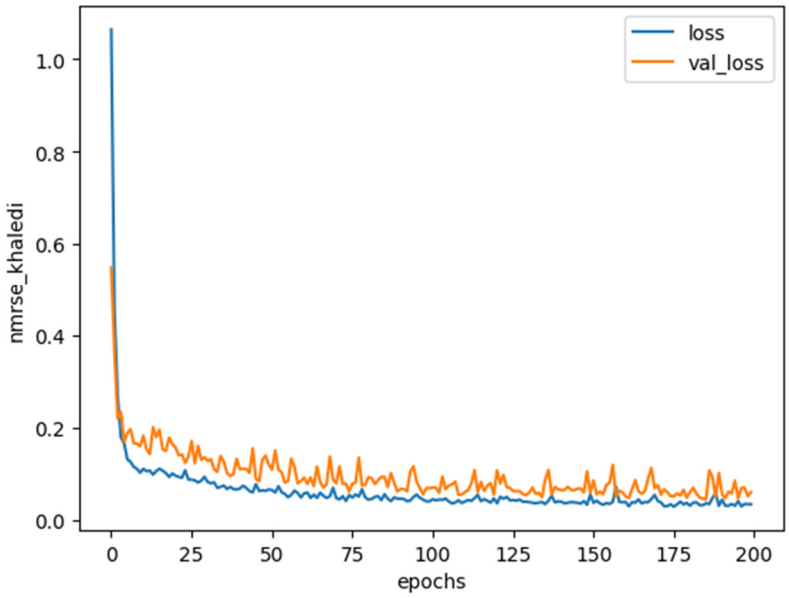
Kompilace je proces sestavení modelu do uceleného celku. K tomu je kromě architektury neuronové sítě zapotřebí definování učícího algoritmu (optimizer), rychlost učení (learning rate) a sledované chybové funkce (loss).

5) Definování zpětného vyvolání nejlepší vah neuronových spojení

```
callbacks_list = [
      keras.callbacks.EarlyStopping(
            monitor="val_loss",
            mode='min',
            restore_best_weights=True,
      )]
```

Neuronová síť automaticky neukládá váhy neuronových spojení během jednotlivých učících epoch, pro vytrénovanou neuronovou síť jsou vždy použité váhy získané v poslední, předem stanovené, učící epoše. To lze změnit definováním tohoto „volání zpátky“, jehož primárním účelem je zastavení v případě malých změn chyby modelu, čehož zde však nebylo využito.

Oscilace hodnoty chybové funkce (loss) a validační chybové funkce (val_loss) v průběhu učícího procesu je zobrazena na obrázku číslo 3.2.



Obrázek 3.2: Chybová funkce v průběhu učícího procesu.

6) Trénovaní modelu

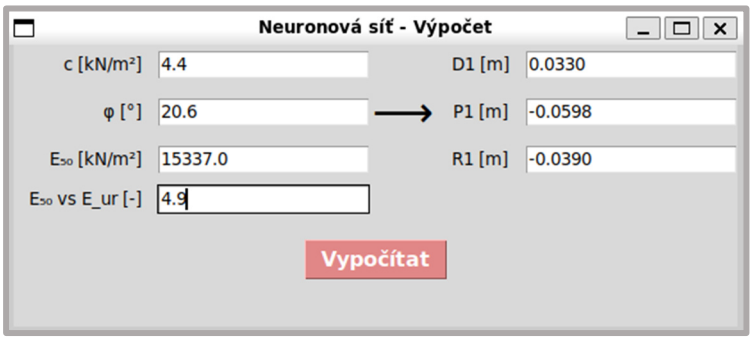
```
model.fit(x=X_train, y=y_train, validation_data=(X_test, y_test),
epochs=500, verbose=0, callbacks= callbacks_list)
```

Model je natrénován na vstupních datech původního modelu X_train a jim odpovídajících výstupech y_train. Validace probíhá na datech X_test a y_test. Učení je zopakováno ve 100 učících epochách, kdy se model datům postupně přizpůsubuje. Nastavení proměnné verbose = 0 způsobí pouze vypnutí zobrazovaní průběhu učení neuronové sítě.

7) Uložení modelu

```
model.save('MM5.keras')
```

Takto uložený model lze znova jednoduše vyvolat pomocí funkce load_model. Tato uložená neuronová síť pak lze pro další použití propojit s připraveným dialogovým oknem, které umožní snadné použití pro výpočet. Vytvořené dialogové okno můžeme vidět na obrázku číslo 3.3.



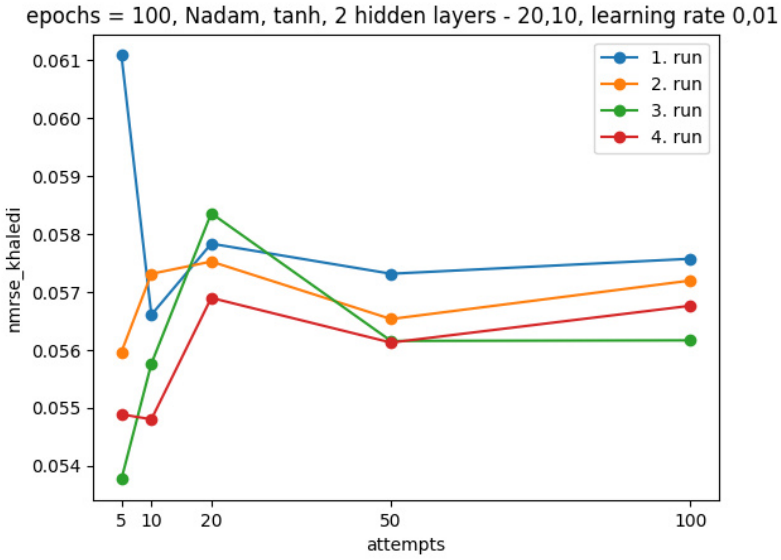
Obrázek 3.3: Dialogové okno pro výpočet vytrénovanou neuronovou sítí.

3.4 Metodika testování metamodelu

V rámci bakalářské práce byly testovány různá nastavení neuronových sítí vytvořených pomocí knihoven Keras a TenserFlow. Cílem bylo nalézt takovou konfiguraci neuronové sítě, která bude co nejpřesněji předpovídat deformace matematického modelu pažící konstrukce popsaného v předchozích kapitolách.

Vytvoření neuronové sítě v programovacím jazyce Python umožnilo částečnou automatizaci testovaní různých parametrů neuronové sítě a zároveň efektivní zpracování velkého objemu dat.

Důležité je zmínit, že přesnost vytrénované neuronové sítě do značné míry závisí na počátečních náhodně zvolených vahách neuronových spojení. Z tohoto důvodu bylo trénování a následné vyhodnocení několikrát opakováno pro každé nastavení neuronové sítě. Pro znázornění problému byl do této části zařazen obrázek 3.4, na kterém je patrný rozptyl průměrů výsledných přesností při různém počtu opakování (attempts) pro 4 spuštění.



Obrázek 3.4: Závislost relevantnosti výsledků na počtu pokusů o vytrénování neuronové sítě.

Pro sledovávání přesnosti byla použita NMRSE podle Khalediho a kol. převzatá z [12]. V bakalářské práci byla použita i další metrika pro testování přesnosti metamodelu označovaná jako v textu jako NMRSE 2.

3.4.1 NMRSE dle Khalediho a kol.

Pro vyhodnocení přesnosti metamodelu, a zároveň pro potřeby jeho trénování, byla primárně použita normovaná střední směrodatná odchylka NMRSE podle Khalediho a kol.:

$$NMRSE = \left\{ \left(\sum_{i=1}^{n_p} \sum_{j=1}^{m} \left(u_j^i - \tilde{u}_j^i\right)^2 \right) \Big/ \left(\sum_{i=1}^{n_p} \sum_{j=1}^{m} \left(u_j^i\right)^2 \right) \right\}^{\frac{1}{2}}, \tag{3.2}$$

kde u_j^i je přesná funkční hodnota pro vzorkovací bod x_i v pozorovacím bodě j, $\tilde{u}_j^i$ označuje předpověď funkční hodnoty v odpovídajícím bodě pomocí metamodelu a n_p a m jsou počty vzorků a pozorovacích bodů. Čím je NRMSE menší, tím přesnější je předpověď metamodelu [12].

3.4.2 NMRSE 2

Tato normovaná střední směrodatná odchylka je definována rovnicí

$$NMRSE\ 2 = \left\{ \frac{1}{n_p} \sum_{i=1}^{n_p} \left(\frac{1}{m} \sum_{j=1}^{m} \frac{\left(u_j^i - \tilde{u}_j^i\right)^2}{{u_j^i}^2} \right) \right\}^{\frac{1}{2}}, \tag{3.3}$$

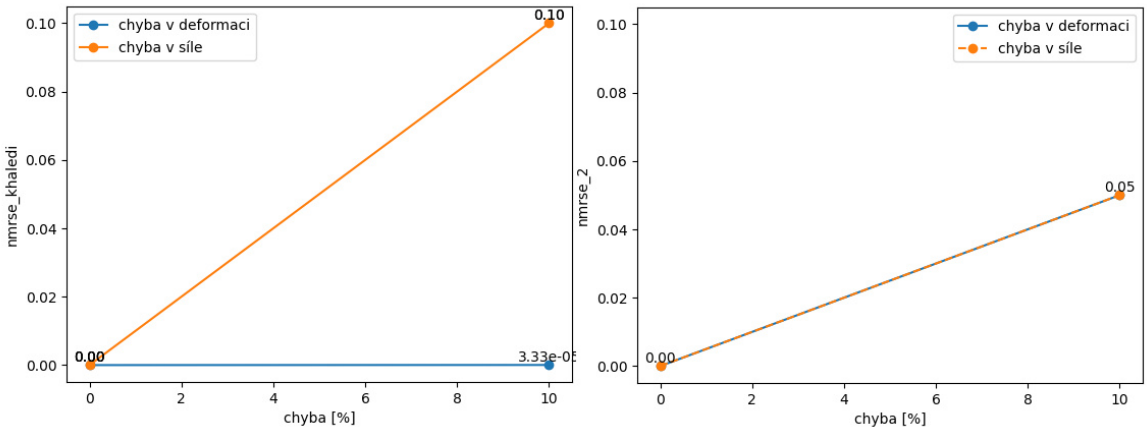
kde u_j^i je přesná funkční hodnota pro vzorkovací bod x_i v pozorovacím bodě j, $\tilde{u}_j^i$ označuje předpověď funkční hodnoty v odpovídajícím bodě pomocí metamodelu a n_p a m jsou počty vzorků a pozorovacích bodů.

Sledování této chyby bylo zařazeno na základě poznatku, že NMRSE dle Khalediho a kol. zohledňuje výrazně více nepřesnost většího sledovaného parametru než parametru menšího. To nezpůsobuje problém při testování přesnosti nastavení neuronové sítě, kdy jsou všechny výstupy znormovány do rozsahu 0 až 1.

Zkreslení výsledků by mohlo nastat v části práce, kde byl testován vliv normování výstupů na konečnou předpověď neuronové sítě. To hlavně pro případ, kdy výstupem neuronové sítě nejsou pouze deformace, ale i parametr jiných rozměrů (síla v táhle kotvy v kN).

Porovnání NMRSE dle Khalediho a kol. a NMRSE 2 je graficky znázorněno na obrázku 3.5.

Rozdílná výsledná NMRSE dle Khalediho a kol. pro procentuálně stejnou chybu různých parametrů je pozorovatelná na levé části obrázku 3.5. V pravé části jsou pak výsledky NMRSE 2 pro stejný případ. Zde je zřejmé, že oba parametry mají na výslednou chybu stejný vliv.



Obrázek 3.5: Porovnání NMRSE dle Khalediho a kol. a NMRSE 2.

Obě metriky pro určení přesnosti metamodelu v porovnání se správným řešením z matematického modelu mohou nabývat hodnot v intervalu $\langle 0; \infty)$. Chyba 0,01 značí, že všechny testované výstupy se odlišují od správného řešení o 1 %, chyba 0,1 o 10 % atd.

3.5 Testovaní neuronových sítí

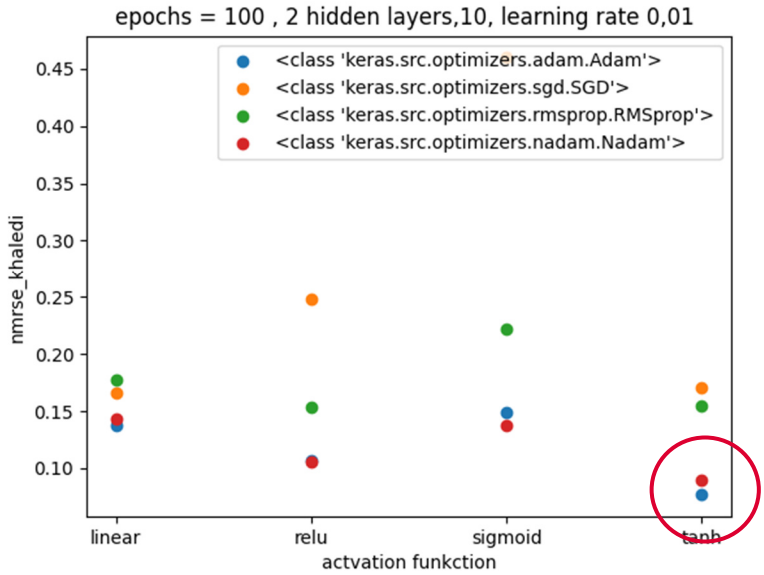
Hlavními proměnnými v testovaných neuronových sítích byly:

- učící algoritmy: SGD, RMSprop, Adam a Nadam (popsány v podkapitole 2.5.5)
- aktivační funkce: Lineární, ReLu, Sigmoid a Tanh (popsány v podkapitole 2.5.3)
- architektura neuronové sítě: počet skrytých neuronových vrstev a počet neuronů v jednotlivých skrytých vrstvách
- počet učících epoch: počet úplných průchodů trénovacích dat neuronovou sítí
- learning rate: rychlost učení, tj. velikost kroků, kterými se neuronová síť přizpůsobuje vstupním datům

3.5.1 Prvotní testování

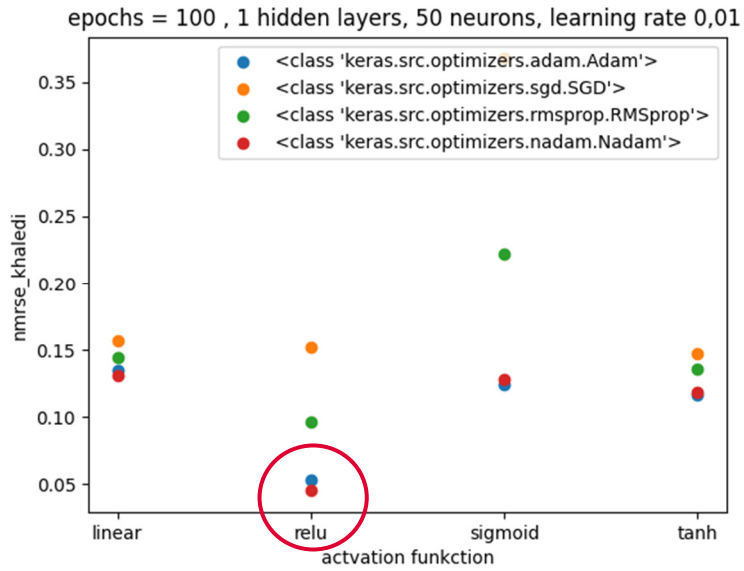
Pro základní představu byly protestovány všechny kombinace zmíněných učících algoritmů a aktivačních funkcí, a to pro neuronovou síť o 1, 2 a 5 skrytých vrstvách o 5, 10, 20, 50 a 100 neuronech. Zároveň byl testován vliv počtu učících epoch. V této podkapitole jsou shrnuty nejzajímavější poznatky.

1. Nejlepší výsledky byly dosaženy pro učící algoritmy Adam a Nadam v kombinaci s aktivační funkcí Tanh pro dvě skryté vrstvy o 10 neuronech (obrázek 3.6).



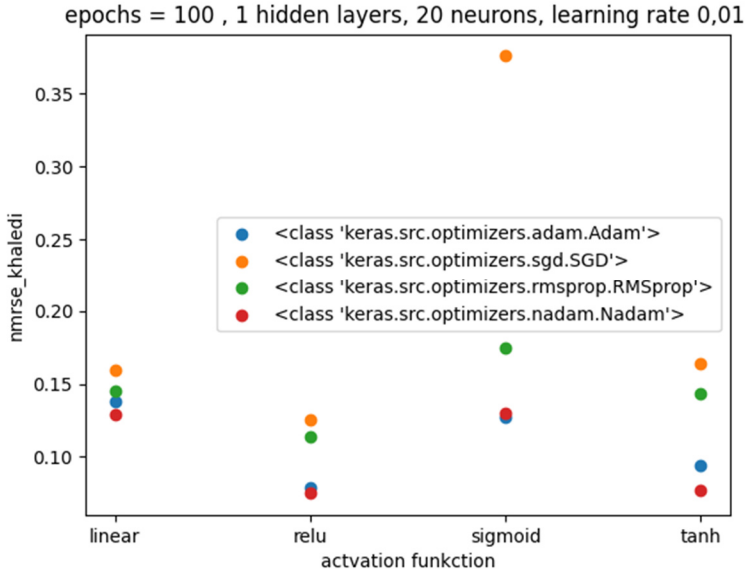
Obrázek 3.6: Porovnání výsledků neuronových sítí s různými učícími algoritmy a aktivačními funkcemi (dvě skryté vrstvy o 10 neuronech).

Obdobné výsledky byly dosaženy i při použití učících algoritmů Adam a Nadam v kombinaci s aktivační funkcí ReLu pro jednu skrytou vrstvu o 50–100 neuronech (obrázek 3.7).



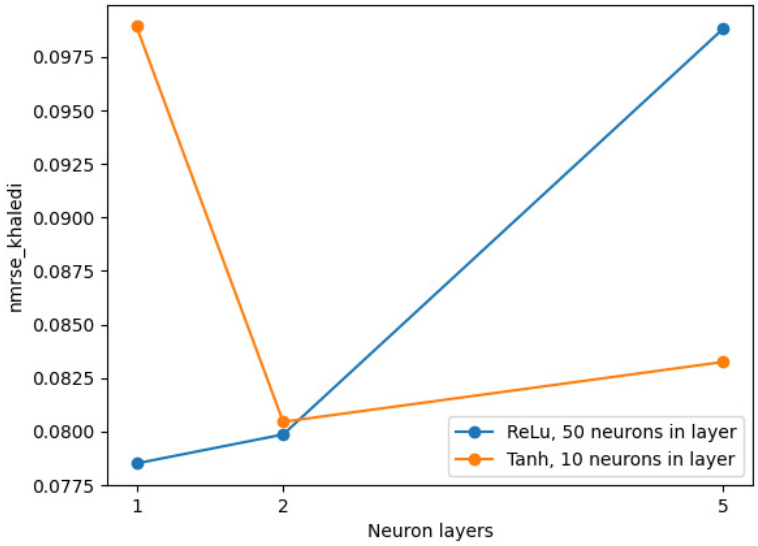
Obrázek 3.7: Porovnání výsledků neuronových sítí s různými učícími algoritmy a aktivačními funkcemi (1 skrytá vrstvy o 50 neuronech).

2. Nejhorší výsledky vykazuje aktivační funkce Sigmoid, a to pro všechny 4 učící algoritmy. Toto zjištění je v souladu s teoretickou části práce, kde je v podkapitole 2.5.3 popsáno její využití primárně v klasifikačních úlohách. U klasifikačních úloh je na rozdíl od regresních účelem dosáhnout hodnot 0 nebo 1, a nikoliv hodnot mezilehlých. Relativně velké hodnoty chyby při použití aktivační funkce Sigmoid jsou patrné na všech obrázcích 3.6, 3.7 a 3.8.



Obrázek 3.8: Porovnání výsledků neuronových sítí s různými učícími algoritmy a aktivačními funkcemi (1 skrytá vrstvy o 20 neuronech).

3. Maximální počet skrytých neuronových vrstev pro efektivní neuronovou síť jsou pro tento případ 2. Pro 100 učících epoch je chyba pro 5 skrytých vrstev větší, což je patrné na obrázku 3.9. Tento rozdíl se při zvětšování počtu učících epoch zmenšuje, ale neúměrně k časové náročnosti trénování takového modelu.



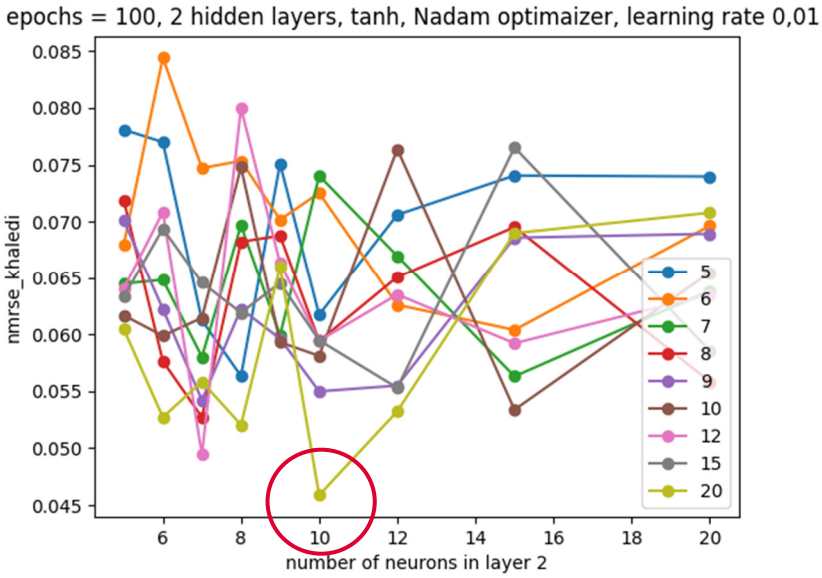
Obrázek 3.9: Vliv počtu skrytých vrstev na velikost chyby neuronové sítě.

3.5.2 Drobné změny architektury

Pro dvě nejpřesnější nastavení z podkapitoly 3.5.1, tedy učící algoritmy Adam a Nadam v kombinaci s aktivační funkcí Tanh (pro 2 skryté vrstvy o 10 neuronech) a ReLu (pro 1 skrytou vrstvu o 50-100 neuronech) byly vyzkoušeny architektury o podobných počtech neuronů.

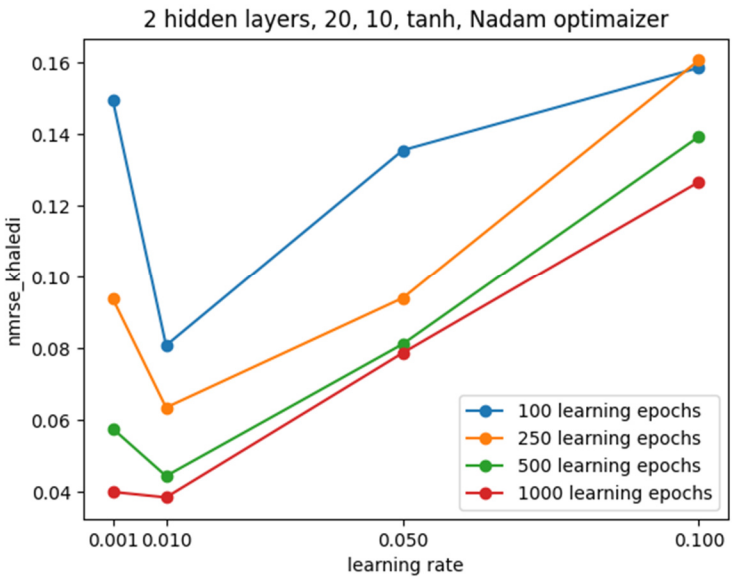
Tyto drobné změny architektury už se pohybují v mnohem menší oblasti, tudíž byl zvětšen počet opakování na 20 pokusů, přičemž vyhodnocován byl průměr dosažených hodnot směrodatné odchylky.

Na základě výsledků analýz uvedených na obrázku 3.10 byla pro další testování vybrána architektura neuronové sítě s 20 neurony v první skryté vrstvě a 10 neurony v druhé skryté vrstvě s nastavenou aktivační funkcí Tanh.



Obrázek 3.10: Průběh směrodatné odchylky (chyby neuronové sítě) pro 5-20 neuronů v první skryté vrstvě.

3.5.3 Rychlost učení a počet učících epoch



Obrázek 3.11: Chyba neuronové sítě při použití různě velkých učících kroků a různém počtu učících epoch.

Na obrázku 3.11 je patrné, že nejlepší velikost kroku učení je 0,01. Při nastavení většího kroku nedokážou učící algoritmy tak přesně určit minimum chybové funkce. Změna vah pro neuronovou sít je poté tak velká, že algoritmus globální minimum nenalezne. Pro menší krok je zas zapotřebí velkého počtu učících epoch.

3.5.4 Normování a nenormování výstupů

Pro zkoušení vlivu normování a nenormování výstupů neuronové vrstvy byla použita nová sada testovacích a validačních dat doplněná o čtvrtý výstup – sílu v táhle kotvy. Důvodem byl záměr otestovat vliv normování výstupů nejen pro dosud používanou neuronovou síť předpovídající hodnoty tří deformací, ale i pro neuronovou síť jejíž výstupy nemají stejný rozměr.

Výsledné hodnoty NRMSE podle Khalediho jsou zaznamenány v tabulce 3.3. Tato veličina však není v tomto případě spolehlivým ukazatelem přesností metamodelu, jak bylo popsáno v podkapitole 3.4.2, a pro vyhodnocení tedy byla použita primárně NMRSE 2 a tabulka 3.4.

V prvním sloupci obou tabulek jsou hodnoty pro neuronovou síť předpovídající pouze deformace, v druhém pak hodnoty pro neuronovou síť s přidaným čtvrtým výstupem.

Tabulka 3.3: Porovnání normování a nenormování výstupů - NMRSE dle Khalediho a kol.

	Výstupem deformace	Výstupem deformace a síla
normované výstupy	0,023213	0,002883
nenormované výstupy	0,273207	0,724249

Tabulka 3.4: Porovnání normování a nenormování výstupů - NMRSE 2.

	Výstupem deformace	Výstupem deformace a síla
normované výstupy	0,093021	0,089989
nenormované výstupy	1,412096	1,906635

Jednoznačným závěrem (vyhodnoceno podle NMRSE 2) je, že nenormování výstupů v obou případech zapříčiňuje zmenšení přesnosti modelu. V případě výstupů různých rozměrů je přínos normovaní ještě větší.

Z tabulky 3.4 je dále patrné, že přidáním výstupu jiného rozměru se při normování výstupů celková přesnost neuronové sítě nezhorší. Pro lepší porovnání změny přesnosti při přidání síly jako výstupu byly pro 2. neuronovou síť zvlášť vypočteny chyby pro předpověď deformací a zvlášť pro předpověď síly v táhle kotvy.

Tabulka 3.5: Chyba při předpovědi deformací konstrukce a síly v táhle kotvy jednou neuronovou sítí.

	Chyba pro předpověď deformací	Chyba pro předpověď síly
nmrse dle Khalediho a kol.	0,022387	0,002192
nmrse 2	0,106572	0,0022396

Hodnoty v tabulce 3.5 byly porovnány s tabulkou 3.6, kde jsou zaznamenány chyby pro předpověď deformací a síly pomocí dvou nezávislých neuronových sítí – jedna se třemi výstupy (3 deformace) a druhá s pouze jedním výstupním parametrem (síla v táhle kotvy).

Tabulka 3.6: Chyba při předpovědi deformací konstrukce a síly v táhle kotvy dvěma samostatnými neuronovými sítěmi.

	Chyba pro předpověď deformací	Chyba pro předpověď síly
nmrse dle Khalediho a kol.	0,023213	0,002089
nmrse 2	0,093021	0,002044

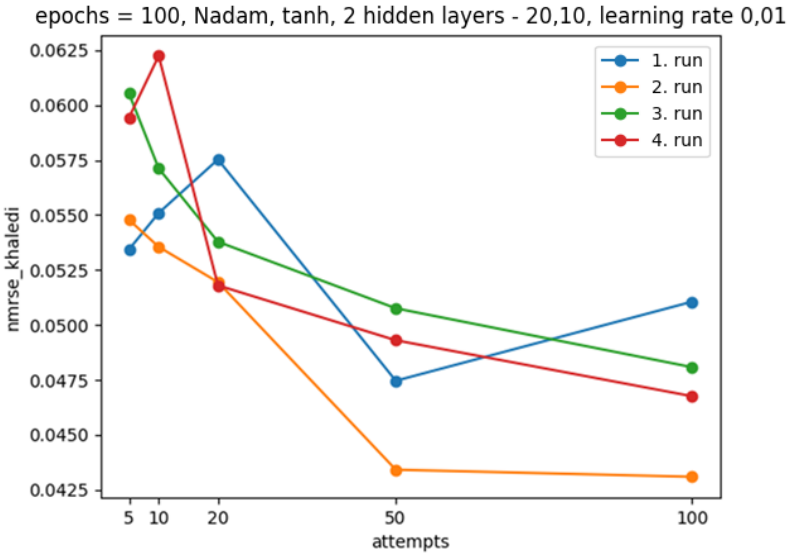
Je zřejmé, že rozdělením výpočtu do dvou neuronových sítí se výpočet nijak nezpřesnil.

Pokud by ale k zpřesnění rozdělením do více neuronových sítí došlo, např. pro větší počet rozdílných výstupních parametrů, nedošlo by k výraznému zkomplikování výpočtu pomocí neuronových sítí. Výpočty různých neuronových sítí se dají pro uživatele jednoduše schovat do jednoho výpočtu na pozadí a výsledky najednou zobrazovat v uživatelském rozhraní.

3.5.5 Nutný počet pokusů pro vytrénování nejpřesnější neuronové sítě

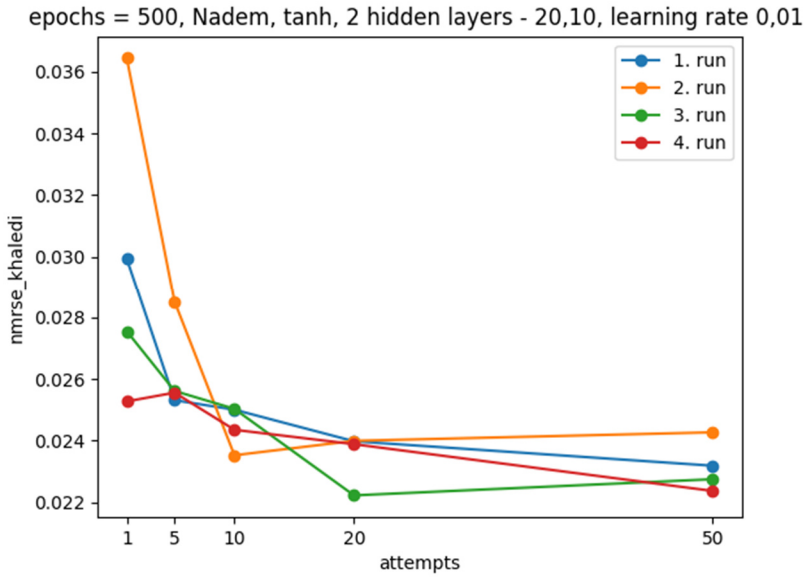
V podkapitole 3.4 již byla zmíněno, že neuronová síť je ovlivněna náhodně zvolenými váhami neuronových spojení, a proto je při vyhodnocování přesnosti potřeba trénování několikrát zopakovat. Obdobně je potřeba při trénování neuronové sítě určené pro další využití potřeba zopakovat trénovaní za průběžného ukládání aktuálně nejpřesnějšího modelu.

Z grafu na obrázku číslo 3.12 je patrné, že při trénování neuronové sítě během 100 učících epoch, smysl má přibližně 50 pokusů o vytrénování co nejpřesnější neuronové sítě.



Obrázek 3.12: Nutný počet pokusů pro vytrénování nejpřesnější neuronové sítě při nastavení 100 učících epoch.

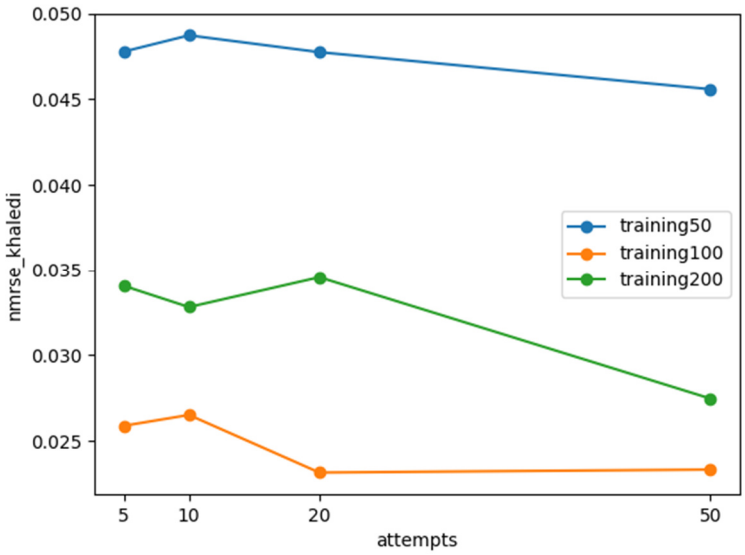
Výhodnější je ale zvolit větší počet učících epoch, protože jak je patrné z obrázku 3.13, při nastavení 500 učících epoch stačí 10 pokusů o vytrénováni neuronové sítě. Při podobné časové náročnosti ($100 \cdot 50 = 500 \cdot 10$) je však dosaženo menší směrodatné odchylky.



Obrázek 3.13: Nutný počet pokusů pro vytrénování nejpřesnější neuronové sítě při nastavení 500 učících epoch.

3.5.6 Vliv počtu testovacích dat

Pro testování vlivu počtu vzorků byly vytvořeno tři nové sady trénovacích dat a to o 50, 100 a 200 vzorcích a pro každou byla zároveň vytvořena sada testovacích dat. Pro každou sadu bylo provedeno stejné testování přesnosti jak v předchozí podkapitole 3.5.5. Na obrázku číslo 3.14 můžeme vidět zprůměrované výsledky 10 testování pro každou z datových sad.



Obrázek 3.14: Vliv počtu vzorků.

Očekávaným výsledkem by bylo, kdyby se přesnost modelu s větším počtem trénovacích dat zpřesňovala, k čemuž ale došlo jen pro sadu 100 oproti sadě 50. Příčinou tohoto výsledku je s největší pravděpodobností velká komplexnost problematiky výběru vzorků pro trénování a vyhodnocování aproximačních metod.

V rámci problematiky výběru vzorků stojí za povšimnutí i porovnání grafu na obrázku 3.12 a oranžové křivky grafu na obrázku číslo 3.14 kde jsou znázorněny výsledky neuronových sítí vytvořených na základě totožného kódu a natrénovaných na dvou různých sadách o 100 vzorcích.

4 ZÁVĚR

Výsledkem práce je vytrénovaná neuronová síť, jenž s průměrnou chybou 2,5 % předpovídá výsledky matematického modelu pro hodnoty vstupů ze stejného oboru hodnot, pro jaké byla vytrénována. Pro názornější ukázku je v příloze bakalářské práce porovnání číselných výsledků pro náhodně vygenerované vstupní parametry. Výpočet neuronovou sítí umožňuje získat výsledky v řádu zlomků sekund, a to s minimálními požadavky na výpočetní techniku.

Hlavním přínosem práce však není vytvoření samotného metamodelu pro jednu konkrétní konstrukci, ale určení nastavení neuronové sítě, které vykazuje největší přesnost metamodelu. Toto nastavení je zjednodušeně zapsáno v tabulce číslo 4.1.

Tabulka 4.1: Nastavení výsledné neuronové sítě

Počet skrytých vrstev	2
Počet neuronů ve skrytých vrstvách	20,10
Učící algoritmus	Nadam
Aktivační funkce	Tanh
Počet učících epoch	500
Learning rate	0,01
Normování výstupů	ANO
Počet pokusů pro vytrénování	10

Předpokladem je, že toto nastavení bude podobně účinné pro předpověď chování jakékoli konstrukce. Tento teoretický předpoklad je však pro budoucí využití potřeba ověřit.

Pro případné využití je velmi výhodné, že díky vytvořenému uživatelskému prostředí může aproximovat výpočty deformací matematického modelu i uživatel bez znalosti programování a bez nutnosti vlastnit licenci výpočetního softwaru. Toho by se dalo využít např. v situaci, kdy je matematická analýza pomocí MKP softwaru zpracována jinou stranou a v určité fázi projektových prací nastává potřeba provedení parametrické studie.

Faktorem ovlivňující přesnost je však kromě nastavení neuronové sítě také kvalita učících dat, přičemž důležitý je nejen počet vzorků, ale i reprezentativnost této sady. Této problematiky se práce dotkla jen velmi okrajově a téma si zaslouží další pozornost.

LITERATURA

[1] MUCHA, W. a W. KUŚ. *Metamodeling as a model order reduction technique in hybrid simulation using RTFEM* [online]. In: . 2018, 020045- [cit. 2025-05-07]. Dostupné z: doi:10.1063/1.5066507

[2] SIMPSON, T.W., J.D. POPLINSKI, P. N. KOCH a J.K. ALLEN. Metamodels for Computer-based Engineering Design: Survey and recommendations. *Engineering with Computers* [online]. 2001, **17**(2), 129-150 [cit. 2025-05-07]. ISSN 0177-0667. Dostupné z: doi:10.1007/PL00007198

[3] ZHAO, D. a D. XUE. A multi-surrogate approximation method for metamodeling. *Engineering with Computers* [online]. 2011, **27**(2), 139-153 [cit. 2025-05-26]. ISSN 0177-0667. Dostupné z: doi:10.1007/s00366-009-0173-y

[4] PIERREVAL, H. A metamodeling approach based on neural networks. *International Journal of Computer Simulation*. 1996, **6**(3), 365.

[5] ARNDT, O., T. BARTH, B. FREISLEBEN a M. GRAUER. Approximating a finite element model by neural network prediction for facility optimization in groundwater engineering. *European Journal of Operational Research* [online]. 2005, **166**(3), 769-781 [cit. 2025-05-07]. ISSN 03772217. Dostupné z: doi:10.1016/j.ejor.2003.09.039

[6] BENBOURAS, M. A., R. KETTAB MITICHE, H. ZEDIRA, A. PETRISOR, N. MEZOUAR a F. DEBICHE. A new approach to predict the compression index using artificial intelligence methods. *Marine Georesources & Geotechnology* [online]. 2019, 2019-07-03, **37**(6), 704-720 [cit. 2025-05-26]. ISSN 1064-119X. Dostupné z: doi:10.1080/1064119X.2018.1484533

[7] NEGRIN, I., M. KRIPKA a V. YEPES. Metamodel-assisted design optimization in the field of structural engineering: A literature review. *Structures* [online]. 2023, **52**, 609-631 [cit. 2025-05-07]. ISSN 23520124. Dostupné z: doi:10.1016/j.istruc.2023.04.006

[8] LANCASTER, P. a K. SALKAUSKAS. Surfaces Generated by Moving Least Squares Methods. *Mathematics of Computation* [online]. 1981, **37**(155), 141-158 [cit. 2025-05-27].

[9] NEALEN, A. *An As-Short-As-Possible Introduction to the Least Squares, Weighted Least Squares and Moving Least Squares Methods for Scattered Data Approximation and Interpolation*. Internal Report. TU Darmstadt, 1990.

[10] LEVIN, D. Mesh-Independent Surface Interpolation. In: *Geometric Modeling for Scientific Visualization*. Springer Berlin Heidelberg, 2004, s. 37-49. ISBN 978-3-642-07263-5 978-3-662-07443-5.

[11] BULJAK, V. Proper orthogonal decomposition and radial basis functions algorithm for diagnostic procedure based on inverse analysis. *FME Transactions*. 2010, **38**(3), 129 - 136.

[12] KHALEDI, K., S. MIRO, M. KÖNIG a T. SCHANZ. Robust and reliable metamodels for mechanized tunnel simulations. *Computers and Geotechnics* [online]. 2014, **61**, 1-12 [cit. 2025-05-03]. ISSN 0266352X. Dostupné z: doi:10.1016/j.compgeo.2014.04.005

[13] BULJAK, V. a G. MAIER. Proper Orthogonal Decomposition and Radial Basis Functions in material characterization based on instrumented indentation. *Engineering Structures* [online]. 2011, **33**(2), 492-501 [cit. 2025-05-27]. ISSN ISSN 0141-0296. Dostupné z: https://doi.org/10.1016/j.engstruct.2010.11.006

[14] SHAHZADI, G. a A. SOULAÏMANI. Deep Neural Network and Polynomial Chaos Expansion-Based Surrogate Models for Sensitivity and Uncertainty Propagation: An Application to a Rockfill Dam: An Application to a Rockfill Dam. *Water* [online]. 2021, **13**(13) [cit. 2025-05-26]. ISSN 2073-4441. Dostupné z: doi:10.3390/w13131830

[15] NOVÁK, L., D. LEHKÝ a D. NOVÁK. Sensitivity Analysis of Engineering Structures Utilizing Artificial Neural Networks and Polynomial Chaos Expansion. In: *Lecture Notes in Computer Science*. Springer Nature Switzerland, 2023, s. 181-196. ISBN 978-3-031-25598-4 978-3-031-25599-1.

[16] SUDRET, B. *Polynomial chaos expansions in 90 minutes* [Online]. Madras: ETH Zurich, 2021 [cit. 2025-05-27]. Dostupné z: http://hdl.handle.net/20.500.11850/508852

[17] GOODFELLOW, I., Y. BENGIO a A. COURVILLE. *Deep learning*. Cambridge, Massachusetts: The MIT Press, [2016]. Adaptive computation and machine learning series. ISBN 978-0-262-03561-3.

[18] KRIESEL, D. *A Brief Introduction to Neural Networks* [online]. 2007 [cit. 2025-05-26]. Dostupné z: http://www.dkriesel.com/en/science/neural_networks

[19] Activation function. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2025 [cit. 2025-05-07]. Dostupné z: https://en.wikipedia.org/wiki/Activation_function

[20] CHOLLET, F. *Deep learning with Python*. Manning, 2018. ISBN 978-1-61729-443-3.

[21] Overfitting. *MathWorks* [online]. 2025 [cit. 2025-05-07]. Dostupné z: https://www.mathworks.com/discovery/overfitting.html

[22] Nadam. In: *Keras* [online]. [cit. 2025-05-07]. Dostupné z: https://keras.io/api/optimizers/Nadam/

[23] SCHANZ, T., P.A. VERMEER a P.G. BONNIER. The hardening soil model: Formulation and verification. In: *Beyond 2000 in Computational Geotechnics*. Routledge, 2019, s. 281-296. ISBN 978-1-315-13820-6.

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

KÓD PRO VYTVOŘENÍ NEURONOVÉ SÍTĚ

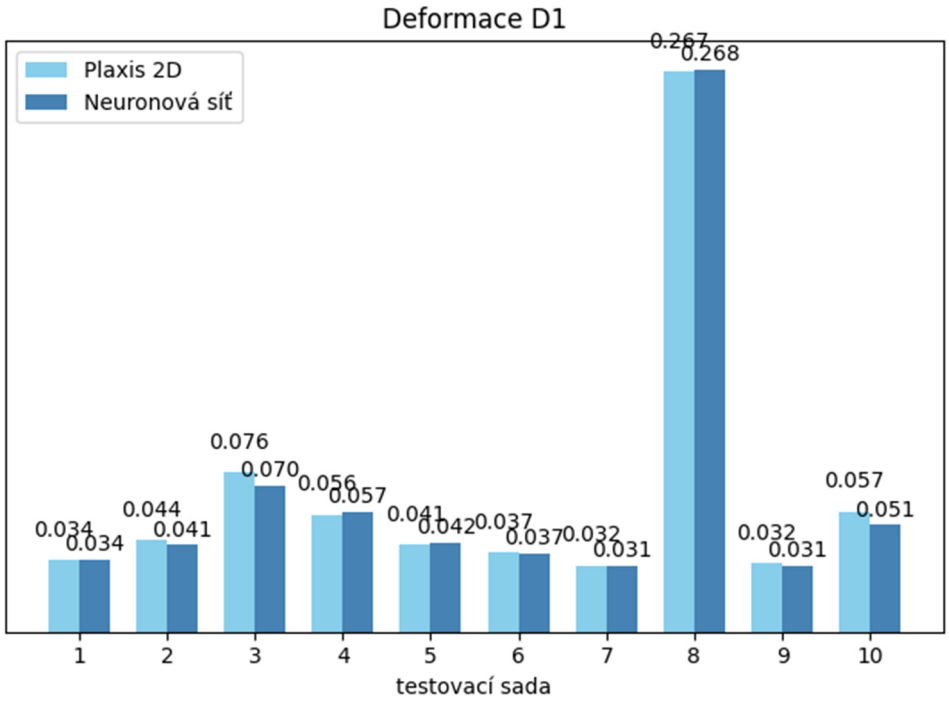
```
import pandas as pd
import tensorflow as tf
import keras
import joblib
from keras import layers
from sklearn.preprocessing import MinMaxScaler
from keras import ops

def read_dataset_snapshot_data(filename):
    df = pd.read_csv(filename, sep="\t")
    readed_data = df.to_numpy()
    input_paramets_raw = readed_data[:,:4]
    snapshot_raw = readed_data[:,4:]

    return input_paramets_raw, snapshot_raw

def nrmse_khaledi_tf(y_test, y_pred):
    Dividend1 = ops.square(y_test - y_pred)
    Dividend2 = ops.sum(Dividend1,axis=1)
    Dividend = ops.sum(Dividend2)
    Divisor1 = ops.square(y_test)
    Divisor2 = ops.sum(Divisor1, axis=1)
    Divisor = ops.sum(Divisor2)

    return ops.sqrt(Dividend/Divisor)

dataset, snapshot =
read_dataset_snapshot_data("output_dataset_training.txt")
dataset_test, snapshot_test =
read_dataset_snapshot_data("output_dataset_test.txt")

scaler_X = MinMaxScaler()
scaler_y = MinMaxScaler()

X_train = scaler_X.fit_transform(dataset)
X_test = scaler_X.transform(dataset_test)

y_train = scaler_y.fit_transform(snapshot)
y_test = scaler_y.transform(snapshot_test)

joblib.dump(scaler_X, 'scaler_x2.pkl')
joblib.dump(scaler_y, 'scaler_y2.pkl')
```

```
nmrse1 = 1
for i in range (1):
    model = keras.Sequential([
        layers.Input(4),
        layers.Dense(20,activation="tanh"),
        layers.Dense(10,activation="tanh"),
        layers.Dense(3),
        ])
    optimizer = keras.optimizers.Nadam(learning_rate=0.01)
    model.compile(optimizer, loss=nrmse_khaledi_tf)

    callbacks_list = [
        keras.callbacks.EarlyStopping(
                monitor="val_loss",
                mode='min',
                restore_best_weights=True,
    )]

    model.fit(x=X_train, y=y_train, validation_data=(X_test, y_test),
    epochs=500, verbose=0, callbacks = callbacks_list)

    predictions = model.predict(X_test)

    current_nmrse1 = nrmse_khaledi_tf(y_test,predictions)
    if current_nmrse1 < nmrse1:
        model.save('MM5.keras')
        nmrse1 = current_nmrse1

print(f"nmrse dle Khalediho a kol. = {nmrse1}")
print("END", "MM5.py")
```

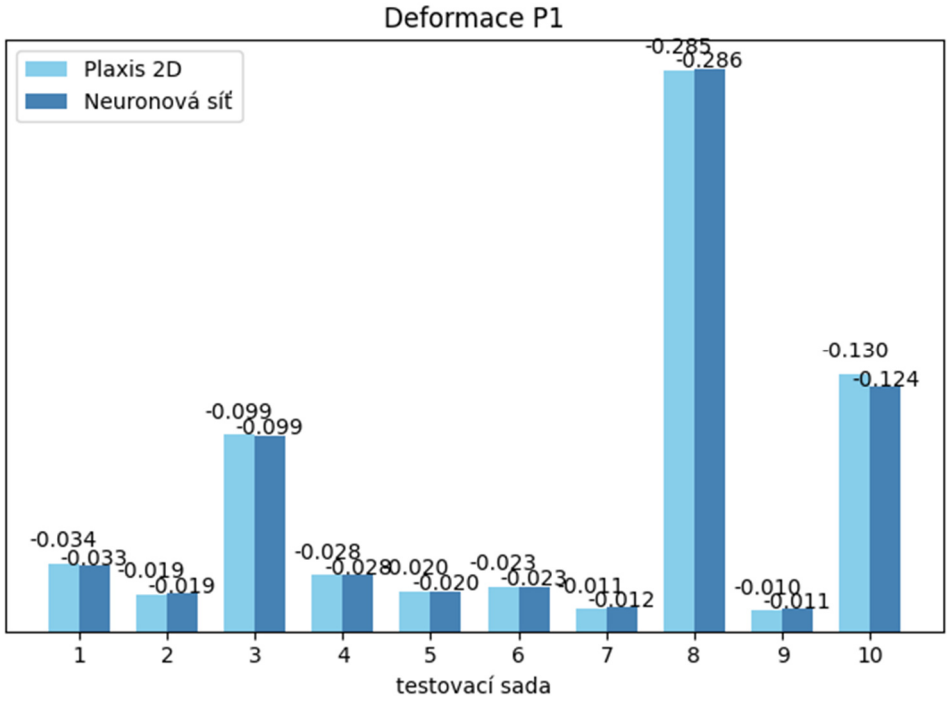
POROVNÁNÍ VÝSLEDKŮ NEURONOVÉ SÍTĚ S VÝSLEDKY MKP

Tabulka 1: Náhodně vygenerovaných hodnot vstupních parametrů.

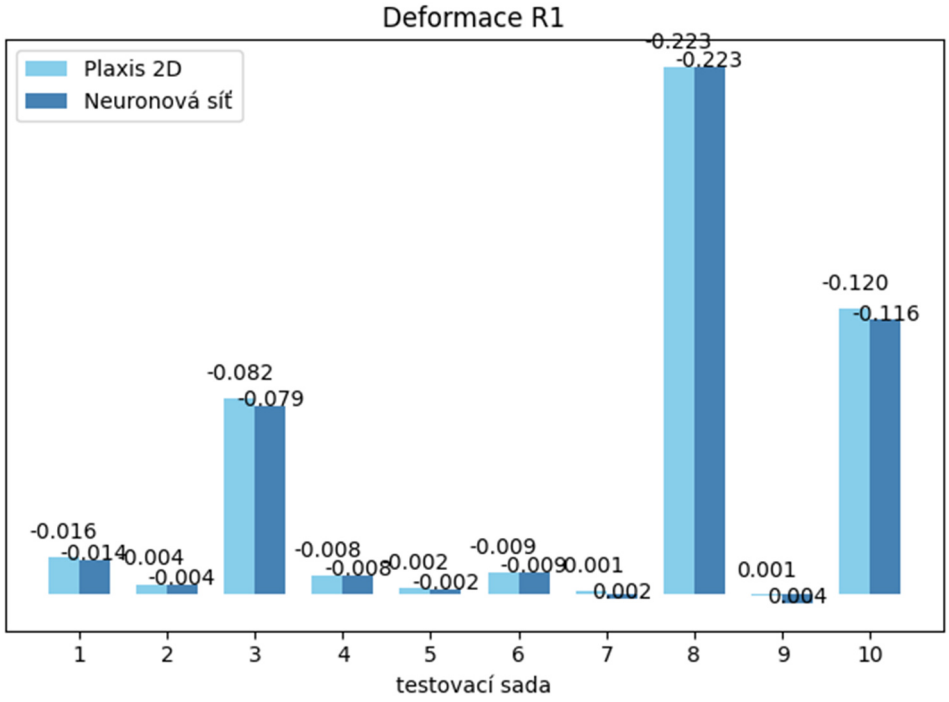
	c	φ	E_{50}^{ref}	$E_{50}^{ref} vs E_{ur}^{ref}$
1. sada	7.1	22.7	14708.3	4.6
2. sada	7.2	28.0	10945.6	5.0
3. sada	0.7	22.6	9282.1	3.4
4. sada	6.0	26.	9778.4	4.3
5. sada	7.6	25.7	17024.6	3.3
6. sada	5.0	26.9	13708.4	4.6
7. sada	7.0	29.7	17485.4	4.4
8. sada	1.4	20.6	2449.7	3.9
9. sada	9.2	29.6	15860.8	4.7
10. sada	0.8	20.5	12146.3	3.7



Obrázek 1: Porovnání výsledků neuronové sítě s původním modelem – deformace $\mathrm{D1}(\mathrm{u}_x)$.



Obrázek 2: Porovnání výsledků neuronové sítě s původním modelem – deformace $\mathrm{P1}(\mathrm{u}_y)$.



Obrázek 3: Porovnání výsledků neuronové sítě s původním modelem – deformace $\mathrm{R1}(\mathrm{u}_x)$.