ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

FAKULTA STROJNÍ

Ústav mechaniky tekutin a termodynamiky

DIPLOMOVÁ PRÁCE

Využití neuronových sítí s fyzikální znalostí při řešení proudění v kavitě

Praha 2024 Bc. Říha Miroslav

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Říha** Jméno: **Miroslav** Osobní číslo: **492460**

Fakulta/ústav: **Fakulta strojní**

Zadávající katedra/ústav: **Ústav mechaniky tekutin a termodynamiky**

Studijní program: **Aplikované vědy ve strojním inženýrství**

Specializace: **Aplikovaná mechanika**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Využití neuronových sítí s fyzikální znalostí při řešení proudění v kavitě

Název diplomové práce anglicky:

Utilization of Physics Informed Neural Networks in Lid-Driven Cavity Flow

Pokyny pro vypracování:

Diskutujte problematiku PINNs (physics-informed neural networks) neuronových sítí, tedy sítí které během procesu učení zohledňují fyzikální podstatu daného problému. Nejprve řešte 1D rovnici vedení tepla, na které ukažte základy tvorby neuronové sítě, možnosti a problémy s tím spojené. Dále řešte proudění v kavitě. Proveďte testování výsledků získaných pomocí PINNs. Sledujte přesnost extrapolace. Podrobně diskutujte získané výsledky.

Seznam doporučené literatury:

dle doporučení vedoucího práce

Jméno a pracoviště vedoucí(ho) diplomové práce:

doc. Ing. Tomáš Hyhlík, Ph.D. ústav mechaniky tekutin a termodynamiky FS

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **15.04.2024** Termín odevzdání diplomové práce: **31.07.2024**

Platnost zadání diplomové práce:

doc. Ing. Tomáš Hyhlík, Ph.D. podpis vedoucí(ho) práce	Ing. Michal Schmirler, Ph.D. podpis vedoucí(ho) ústavu/katedry	doc. Ing. Miroslav Španiel, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

24.4.2024
Datum převzetí zadání

Podpis studenta

Anotační list

Jméno autora: Bc. Miroslav Říha

Fakulta: Fakulta strojní

Zadávající ústav: Ústav mechaniky tekutin a termodynamiky

Studijní program: Aplikované vědy ve strojním inženýrství

Studijní obor: Aplikovaná mechanika

Rok: 2024

Název diplomové práce: Využití neuronových sítí s fyzikální znalostí při řešení proudění v kavitě

Název diplomové práce anglicky: Utilization of physics informed neural networks in lid-driven cavity flow

Vedoucí práce: doc. Ing. Tomáš Hyhlík, Ph.D.

Bibliografické údaje: počet stran 58

počet obrázků 34

počet tabulek 4

Klíčová slova: Neuronová síť, chybová funkce, kavita

Key words: Neural network, cost function, cavity

Anotace: Diplomová práce se v první části zabývá teoretickým rozborem problematiky neuronových sítí, jejich funkcí a architekturou. Ve druhé části následuje popis tvorby sítě na příkladu rovnice vedení tepla v 1D. Poslední část obsahuje popis neuronové sítě řešící proudění v kavitě a diskusi výsledků jejích predikcí. Z výsledků je patrná schopnost sítě přesně predikovat pro data z tréninkového datasetu. Její schopnost extrapolace je omezená a klesá se zvyšující se vzdáleností od tréninkových dat.

Abstract: The first part of the thesis deals with the theoretical analysis of neural networks, their functions and architecture. In the second part, a description of the network creation is followed by an example of the heat conduction equation in 1D. The last part contains a description of the neural network solving the flow in the cavity and a discussion of the results of its predictions. The results show the ability of the network to make accurate predictions for the training dataset. Its ability to extrapolate is limited and decreases with increasing distance from the training data.

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval zcela samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.

Nemám závažný důvod proti užití tohoto školního díla ve smyslu § 60 Zákona č.121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne …….. ….…….

Podpis

Poděkování

Rád bych poděkoval vedoucímu této práce doc. Ing. Tomáši Hyhlíkovi, Ph.D. za jeho přínosné a odborné rady. Dále bych rád poděkoval mé přítelkyni a rodině za veškerou podporu během studia.

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

V posledních letech došlo k velkému pokroku v oblasti strojového učení a v důsledku toho i k revoluci řešení problémů v některých oblastech. Rapidní rozvoj strojového učení, jakožto odvětví umělé inteligence, je umožněn stále výkonnějším hardwarem a také dostupností obrovského množství informací ve všech možných oborech. S jeho produkty se můžeme běžně setkávat, ať už to jsou softwarové prvky moderních automobilů či digitální asistent ChatGPT. K tréninku běžné neuronové sítě je potřeba velké množství informací a výpočetní kapacity, a proto je pro některé inženýrské aplikace, řešící fyzikální problémy, jejich využití omezené a drahé.

Avšak jeden z nově vzniklých typů neuronové sítě se zdá být pro tuto problematiku ideální. Jedná se o *PINN* (physics-informed neural network), tedy neuronovou síť, která na základě znalosti fyziky popisující daný problém, může získávat data z řídících rovnic a tím zjednodušit a zlevnit její přípravu a samotné učení.

Tato práce se zaměřuje na využití strojového učení při CFD výpočtech, konkrétně proudění v kavitě. Obecně je výsledek řešení problému proudění velmi závislý na znalostech výpočtáře, a tedy i na správném nastavení výpočtu. I při správné konfiguraci a využití nejnovějšího softwaru může výpočet řešení v některých případech trvat i desítky hodin. A zde se naskytuje příležitost pro umělou inteligenci. Cílem této práce je vytvořit neuronovou síť, která bude umět predikovat pole 2D proudění v kavitě a následně její predikce otestovat. Testování bude provedeno na validačním datasetu, který prověří schopnost sítě interpolovat v rozsahu tréninkových dat. Dále budou prověřeny výsledky predikce extrapolací z dat mimo tréninkový dataset. Na základě těchto testů bude zhodnoceno, zda je síť korektně naučena a s jakou přesností předpovídá výsledky. Důležitou částí práce bude nejen správné vyhodnocení výsledných predikcí, ale i příprava samotné neuronové sítě, vyladění jejích parametrů učení a také správná implementace fyziky do výpočtu chybové funkce.

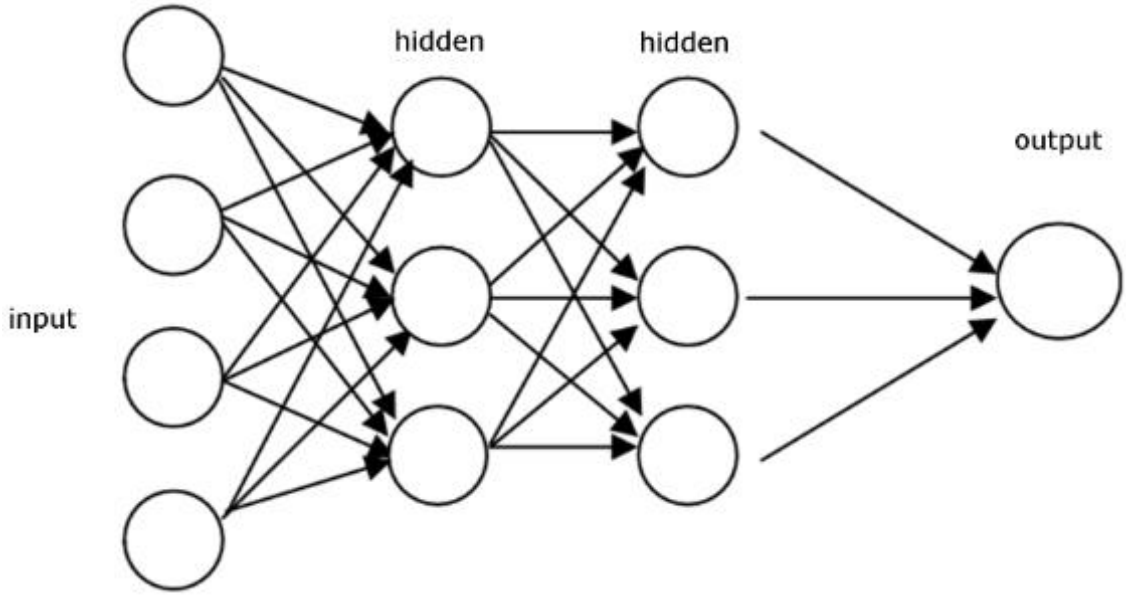
2 Teorie a metody neuronových sítí

Strojové učení (*machine learning)*, je jednou z mnoha disciplín spadajících do oblasti umělé inteligence (*artificial intelligence)*. Hlavní snahou tohoto oboru je využití dat a vhodných algoritmů ke tvorbě programů, schopných se učit a zlepšovat spolu se získáváním nových zkušeností. Výsledné programy mohou obecně sloužit k vytváření predikcí či hodnocení dat. Jejich schopnosti a přesnost výsledků jsou závislé na mnoha parametrech, především jsou to vstupní data a parametry procesu učení.

Neuronové sítě (*neural networks)* jsou odvětvím strojového učení. Disciplína zabývající se jimi se nazývá hluboké učení (*deep learning)*. Její počátky sahají do 40. let 20. století, kdy v roce 1943 proběhla první snaha o matematický popis fungování neuronů a již v roce 1959 byla vytvořena neuronová síť s názvem MADALINE, která jako první řešila reálný problém, využívala adaptivního filtru k rušení ozvěn na telefonních linkách. Avšak historie moderních neuronových sítí se začala psát až v roce 2006, kdy Geoffrey Hinton přišel s vylepšeným algoritmem učení, který umožnil efektivní trénink sítí s více vrstvami (*deep neural networks)*, a tím byla odstartována revoluce umělé inteligence [1][2].

2.1 Princip a architektura

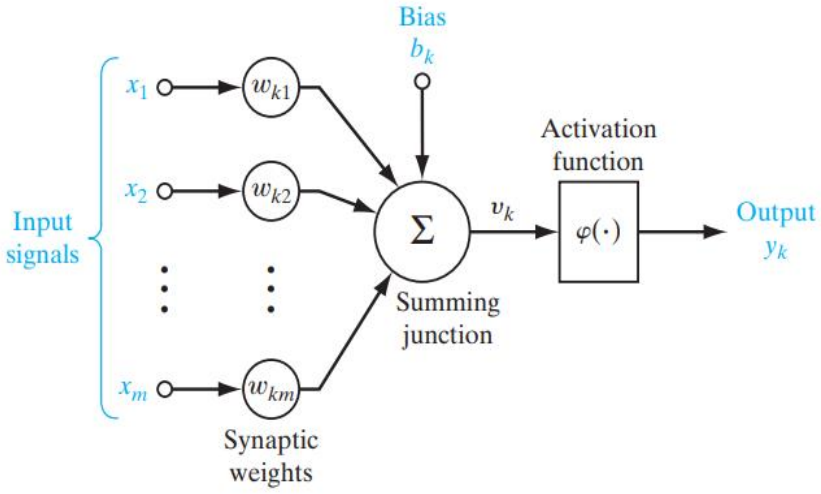
Neuronovou síť lze popsat jako skupinu výpočetních vrstev, které dle vstupního vektoru počítají odpovídající výstupní vektor či skalár. Jednotlivé vrstvy se skládají z neuronů a jsou propojeny řadou synapsí, spojovacích článků. Každá synapse má přiřazenou jinou váhu, tedy ovlivňuje připojený neuron jinou mírou. Vrstvy lze rozdělit na několik druhů. První či vstupní (*input*) zpracovává původní vektor, poslední, výstupní (*output*) zobrazuje výsledný vektor, skalár. Veškeré zbylé vrstvy, nacházející se mezi vstupní a výstupní, se nazývají skryté (*hidden layers*) a probíhá v nich výpočetní proces. Výše popsané schéma sítě je zobrazeno na následujícím obrázku (**Obrázek 2.1**). Jednotlivé sloupce zde představují vrstvy, každá kružnice reprezentuje neuron a šipky znázorňují synapse, tedy vzájemné vztahy neuronů. V plně propojené neuronové síti během výpočtu každý neuron ze skryté vrstvy dostává vstupní hodnoty od každého neuronu z předešlé vrstvy a zároveň jeho výstupní hodnotu předává každému neuronu do vrstvy následující. Tato architektura sítě se nazývá dopředná neuronová sít FNN (*feedforward neural network*). Informace tedy teče pouze v jednom směru od vstupní přes skryté až po výstupní vrstvy. Druhým typem architektury je rekurentní neuronová síť (*recurrent neural network*). V ní se může část informací vracet ,,proti směru‘‘ a ovlivňovat tak výstup z předchozích uzlů. Tato práce je zaměřena především na FNN.



Obrázek 2.1 *Schéma neuronové sítě [4]*

2.1.1 Neurony

Neurony jsou tvořeny matematickou funkcí koncipovanou stejně jako model biologických neuronů v mozku. Zde se projevuje podobnost a inspirace lidským mozkem. Neurony představují základní jednotku každé sítě, rozhodují o jejím fungování a zpracovávají veškeré informace. Jejich výstupem je právě jedna skalární hodnota. Model neuronu lze rozdělit na 3 části: soustavu vstupních synapsí, sumu sčítající vážené vstupní signály a aktivační funkci transformující hodnotu výstupu. Blokové schéma modelu neuronu, označeného *k*, lze zobrazit následovně (**Obrázek 2.2**):



Obrázek 2.2 *Model neuronu k [3]*

Synapse obsahují vstupní hodnotu x_j vycházející z předchozího neuronu a zároveň jsou charakterizovány hodnotou váhy w_{kj}. Konkrétní hodnota signálu x_j se násobí příslušnou váhou w_{kj} a vzniká tak vážený vstupní signál. Hodnota váhy může nabývat jak kladných, tak i záporných hodnot a reprezentuje závislost neuronu *k* na dané vazbě. Větší hodnoty vah tedy zobrazují vyšší vliv dané synapse. Sumou všech vážených vstupních signálů vzniká lineární sloučení vstupů u_k. Model dále obsahuje hodnotu *bias* označenou b_k. Ta má v závislosti na hodnotě za následek přímé zvýšení či snížení vstupní hodnoty do aktivační funkce, akčního potenciálu v_k. Aktivační funkce slouží k transformaci akčního potenciálu na nelineární funkci a zároveň může omezovat či normovat hodnotu výstupu. Neuron *k* lze tedy matematicky popsat párem rovnic:

$$u_k = \sum_{j=1}^{m} w_{kj} x_j \tag{2.1}$$

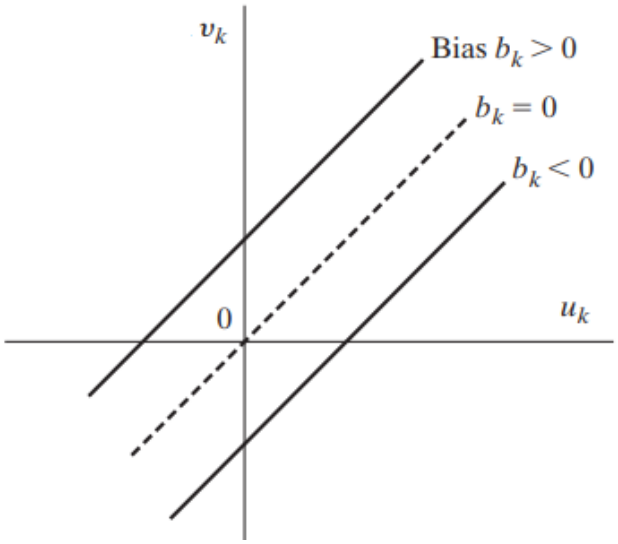
a

$$y_k = \varphi(u_k + b_k) \tag{2.2}$$

kde $x_1, x_2 \ldots, x_m$ jsou vstupní signály a $w_{k1}, w_{k2} \ldots, w_{km}$ jsou odpovídající váhy neuronu *k*, u_k je výstup lineárního sloučení vstupů, b_k je *bias*, y_k výstupní signál neuronu a φ představuje aktivační funkci neuronu. Použití *bias* b_k má za následek transformaci výstupu u_k dle následujícího vzorce.

$$v_k = u_k + b_k \tag{2.3}$$

V závislosti na znaménku *bias* b_k je vztah mezi aktivačním potenciálem v_k a výstupem lineárního sloučení vstupů u_k upraven dle následujícího zobrazení (**Obrázek 2.3**). Za povšimnutí stojí, že v důsledku této transformace graf neprochází počátkem [3].



Obrázek 2.3 *Transformace způsobená biasem [4]*

2.1.2 Aktivační funkce

Pokud bychom při používání neuronových sítí speciálně nedefinovali aktivační funkce, pak by se chovali pouze jako lineární regrese. Výstupní signál by představoval lineární funkci, tedy polynom prvního řádu, s vnesenou chybou. Ačkoliv by práce s těmito polynomy byla jednoduchá, jejich složitost je natolik omezená, že nemají vysoké rozlišovací schopnosti a schopnosti řešit komplexní problémy. Nehledě na počet skrytých vrstev v modelu, bez narušení jeho linearity bychom nikdy nedosáhli rozpoznání vzorů a vztahů mezi vstupními daty. U neuronových sítí je žádoucí, aby byly schopny se nejen učit a počítat lineární funkce, ale také provádět mnohem komplikovanější operace jako je rozpoznávání obrazu, zvuku či řeči.

Proto jsou zavedeny aktivační funkce φ, které slouží k narušení linearity modelu a umožňují práci s vysokodimenzionálními, nelineárními sadami dat. Jsou aplikovány na aktivační potenciál v_k a přímo transformují výstupní hodnotu a vytváří výstupní signál y_k, který je předávám do další vrstvy. Důležitou požadovanou vlastností aktivační funkce je diferencovatelnost, která je potřebná při učení sítě při implementaci algoritmu zpětného šíření chyby (*backpropagation*). Ve stručnosti se jedná optimalizační proces, který počítá chybu odhadu sítě vzhledem ke zvoleným vahám a *biasu* a na základě této chyby optimalizuje jejich hodnoty pro snížení chyby. Tento pojem bude více vysvětlen v další kapitole týkající se učení (2.2).

Existuje mnoho druhů různých aktivačních funkcí, které se liší oblastmi využití. Jejich vhodná volba může přímo ovlivnit rychlost učení, konvergence sítě a také přesnost. Některé z nich mohou

například limitovat hodnoty výstupního signálu neuronu, takové funkce se nazývají *squashing functions*. Zde bude uvedeno pár nejdůležitějších příkladů [5].

Nejjednodušším typem je funkce nazývaná ***Threshold***, ta funguje jen jako vypínač. Určuje, zda bude výstup nulový, neuron neaktivní či zda bude nenulový, neuron bude aktivní. Matematicky ji lze popsat následovně

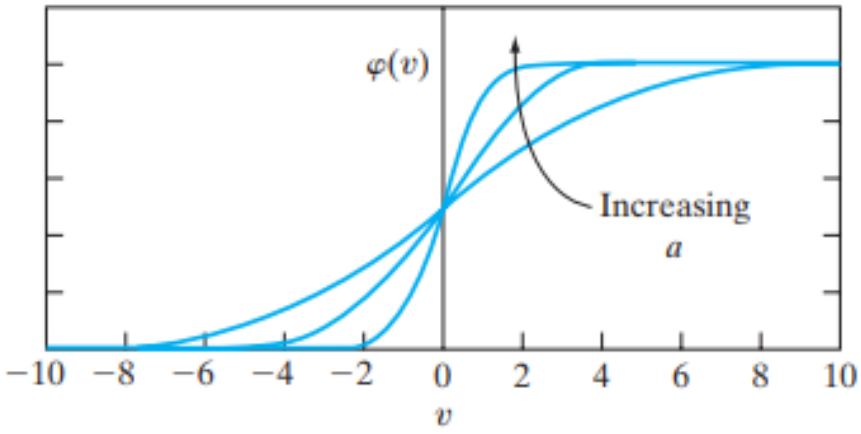
$$\varphi(v) = \begin{cases} 1, & pokud\ v \geq 0 \\ 0, & pokud\ v < 0 \end{cases} \tag{2.4}$$

Threshold je velmi jednoduchá a její hlavní nevýhoda je v nulové derivaci. Takže při použití *backpropagation* algoritmu neposkytuje žádné poznatky, jak optimalizovat hodnoty vah a *bias.*

Nejčastěji používaná je funkce s názvem ***Sigmoid***. Jde o nelineární funkci, která plynule transformuje hodnoty v rozmezí 0,1 a zároveň není symetrická vůči počátku, takže všechny ovlivněné výstupní hodnoty mají stejné znaménko. Její předpis lze zapsat

$$\varphi(v) = \frac{1}{1 + e^{-av}} \tag{2.5}$$

kde *a* je parametr určující její sklon. Jak je z předpisu patrné, *Sigmoid* je spojitě diferencovatelná, a tedy poskytuje dostatečnou informaci pro učení sítě.



Obrázek 2.4 *Sigmoid funkce [3]*

Dalším příkladem je funkce ***Tanh*** (*hyperbolic tangent function*), ta se velmi podobá *Sigmoid*, pouze je v porovnání s ní symetrická vůči počátku, a to může být pro některá použití přínosné.

Je definována jako:

$$\varphi(v) = \tanh(v) \tag{2.6}$$

Poslední, aktuálně často používanou aktivační funkcí je ***ReLU*** (*Rectified Linear Unit*). Její výhoda v porovnání s ostatními tkví v postupném aktivování neuronů a s tím spojenou zvýšenou výpočetní efektivitou, ze které vyplývá urychlení procesu učení. Neurony jsou deaktivovány pouze v případě, že je jejich výstupem číslo nekladné. Jediný problém nastává při učení v bodě 0, ve kterém není diferencovatelná a je zde třeba uměle dodefinovat její hodnotu. Tomuto popisu je odpovídající následující zápis

$$\varphi(v) = max(0, v) \tag{2.7}$$

2.2 Učení neuronové sítě

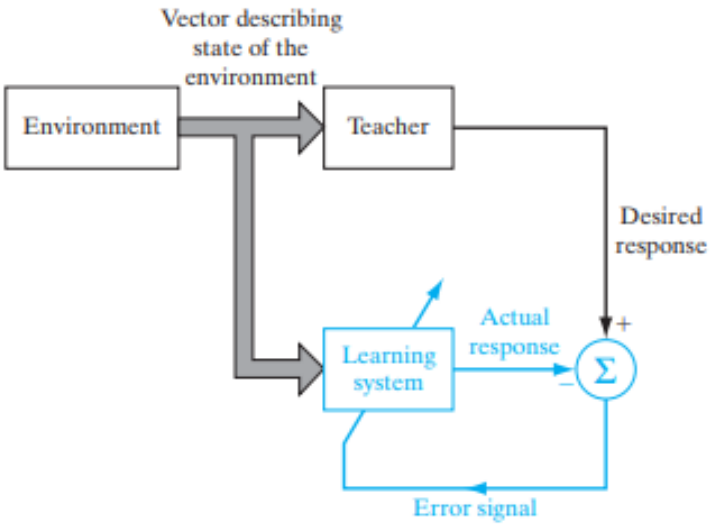
V předchozích kapitolách byly obecně představeny neuronové sítě, jejich architektura, fungování neuronů a důležité aktivační funkce. Nyní vzestává otázka, jak všechny tyto poznatky využít a síť naučit fungování dle našich požadavků. Jednoduchou odpovědí je správné nastavení parametrů, tedy hodnot vah a *bias.* K tomu nám slouží procesy učení, kdy se síť sama iteračním procesem učí a mění hodnoty parametrů tak, aby vyhověly. Hlavní motivace k tomu se liší v závislosti na typu učení.

Zde se opět nabízí paralela k lidskému mozku, pro který existují různé druhy učení, a stejné je to i u neuronových sítí. Procesy učení, na jejichž základě se neuronové sítě učí, lze kategorizovat následovně: učení s učitelem a bez učitele. Druhá forma bez učitele může být dále rozdělena do dvou podkategorií, posilující učení (*reinforcement learning*) a učení bez dozoru (*unsupervised learning*). Každá kategorie má své výhody a oblasti využití, nelze proto o některé obecně tvrdit, že by byla výhodnější či lepší.

2.2.1 Učení s učitelem

Učení s učitelem (*supervised learning*) si lze koncepčně můžeme představit jako interakci učitele, který má znalosti celého prostředí, ty jsou reprezentovány sadami vstupních a odpovídajících, očekávaných výstupních dat, a neuronové sítě, pro kterou je toto prostředí neznámé, snaží se jeho fungování napodobit. Předpokládejme nyní, že je učitel i neuronová síť vystavena tréninkovému příkladu. Díky svým znalostem může učitel síti ukázat požadované řešení pro daný příklad. Toto

řešení představuje optimální odezvu pro síť. Ta si vzhledem k tréninkovému příkladu a vzhledem k chybovému signálu upraví hodnoty svých parametrů. Chybový signál je definován jako funkce rozdílu požadované odezvy a skutečné odezvy sítě. Úpravy se provádí iteračně krok za krokem, kdy každému kroku odpovídá nový tréninkový příklad, a proto je třeba mít vhodné množství dat pro dostačující trénink. Zprvu mohou být odhadované výsledky zcela náhodné, ale s postupem učení se zvyšuje schopnost sítě emulovat učitele, až do bodu, kdy dojde k minimalizaci chybové signálu a odhad dosáhne svého optima. Tímto způsobem jsou znalosti prostředí učitele předány neuronové síti ve formě pevně stanovených hodnot vah a *bias,* představujících dlouhodobou paměť. Poté, co je dosaženo optima, můžeme odstranit učitele a nechat síť, aby se s novými příklady ze stejného prostředí vypořádala sama.



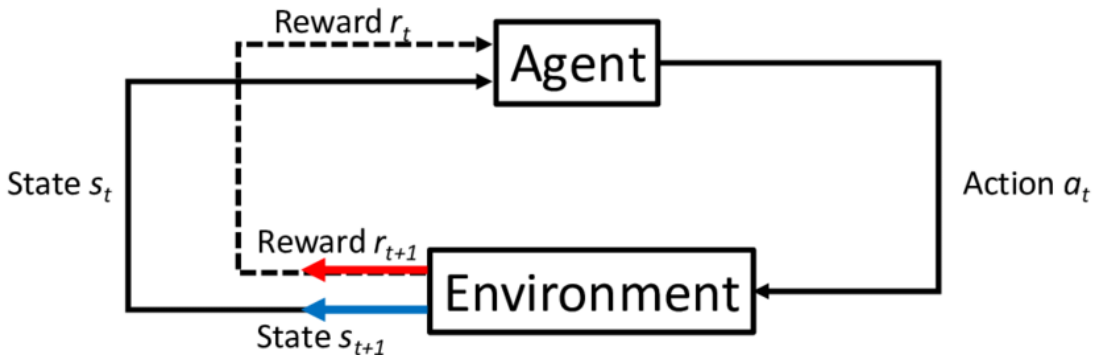
Obrázek 2.5 *Supervised learning [3]*

Tato forma učení je základem pro všechny další, obsahující korekci chyb. Schéma celého procesu (**Obrázek 2.5**) obsahuje uzavřenou smyčku systému se zpětnou vazbou (modře), ale neznámé prostředí se nachází mimo ni. Měřítkem efektivity tohoto systému může být střední kvadratická chyba definovaná jako funkce parametrů sítě. Lze ji vizualizovat v multidimenzionálním grafu jako povrch chyby (*error surface*) s parametry vah a *bias* jako souřadnicemi. Postupná optimalizace je v tomto grafu vidět jako posun povrchu chyby k minimu.

2.2.2 Učení bez učitele

Při učení bez učitele, neexistuje žádný učitel, ani žádné označené příklady, kterými by se síť mohla inspirovat. Celý proces učení proto funguje na zcela jiném principu.

Jak již bylo zmíněno, tak první podkategorií je **posilující učení** (*reinforcement learning*). To k učení využívá metody pokus-omyl. Hlavními dvěma entitami jsou zde agent a prostředí (*environment*). Agent se snaží na základě zpětné vazby optimalizovat svoje řešení daného problému. Za akce vedoucí ke správným výsledkům je odměňován a za chyby je naopak trestán. Prostředí představuje zcela neznámé okolí agenta a určuje směr jeho učení. Proces (**Obrázek 2.6**) učení probíhá v krocích, kdy agent obdrží aktuální stav s_t a odměnu r_t a na jejich základě vykoná akci a_t, která vede k aktualizaci stavu s_{t+1} a odměny r_{t+1}, ty jsou opět sděleny agentovi. Tento proces může být velmi zdlouhavý. Jeho předností je však nepotřebnost jakéhokoliv učitele či vstupních dat [6].



Obrázek 2.6 *Schéma procesu reinforcement learning [7]*

Druhou podkategorií tohoto typu učení je již zmíněné **učení bez dozoru** (*unsupervised learning*). Zde není přítomen ani učitel ani systém odměňující činnosti sítě. Jednou z možných motivací k učení může být použití konkurenční vrstvy, kde spolu jednotlivé neurony soutěží a pouze ten s nejvhodnějším výstupem je aktivován.

2.2.3 Gradient descent

Uvažujme nyní neuronovou síť s jednou a více skrytými vrstvami. Nechť τ představuje sadu tréninkových dat

$$\tau = \{x(n), d(n)\}_{n=1}^{N} \tag{2.8}$$

Dále nechť $y_j(n)$ označuje funkční výstup produkovaný neuronem j (viz Obrázek 2.7) ve výstupní vrstvě, který vznikl na základě vstupu $x(n)$ do vstupní vrstvy. Chybový signál (*error signal*) jakožto část výstupu je definován následovně

$$e_j(n) = d_j(n) - y_j(n) \tag{2.9}$$

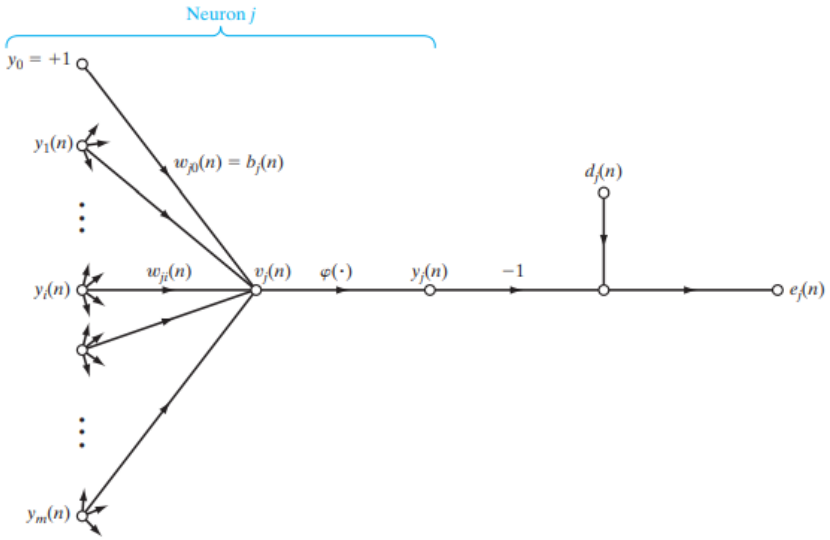
kde $d_j(n)$ je j-tý prvek požadovaného výstupního vektoru. S využitím chyby lze definovat chybovou funkci neuronu j jako

$$\xi_j(n) = e_j^2(n) \tag{2.10}$$

Sečtením příspěvků chybových funkcí všech neuronů výstupní vrstvy můžeme vyjádřit celkovou chybovou funkci

$$\xi(n) = \sum_{j \in C} \xi_j(n) = \sum_{j \in C} [d_j(n) - y_j(n)]^2 \tag{2.11}$$

kde soubor C obsahuje všechny neurony výstupní vrstvy.



Obrázek 2.7 *Graf toku informace neuronem j [3]*

Algoritmus *gradient descent* je využíván k hledání minima funkce celkové chyby $\xi(n)$. Během tohoto procesu používáme gradient této funkce $\nabla\xi(n)$, který slouží jako vodítko ke změně parametrů sítě *w, b.* Jeho vyjádření je následující:

$$\nabla\xi(n) = \frac{\partial\xi(n)}{\partial w_{ji}(n)} \tag{2.12}$$

Zde došlo ke změně zápisu oproti předešlým řádkům. Rozdílná notace vah *w* a *bias b* byla sjednocena, váha synapse w_{j0} (s odpovídajícím vstupem $y_0 = 1$) se rovná *biasu* b_j aplikovanému na neuron *j* (viz **Obrázek 2.7**).

Obecně gradient v daném bodě vyjadřuje směr nejstrmějšího stoupání funkce, proto pokud chceme funkci minimalizovat je třeba jeho hodnotu přenásobit -1 a tím otočit směr na nejstrmější klesání. Využitím výše popsaného lze korekci $\Delta w_{ji}(n)$ aplikovanou na parametry sítě $w_{ji}(n)$ definovat dle vztahu

$$\Delta w_{ji}(n) = -\eta \nabla \xi(n) = -\eta \frac{\partial \xi(n)}{\partial w_{ji}(n)} \tag{2.13}$$

parametr η zde značí rychlost učení (*learning rate*), jeho hodnota je vždy kladná a definuje, jak velkými kroky se ve směru největšího klesání budeme pohybovat. Větší hodnoty rychlosti učení mohou vést k rychlejšímu učení sítě, ale za cenu menší přesnosti. Menší kroky při minimalizaci naopak zvyšují rozlišení, s jakým může síť upravovat své váhy a *bias,* ale mohou podstatně navýšit dobu učení. Proto také vznikly algoritmy, které parametr rychlosti učení během procesu upravují a optimalizují tak celý proces. Algoritmus *gradient descent* je základním a nejjednodušším z mnoha. Na jeho základě či jako kombinace s dalšími vznikly také další používané algoritmy například ADAM nebo LBFGS.

Nyní máme k dispozici matematický algoritmus k optimalizaci parametrů neuronové sítě. Dokážeme pomocí něj minimalizovat chybovou funkci. Otázkou je, jak ho zakomponovat do iteračního procesu učení [3].

2.2.4 Backpropagation

Backpropagation algoritmus nám poskytuje způsob, jak počítat gradient chybové funkce a postupným zpětným šířením přiřadit jednotlivým neuronům jejich odpovídající chyby. V kombinaci s metodou *gradient descent* jsme tedy schopni chyby přiřazené konkrétním neuronům upravovat za účelem minimalizace chybové funkce, tedy zpřesnění odhadů sítě.

K popisu tohoto algoritmu je použito několik rovnic, první z nich (2.14) poměřuje ve své první části, jak rychle se chybová funkce ξ mění v závislosti na výstupech neuronů poslední vrstvy y_j^L. Druhá část měří, jak rychle se mění aktivační funkce φ v závislosti na v_j^L.

$$\delta_j^L = \frac{\partial \xi}{\partial y_j^L} \varphi'(v_j^L) \tag{2.14}$$

Pokud ku příkladu ξ téměř nezávisí na konkrétním výstupu neuronu *j*, potom bude hodnota δ_j^L podle očekávání malá.

Druhá rovnice (2.15) popisuje výpočet vedoucí k hodnotě chyby δ^l vycházející ze znalosti chyby následující vrstvy δ^{l+1}. Člen $(w^{l+1})^T$ představuje transponovanou matici vah následující vrstvy. Tento výraz si lze představit jako šíření chyby zpět skrze síť. Poslední část $\odot\varphi'(v_j^L)$ obsahující Hadamardův součin posune chybu zpět aktivační funkcí a obdržíme tedy chybu δ^l vážených vstupů do vrstvy *l*.

$$\delta^l = \left(\left(w^{l+1}\right)^T \delta^{l+1}\right) \odot\varphi'(v_j^L) \tag{2.15}$$

Třetí a poslední doplňující rovnice, popisující gradient chybové funkce již byla popsána výše (2.12).

Postup *backpropagation* algoritmu lze shrnout do tří kroků:

1. Vložení vstupního souboru dat k učení
2. Pro každý tréninkový příklad *x* se provede následující:
    2a. Pro každé *l=2,3,…,L* proběhne dopředný výpočet $v^{x,l}$ a následně proběhne průchod aktivační funkce $\varphi(v^{x,l})$
    2b. Výpočet chyby $\delta_j^{x,L}$ dle rovnice 2.14
    2c. Zpětné rozšíření chyby pro každé *l=L-1,L-2,…,2* dle rovnice 2.15
3. Použití minimalizační metody např. *gradient descent,* pro každé *l=L,L-1,…,2* se upraví váhy a *bias* dle pravidla $w^l \rightarrow w^l - \Delta w_{ji}(n)$, viz (2.13)

Z původních náhodných hodnot vah a *bias* se tedy síť postupně za použití algoritmů *gradient descent* a *backpropagation* učí, minimalizuje chybovou funkci úpravami vah a *bias [8]*.

2.3 Neuronové sítě s fyzikální znalostí

Hlavním zaměřením této práce jsou neuronové sítě s fyzikální znalostí (*physics informed neural networks)* zkráceně také PINN. Jejich odlišnost od ostatních typů sítí je především v definici a výpočtu chybové funkce. U ostatních typů může chyba vycházet pouze z odlišnosti predikce a požadované hodnoty tréninkového datasetu. Chybová funkce PINN sítě obvykle bývá složitější, její součástí bývají fyzikální rovnice, popisující řešený problém. Často jsou tyto řídící rovnice diferenciálního typu, a proto je třeba v průběhu učení počítat parciální derivace predikcí a z nich s pomocí rovnic dopočítávat chybu. Tento proces může být mnohdy výpočtově náročný, avšak v jeho důsledku může síť lépe vystihovat fyzikální podstatu problému a zachycovat složitější vzorce, které by při použití pouze výsledku například numerických výpočtů nebyly zřejmé. Součástí chybové funkce mohou být dále také okrajové či počáteční podmínky. Ty neuronové síti dávají jakési opěrné body, ze kterých může při učení vycházet a dále zlepšují predikce na počátku či na krajích řešené oblasti. Obecně může síť správně řešit daný problém ve středu uvažované oblasti a kvůli tomu, že se převážná většina bodů nachází ve středu, může síť přehlížet počáteční či okrajové hodnoty. Zavedením okrajových a počátečních podmínek lze tomuto problému předejít.

Implementací fyziky do procesu učení získáváme více možností, jak síť směrovat správným směrem k požadovaným výsledkům. Společně s rostoucím množstvím nástrojů a složitostí chybové funkce roste i složitost celého procesu a může být složitější stanovit parametry učení či najít optimální nastavení. Stejně jako u každého jiného typu neuronové sítě, i zde platí, že není možné přímo předpovídat výsledky procesu učení a mnohdy je třeba zkoušet různá nastavení a testovat různé kombinace parametrů učení.

2.4 Řešení 1D vedení tepla

První řešenou úlohou, představující prvotní seznámení s problematikou neuronových sítí, byla rovnice vedení tepla v jedné dimenzi. Jedná se o parciální diferenciální rovnici parabolického typu popisující šíření tepla v jednorozměrné tyči. Její výhodu představuje jednoduše získatelné přesné numerické řešení, jinými slovy lehce získat data potřebná k strojovému učení.

Uvažovaná řešená rovnice vedení tepla tedy byla:

$$\frac{\delta T}{\delta t} = \alpha * \frac{\delta^2 T}{\delta x^2} \tag{2.16}$$

kde α představuje součinitel tepelné vodivosti a v našem konkrétním případě měl hodnotu $\alpha = 2\, m^2/s$. Řešená tyč měla hlavní rozměr 2 metry a čas probíhal od 0 do 0,2 vteřin. Pro co

nejvyšší názornost a jednoduchost této úlohy byly zvoleny Dirichletovy podmínky na okrajích tyče a počáteční podmínka určena kvadratickou funkcí. Shrnutí celého numerického zadání, včetně konkrétních podmínek je zobrazeno v následujících řádcích:

Diferenciální rovnice	$T(x,t)\colon \frac{\delta T}{\delta t} = 2\frac{\delta^2 T}{\delta x^2}$
Okrajové podmínky	$T(0,t) = T(2,t) = 0$
Počáteční podmínka	$T(x,0) = -2x^2 + 4x$

Rovnice byla numericky řešena pomocí explicitního Eulerova schématu na síti o rozměru $(x,t) = (100{,}3961)$. Rozměry sítě numerického schématu byly zvoleny na základě velikosti sítě, na které bude učena neuronová síť, odpovídají tedy rozměrům 100 x 100 bodů. Zjemnění ve směru časové osy bylo třeba pro splnění podmínky stability zmíněného schématu. Hodnota končí neobvykle 1, protože bylo třeba synchronizovat hodnoty času tak, aby obsahovaly stejné hodnoty jako síť strojového učení.

V dalších řádcích bude popsán krok po kroku proces tvorby neuronové sítě, vycházející z kompletní znalosti problematiky, tedy numerického řešení, počáteční a okrajové podmínky. Při popisu budou použity obrázky kódu neuronové sítě.

Celá síť byla programována v jazyce Python, konkrétně byla využita knihovna PyTorch. Nejprve tedy bylo potřeba importovat tuto a další potřebné knihovny.

```
import torch
import numpy as np
import torch.nn as nn
import torch.optim as optim
import matplotlib.pyplot as plt
```

Obrázek 2.8 *Použité knihovny*

Přiložený obrázek ukazuje importované knihovny včetně použitých zkratek, tak aby bylo v dalších odstavcích jasnější čtení přiloženého kódu. *NumPy* přidává důležitou podporu výpočetních operací s maticemi a velkými sadami dat. *Torch.nn* jakožto část knihovny *PyTorch* umožňuje jednoduše definovat neuronové sítě a následně s nimi manipulovat, *torch.optim* obsahuje sadu optimalizačních algoritmů, ze kterých je možné volit ten nejvhodnější pro správnou konvergenci

učení. Poslední používanou knihovnou je *matplotlib.pyplot*, umožňující tvořit grafy a lépe vizualizovat průběh a výsledky procesu strojového učení.

V prvním kroku je třeba definovat neuronovou síť, její rozměry, počet vstupních a výstupních veličin a také metodu, jak budou data sítí procházet, společně s aktivační funkcí.

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.hidden1 = nn.Linear(2, 50)     # input layer (x, t)
        self.hidden2 = nn.Linear(50, 50)    # hidden layer
        self.hidden3 = nn.Linear(50, 50)    # hidden layer
        self.hidden4 = nn.Linear(50, 50)    # hidden layer
        self.hidden5 = nn.Linear(50, 50)    # hidden layer
        self.hidden6 = nn.Linear(50, 50)    # hidden layer
        self.output = nn.Linear(50, 1)      # output layer (u)
```

Obrázek 2.9 *Definice neuronové sítě*

Na prvním řádku je pomocí *class Net(nn.Module)* definována třída s názvem *Net*, která dědí vlastnosti již existující třídy nn.Module a ta se stává její nadřazenou. V dalších dvou řádcích *def __init__(self)* a *super(Net, self).__init__()* je volán konstruktor, díky kterému je možné inicializovat vlastnosti vytvořené třídy a nastavit definovaný objekt správně v paměti. Zároveň tyto příkazy zajišťují správnou dědičnost všech vlastností nadřazené třídy. V podstatě jsou tedy vytvořeny základy neuronové sítě pojmenované *Net*, která se bude chovat stejně jako třída *nn.Module* definovaná v knihovně PyTorch. Následně je možné nastavit inicializované vlastnosti, zde se jedná o vrstvy neuronové sítě. *Self.hidden1 = nn.Linear(2, 50)* definuje první vstupní vrstvu, kde zapsané parametry říkají, že vrstva *hidden1* má 2 vstupy a 50 výstupů. Volba množství vstupů závisí na řešené problematice, v případě 1D nestacionární úlohy stačí jedna prostorová a jedná časová souřadnice. Výstupy je nutno volit s přihlédnutím k činnosti sítě. Vyšší množství může zlepšit rozlišovací schopnost sítě, ale zároveň zpomaluje celý proces učení. V podobném stylu zde definujeme celkem 7 vrstev, poslední, výstupní vrstva má nastaven pouze 1 výstup, to by měla být v tomto případě predikovaná hodnota teploty *T*.

Dále je potřeba dodefinovat, jak spolu jednotlivé vrstvy musí interagovat a dát jim vazby spojující jednotlivé vstupy a výstupy.

```
def forward(self, x_t):
    hidden1_out = torch.tanh(self.hidden1(x_t))
    hidden2_out = torch.tanh(self.hidden2(hidden1_out))
    hidden3_out = torch.tanh(self.hidden3(hidden2_out))
    hidden4_out = torch.tanh(self.hidden4(hidden3_out))
    hidden5_out = torch.tanh(self.hidden5(hidden4_out))
    hidden6_out = torch.tanh(self.hidden6(hidden5_out))
    u = self.output(hidden6_out)
    return u
```

Obrázek 2.10 *Definice průchodu daty sítí*

První řádek *def forward(self, x_t)* definuje metodu *forward*, která určuje dopředný průchod daty sítí, určuje tedy způsob transformace vstupů na výstupy. Argument *x_t* představuje vstupní data, v další části kódu je tedy potřeba tento tensor odpovídající časoprostorové síti definovat. Následně je třeba formulovat vztahy mezi jednotlivými vrstvami. V řádku *hidden1_out = torch.tanh(self.hidden(x_t))* jsou vstupní data propuštěna skrze první vrstvu *hidden1* a následně je na ně aplikována aktivační funkce *tanh*, která normalizuje výstupy *hidden1_out* na hodnotu v intervalu (-1,1). V následujícím řádku je výstup první vrstvy propuštěn skrze druhou vrstvu a v podobném duchu jsou popsány vazby mezi všemi dalšími vrstvami. Poslední řádek určuje *T* výstupem celé sítě. Volba konkrétních aktivačních funkcí opět závisí na řešené problematice a je možné je měnit v závislosti na výsledcích učení. Výhodou vybrané funkce je její symetrie kolem nuly, která zvyšuje stabilitu učení a zároveň její komplexnost předchází problémům, vyskytujícím se u jednodušších funkcí.

Nyní, když je plně definována neuronová síť se všemi potřebnými vazbami, je třeba doplnit optimalizační algoritmus a metody, jakými bude probíhat proces učení.

```
net = Net().to(device)
optimizer = optim.Adam(net.parameters(), lr=0.01)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=2000, gamma=0.1)
criterion = nn.L1Loss()
```

Obrázek 2.11 *Definice optimalizačního algoritmu*

První řádek další části přiloženého kódu *net = Net().to(device)* vytvoří, seskládá neuronovou síť s výše definovanou architekturou a tím ji umožní provádět výpočty v dalších částech skriptu. *optimizer = optim.Adam(net.parameters(), lr=0.01)* nastavuje optimalizační algoritmus používaný během učení. Volba konkrétního algoritmu přímo ovlivní konvergenci a rychlost učení. Zvolený *ADAM* je jednou z nejpoužívanějších a nejrobustnějších možností, zároveň není výpočtově náročný a měl by měl být schopen rychlé konvergence. Parametr *lr* nastavuje počáteční hodnotu rychlosti učení (*learning rate*) na 0,01. Obecně je výhodné s průběhem celého procesu učení jeho hodnotu snižovat, protože pro příliš vysoké hodnoty algoritmus nedokáže dělat dostatečně malé kroky

směrem k minimu chybové funkce a tím nekonverguje. Proto je v dalším řádku definován *scheduler,* který v pravidelných intervalech redukuje rychlost učení. V zobrazeném kódu (viz **Obrázek 2.11**) je nastaven tak, aby po každých 2000 krocích učení vynásobil hodnotu rychlosti učení 0.1 a tím ji efektivně periodicky snižuje na desetinu. Opět je možné použít jiné alternativy, které snižují optimalizační krok exponenciálně nebo pouze jsou podmíněny hodnotami metrik. Pro tento jednoduchý případ není potřeba používat složitější algoritmus s vyšší adaptibilitou a tím zvyšovat výpočetní náročnost. Poslední řádek *criterion = nn.L1Loss()* nastavuje, jakým způsobem bude počítána chybová funkce. Funkce *L1Loss* počítá absolutní střední chybu mezi výstupem a referenčními hodnotami.

Kód, který byl doposavad popisován, vytvoří obecnou neuronovou síť o 7 vrstvách s 1 vstupní a 2 výstupními hodnotami. Používá algoritmus ADAM k minimalizaci chybové funkce a hodnota rychlosti učení je periodicky redukována každých 2000 iterací. Dále je třeba si připravit časoprostorovou síť a rovnice popisující problematiku vedení tepla, díky kterým bude možné formovat neuronovou síť.

```
# Training data
x = np.linspace(0, 2, 100)
t = np.linspace(0, 0.2, 100)
X, T = np.meshgrid(x, t)
# Convert to tensors
x_t_data = torch.tensor(np.hstack((X.flatten()[:,None], T.flatten()[:,None])), dtype=torch.float32).to(device)

# Initial condition
IC = -2*x**2+4*x
IC = torch.tensor(IC, dtype=torch.float32).to(device)

# List for storing of loss values
losses = []
```

Obrázek 2.12 *Příprava časoprostorové sítě a formulace počáteční podmínky*

S využitím funkce *linspace* z knihovny NumPy jsou v prvních dvou řádcích vytvořena pole o 100 rovnoměrně rozložených bodech. V případě prostorové souřadnice jsou rozloženy mezi hodnotami 0 a 2, pro čas jsou to hodnoty 0 a 0,2. Řádek *X, T = np.meshgrid(x, t)* vytváří z dvou jednorozměrných polí (polohy a času) jedno dvourozměrné časoprostorové pole. V dalším řádku je provedeno hned několik operací, první z nich je *X.flatten()[:,None]*, tato funkce převádí dvourozměrné pole *X* respektive *T* na jednorozměrné a následně zpět, avšak oproti původnímu tvaru má v novém poli každá hodnota vlastní sloupec respektive řádek. Následně jsou vzniklé produkty spojeny funkcí *hstack*, která je po sloupcích spojí do jediného 2D pole, které má na každém řádku dvojici odpovídajících hodnot *X* a *T*. Z původních polí *x* a *t* o rozměrech (100) tedy vzniká výsledné pole o rozměru (10 000, 2). Vzniklé pole je nakonec převedeno funkcí *torch.tensor* na tensor, se kterým pracuje knihovna PyTorch. V následujícím řádku je formulována počáteční podmínka ve tvaru $IC = 2x^2 + 4x$, ta je následně také převedena na tensor. Poslední řádek *losses = []* připraví prázdný

seznam, který je dále použit k ukládání hodnot chybové funkce v každé iteraci. Na jeho základě je možné dobře vizualizovat a analyzovat průběh učení.

A konečně se dostáváme k nejdůležitější části kódu, tedy k cyklu, ve kterém je počítána chybová funkce a ve kterém probíhá celé učení. Tento cyklus je pro lepší přehlednost a srozumitelnější popis rozdělen do dvou obrázků (**Obrázek 2.13**, **Obrázek 2.14**)

```
# Define number of iterations
iterations = 10000
    # Training loop
for epoch in range(iterations):
        net.zero_grad()

        x_t_data.requires_grad = True
        u_pred = net(x_t_data)

        # Compute loss based on IC
        u_pred_IC = u_pred[:100]
        loss_IC = criterion(u_pred_IC, IC)
        loss = loss_IC

        # Enforce boundary conditions
        u_bc0 = net(torch.tensor([[0., t] for t in np.linspace(0, 0.2, 100)], dtype=torch.float32).to(device))
        u_bc1 = net(torch.tensor([[2., t] for t in np.linspace(0, 0.2, 100)], dtype=torch.float32).to(device))
        loss += criterion(u_bc0, torch.zeros_like(u_bc0))
        loss += criterion(u_bc1, torch.zeros_like(u_bc1))
```

Obrázek 2.13 *Formulace cyklu učení – první část*

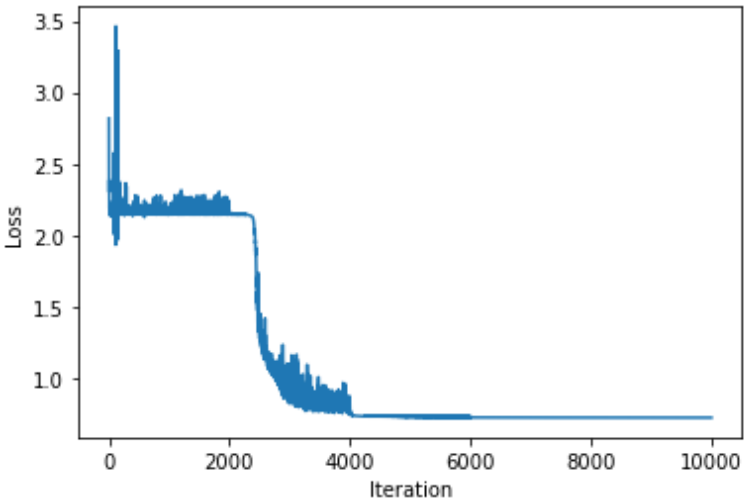
Na začátku je definován *for* cyklus, který bude probíhat po dobu rovnající se počtu iterací, ten je specifikován před cyklem. Nejprve je třeba vynulovat všechny gradienty proměnných funkcí *net.zero_grad()*, aby nedocházelo k jejich kumulaci z předešlých iterací. *x_t_data.requires_grad = True* povoluje výpočet gradientů x_t dat, který je potřebný při výpočtu chybové funkce. Dále je proveden dopředný průchod sítí vstupními daty *x_t_data* a je jim přiřazen odpovídající výstup *u_pred*. Nyní na řadu přichází výpočet chyby. Z predikce je vyjmuto prvních 100 hodnot, odpovídajících počáteční podmínce *u_pred_IC = u_pred[:100],* tyto hodnoty jsou následně porovnány s již vypočtenými hodnotami podmínky za pomocí chybové funkce *criterion* definované výše a výsledek je nazván chybou (*loss*). K chybě jsou následně přičteny další chyby vzniklé nesplněním okrajových podmínek *loss += criterion(u_bc0, torch.zeros_like(u_bc0))*. Chyba je vypočtena porovnáním tensoru predikcí sítě pro krajní body, spočtených na řádku *u_bc0 = net(torch.tensor([[0., t] for t in np.linspace(0, 0.2, 100)],….)* a nulových tensorů o stejných rozměrech, protože z okrajové podmínky mají být hodnoty krajních bodů nulové.

```
# Enforce heat equation
du_dt = torch.autograd.grad(u_pred.sum(), x_t_data, create_graph=True)[0][:, 1:2]
du_dx = torch.autograd.grad(u_pred.sum(), x_t_data, create_graph=True)[0][:, 0:1]
d2u_dx2 = torch.autograd.grad(du_dx.sum(), x_t_data, create_graph=True)[0][:, 0:1]
residual = du_dt - 2*d2u_dx2
# residual_abs = torch.abs(residual)
loss_PDE =criterion(residual, torch.zeros_like(residual))
loss += loss_PDE

# Enforce numerical solution
u_pred_reshaped = u_pred.view(100, 100)
u_numer_tensor = torch.tensor(u_numer, dtype=torch.float32).to(device)
loss_numer = criterion(u_pred_reshaped, u_numer_tensor)
loss += loss_numer

loss.backward()
optimizer.step()
scheduler.step()

# Save current loss for later graphing
losses.append(loss.item())
```

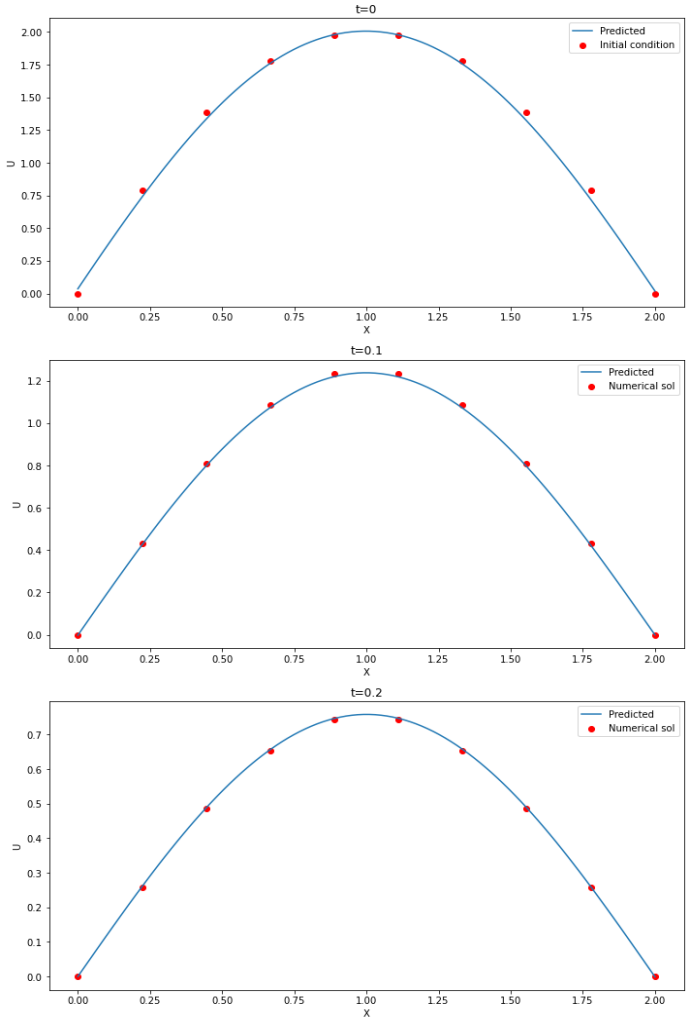
Obrázek 2.14 *Druhá část cyklu učení*

Pro výpočet chyby od diferenciální rovnice je třeba spočítat derivace funkce predikce, to je provedeno následovně: *du_dt = torch.autograd.grad(u_pred.sum(), x_t_data, create_graph=True)[0][:, 1:2],* zde je vypočtena derivace predikce *u_pred* vzhledem k času, který je specifikován na konci výrazem *[:, 1:2].* Obdobným způsobem je dopočtena první i druhá prostorová derivace, rozdíl je pouze ve specifikaci derivace podle souřadnice *[:, 0:1].* Z obdržených derivací lze jednoduše stanovit hodnotu *residual = du_dt – 2*d2u_dx2,* vyjadřující diferenciální rovnici. Hodnota residua je porovnána s nulovým tensorem o stejném rozměru a celková chyba je opět rozšířena o tuto hodnotu *loss += loss_PDE*. Posledním přídavkem do celkové hodnoty chybové funkce je odchylka od numerického řešení. Pro její stanovení je nutné nejprve přetvořit tensor predikce na 2D *u_pred_reshaped = u_pred.view(100, 100)* a z pole výsledků numeriky vytvořit tensor pomocí funkce *torch.tensor.* Porovnáním těchto dvou vznikne poslední přídavek a hodnota *loss* obsahuje vše potřebné pro správné naučení sítě. Na konci cyklu je třeba iniciovat výpočet gradientů chyby vzhledem k parametrům modelu *loss.backward()* a na jejich základě aktualizovat parametry modelu *optimizer.step(),* aktivovat redukci rychlosti učení *scheduler.step()* a jako úplně poslední zaznamenat hodnotu chybové funkce do připraveného seznamu *losses.append(loss.item()).*

Ze seznamu, ukazujícího vývoj chyby v průběhu iterací, je možné vykreslit graf. V našem případě je vidět jasná konvergence k minimu (viz **Obrázek 2.15**). Výsledná predikce je zobrazena společně s body počáteční podmínky a body numerického řešení v dalším obrázku (**Obrázek 2.16**).



Obrázek 2.15 *Graf vývoje chyby v průběhu iterací učení*



Obrázek 2.16 *Výsledná predikce naučené sítě*

2.5 Rozbor problematiky

Jak již bylo řečeno v úvodu, během několika posledních dekád došlo k obrovskému posunu v oblasti výpočtů proudění (CFD), numericky řešící jak nestlačitelné, tak stlačitelné problémy proudění tekutin, popsané Navierovými – Stokesovými rovnicemi. Stále je však při aplikaci na reálné problémy obtížné bezchybně začlenit experimentální data do existujících algoritmů a také je pro složitější průmyslové problémy generování sítě příliš časově náročné a vyžaduje zkušenosti a cit pro věc. Kromě toho je také velký prostor pro zlepšení řešení inverzních úloh, kdy je například pro neznámé okrajové podmínky řešení velmi složité a vyžaduje konkrétní modifikace a tvorbu nových počítačových kódů. Obecně lze aplikaci strojového učení na mechaniku tekutin rozdělit na dva typy: úplnou a částečnou regresi, které se snaží daný modelový problém řešit buď zcela, nebo částečně. Nejčastějším použitím úplné regrese je predikce dynamiky proudění za daných podmínek, či inverzní problematika řešící neznámé parametry proudění. Příkladem tohoto využití neuronových sítí integrujících experimentální data s různými formulacemi Navierových – Stokesových rovnic je série příspěvků publikovaných od Karniadakisovy skupiny, a to pro nestlačitelné [12],[13],[11] i stlačitelné [14] proudění a také pro biomedicínské aplikace [15]. Hybridní regrese nachází využití v modelování uzávěru turbulence. V turbulentním proudění je kvůli širokému spektru rozměrů vírů téměř nemožné řešit víry malých rozměrů. Běžné je řešit pouze velké víry například v RANS (*Reynolds averaged Navier – Stokes*) rovnicích či modelu LES (large – eddy simulations) a vliv zbylých vírů modelovat pomocí uzávěru. Právě zde lze aplikovat částečnou regresi například pro vyhodnocení Reynoldsových napětí v RANS [16][17] nebo pro návrh statistik malých měřítek v LES [18][19][20].

V následujících řádcích budou podrobněji rozebrány práce využívajících neuronové sítě v mechanice tekutin. První z nich řeší základní úloh, tedy proudění v kavitě, Hagenovo – Poiseuillovo a Couettovo proudění [11], zde jsou porovnány dvě různé formulace Navierových – Stokesových rovnic a výsledkem je vyhodnocení vhodnosti jejich využití ve strojovém učení, práce dále sleduje vliv kvality okrajových podmínek na učení a možnosti přeučení. V druhé práci autor nejprve zkoumá vliv omezené znalosti výsledku při učení a následně učí sítě predikovat parametricky popsaný letecký profil [9]. Poslední práce se také zabývá predikcí proudění, avšak je zde použita konvoluční neuronová síť parametrizující geometrii leteckého profilu a poté vícevrstvý perceptron, řešící proudění bez znalosti fyziky [10].

Autoři práce zabývající se rozdílnými formulacemi Navierových – Stokesových rovnic uvažují veškeré výpočty ve dvourozměrném prostoru, pro nestlačitelnou tekutinu a řešené proudění je ustálené, viskózní a laminární. První formulace je rychlostně – tlaková (VP – velocity-pressure), ve druhé formulaci je použit Cauchyho tenzor napětí (ST – Cauchy stress tensor). Referenční řešení představují výsledky numerických výpočtů. Nejprve autoři řešili vliv hloubky neuronové sítě, tedy

počtu skrytých vrstev a zároveň vliv počtu kolokačních a krajních bodů. V první části síť obsahovala fixně 4 skryté vrstvy o 40 neuronech a postupně se měnil poměr kolokačních bodů ku krajním na hodnoty 4:1, 8:1 a 16:1. Následně byl zafixován poměr bodů na 8:1, poté 16:1 a měnila se pouze hloubka sítě. Z výsledků je patrné, že je pro VP tvar obecně potřeba více iterací ke konvergenci, a tedy je potřeba delší výpočtový čas. Tento fakt je způsoben derivacemi vyšších řádů, které tato formulace obsahuje. Dále je patrné, že s rostoucí hloubkou a počtem kolokačních bodů roste také potřebný výpočetní čas. Pro proudění Hagena – Poiseuilla a Couetta dosahuje síť vycházející z ST formulace menší průměrné chyby, tedy je obecně méně citlivá na změny v parametrech sítě. Avšak pro řešení proudění v kavitě ST síť nebyla schopna najít správné řešení. Co se týče optimálních parametrů pro jednotlivá proudění, tak pro Hagenovo – Poiseuillovo je to ST formulace, hloubka 6 vrstev a poměr 8:1. Řešení Couettova proudění dosahuje nejlepších výsledků pro stejnou konfiguraci. Proudění v kavitě nejlépe řeší síť vycházející z VP formulace o 8 vrstvách a poměru bodů 8:1.

V další části byl sledován vliv chybových vah, optimalizujících váhy sítě, a vliv okrajových podmínek. Okrajové podmínky na vstupu byly buď definovány konstantně anebo obsahovaly šum, který byl definován pomocí Gaussova bílého šumu o amplitudě 10 % z konstanty. Chybové váhy byly modifikovány buď konstantně či pomocí adaptivní funkce. Dle očekávání sítě dosahovaly lepšího řešení při učení s konstantními okrajovými podmínkami. Avšak i když obsahovaly šum, stále byly schopny poměrně přesně predikovat výsledky. Použití adaptivních vah výrazně zvýšilo přesnost sítí s VP formulací při řešení problematiky Hagenova – Poiseuilleova a Couettova proudění. Naopak ST formulace si vedla lépe při konstantních vahách. Při řešení proudění v kavitě adaptivní váhy zhoršovaly výsledky. Autoři konstatují, že pro zvyšující se komplexitu problému, způsobenou přidáním adaptability, je použité nastavení nedostačující.

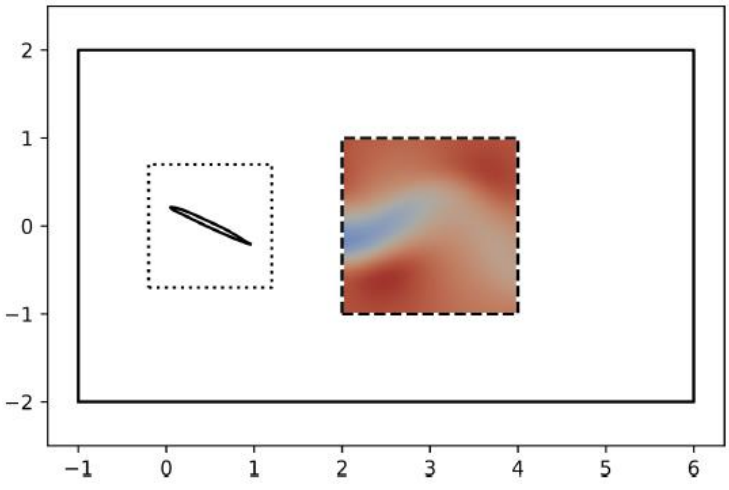
Poslední část se zaměřovala na přenos znalostí již naučené sítě na novou problematiku. Hlavní snahou zde je urychlit výpočet, když jsou k dispozici informace předchozích simulací. Využitím přenosu znalostí je možné přeučit stávající síť na daný problém místo kompletního tréninku sítě nové. Řešeny byly tři různé případy: změna okrajové podmínky, změna výpočtové domény a změna obou předchozích. V následující tabulce jsou shrnuty výsledky.

Tabulka 1 *Výsledky přenosu učení (T.L.), case 1 označuje změnu OP, case 2 změnu domény, case 3 obojí [11]*

Cases	T. L.	Error ($\times 10^{-2}$)		Iterations ($\times 10^{3}$)		Times (min.)	
		VP	ST	VP	ST	VP	ST
Case 1	-	2.01 ± 0.46	0.65 ± 0.46	16.82 ± 2.51	25.96 ± 5.69	15.22 ± 2.27	25.96 ± 5.69
	Yes	1.48 ± 0.35	1.34 ± 0.41	3.61 ± 1.38	2.6 ± 0.80	2.91 ± 1.11	1.10 ± 0.34
Case 2	-	3.33 ± 0.44	0.64 ± 0.38	17.69 ± 1.44	25.22 ± 2.71	16.01 ± 1.30	12.21 ± 1.77
	Yes	3.44 ± 0.55	3.08 ± 1.40	7.61 ± 1.03	5.78 ± 1.13	6.13 ± 0.83	2.46 ± 0.48
Case 3	-	17.12 ± 5.37	200.18 ± 4.76	15.98 ± 0.69	12.99 ± 0.58	28.12 ± 1.21	5.65 ± 0.25
	Yes	37.88 ± 8.94	204.74 ± 3.38	7.08 ± 1.63	4.14 ± 0.79	5.70 ± 1.31	1.77 ± 0.34

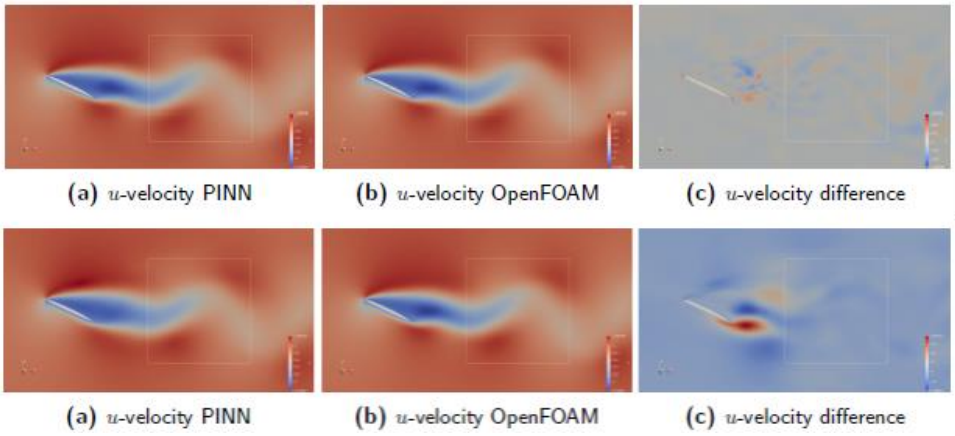
Z výsledků je patrné, že formulace SP dosahuje přesnějších výsledků pro první a druhý případ, avšak pro třetí výrazně exceluje VP forma. Při přenosu učení došlo k dramatickému poklesu počtu iterací a potřebného času k učení v porovnání s učením kompletně nové sítě. Pokud však srovnáme přesnost, tak sítě přeučené ji mají sníženou. Ke konci autoři vyzdvihují potenciál tohoto typu učení a možnosti budoucího výzkumu [11].

První práce popisující predikci proudění v okolí leteckého profilu zkoumá schopnost učení z omezené znalosti výsledku. K výpočtům využívá PINN, konkrétně síť o 9 vrstvách s 128 neurony v každé. Využita byla také postupná změna parametru rychlosti učení, který klesal každých 2000 iterací. Výpočtová doména byla v prostoru proložena náhodně generovanými kolokačními body, ve kterých byla hodnocena residua Navierových – Stokesových rovnic. Následující obrázek ukazuje výpočtovou doménu, kde je v okolí leteckého profilu vyznačena oblast se zvýšenou hustotou kolokačních bodů, která by měla vést ke zvýšení přesnosti. Také je zde barevně vyznačena oblast, ve které jsou známa data.



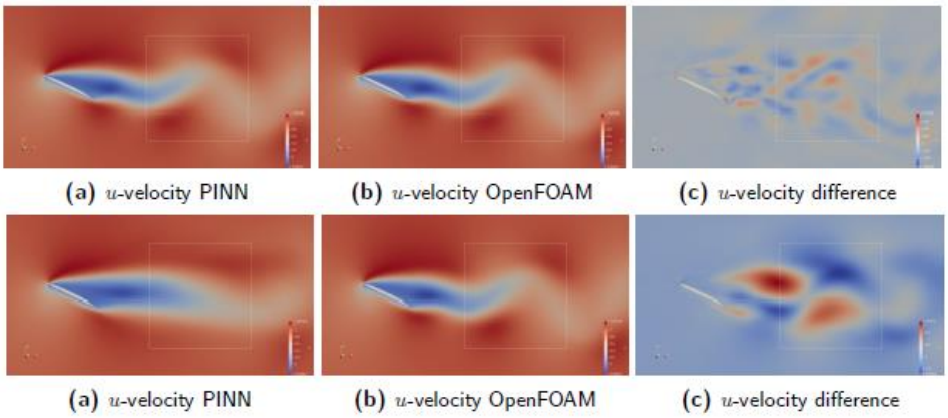
Obrázek 2.17 *Výpočtová doména neuronové sítě [9]*

Neuronová síť byla učena pro dvě různé sady dat. První obsahovala veškeré informace ve vyznačené oblasti. Druhá měla ve vyznačené oblasti k dispozici pouze informace o tlaku. Porovnání výsledků predikce sítě, vycházející z plné znalosti s výsledky z OpenFOAMu ukazuje dobrou shodu. Největší rozdíly jsou pozorovány v oblasti mezi profilem a vyznačenou oblastí (viz **Obrázek 2.18** Výsledky predikce první sítě se znalostí veškerých parametrů, první řádek představuje počáteční čas, druhý řádek čas t+3 s). Síť také vykazuje problémy při vypořádávání se s časem při kontinuálním výpočtu. Nejlepších odhadů je tedy dosahováno na počátku časové osy, kde je odhad nejblíže počátečním podmínkám.



Obrázek 2.18 *Výsledky predikce první sítě se znalostí veškerých parametrů, první řádek představuje počáteční čas, druhý řádek čas t+3 s [9]*

Porovnání výsledků druhé sítě, vycházející jen ze znalosti tlaku, ještě více podtrhuje neschopnost sítě vypořádat se s časem. Tato síť vykazuje pro hodnoty času dále od počátku výrazně vyšší nepřesnost (viz **Obrázek 2.19** Výsledky predikce druhé sítě se znalostí pouze tlaku, první řádek představuje počáteční čas, druhý řádek čas t+3 s).



Obrázek 2.19 *Výsledky predikce druhé sítě se znalostí pouze tlaku, první řádek představuje počáteční čas, druhý řádek čas t+3 s [9]*

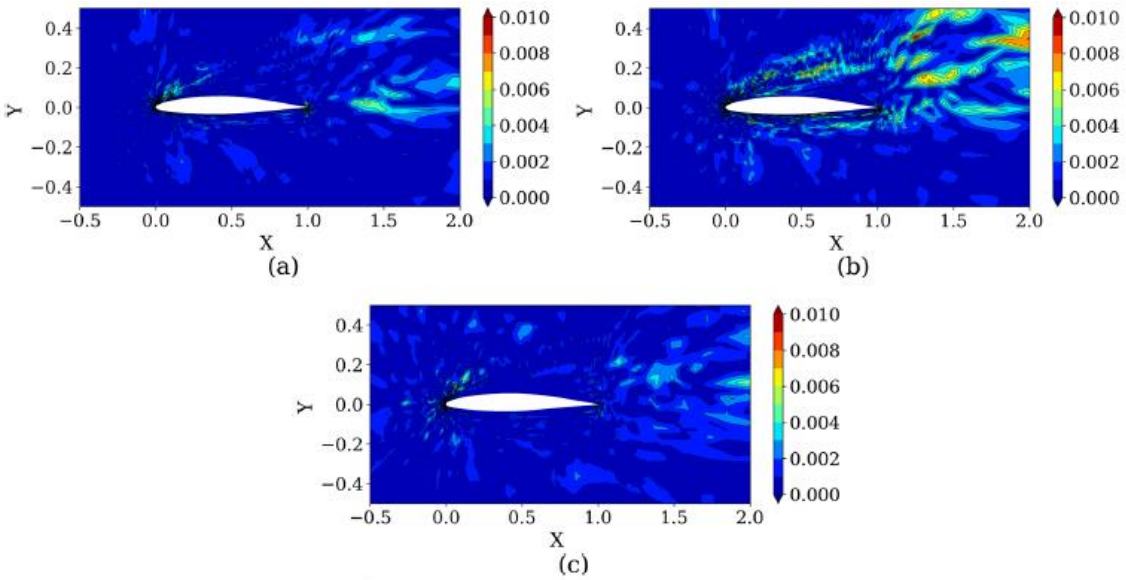
V dalších částech autor již vzhledem k výsledkům z předešlé části uvažuje ustálené proudění, tedy odstraňuje z predikce čas, který způsoboval sítím problémy. Řešeny jsou symetrické letecké profily NACA a nastavení PINN je až na pár detailů stejné jako v předchozí úloze. K učení je využita znalost celé výpočtové domény včetně okrajových podmínek založena na výsledcích numerických výpočtů. Trénovaná síť o 12 vrstvách s 64 neurony v každé byla učena po dobu 150 000 iterací. Z výsledků je patrné, že je natrénovaná PINN schopná přesně předpovědět řešení pro profily, které tloušťkou leží mezi dvěma tréninkovými vzorky. Přesnost je zde dokonce vyšší oproti původním numerickým simulacím. Naopak u vzorků, které jsou tlustší, či tenčí než dva tréninkové vzorky, má síť problémy s korektním naučením vztahů geometrie a proudového pole. Z těchto výsledků vyplývá, že síť dosahuje lepších výsledků při interpolaci než při extrapolaci.

Poslední článek řeší podobnou úlohu jako předešlá práce, ale používá jiné nástroje. Snahou autorů bylo nahradit konvenční CFD výpočet obtékání leteckého profilu, řešící Navierovy – Stokesovy rovnice na výpočetní síti s ohledem na okrajové podmínky, za účelem získání řešení toku proudění. Nahrazení spočívalo v aproximaci proudového pole funkcí geometrie profilu křídla, Reynoldsova čísla a úhlu náběhu za použití neuronových sítí bez jakéhokoliv řešení Navierových – Stokesových rovnic. Tento přístup se skládá ze dvou hlavních kroků. V prvním je použita konvoluční neuronová síť, která z leteckého profilu určí geometrické parametry. Ve druhém kroku jsou tyto parametry společně s Reynoldsovým číslem a úhlem náběhu použity jako vstup do další sítě, vícevrstvého perceptronu, který představuje přibližný model pro aproximaci pole proudění. Nutnost využití dvou sítí tkví v potřebě zpracovat obraz geometrie profilu. K tomuto úkolu je ideálním nástrojem právě konvoluční síť, která si narozdíl od vícevrstvého perceptronu umí dobře poradit s velkým množstvím vstupů, vycházejícího ze zpracování obrázků geometrie. Databáze

použitá k učení sítí byla vytvořena v programu OpenFOAM, který řeší Navierovy – Stokesovy rovnice.

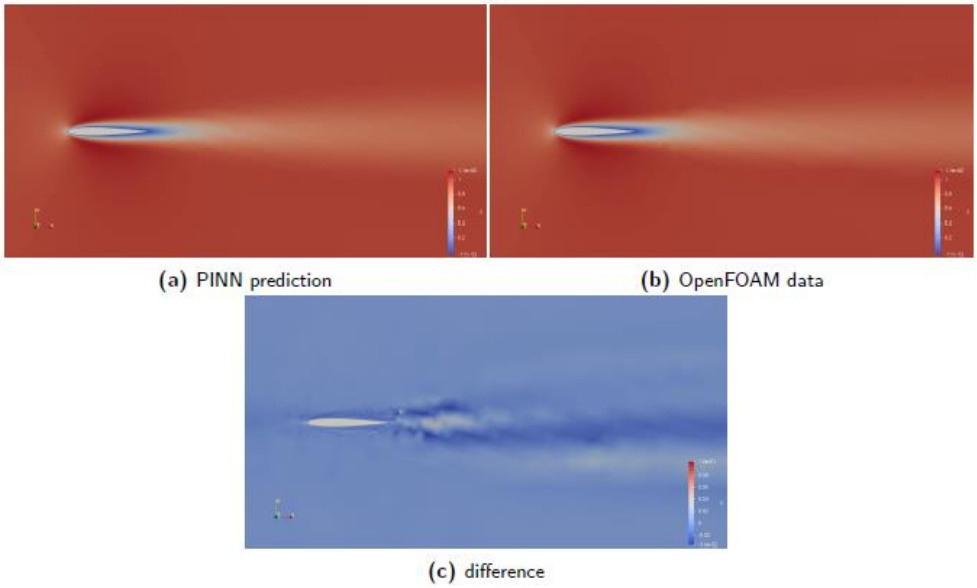
Testovány byly celkem tři různé perceptrony, každý měl 20 vstupních parametrů a 3 výstupy. Rozdíl byl pouze v počtech skrytých vrstev a v neuronech, které obsahovaly. První síť obsahovala 8 skrytých vrstev o 800 neuronech, druhá 10 vrstev o 1000 neuronech a třetí 12 vrstev s 1200 neurony. Veškeré vstupy byly normalizovány, aby měly hodnoty v rozmezí 0–1. Z celkového datasetu bylo použito 90 % k procesu učení a zbylých 10 % pro validaci modelu. Nejprve bylo provedeno několik testů, ze kterých při porovnání výsledků všech třech sítí vyplynulo, že druhá síť o rozměru 10x1000 dosahuje nejmenší chyby, a proto byla nadále použita jako hlavní.

Z celkových 13 profilů použitých k učení byl vybrán vzorek NACA 63415, reprezentující schopnost modelu predikovat tlakové a rychlostní pole. V následujících obrázcích jsou zobrazeny absolutní rozdíly mezi predikcí a CFD výpočtem [10].



Obrázek 2.20 *Srovnání absolutní chyby mezi CFD a predikcí. (a) tlak, (b) rychlost x, (c) rychlost y [10]*

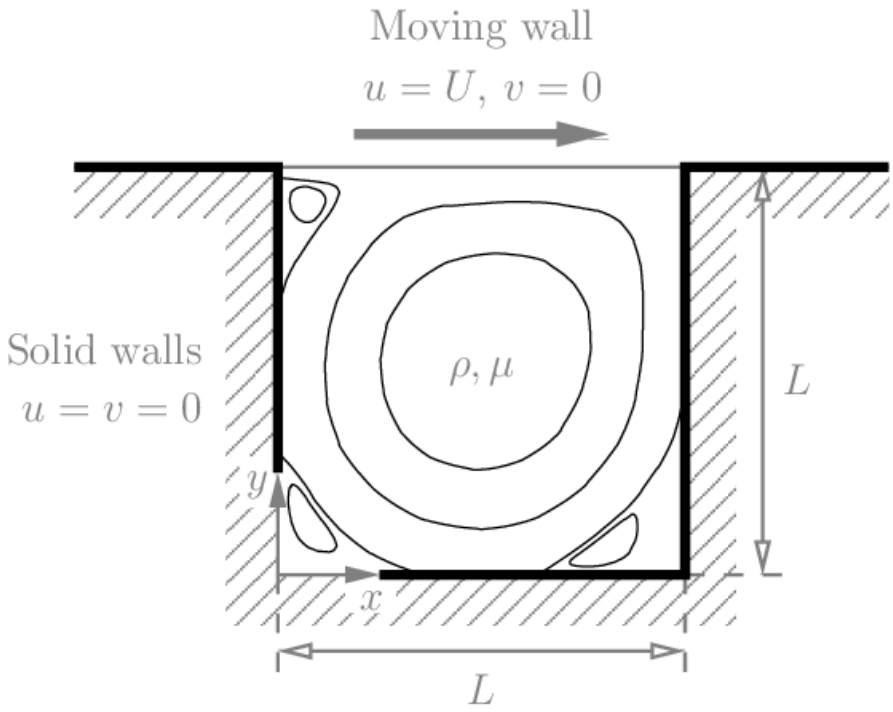
Celkový čas procesu učení obou sítí počítaných v článku [10] byl 1493,5 hodin, což ve srovnání s 6 hodinami potřebnými k naučení sítě [9] představuje obrovskou časovou ztrátu. Pro představu o přesnosti a srovnání výsledků perceptronu a PINN je přiložen následující obrázek s výsledky první práce.



Obrázek 2.21 *Výsledek predikce PINN a jeho srovnání s výsledkem numerického výpočtu [9]*

3 Proudění v kavitě

Proudění v dutině poháněné pohybem horní stěny (*lid-driven cavity)* je velmi dobře známý a řešený problém řešící viskózní nestlačitelné proudění. Často je používán jako benchmark numerických metod díky své jednoduchosti geometrie a složitosti chování tekutiny. Řešena je jednoduchá geometrie čtvercové oblasti (viz **Obrázek 3.1**). Horní stěna se pohybuje konstantní rychlostí a ostatní stěny jsou pevně zafixovány. V důsledku pohybu horní stěny vzniká v dutině uprostřed velký vír, v dolních rozích menší víry a eventuelně vzniká další vír v levém horním rohu. Rozměry a přesné umístění vírů závisí na hodnotě Reynoldsova čísla.



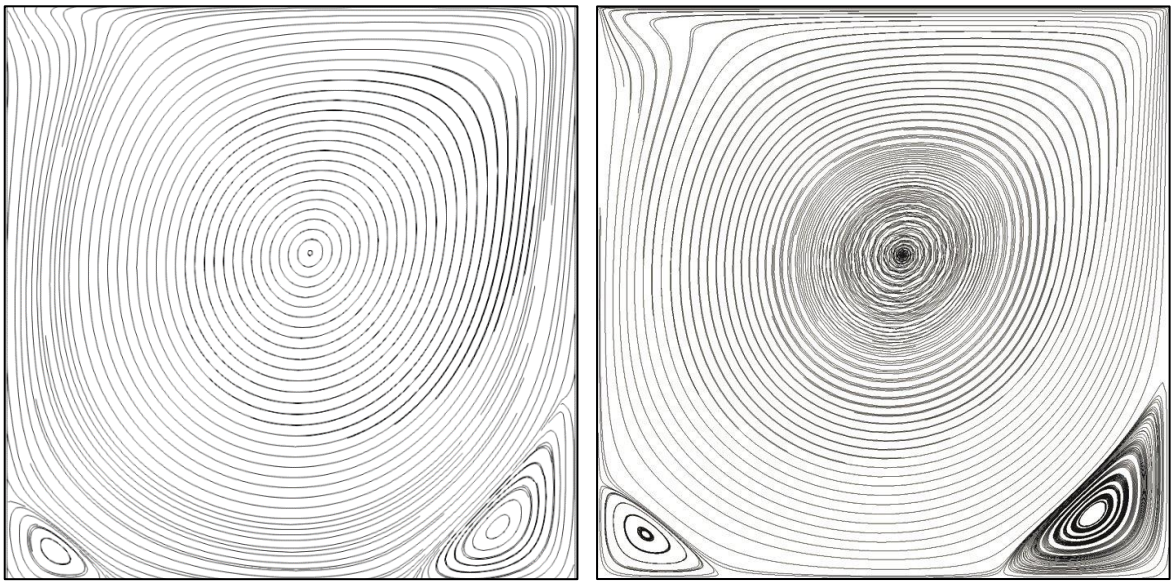
Obrázek 3.1 *Schéma proudění v kavitě [23]*

3.1 Numerické výpočty

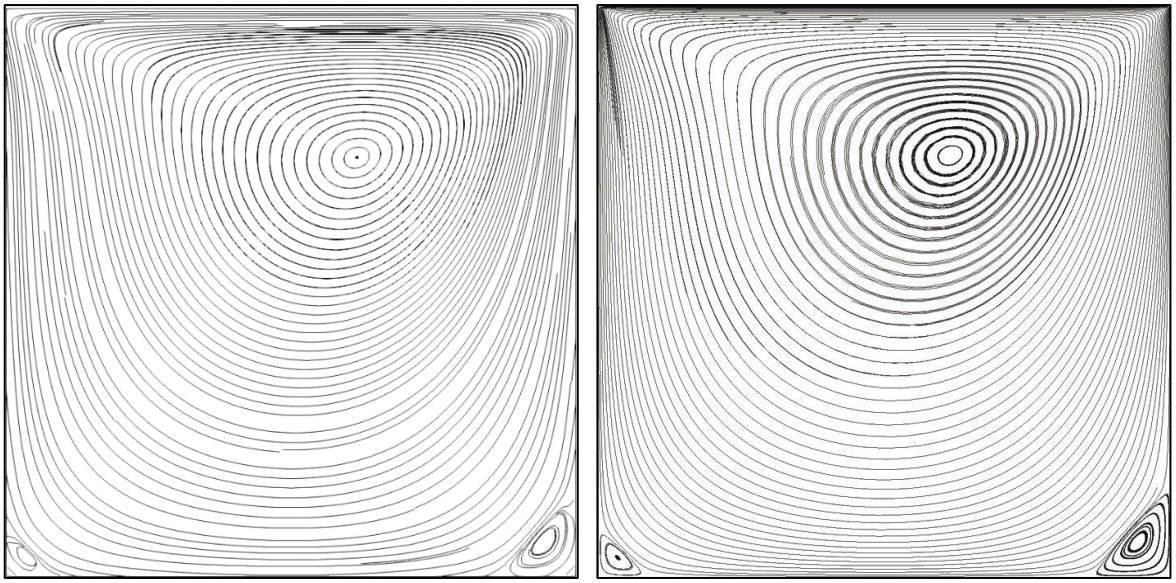
Nejprve je třeba definovat velikost kavity a okrajové podmínky. V řešeném případu bude mít dutina rozměry 1x1 metr. Víko se bude pohybovat rychlostí *u = 1* m/s. Proudění bude řešeno jako stacionární. Protože je cílem vytvořit neuronovou síť, která při svém odhadu výsledku bude zohledňovat hodnoty Reynoldsova čísla, je třeba si stanovit, v jakém rozmezí se jeho hodnoty budou pohybovat, aby bylo proudění stále stacionární. Z dostupných článků plyne, že nestacionarita proudění se začíná objevovat mezi hodnotami 5000 – 10000, kde jde o periodické chování, které později pro vyšší Reynoldsova čísla přechází v chování chaotické[21][22]. Proto byly zvoleny

hodnoty Reynoldsova čísla v rozmezí 100 – 1100. Při konstantní rychlost víka dochází při postupném nárůstu Reynoldsova čísla k snižování kinematické viskozity. To má postupně za následek vznik a zvětšování vírů ve spodních rozích (viz **Obrázek 3.2**, **Obrázek 3.3**).

Vstupní data byla získána numerickými výpočty v programu OpenFOAM. Jako řešič byl použit icoFOAM. Dutina byla aproximována čtvercovou sítí se stranou o rozměru 50 buněk. Okrajové podmínky byly nastaveny, dle již zmíněných parametrů. Rychlost vrchní stěny byla 1 m/s a ostatní stěny dostaly podmínku noSlip. Hodnota kinematické viskozity μ byla vždy dopočtena z požadovaného Reynoldsova čísla. Po provedení výpočtů byly výsledky kvůli validaci porovnány s daty dostupnými na internetu. Částečné srovnání je zobrazeno na následujících dvou obrázcích.



Obrázek 3.2 *Srovnání výsledků numerických výpočtů (vlevo) s referenčním řešením [24] (vpravo) pro Re = 1000*



Obrázek 3.3 *Srovnání výsledků numerických výpočtů (vlevo) s referenčním řešením [24] (vpravo) pro Re = 100*

3.2 Neuronová síť

V dalším kroku bylo třeba si připravit neuronovou síť (viz **Obrázek 3.4**) schopnou řešit danou problematiku. Po mnoha pokusech s různými parametry učení a rozměry sítě bylo zvoleno 8 vrstev. Skryté vrstvy obsahovaly 350 neuronů každá. Menší množství neuronů či vrstev vedlo buď k pomalejší konvergenci nebo byl proces učení nestabilní. V důsledku nestability učení bylo třeba proces opakovat a doufat v pozitivní výstup, tedy správně naučenou síť. Výsledná síť konverguje rychleji a není třeba proces učení opakovat. První, vstupní vrstva je nastavena na 3 vstupní hodnoty, a to souřadnice *x, y* a Reynoldsovo číslo *Re*. Výstup tvoří hodnota ψ představující proudovou funkci a tlak *p.* Zvolená formulace pomocí proudové funkce se jevila být výhodnější, protože vedla k menšímu počtu výstupních hodnot. Jako aktivační funkce je zvolena *tanh* neboli hyperbolická tangenciální funkce. Její výhoda spočívá v tom, že výsledek normuje do intervalu <-1,1> a také je nelineární, díky čemuž se síť zvládne přizpůsobit i komplexnějším vzorům. Následně byl zvolen optimalizační algoritmus ADAM [25].

```python
# Define Network
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.hidden1 = nn.Linear(3, 350)     # input layer (x, y, Re)
        torch.nn.init.xavier_normal_(self.hidden1.weight)
        self.hidden2 = nn.Linear(350, 350)    # hidden layer
        torch.nn.init.xavier_normal_(self.hidden2.weight)
        self.hidden3 = nn.Linear(350, 350)    # hidden layer
        torch.nn.init.xavier_normal_(self.hidden3.weight)
        self.hidden4 = nn.Linear(350, 350)    # hidden layer
        torch.nn.init.xavier_normal_(self.hidden4.weight)
        self.hidden5 = nn.Linear(350, 350)    # hidden layer
        torch.nn.init.xavier_normal_(self.hidden5.weight)
        self.hidden6 = nn.Linear(350, 350)    # hidden layer
        torch.nn.init.xavier_normal_(self.hidden6.weight)
        self.hidden7 = nn.Linear(350, 350)    # hidden layer
        torch.nn.init.xavier_normal_(self.hidden7.weight)
        self.output = nn.Linear(350, 2)      # output layer (psi, p)
        torch.nn.init.xavier_normal_(self.output.weight)

    def forward(self, x_y_Re):
        hidden1_out = torch.tanh(self.hidden1(x_y_Re))
        hidden2_out = torch.tanh(self.hidden2(hidden1_out))
        hidden3_out = torch.tanh(self.hidden3(hidden2_out))
        hidden4_out = torch.tanh(self.hidden4(hidden3_out))
        hidden5_out = torch.tanh(self.hidden5(hidden4_out))
        hidden6_out = torch.tanh(self.hidden6(hidden5_out))
        hidden7_out = torch.tanh(self.hidden7(hidden6_out))
        psi, p = torch.split(self.output(hidden7_out), 1, dim=1) # split output into two tensors
        p = p # linear activation
        psi = torch.sigmoid(psi) # sigmoid activation
        return psi, p

net = Net().to(device)
optimizer = optim.Adam(net.parameters(), lr=0.000099)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=300, gamma=0.1)
```

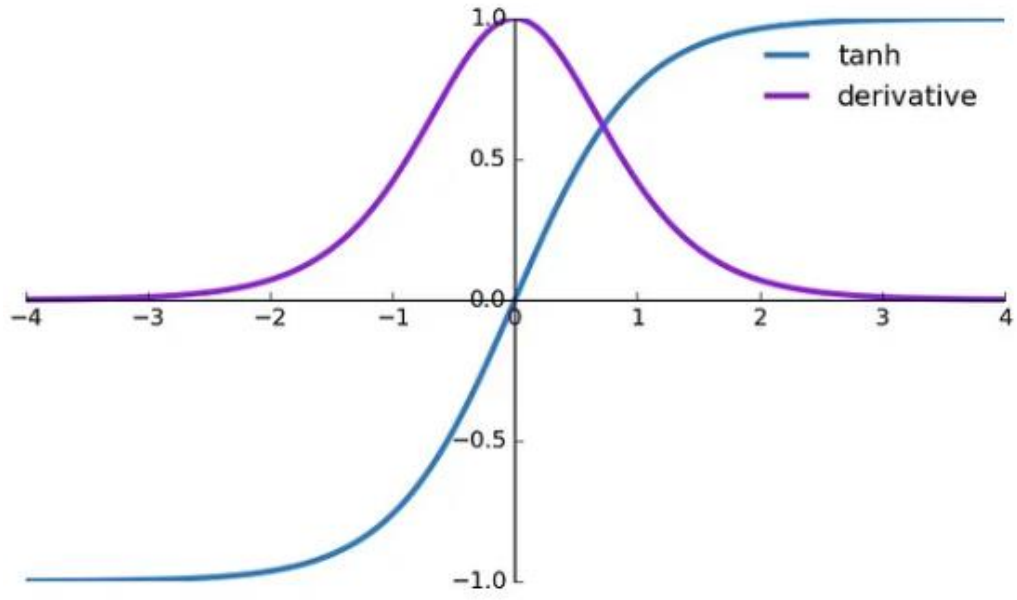
Obrázek 3.4 *Definice neuronové sítě, optimalizačního algoritmu a scheduleru*

Dále bylo třeba správně definovat chybovou funkci a zjistit, jaký mají jednotlivé inkrementy vliv. Jinými slovy bylo třeba zjistit, zdali nemá některá z dílčích chyb výrazně vyšší hodnoty než ostatní a tím pádem i vyšší vliv na učení. Pokud by například hodnota dílčí chyba počítaná z nesplnění okrajových podmínek byla 1000krát větší než ostatní, bylo by možné, že by se síť učila převážně, jak vyhovět okrajovým podmínkám, a ne jak správně odhadnout i zbytek oblasti. Naopak pokud by chyba od okrajových podmínek byla 1000krát menší, síť by mohla správně odhadovat střed oblasti a na okrajích by hodnoty nemusely dávat smysl. Na základě výsledků několika zkušebních testů, kdy byly pozorovány hodnoty všech dílčích funkcí, byly upraveny hodnoty chyb tak, že byl zvýšen vliv rychlostních dat získaných programem OpenFOAM na pětinásobek a vliv okrajových podmínek snížen na čtvrtinu. Díky snížení hodnoty chyby od okrajových podmínek je síť schopná lépe vystihnout i zbytek oblasti, pokud by k tomuto snížení nedošlo, síti by trvalo výrazně déle vystihnout správně celou oblast, pokud by toho byla vůbec schopná. Výsledkem již zmíněných testů byla také počáteční hodnota rychlosti učení 0,000099, ta na začátku učení umožňuje síti rychle konvergovat ke správnému řešení. Pro příliš vysokou hodnotu by síť při hledání minima funkce nemusela být schopná správné minimum najít, mohla by ho přeskočit či najít jiné lokální, suboptimální minimum.

Příliš malá hodnota rychlosti učení by způsobila příliš pomalou konvergenci a výpočet by trval dlouho.

Poté, co síť začne směřovat ke správnému minimu, je potřeba hodnotu rychlosti učení postupně snižovat a tím zvýšit přesnost, se kterou se minimu přiblíží. Proto je v kódu také definován *scheduler*, který snižuje hodnotu rychlosti učení po každých 300 iteracích na desetinu. Neméně důležitá byla také úprava chybové funkce. Pro prvních 250 iterací se síť učila pouze ze znalosti dat získaných numerickými výpočty. Chyby z okrajových podmínek, Navierových - Stokesových rovnic a rovnice kontinuity byly přidány později. Důvodem bylo nasměrování sítě pomocí dat správným směrem a následné využití rovnic k zvýšení přesnosti a urychlení optimalizace.

Protože síť zpočátku konvergovala pouze k nulovému řešení a nebyla schopna se od něho oprostit, bylo potřeba implementovat inicializační schéma. Použita byla Xavier (Glorot) inicializace [26], která řeší problematiku sítě, která se „neučí‘‘ a konverguje k 0, tím že naruší symetrii počátečních vah, které jsou jinak pro všechny neurony stejné. Toto schéma také napomáhá rychlejšímu tréninku a zároveň předchází problému tzv. mizejícího gradientu (*vanishing gradient*), který u aktivační funkce *tanh* může nastávat. Protože *tanh* normuje veškeré vstupy do intervalu < - 1, 1 >, derivace velmi velkých či velmi malých hodnot se během zpětného šíření (*backpropagation*) mohou rychle blížit 0 a tím prakticky vymizet (viz **Obrázek 3.5**). Xavier inicializace vyrovnává odchylky mezi vstupy a výstupy jednotlivých neuronů, a tím udržuje nenulové gradienty napříč vrstvami.

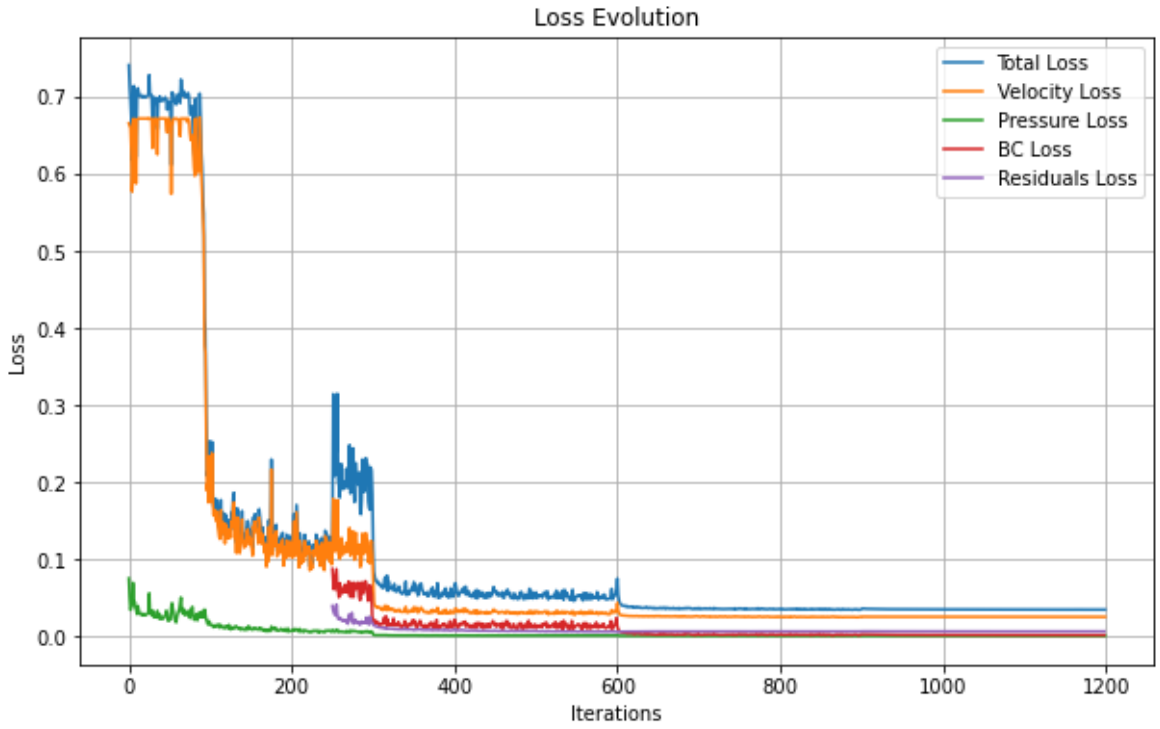


Obrázek 3.5 *Aktivační funkce tanh a její derivace [27]*

Je třeba zmínit, že obecně je proces optimalizace parametrů učení, jak hodnoty rychlosti učení, změny rychlosti učení a chybové funkce tak i množství vrstev a neuronů v nich, velmi obtížný a v některých případech je třeba se spolehnout na vlastní odhad či na metodu pokus omyl. Ne vždy se dá přesně dopředu určit, jaký bude výstup, jak rychle či zdali vůbec bude síť konvergovat ke správnému výsledku.

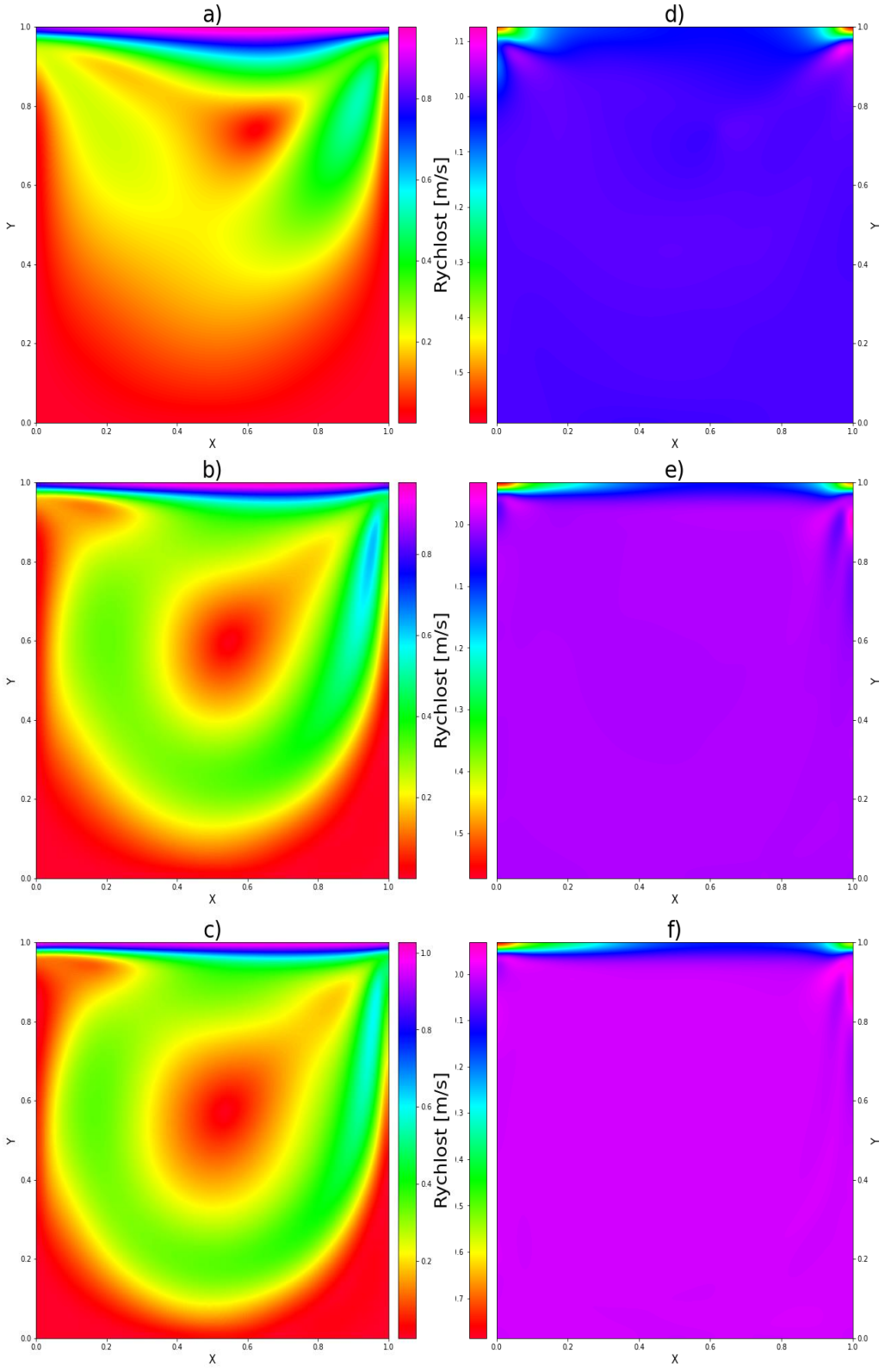
Celý proces učení proběhl v hlavním cyklu o 1200 iteracích. V rámci jedné iterace síť postupně prošla Reynoldsova čísla od 100 do 900 s krokem 4, učila se tedy na Reynoldsových číslech 100, 104, 108…900. Pro každé z nich byl spočten její odhad na síti o rozměru 50x50 buněk a na jeho základě i hodnota chybové funkce. Díky ní si mohla síť následně za pomoci *backpropagation* algoritmu spočítat gradienty chybové funkce a ty využít s optimalizačním algoritmem ADAM k úpravě svých vah. Zbylé výsledky spočtené pro Reynoldsova čísla 902 – 1100 budou použity při zkoumání sítě v režimu extrapolace.

Na následujících obrazcích jsou zobrazeny výsledné predikce a graf vývoje chybové funkce v průběhu iterací. Z grafu (**Obrázek 3.6**) je patrné, že prvních přibližně 100 iterací síť hledala správný „směr‘‘ a hodnota chyby stagnovala. Poté je patrný výrazný pokles kolem 100. iterace, zde byl pravděpodobně zachycen hlavní vír uprostřed kavity a společně s ním průběh tlaku, jak je patrné z poklesu obou chyb. Další výrazná změna nastala při 250. iteraci, kdy byly přidány chyby způsobené nesplněním okrajových podmínek a řídících rovnic. Dále jsou patrné skoky každých 300 iterací, kdy docházelo ke snížení hodnoty rychlosti učení. Na první pohled by se mohlo zdát, že v úsecích mezi těmito skoky nedocházelo k významným poklesům, avšak i přes to, že hodnota chyby neklesala, síť pravděpodobně zlepšovala svůj odhad.

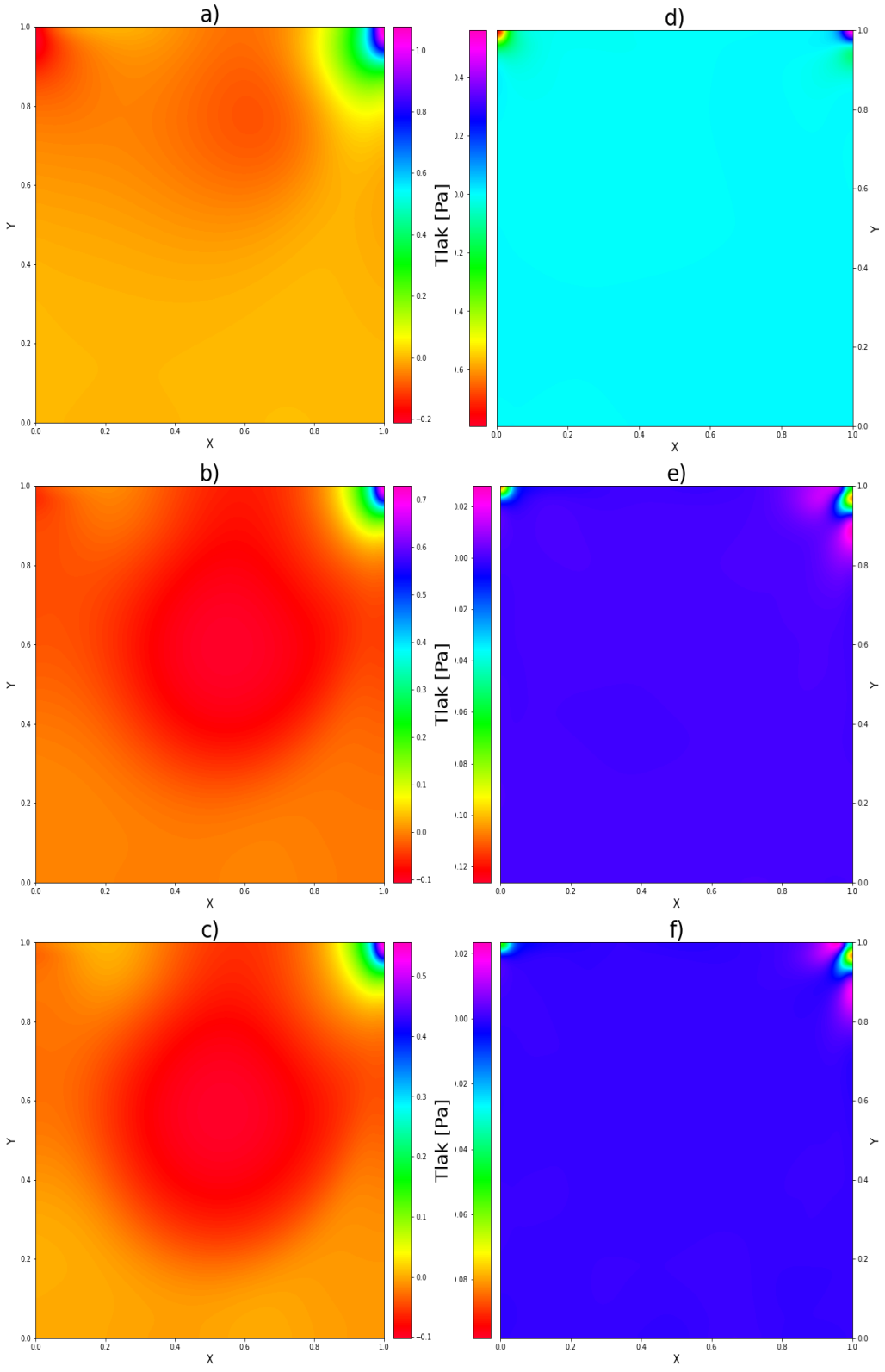


Obrázek 3.6 *Vývoj chyby v průběhu iterací*

Z výsledných predikcí je patrné, že největší nedostatky rychlostního pole jsou v prvních dvou řádcích buněk, zde je největším problémem správně vystihnout rychlost víka. Řešením tohoto problému by mohlo být výrazné zjemnění sítě. To by ale mělo za následek ohromné zvýšení výpočetní náročnosti a tomu odpovídající nárůst výpočetního času. Zároveň je patrný nedostatek v širší oblasti horních rohů, a to především při nízkých Reynoldsových číslech. U hodnoty Reynoldsova čísla 140 tato chyba začíná mizet a pro konečné hodnoty zcela vymizí (viz **Obrázek 3.7**). Predikce tlaku ukazují podobný fenomén, a to vyšší odchylky v horních rozích, především pak v pravém horním, kde je tlak nejvyšší (viz **Obrázek 3.8**).



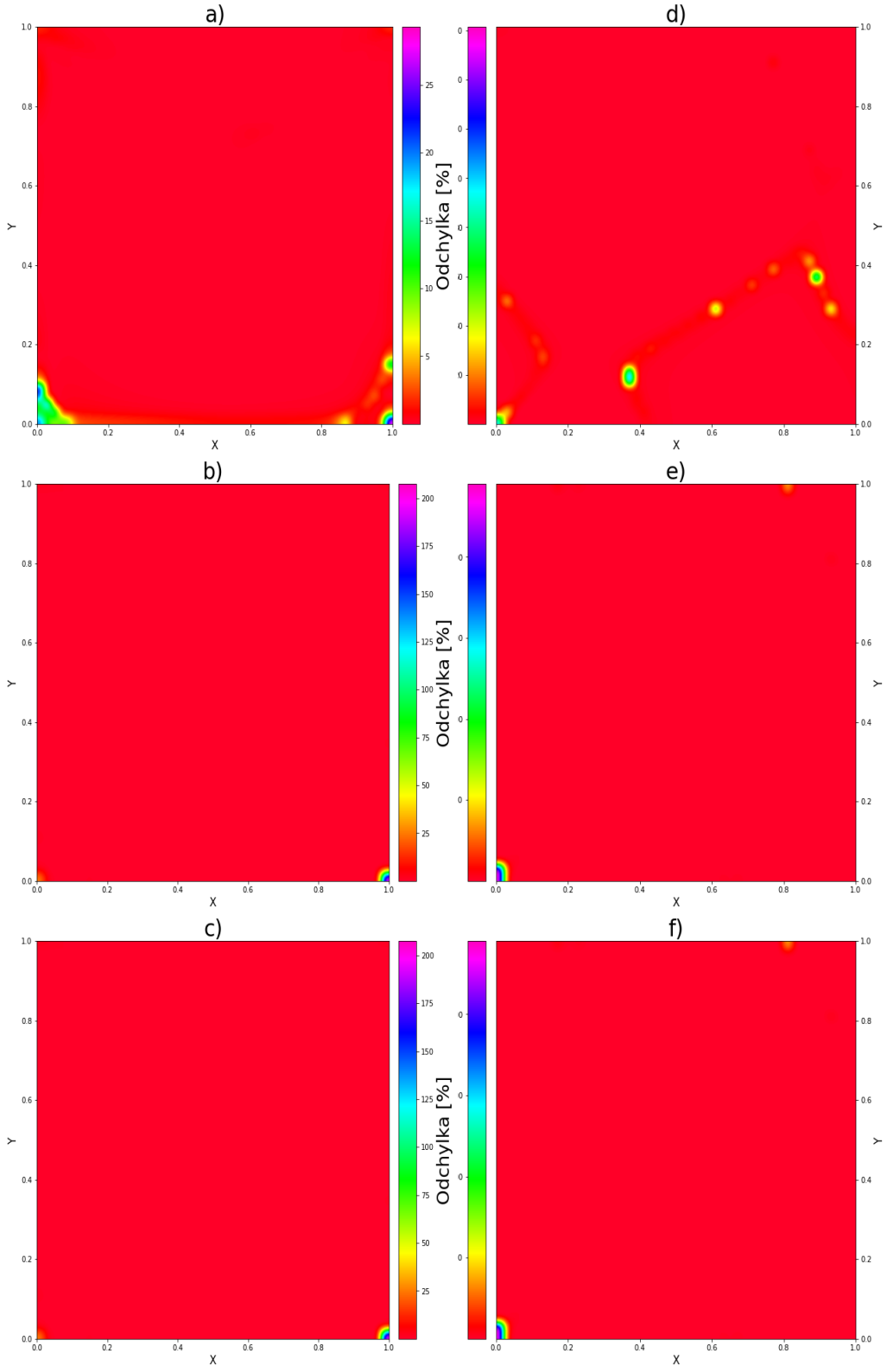
Obrázek 3.7 *Výsledná predikce rychlostního pole pro a) Re 100, b) Re 500, c) Re 900 a absolutní odchylka dané predikce od numerických dat d) Re 100, e) Re 500, f) Re 900*



Obrázek 3.8 *Výsledná predikce tlaku pro a) Re 100, b) Re 500, c) Re 900, a absolutní odchylka dané predikce od numerických dat d) Re 100, e) Re 500, f) Re 900*

Pro podrobnější pohled na schopnost sítě predikovat je vhodné spočítat také relativní odchylku odhadu. Z ní je patrné, že i rychlosti v oblasti spodních rohů dělají síti určité problémy, především pak v pravém rohu a nejvíce pro nižší Reynoldsova čísla, jak je patrné na dalším obrázku (viz **Obrázek 3.9**). Tato chyba přetrvává přibližně do hodnot Re 200 – 220, kde se predikce začínají zpřesňovat. V samotných dolních rozích však chyba zůstává a dochází k nárůstu relativní chyby rychlosti do obrovských hodnot. Přičemž v pravém dolním rohu je odchylka výrazně vyšší. Při pohledu na stupnici vždy maximální hodnota reprezentuje chyby v jednom z dolních rohů. Tento extrém může výrazně ovlivnit a zkreslit průměrnou hodnotu odchylky. Při jeho zanedbání je průměrná relativní odchylka rychlosti rovna 4,613 %. Tato hodnota je především ovlivněna nepřesnými odhady v blízkosti stěn, které sice nerostou tolik jako v rozích, ale nejsou ani srovnatelně přesné jako odhady ve středu oblasti. Pokud bychom tedy originální síť o rozměrech 50x50 zmenšili na rozměry 44x44 odebráním 3 krajních vrstev u všech stěn, velikost průměrné relativní odchylky rychlosti by klesla na hodnotu 0,904 %. Tento fakt jen potvrzuje tvrzení o přesnosti ve středu uvažované oblasti.

Predikce tlaku se téměř všude přibližuje numerickým hodnotám, s výjimkou několika vždy náhodně rozložených bodů (viz **Obrázek 3.9**), které obsahují velkou relativní odchylku. Tento fenomén se zdá být výraznější pro nižší Reynoldsova čísla a s jejich nárůstem téměř vymizí s výjimkou levého dolního rohu oblasti. Při zanedbání těchto extrémů je průměrná relativní odchylka tlaku rovna 12,11 %. Přesnost predikce se opět ve středu oblasti zvyšuje, na zmenšené síti 44x44 průměrná relativní odchylka nabývá hodnoty 6,587 %, což jen podtrhuje fakt, že síť lépe vystihuje střední část kavity.



Obrázek 3.9 *Relativní odchylka predikce od numerických výsledků rychlosti pro a) Re 100, b) Re 500, c) Re 900 a tlaku pro d) Re 100, e) Re 500, f) Re 900*

3.2.1 Validace

Validace je důležitou součástí procesu posuzování schopnosti modelu vypořádat se s neznámými vstupy. Jedná se o formu interpolace, kdy jsou modelu předložena Reynoldsova čísla v rozmezí 102 – 898 s krokem 4, tedy hodnoty 102, 106, 110…898. Jsou to tedy hodnoty z rozsahu, na kterém se síť učila, avšak jsou to konkrétní hodnoty, které nikdy předtím neviděla. Požadovaná predikce je opět na stejné síti o rozměru 50x50 buněk. Tato kontrola je klíčová, protože model, který se výborně naučí pouze tréninková data a není schopný předpovídat cokoliv jiného, není použitelný. Tento problém se nazývá *overfitting.* V ideálním případě by měl model předpovídat hodnoty se stejnou přesností pro tréninkový i validační dataset.

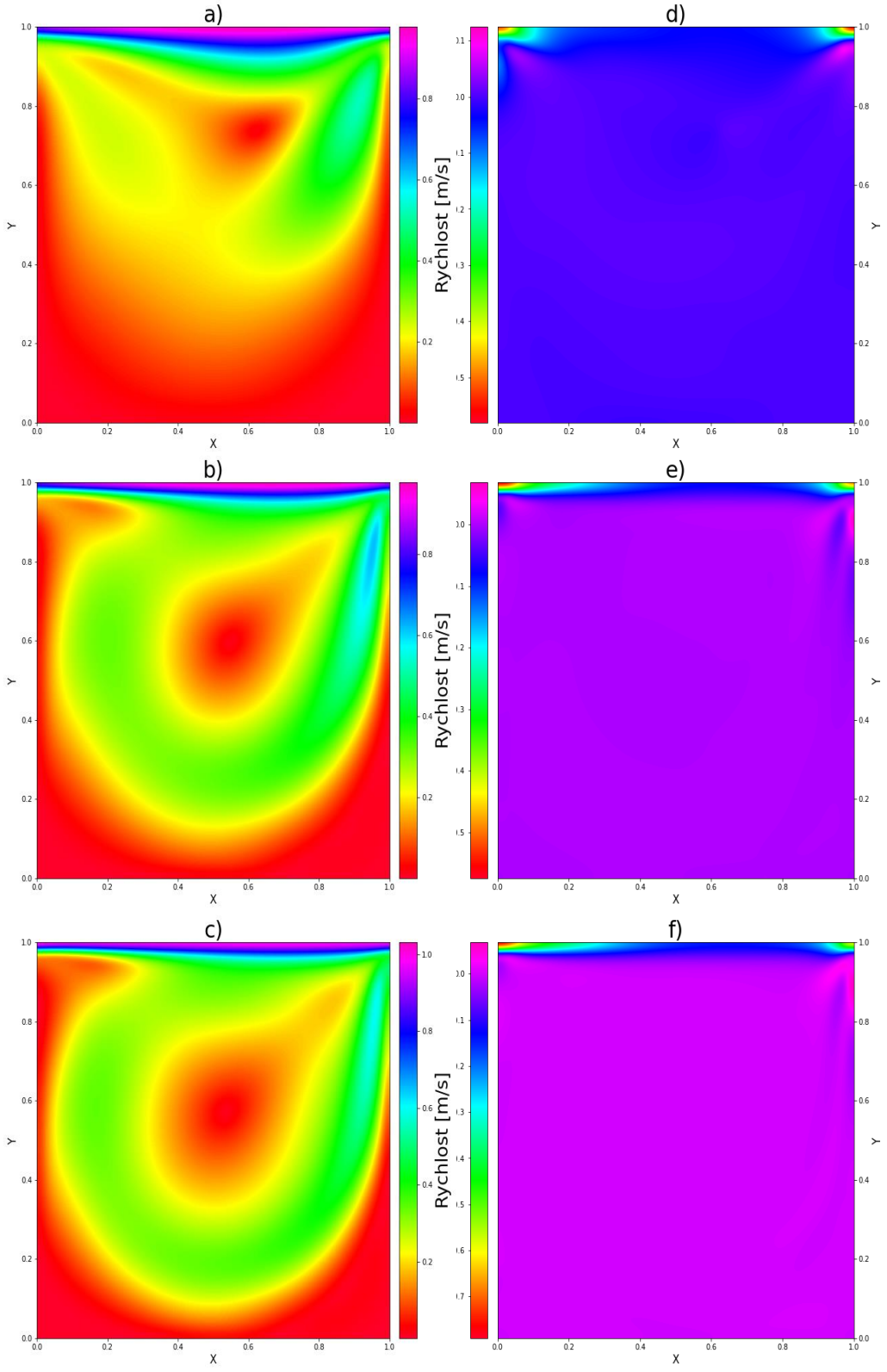
Jak je z následujících obrázků patrné (viz **Obrázek 3.10**, **Obrázek 3.11**), síť predikuje velmi srovnatelně pro nová i tréninková data. Vyskytují se stejné chyby u predikce rychlost v horních rozích a stejně také začínají mizet pro Reynoldsovo číslo 140.

Toto tvrzení potvrzuje i průměrná hodnota relativní odchylky rychlosti bez extrémů o hodnotě 4,566 %, která je téměř shodná s hodnotou při tréninku. Předpověď tlaku je také velmi blízká tréninku, její průměrná hodnota relativní odchylky bez extrémů je 12,91 %. Pokud bychom se opět z původní sítě 50x50 odstraněním 3 krajních sloupců a řádků omezili na síť 44x44, hodnoty průměrných relativních odchylek opět klesnou. Pro rychlost má tato odchylka hodnotu 0,897 % a pro tlak 6,531 %. Zde se znovu projevuje snížená schopnost sítě přesně predikovat v blízkosti stěn. Porovnání hodnot odchylek je zobrazeno v následující tabulce. Z ní je také patrné, jak mohou obrovské extrémy na jednom místě ovlivnit celkovou hodnotu. Predikce tlaku se zdají být zcela nepřesné, avšak po odstranění pár extrémů je jasné, že ve většině oblasti přesné jsou.

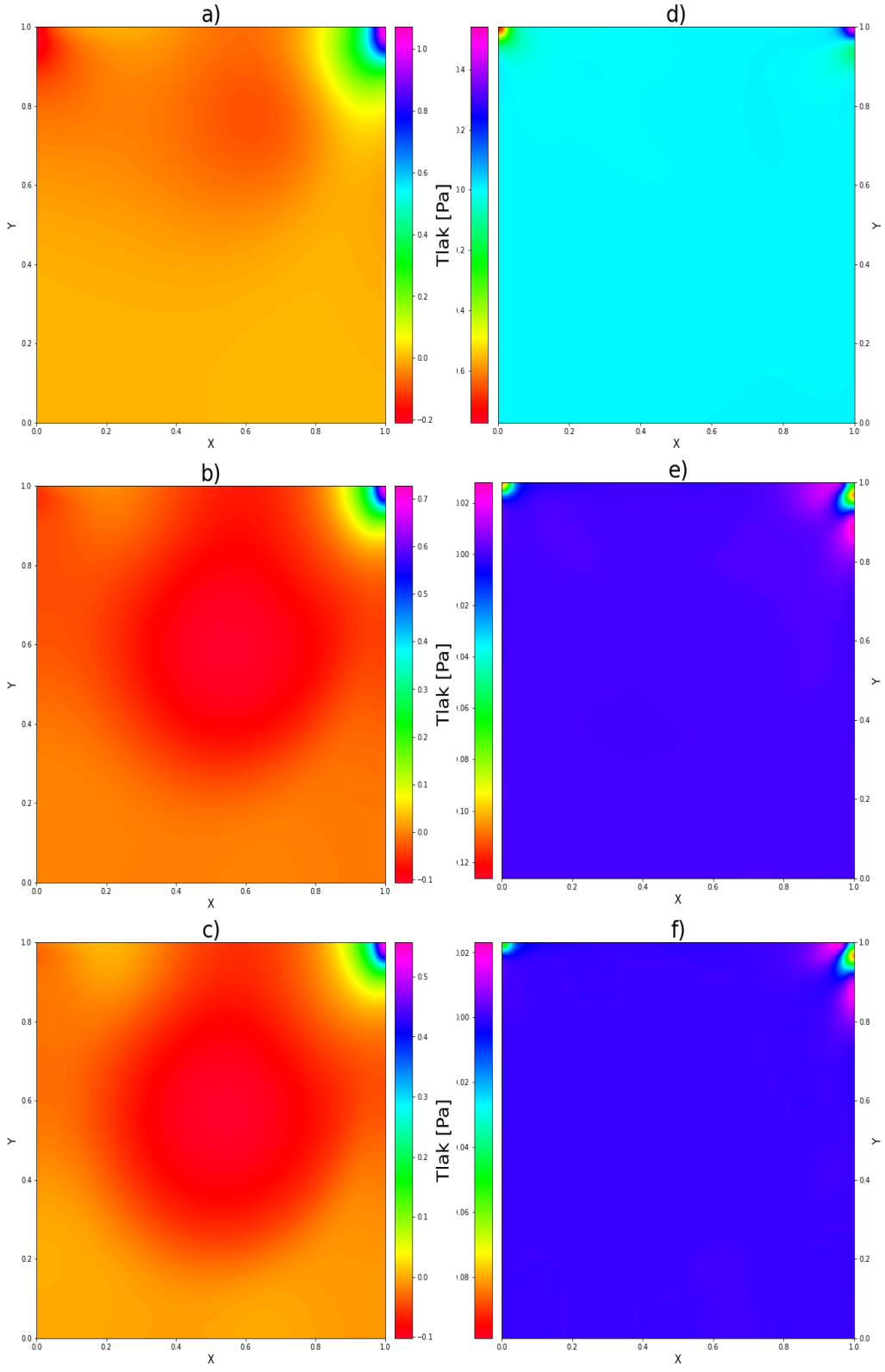
Tabulka 2 *Průměrné relativní odchylky predikce sítě od numerických dat pro trénink a validaci*

		Bez úpravy		Odstraněny extrémy		Síť 44x44	
		Trénink	Validace	Trénink	Validace	Trénink	Validace
Relativní odchylka [%]	Rychlost	19,18	19,15	4,613	4,566	0,904	0,897
	Tlak	182,5	87,12	12,11	12,91	6,587	6,531

Celkově lze na základě výše zmíněných srovnání říci, že nedošlo k problému *overfittingu.* Výsledná síť úspěšně prošla validací a je tedy schopná korektně a srovnatelně přesně interpolovat data, která nikdy neviděla.



Obrázek 3.10 *Predikce rychlostního pole validace pro Reynoldsova čísla a) Re 102, b) Re 502, c) Re 898 a absolutní odchylka dané predikce od numerických dat d) Re 102, e) Re 502, f) Re 898*



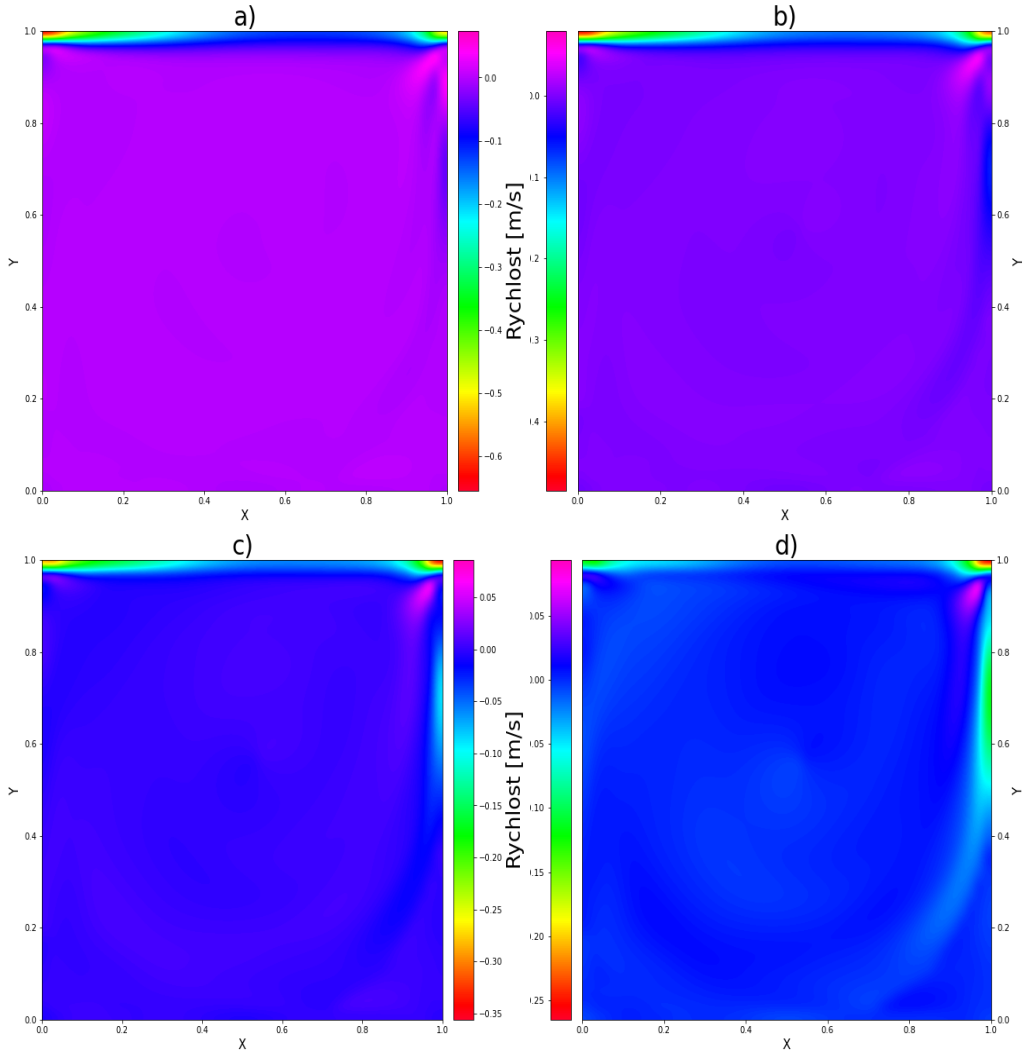
Obrázek 3.11 *Predikce tlaku validačního datasetu pro Reynoldsova čísla a) Re 102, b) Re 502, c) Re 898, a absolutní odchylka dané predikce od numerických dat d) Re 102, e) Re 502, f) Re 898*

3.2.2 Extrapolace

Z minulé kapitoly vyplývá, že je naučená síť schopná poměrně přesně predikovat výsledky z rozsahu tréninkového datasetu. Extrapolace představuje výrazně náročnější úkol, kdy je po modelu vyžadováno provádět předpovědi pro Reynoldsova čísla mimo tréninkový dataset. Testována je tedy schopnost učená data nejen interpolovat, ale i zobecnit. Zde může síti pomoci právě znalost fyzikálních rovnic popisujících proudění v kavitě nebo znalost okrajových podmínek. Řídící rovnice totiž nejsou omezeny jen na určitý úsek, ale popisují danou problematiku v celém rozsahu.

Do sítě jsou jako vstup vložena Reynoldsova čísla v rozsahu 902 – 1100 s krokem 2, jsou to tedy hodnoty 902, 904, 906…1100. Predikce je opět požadována na síti 50x50 buněk. Predikce pro nejnižší Reynoldsova čísla mají podobný charakter jako výsledky interpolace, proto zde budou zobrazeny obrázky výsledků pouze pro hodnoty Re 950, 1000, 1050 a 1100. Z nich je nejlépe vidět měnící se charakter predikce modelu.

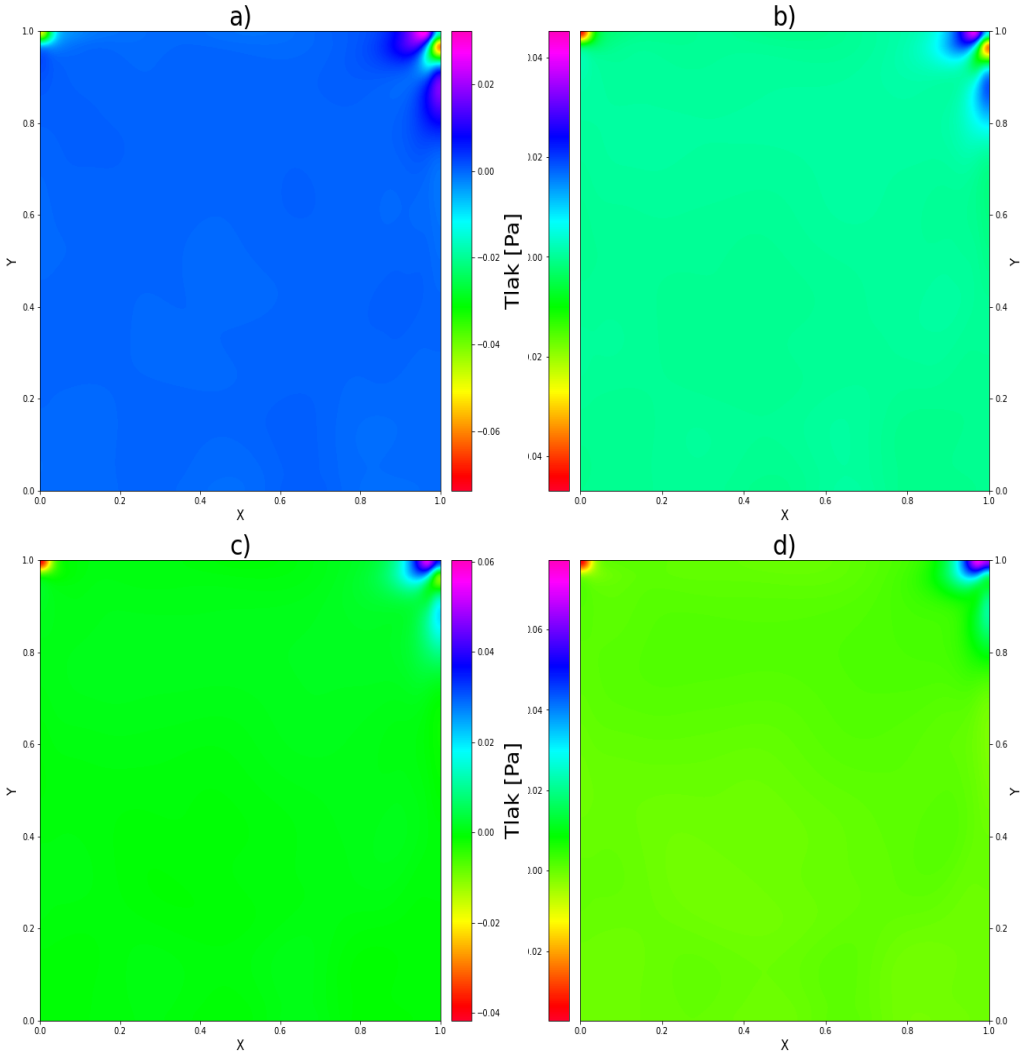
Při pohledu na absolutní odchylky predikce rychlosti (viz **Obrázek 3.12**) je vidno, jak se se zvyšující vzdáleností od tréninkového datasetu zhoršuje predikce. Největší chyba se začíná vyskytovat na pravé straně hlavního víru a postupně se šíří. Pro hodnotu Reynoldsova čísla 1100 chyba začíná prorůstat až do pravého dolního rohu. Zároveň je patrný nárůst nepřesnosti ve středu hlavního víru a na jeho okrajích. Dalo by se očekávat, že s dalším nárůstem vzdálenosti od tréninkových dat chyba nadále poroste, až do bodu, kdy bude predikce zcela špatně a nepřesná.



Obrázek 3.12 *Absolutní odchylka predikce rychlostního pole extrapolací od numerických výpočtů pro Reynoldsova čísla a) Re 950, b) Re 1000, c) Re 1050, d) Re 1100*

Při pohledu na absolutní chybu predikce tlaku (viz **Obrázek 3.13**) by se mohlo zdát, že žádný výrazný pokles přesnosti nenastal, není patrný růst chyby v konkrétních místech. Chyby se z žádného rohu nešíří dále do zbytku oblasti, ani nevznikají další extrémy. Avšak z přiložené tabulky (viz

Tabulka 4) je jasné, že přesnost této predikce také postupně klesá.



Obrázek 3.13 *Absolutní odchylka predikce tlaku (extrapolací) od numerických výsledků*

Relativní odchylky výsledků extrapolace následují stejný trend jako u výsledků učení, a proto je zde uveden jen slovní popis. U rychlosti dochází opět k velkému nárůstu odchylky v pravém dolním rohu a u tlaku je to opět v levém dolním rohu. Tyto odchylky opět výrazně převyšují vše ostatní. V jiných místech se žádné další výkyvy ani extrémy nevyskytují.

Jak již bylo zmíněno v předešlých řádcích, přesnost odhadů sítě jak pro rychlostní, tak pro tlakové pole postupně klesá. Tento fakt je patrný z přiložených tabulek (viz **Tabulka 3**,

Tabulka 4), kde po odstranění extrémů, ve stejném stylu jako u tréninku a validace, hodnoty odchylek postupně rostou. Řešení tohoto problému by mohlo být navýšení počtu neuronů v jednotlivých vrstvách, avšak i toto řešení není všespásné a má pouze omezený účinek. Hodnoty v tabulkách pro ořezanou výpočetní síť opět potvrzují předešlé tvrzení, že model lépe predikuje stav ve středu oblasti. Z tabulek je také patrná nepřesnost a chaotičnost extrémních odchylek v rozích. U predikcí rychlostního pole (Bez úpravy) to má za následek postupný pokles relativní odchylky, což je v rozporu s ostatními výsledky. U predikce tlaku (Bez úpravy) je vidět, jak jsou hodnoty odchylek zcela náhodné.

Tabulka 3 *Relativní odchylky predikcí rychlostního pole od výsledků numerických výpočtů*

Re	Bez úpravy [%]	Odstraněny extrémy [%]	Síť 44x44 [%]
950	23,42	4,673	0,831
1000	19,40	5,132	1,268
1050	17,42	5,899	1,984
1100	16,93	7,201	2,903

Tabulka 4 *Relativní odchylky predikcí tlakového pole od výsledků numerických výpočtů*

Re	Bez úpravy [%]	Odstraněny extrémy [%]	Síť 44x44 [%]
950	177,7	12,85	5,352
1000	342,7	16,74	6,237
1050	1147	15,10	7,479
1100	120,2	18,88	9,372

Relativní odchylka rychlostního pole pro data nejvzdálenější od tréninkového datasetu, tedy pro Reynoldsovo číslo 1100, nabývá přibližně hodnoty 7,2 %, což se ještě jeví být přijatelné. Při porovnání s tlakem má rychlost více než 2,5krát menší odchylku. Stejný fenomén se vyskytuje i u výsledků predikcí z rozsahu tréninkových dat, proto lze usoudit, že se síť primárně lépe naučila data a skryté vzorce popisující rychlostní pole, a to především ve středu oblasti.

4 Závěr

V této práci byl postupně vysvětlen princip, architektura a funkce neuronových sítí. Následně byl proveden rozbor dané problematiky a byla naprogramována zkušební neuronová síť, řešící rovnici vedení tepla v 1D. Účelem této sítě bylo prvotní praktické seznámení s problematikou a také tvorba jednoduchého exempláře, na kterém lze popsat proces tvorby neuronové sítě. Tvorba této sítě je zde rozebrána a komentována krok za krokem, od volby vhodných aktivačních funkcí až po výpočet chybové funkce.

V další části byla nejprve provedena rešerše rozebírající využití neuronových sítí v CFD. Následně byl popsán řešený fenomén proudění v kavitě. Dále byla v této části popsána geometrie a způsob získání referenčních dat, použitých k tréninku. Konkrétně se jedná o výsledky numerických výpočtů z programu OpenFOAM. Výpočty byly provedeny pro Reynoldsova čísla v rozsahu 100 – 1100. Část z nich v rozsahu 100 – 900 je určena k tréninku jakožto tréninkový dataset, zbytek v rozsahu 902 – 1100 slouží k testování schopnosti naučené sítě extrapolovat. Následně byla popsána a vysvětlena architektura samotné sítě. Ta se skládá celkem z 8 vrstev, vstupní vrstva očekává 3 vstupní hodnoty: souřadnice *x, y* a Reynoldsovo číslo *Re*. Skryté vrstvy obsahují 350 neuronů každá a výstupní vrstva generuje 2 hodnoty: ψ představující proudovou funkci a *p* představující tlak. Použit je optimalizační algoritmus ADAM. Dále je popsán proces učení a následně jeho výsledky. Učení proběhlo v cyklu o 1200 iteracích, kde v prvních 250 iteracích byla chybová funkce počítána pouze z numerických dat. To mělo síti pomoci lépe vystihnout obecné vzory řešeného problému. Následně jsou přidány všechny ostatní složky chybové funkce. Výsledky ukazují dobrou shodu predikcí s numerickými výpočty, s výjimkou několika míst, převážně v rozích, kde vznikají extrémy relativní odchylky. V další části byla provedena validace, kde jsou do sítě vložena data z tréninkového rozsahu, avšak jsou to data, která ještě neviděla. Jedná se tedy o kontrolu schopnosti sítě interpolovat. Výsledky ukazují, že se síť naučila data z tréninkového datasetu korektně a nedochází k problému zvanému *overfitting*. Nutné je konstatovat, že jsou predikce jak tréninkového, tak validačního datasetu přesnější ve středu oblasti, s přiblížením ke stěně přesnost odhadu klesá.

V poslední části jsou síti prezentována data mimo rozsah tréninku a je jimi testována její schopnost extrapolovat. Zde se potvrzuje obecně známé tvrzení, že neuronové sítě předpovídají výsledky lépe v režimu interpolace. Pro hodnoty blízké tréninku byla přesnost odhadů sítě srovnatelná s interpolací, avšak se zvyšující se vzdáleností dat od tréninkového datasetu, tedy s rostoucím Reynoldsovým číslem, dle očekávání přesnost klesá.

5 Reference

[1] HEATON, Jeff. Applications of Deep Neural Networks with Keras [online]. 2022 Dostupné z: arXiv:2009.05673 [cs.LG]. [cit. 2023-11-16].

[2] History: The 1940's to the 1970's [online]. Dostupné z: https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/History/history1.html. [cit. 2023-11-16].

[3] HAYKIN, Simon S. Neural networks and learning machines [online]. 3rd ed. Upper Saddle River: Pearson Education, 2009 ISBN 978-0-13-147139-9. [cit. 2023-11-17].

[4] CIABURRO, Giuseppe a Balaji VENKATESWARAN. Neural Networks with R [online]. Dostupné z: https://subscription.packtpub.com/book/data/9781788397872/pref01. [cit. 2023-11-16].

[5] SHARMA, Siddharth, Simone SHARMA a Anidhya ATHAIYA. ACTIVATION FUNCTIONS IN NEURAL NETWORKS. In: International Journal of Engineering Applied Sciences and Technology [online]. 4. 2020, s. 310 - 316 Dostupné z: 10.33564/IJEAST.2020.v04i12.054. [cit. 2023-11-17].

[6] PIKMAN, Jan. Federated Learning for Robotic Navigation. Online, Bakalářská práce. Praha: ČVUT, 2022. Dostupné z: https://dspace.cvut.cz/handle/10467/101275. [cit. 2023-11-19].

[7] DELL'AVERSANA, Paolo. Reinforcement learning in optimization problems. Applications to geophysical data inversion. AIMS Geosciences [online]. 2022, 8(3), 488-502ISSN 2471-2132. Dostupné z: doi:10.3934/geosci.2022027. [cit. 2023-11-19].

[8] NIELSEN, Michael A. Neural networks and deep learning [online]. Determination press, 2015 Dostupné z: http://neuralnetworksanddeeplearning.com/. [cit. 2023-12-15].

[9] HUANG, Dobbin. Physics-informed neural networks for fluid mechanics. Online. Delft: Technická univerzita Delft, 2022. Dostupné z: https://repository.tudelft.nl/islandora/object/uuid:e736cf29-f463-4b1f-9e18-aa7532dc1209?collection=education. [cit. 2023-12-19].

[10] SEKAR, Vinothkumar; JIANG, Qinghua; SHU, Chang a KHOO, Boo Cheong. Fast flow field prediction over airfoils using deep learning approach. Online. Physics of Fluids. 2019, roč. 31, č. 5. ISSN 1070-6631. Dostupné z: https://doi.org/10.1063/1.5094943. [cit. 2023-12-19].

[11] AMALINADHI, Cahya; PALAR, Pramudita S.; STEVENSON, Rafael a ZUHAL, Lavi. On Physics-Informed Deep Learning for Solving Navier-Stokes Equations. Online. In: AIAA SCITECH 2022 Forum. Reston, Virginia: American Institute of Aeronautics and Astronautics, 2022. ISBN 978-1-62410-631-6. Dostupné z: https://doi.org/10.2514/6.2022-1436. [cit.2023-12-22].

[12]JIN, Xiaowei, Shengze CAI, Hui LI a KARNIADAKIS, George Em. NSFnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations. In: Journal of Computational Physics [online]. 2021, s. 426 ISBN 109951. ISSN 0021-9991. Dostupné z: doi:https://doi.org/10.1016/j.jcp.2020.109951. [cit. 2024-01-25].

[13]RAISSI, Maziar; YAZDANI, Alireza a KARNIADAKIS, George Em. Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. Online. In: Science. 2020, s. 1026-1030. ISSN 0036-8075. Dostupné z: https://doi.org/10.1126/science.aaw4741. [cit. 2024-01-25].

[14]MAO, Zhiping; JAGTAP, Ameya D. a KARNIADAKIS, George Em. Physics-informed neural networks for high-speed flows. Online. In: Computer Methods in Applied Mechanics and Engineering. 2020. ISSN 00457825. Dostupné z: https://doi.org/10.1016/j.cma.2019.112789. [cit. 2024-01-25].

[15]YIN, Minglang; ZHENG, Xiaoning; HUMPHREY, Jay D. a KARNIADAKIS, George Em. Non-invasive inference of thrombus material properties with physics-informed neural networks. Online. In: Computer Methods in Applied Mechanics and Engineering. 2021. ISSN 00457825. Dostupné z: https://doi.org/10.1016/j.cma.2020.113603. [cit. 2024-01-25].

[16]WANG, Jian-Xun; WU, Jin-Long a XIAO, Heng. Physics-informed machine learning approach for reconstructing Reynolds stress modeling discrepancies based on DNS data. Online. Physical Review Fluids. 2017, roč. 2, č. 3. ISSN 2469-990X. Dostupné z: https://doi.org/10.1103/PhysRevFluids.2.034603. [cit. 2024-02-18].

[17]XIAO, H.; WU, J.-L.; WANG, J.-X.; SUN, R. a ROY, C.J. Quantifying and reducing model-form uncertainties in Reynolds-averaged Navier–Stokes simulations: A data-driven, physics-informed Bayesian approach. Online. Journal of Computational Physics. 2016, roč. 324, s. 115-136. ISSN 00219991. Dostupné z: https://doi.org/10.1016/j.jcp.2016.07.038. [cit. 2024-02-18].

[18]BECK, Andrea; FLAD, David a MUNZ, Claus-Dieter. Deep neural networks for data-driven LES closure models. Online. Journal of Computational Physics. 2019, roč. 398. ISSN 00219991. Dostupné z: https://doi.org/10.1016/j.jcp.2019.108910. [cit. 2024-02-18].

[19]SHARMA, Pushan; CHUNG, Wai Tong; AKOUSH, Bassem a IHME, Matthias. A Review of Physics-Informed Machine Learning in Fluid Mechanics. Online. Energies. 2023, roč. 16, č. 5. ISSN 1996-1073. Dostupné z: https://doi.org/10.3390/en16052343. [cit. 2024-02-18].

[20]CAI, Shengze; MAO, Zhiping; WANG, Zhicheng; YIN, Minglang a KARNIADAKIS, George Em. Physics-informed neural networks (PINNs) for fluid mechanics: a review. Online. Acta Mechanica Sinica. 2021, roč. 37, č. 12, s. 1727-1738. ISSN 0567-7718. Dostupné z: https://doi.org/10.1007/s10409-021-01148-1. [cit. 2024-02-18].

[21]AGUIRRE, A.; CASTILLO, E.; CRUCHAGA, M.; CODINA, R. a BAIGES, J. Stationary and time-dependent numerical approximation of the lid-driven cavity problem for power-law fluid flows at high Reynolds numbers using a stabilized finite element formulation of the VMS type. Online. Journal of Non-Newtonian Fluid Mechanics. 2018, roč. 257, s. 22-43. ISSN 03770257. Dostupné z: https://doi.org/10.1016/j.jnnfm.2018.03.014. [cit. 2024-05-09].

[22]The Lid-Driven Square Cavity Flow: From Stationary to Time Periodic and Chaotic. Online. In: Communications in Computational Physics. 2007, s. 900 - 932. ISSN 1815-2406. Dostupné z: https://global-sci.org/intro/article_detail/cicp/7932.html#. [cit. 2024-05-09].

[23]REDAL, Héctor; CARPIO, Jaime; GARCÍA-SALABERRI, Pablo A. a VERA, Marcos. DynamFluid: Development and Validation of a New GUI-Based CFD Tool for the Analysis of Incompressible Non-Isothermal Flows. Online. Processes. 2019, roč. 7, č. 11. ISSN 2227-9717. Dostupné z: https://doi.org/10.3390/pr7110777. [cit. 2024-05-09].

[24] Benchmark: Lid-driven Cavity (2d). Online. Zeta Comptutational resources. 2024. Dostupné z: http://www.zetacomp.com/benchmarks/lid-driven-cavity-2d.asp. [cit. 2024-05-25].

[25]What is Adam Optimizer? Online. GeeksforGeeks. Dostupné z: https://www.geeksforgeeks.org/adam-optimizer/. [cit. 2024-05-24].

[26]Xavier initialization. Online. GeeksforGeeks. Dostupné z: https://www.geeksforgeeks.org/xavier-initialization/. [cit. 2024-05-24].

[27]Activation Functions with Derivative and Python code: Sigmoid vs Tanh Vs Relu. Online. In: Medium. 2019. Dostupné z: https://medium.com/@omkar.nallagoni/activation-functions-with-derivative-and-python-code-sigmoid-vs-tanh-vs-relu-44d23915c1f4. [cit. 2024-05-11].