Diplomová práce

MODELOVÁNÍ NEURONOVÝCH SÍTÍ VE VIRTUÁLNÍ REALITĚ

Bc. Ondřej Úlehla

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: doc. Ing. Mgr. Petr Klán, CSc.
9. ledna 2025

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Modelování neuronových sítí ve virtuální realitě
Student:	Bc. Ondřej Úlehla
Vedoucí:	doc. Ing. Mgr. Petr Klán, CSc.
Studijní program:	Informatika
Obor / specializace:	Webové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2025/2026

Pokyny pro vypracování

Realizujte systém modelování neuronových sítí ve virtuální realitě. Parametry neuronových sítí bude kódovat webová aplikace. Vizuální program virtuálního 3D prostředí přečte a dekóduje webová data a postaví podle nich funkční 3D model neuronové sítě (3D mozek). Model bude sloužit jako řídicí nebo ovládací prvek objektů virtuálního prostředí.

Postupujte podle následujících kroků:

1. Seznamte se s základními principy umělých neuronových sítí.
2. Navrhněte webovou aplikaci včetně vhodného způsobu popisu neuronových sítí.
3. Implementujte a testujte webovou aplikaci.
4. Seznamte se s principy vizuálního programování v prostředí metaverse Resonite.
5. Navrhněte virtuální 3D svět pro generování modelů neuronových sítí.
6. Vytvořte 3D svět v prostředí metaverse Resonite.
7. Propojte webovou aplikaci s virtuálním 3D světem.
8. Implementujte automatické generování modelů neuronových sítí s použitím webové aplikace.
9. Použijte modely neuronových sítí v rámci 3D scény a testujte jejich správnost.
10. Popište podrobně a zveřejněte virtuální svět pro generování modelů neuronových sítí.

Elektronicky schválil/a Ing. Jaroslav Kuchař, Ph.D. dne 24. června 2024 v Praze.

České vysoké učení technické v Praze
Fakulta informačních technologií
© 2025 Bc. Ondřej Úlehla. Všechna práva vyhrazena.
Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí a nad rámec oprávnění uvedených v Prohlášení, je nezbytný souhlas autora.

Odkaz na tuto práci: Úlehla Ondřej. *Modelování neuronových sítí ve virtuální realitě*. Diplomová práce. České vysoké učení technické v Praze, Fakulta informačních technologií, 2025.

Chtěl bych poděkovat především vedoucímu práce panu doc. Ing. Mgr. Petr Klánovi, CSc. za jeho odborné vedení a konzultace během vytváření této práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 9. ledna 2025

Abstrakt

Tato diplomová práce se zabývá vývojem systému pro modelování a vizualizaci neuronových sítí ve virtuální realitě. Cílem je propojit webovou aplikaci, která slouží ke konfiguraci parametrů neuronové sítě, s imerzivním 3D prostředím, kde je síť automaticky generována a použita k řízení virtuálních objektů. V rámci práce jsou nejprve představeny základní principy umělých neuronových sítí. Následně je popsán návrh a implementace webové aplikace, která poskytuje uživatelské rozhraní pro definici architektury a parametrů neuronové sítě. Další částí práce je pak integrace této aplikace s virtuálním světem vytvořeným na platformě Resonite, která umožňuje vizuální programování a tvorbu interaktivních 3D scén. Výsledný systém demonstruje automatizované generování modelu neuronové sítě na základě parametrů zadaných ve webové aplikaci. Práce tak představuje řešení propojující oblast webových technologií, umělé inteligence a virtuální reality.

Klíčová slova umělé neuronové sítě, virtuální realita, 3D vizualizace, webové aplikace, parametrizace neuronových sítí, generování modelů, vizuální programování, Resonite, metaverse

Abstract

This thesis focuses on the development of a system for modeling and visualization of neural networks in virtual reality. The goal is to connect a web application, which serves for configuring the parameters of a neural network, with an immersive 3D environment where the network is automatically generated. The thesis first introduces the fundamental principles of artificial neural networks and the methods for modeling them. Subsequently, it describes the design and implementation of a web application that provides a user interface for defining the architecture and parameters of a neural network. Next part of the work lies in the integration of this application with a virtual world created on the Resonite platform, which enables visual programming and creation of

interactive 3D scenes. The resulting system demonstrates the automated generation of a neural network model based on parameters specified in the web application. The thesis thus presents an innovative solution connecting the fields of web technologies, artificial intelligence, and virtual reality.

Keywords artificial neural networks, virtual reality, 3D visualization, web applications, parameterization of neural networks, model generation, visual programming, Resonite, metaverse

Obsah

Seznam obrázků

Seznam zkratek

API	Application Programming Interface
CDN	Content Delivery Network
CNN	Convolutional Neural Network
CORS	Cross-Origin Resource Sharing
CPU	Central Processing Unit
DNS	Domain Name System
ELU	Exponential Linear Unit
GCP	Google Cloud Platform
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
I/O	Input/Output
JSON	JavaScript Object Notation
LIME	Local Interpretable Model-agnostic Explanations
LSTM	Long Short-Term Memory
MLP	Multi-Layer Perceptron
MSE	Mean Squared Error
PR	Precision-Recall
PReLU	Parametric Rectified Linear Unit
QDs	Quantum Dots
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
ROC	Receiver Operating Characteristic
SHAP	SHapley Additive exPlanations
SSL	Secure Sockets Layer
STP	Short-Term Plasticity
SVG	Scalable Vector Graphics
UI	User Interface
UNS	Umělé Neuronové Sítě
VR	Virtual Reality
WebGL	Web Graphics Library

Kapitola 1
Úvod

Umělá inteligence a neuronové sítě zažívají v posledních letech obrovský rozmach. S jejich pomocí řešíme komplexní problémy, od rozpoznávání obrazu přes zpracování přirozeného jazyka až po autonomní řízení vozidel. Zároveň se virtuální realita (VR) stává stále dostupnější a nabízí nové možnosti interakce a vizualizace. Propojení těchto dvou oblastí otevírá dveře velké spoustě aplikací, které mohou změnit způsob, jakým chápeme a pracujeme s neuronovými sítěmi. Tato diplomová práce se zaměřuje na vývoj systému, který umožňuje modelování a vizualizaci neuronových sítí ve virtuální realitě. Cílem je vytvořit intuitivní a interaktivní prostředí, kde uživatelé mohou snadno navrhovat, upravovat a zkoumat architekturu neuronových sítí. Klíčovým aspektem je integrace webové aplikace, která slouží jako rozhraní pro konfiguraci parametrů sítě, s imerzivním 3D prostředím, kde je síť automaticky generována a slouží k vizualizaci. Práce se nejprve věnuje teoretickým základům umělých neuronových sítí, jejich architektuře a principům fungování. Následně je popsán návrh a implementace webové aplikace, která poskytuje uživatelské rozhraní pro specifikaci parametrů neuronové sítě. Aplikace umožňuje definovat počet vrstev, počet neuronů v každé vrstvě, typ aktivačních funkcí, inicializaci vah a další klíčové aspekty. Důraz je kladen na vytvoření intuitivního a efektivního rozhraní, které uživatelům usnadní proces konfigurace sítě. Jádrem práce je integrace webové aplikace s virtuálním světem vytvořeným na platformě Resonite, která podporuje vizuální programování a tvorbu interaktivních 3D scén. Popisuje se způsob přenosu parametrů z webové aplikace do virtuálního prostředí a proces automatického generování modelu neuronové sítě na základě těchto parametrů. Výsledná 3D vizualizace umožňuje uživatelům prozkoumat strukturu sítě, sledovat tok dat a interagovat s modelem v reálném čase. Navržený systém představuje řešení, které propojuje webové technologie, umělou inteligenci a virtuální realitu. Kombinace konfigurovatelné webové aplikace a automaticky generovaných 3D modelů neuronových sítí přináší možnosti vizualizace, experimentování a pochopení principů fungování těchto komplexních systémů. Výsledky této práce mohou nalézt uplatnění v oblasti vzdělávání i v

praktických aplikacích, kde je potřeba názorně prezentovat a zkoumat chování neuronových sítí. Zároveň otevírají dveře dalšímu rozšiřování aplikace a vývoji v oblasti propojení umělé inteligence a virtuální reality.

Kapitola 2
Cíle práce

Tato diplomová práce si klade za cíl navrhnout, implementovat a otestovat systém pro modelování a vizualizaci neuronových sítí ve virtuální realitě. Tento systém propojí webovou aplikaci pro konfiguraci parametrů neuronových sítí s virtuálním 3D prostředím, kde budou modely automaticky generovány a vizualizovány. Dílčí cíle práce vycházejí z jednotlivých bodů zadání a pokrývají klíčové aspekty vývoje tohoto systému:

- Prostudovat základní principy a fungování umělých neuronových sítí pro získání potřebných teoretických znalostí.
- Navrhnout, implementovat a otestovat webovou aplikaci jako rozhraní pro konfiguraci parametrů neuronových sítí, včetně způsobu jejich kódování s využitím vhodných technologií a frameworků.
- Seznámit se s principy vizuálního programování v prostředí Resonite pro tvorbu virtuálního 3D světa.
- Navrhnout a implementovat 3D svět v prostředí Resonite s využitím dostupných nástrojů a funkcionalit.
- Zajistit propojení webové aplikace s virtuálním 3D světem pro přenos parametrů a generování modelů.
- Implementovat logiku pro automatické generování modelů neuronových sítí na základě parametrů z webové aplikace.
- Otestovat a vyhodnotit funkčnost vygenerovaných modelů ve virtuálním prostředí.
- Zdokumentovat proces vývoje, popsat implementaci a dosažené výsledky, zveřejnit virtuální svět pro širší využití.

Splněním těchto cílů vznikne systém, který propojí oblast umělých neuronových sítí s virtuální realitou. Výsledné řešení umožní uživatelům navrhovat, konfigurovat a vizualizovat neuronové sítě v interaktivním 3D prostředí.

Kapitola 3

Základní principy umělých neuronových sítí

Umělé neuronové sítě (ANN) představují výkonný výpočetní model inspirovaný biologickými neuronovými sítěmi, který v posledních desetiletích výrazně ovlivnil vývoj v oblasti umělé inteligence a strojového učení. Tato kapitola poskytuje detailní pohled na základní principy ANN, jejich architekturu a fungování. Začíná představením historického vývoje od prvních matematických modelů neuronů až po současné architektury hlubokého učení. Následně se věnuje matematickým základům, včetně podrobného rozboru struktur jednotlivých neuronů, jejich propojení a způsobů učení. V kapitole jsou popsány různé typy aktivačních funkcí, metody učení a optimalizační techniky používané při trénování neuronových sítí. Zvláštní pozornost je věnována problematice zpětné propagace chyby a gradient descent algoritmu, které tvoří základ moderních učících metod [1]. Text se také zabývá praktickými aspekty implementace neuronových sítí, včetně regularizačních technik, problematiky přetrénování a metod pro zlepšení generalizace [2]. Kapitola je zakončena popisem současných trendů a výzev v oblasti neuronových sítí, s důrazem na nové směry výzkumu a aplikační možnosti [3].

Použitá notace

V této práci používáme následující matematickou notaci:

- Vektory značíme tučným písmem malým: $\mathbf{x}$, $\mathbf{h}$, $\mathbf{b}$
- Matice značíme tučným písmem velkým: $\mathbf{W}$, $\mathbf{U}$
- Skalární hodnoty značíme běžným písmem: w, x, b
- Index vrstvy značíme horním indexem v závorce: $\mathbf{h}^{(l)}$
- Časový index značíme dolním indexem: $\mathbf{h}_t$

- Index neuronu značíme dolním indexem: x_i, w_{ij}

3.1 Historie a vývoj neuronových sítí

Historie vývoje umělých neuronových sítí představuje fascinující cestu vědeckého poznání, během níž se střídala období významných objevů s obdobími relativní stagnace. Tento vývoj lze rozdělit do několika klíčových etap, které formovaly současné chápání a využití neuronových sítí [4].

3.1.1 Počátky (1943-1969)

První formální model neuronu představili v roce 1943 Warren McCulloch a Walter Pitts. Jejich práce „A Logical Calculus of the Ideas Immanent in Nervous Activity“ položila základy pro pochopení neuronů jako logických přepínačů schopných provádět výpočetní operace [5]. Tento model, ačkoliv značně zjednodušený oproti biologickým neuronům, demonstroval možnost vytvoření výpočetního systému inspirovaného lidským mozkem.

Zásadní průlom přišel v roce 1957, kdy Frank Rosenblatt vyvinul perceptron, první praktickou implementaci neuronové sítě schopné učení [6]. Rosenblatt dokázal, že perceptron může být použit pro klasifikaci lineárně separovatelných vzorů a formuloval perceptronový konvergenční teorém, který garantoval nalezení řešení pro takové problémy. Jeho práce zahrnovala:

- Definici učícího algoritmu pro adaptaci vah
- Důkaz konvergence pro lineárně separovatelné problémy
- Demonstraci praktického využití v oblasti rozpoznávání vzorů
- Vytvoření první hardwarové implementace neuronové sítě

Matematicky lze perceptron popsat následující rovnicí:

$$y = f\left(\sum_{i=1}^{n} w_i x_i - \theta\right) \tag{3.1}$$

kde:

- y reprezentuje výstup perceptronu
- w_i jsou váhy spojení
- x_i jsou vstupní hodnoty
- θ je prahová hodnota
- f je aktivační funkce (původně skoková funkce)

3.1.2 První útlum (1969-1986)

V roce 1969 Marvin Minsky a Seymour Papert publikovali knihu „Perceptrons“ [7], která měla zásadní vliv na další vývoj oboru. V této práci matematicky dokázali fundamentální omezení jednoduchých perceptronů, zejména:

- Neschopnost řešit nelineárně separovatelné problémy
- Omezení při zpracování prostorových vztahů
- Problémy s invariancí vůči posunu a rotaci
- Nedostatečnou výpočetní kapacitu pro komplexní úlohy

Jejich analýza problému XOR se stala klasickou ukázkou omezení jednoduchých perceptronů. Pro funkci XOR platí:

$$\text{XOR}(x_1, x_2) = \begin{cases} 1 & \text{pro } (x_1 = 1 \land x_2 = 0) \lor (x_1 = 0 \land x_2 = 1) \\ 0 & \text{pro } (x_1 = 0 \land x_2 = 0) \lor (x_1 = 1 \land x_2 = 1) \end{cases} \tag{3.2}$$

3.1.3 Renesance (1986-současnost)

Zásadní průlom v oblasti neuronových sítí přišel s objevem algoritmu zpětného šíření chyby (backpropagation) v roce 1986, který představili David Rumelhart, Geoffrey Hinton a Ronald Williams [8]. Tento algoritmus umožnil efektivní trénování vícevrstvých sítí a překonal omezení původních perceptronů. V jeho základní podobě lze algoritmus popsat následující rovnicí:

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} \tag{3.3}$$

kde:

- Δw_{ij} představuje změnu váhy mezi neurony i a j
- η je učící koeficient
- E je chybová funkce
- $\frac{\partial E}{\partial w_{ij}}$ je parciální derivace chyby vzhledem k váze

Tato základní forma byla později rozšířena do maticové podoby pro efektivní implementaci, jak bude popsáno v sekci 3.5.3 o zpětné propagaci chyby.

3.2 Biologická inspirace a matematický model

Umělé neuronové sítě vycházejí z poznatků o fungování biologického nervového systému [1]. Tato inspirace není náhodná - biologické neuronové sítě představují vysoce efektivní výpočetní systém, který se vyvinul během milionů let evoluce. Pochopení této analogie je klíčové pro porozumění principům umělých neuronových sítí a jejich schopnosti učit se a adaptovat.

3.2.1 Biologický neuron

Biologický neuron představuje základní stavební jednotku nervového systému. Jeho struktura je výsledkem evolučního procesu optimalizace pro zpracování a přenos informací. Skládá se z následujících klíčových částí:

- **Dendrity** - jsou vstupní struktury neuronu, které přijímají signály od ostatních neuronů. Jejich větvená struktura umožňuje přijímat signály od tisíců jiných neuronů.
- **Soma** (tělo neuronu) - integruje přijaté signály a rozhoduje o generování výstupního signálu. V těle neuronu probíhají komplexní biochemické procesy, které určují, zda neuron „vypálí“ signál.
- **Axon** - představuje výstupní dráhu neuronu, po které se šíří akční potenciál. Axon může být velmi dlouhý a může se větvit, což umožňuje přenos signálu k mnoha cílovým neuronům.
- **Synapse** - jsou specializovaná spojení mezi neurony, kde dochází k přenosu signálu. Synaptická plasticita, tedy schopnost měnit sílu synaptických spojení, je základem učení v biologických neuronových sítích.

Přenos signálu v biologickém neuronu je komplexní proces založený na elektrochemických mechanismech. Když soma obdrží dostatečně silné excitační signály přes dendrity, generuje elektrický impulz (akční potenciál), který se šíří po axonu k synapsím. Na synapsích dochází k uvolnění neurotransmiterů, které ovlivňují cílové neurony. Tento proces lze matematicky popsat pomocí rovnice pro membránový potenciál:

$$C_m \frac{dV_m}{dt} = -g_L(V_m - E_L) + \mathbf{I}_{\text{syn}}(t) \tag{3.4}$$

kde C_m reprezentuje kapacitu membrány, V_m je membránový potenciál, g_L představuje vodivost membrány, E_L je klidový potenciál a $\mathbf{I}_{\text{syn}}(t)$ představuje synaptický proud. Tato rovnice popisuje, jak se mění elektrický potenciál neuronu v čase v závislosti na vstupních signálech.

Zatímco biologické neurony jsou komplexní systémy s mnoha jemnými detaily, pro účely strojového učení potřebujeme jejich zjednodušený matematický model. Tento model zachovává klíčové vlastnosti důležité pro zpracování informací, zatímco abstrahuje od biologických detailů.

3.2.2 Od biologického k umělému neuronu

Při vytváření matematického modelu umělého neuronu [5] se zaměřujeme na klíčové funkční aspekty biologického neuronu, které jsou podstatné pro zpracování informací:

- **Integrace vstupů** - podobně jako dendrity přijímají signály od jiných neuronů, umělý neuron přijímá vážené vstupy.
- **Prahování** - analogicky k rozhodování somy o generování akčního potenciálu, umělý neuron používá aktivační funkci.
- **Šíření signálu** - stejně jako axon přenáší signál k dalším neuronům, výstup umělého neuronu je distribuován k neuronům v následující vrstvě.

Matematický model neuronu vyjadřuje tyto principy pomocí následující rovnice:

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{3.5}$$

kde:

- x_i jsou vstupní hodnoty reprezentující signály přicházející od jiných neuronů
- w_i jsou váhy spojení analogické k synaptickým vahám v biologickém systému
- b je bias (práh) odpovídající základní úrovni excitace neuronu
- f je aktivační funkce modelující nelineární charakter odpovědi biologického neuronu

Tato abstrakce zachovává klíčové vlastnosti biologického neuronu důležité pro zpracování informací, zatímco zjednodušuje nebo vypouští aspekty, které nejsou pro výpočetní model podstatné. Například časové charakteristiky přenosu signálu nebo metabolické procesy v neuronu nejsou v základním modelu umělého neuronu zahrnuty.

3.2.3 Role synapsí v učení

Synapse hrají v biologických neuronových sítích klíčovou roli v procesu učení [8]. Jejich schopnost měnit sílu spojení (synaptická plasticita) je základem formování paměti a adaptace nervového systému. Tento princip je emulován v umělých neuronových sítích prostřednictvím úpravy vah během procesu učení.

Hebbův princip učení, formulovaný Donaldem Hebbem v roce 1949, popisuje základní mechanismus synaptické plasticity: „Když axon neuronu A je

dostatečně blízko k excitaci neuronu B a opakovaně nebo perzistentně se podílí na jeho aktivaci, dochází v jednom nebo obou neuronech k metabolickým změnám tak, že se efektivita A jako jeden z neuronů aktivujících B zvyšuje.“[9]

Tento biologický princip je v umělých neuronových sítích implementován prostřednictvím algoritmu zpětného šíření chyby, který upravuje váhy spojení na základě chyby výstupu sítě. Matematicky lze tuto adaptaci vyjádřit jako:

$$\Delta w_{ij} = \eta \cdot \delta_j \cdot x_i \tag{3.6}$$

kde η je rychlost učení, δ_j je chyba výstupu neuronu j a x_i je vstup od neuronu i. Tato rovnice reprezentuje zjednodušený model synaptické plasticity v umělých neuronových sítích.

3.3 Aktivační funkce

Aktivační funkce představují klíčový prvek neuronových sítí, který zavádí nelinearitu do systému a umožňuje modelování komplexních vztahů [10]. Jejich role je zásadní pro schopnost sítě učit se a reprezentovat složité vzory v datech. Historický vývoj aktivačních funkcí odráží postupné porozumění problematice trénování hlubokých neuronových sítí a řešení problémů jako je mizející gradient [11].

3.3.1 Historie a vývoj aktivačních funkcí

První modely neuronových sítí používaly jednoduché prahové funkce, které přímo napodobovaly princip „všechno nebo nic“ biologických neuronů [7]. Tyto funkce však měly zásadní omezení - nebyly diferencovatelné, což znemožňovalo použití gradientních optimalizačních metod. Postupný vývoj vedl k zavedení spojitých aktivačních funkcí, které tento problém řešily.

3.3.2 Lineární aktivační funkce

Nejjednodušší aktivační funkcí je lineární funkce:

$$f(x) = ax \tag{3.7}$$

kde a je konstanta (obvykle 1). Přestože je tato funkce matematicky jednoduchá, má několik zásadních omezení:

- Neumožňuje modelování nelineárních vztahů, což významně omezuje expresivní schopnost sítě
- Derivace je konstantní, což může vést k problémům při trénování
- Složení více lineárních funkcí je opět lineární funkce, čímž se ztrácí výhoda vícevrstvé architektury

V praxi se lineární aktivační funkce používá především ve výstupní vrstvě pro regresní úlohy, kde chceme, aby síť mohla generovat libovolné reálné číslo.

3.3.3 Sigmoidální funkce

Sigmoid představuje historicky významnou aktivační funkci, která dominovala raným obdobím hlubokého učení:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{3.8}$$

Její popularita vycházela z několika klíčových vlastností:

Výhody:

- Spojitost a diferencovatelnost v celém definičním oboru
- Omezený výstupní rozsah (0,1), což je užitečné pro pravděpodobnostní interpretaci výstupů
- Biologická interpretovatelnost - podobnost s aktivací biologických neuronů
- Jednoduchá derivace: $f'(x) = f(x)(1 - f(x))$

Nevýhody:

- Saturace při velkých hodnotách vstupu vede k problému mizejícího gradientu
- Výstup není centrován kolem nuly, což může komplikovat trénování
- Výpočetně náročnější než modernější alternativy

3.3.4 Hyperbolický tangens

Hyperbolický tangens představuje alternativu k sigmoidální funkci s výhodnějšími vlastnostmi pro trénování:

$$f(x) = \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{3.9}$$

Klíčové charakteristiky zahrnují:

- Výstupní rozsah (-1,1), což poskytuje přirozenější centrování kolem nuly
- Strmější gradient než sigmoid, což může urychlit učení
- Podobně jako sigmoid trpí saturací při velkých vstupech
- Derivace: $f'(x) = 1 - \tanh^2(x)$

3.3.5 ReLU (Rectified Linear Unit)

ReLU představuje průlom v designu aktivačních funkcí a v současnosti je jedna z nejpoužívanější aktivační funkcí v hlubokých neuronových sítích [11]:

$$f(x) = \max(0, x) \tag{3.10}$$

Její úspěch je založen na několika klíčových výhodách:

Výhody:

- Výpočetní efektivita - jednoduchá implementace a rychlý výpočet
- Redukce problému mizejícího gradientu díky konstantnímu gradientu pro kladné vstupy
- Biologická interpretovatelnost - podobnost s chováním reálných neuronů
- Podpora řídkých reprezentací - přirozená regularizace sítě
- Neomezený výstupní rozsah pro kladné hodnoty

Nevýhody:

- „Dying ReLU“ problém - neurony se mohou „vypnout“ a zůstat neaktivní
- Nespojitá derivace v bodě 0
- Neomezený výstup může v některých případech vést k nestabilitě

3.3.6 Moderní varianty aktivačních funkcí

Na základě zkušeností s klasickými aktivačními funkcemi byly vyvinuty různé modifikace řešící jejich nedostatky:

Leaky ReLU:

$$f(x) = \begin{cases} x & \text{pro } x \geq 0 \\ \alpha x & \text{pro } x < 0 \end{cases} \tag{3.11}$$

kde α je malá konstanta (typicky 0.01). Tato modifikace řeší problém „dying ReLU“ tím, že umožňuje malý gradient i pro záporné hodnoty.

Parametric ReLU (PReLU): Podobná Leaky ReLU, ale parametr α je učen během trénování sítě.

ELU (Exponential Linear Unit):

$$f(x) = \begin{cases} x & \text{pro } x \geq 0 \\ \alpha(e^x - 1) & \text{pro } x < 0 \end{cases} \tag{3.12}$$

Kombinuje výhody ReLU s robustností vůči šumu díky saturaci v záporné oblasti.

3.3.7 Výběr aktivační funkce

Volba vhodné aktivační funkce závisí na několika faktorech:

- **Typ úlohy** - například sigmoid je vhodný pro binární klasifikaci, ReLU pro hluboké sítě
- **Hloubka sítě** - hlubší sítě obvykle vyžadují aktivační funkce odolné vůči mizejícímu gradientu
- **Výpočetní omezení** - v embedded systémech může být důležitá výpočetní efektivita
- **Požadavky na regularizaci** - některé aktivační funkce poskytují přirozenou regularizaci

V praxi se často používá kombinace různých aktivačních funkcí v různých částech sítě. Například ReLU ve skrytých vrstvách a sigmoid ve výstupní vrstvě pro klasifikační úlohy.

3.4 Architektury neuronových sítí

Architektury neuronových sítí definují způsob organizace a propojení neuronů v síti. Různé architektury jsou optimalizovány pro řešení specifických typů úloh [3]. Vývoj architektur neuronových sítí byl značně ovlivněn jak biologickými poznatky o struktuře mozku, tak praktickými požadavky na řešení konkrétních problémů strojového učení.

3.4.1 Dopředné neuronové sítě

Dopředné neuronové sítě (feed-forward neural networks) představují základní a historicky první architekturu, kde signál postupuje striktně jednosměrně od vstupní vrstvy přes skryté vrstvy k výstupní vrstvě. Tato architektura je založena na hierarchickém zpracování informací, kde každá vrstva extrahuje stále abstraktnější reprezentace vstupních dat.

Jednovrstvý perceptron

Jednovrstvý perceptron, vyvinutý Frankem Rosenblattem [6], představuje nejjednodušší formu dopředné sítě:

- **Struktura:** Obsahuje pouze vstupní a výstupní vrstvu
- **Omezení:** Schopen řešit pouze lineárně separovatelné problémy
- **Historický význam:** Demonstroval možnost strojového učení
- **Matematický model:** Výstup je dán rovnicí (3.5), kde se používá modernější notace s biasem b místo prahové hodnoty θ.

Vícevrstvý perceptron (MLP)

Vícevrstvý perceptron překonává omezení jednovrstvého perceptronu přidáním jedné nebo více skrytých vrstev. Pro l-tou vrstvu sítě můžeme výpočet popsat následovně:

$$\mathbf{h}^{(l)} = f(\mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}) \tag{3.13}$$

kde:

- $\mathbf{h}^{(l)} \in \mathbb{R}^{n_l}$ je vektor aktivací neuronů v l-té vrstvě
- $\mathbf{W}^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$ je matice vah mezi vrstvami $l-1$ a l
- $\mathbf{b}^{(l)} \in \mathbb{R}^{n_l}$ je vektor biasů pro l-tou vrstvu
- n_l označuje počet neuronů v l-té vrstvě

Klíčové vlastnosti MLP:

- **Univerzální aproximace:** Teoreticky schopný aproximovat libovolnou spojitou funkci
- **Hierarchické učení:** Každá vrstva se specializuje na různé úrovně abstrakce
- **Škálovatelnost:** Možnost přizpůsobit velikost sítě složitosti úlohy

3.4.2 Konvoluční neuronové sítě (CNN)

Konvoluční neuronové sítě [12] jsou specializovanou architekturou inspirovanou strukturou vizuální kůry mozku. Jsou optimalizovány pro zpracování dat s mřížkovou topologií, jako jsou obrázky nebo časové řady.

Základní principy CNN

Konvoluční sítě jsou založeny na třech klíčových konceptech:

- **Lokální receptivní pole:** Každý neuron zpracovává pouze malou část vstupních dat
- **Sdílené váhy:** Stejné konvoluční filtry jsou aplikovány napříč celým vstupem
- **Pooling:** Redukce dimenzionality pomocí podvzorkování

Matematicky lze konvoluční vrstvu popsat jako:

$$\mathbf{h}_{ij}^{(l)} = f\left(\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} w_{mn}^{(l)} x_{(i+m)(j+n)}^{(l-1)} + b^{(l)}\right) \tag{3.14}$$

kde w_{mn}^{l} jsou váhy konvolučního jádra a x_{ij}^{l-1} je vstup z předchozí vrstvy.

Architektura CNN

Typická CNN se skládá z následujících typů vrstev:

- **Konvoluční vrstvy:** Aplikují filtry pro detekci příznaků
- **Aktivační vrstvy:** Zavádějí nelinearitu (typicky ReLU)
- **Pooling vrstvy:** Redukují prostorové dimenze
- **Plně propojené vrstvy:** Klasifikace na základě extrahovaných příznaků

3.4.3 Rekurentní neuronové sítě (RNN)

Rekurentní neuronové sítě jsou specializovány na zpracování sekvencí a časových řad. Jejich klíčovou vlastností je schopnost udržovat vnitřní stav, který funguje jako druh paměti.

Základní principy RNN

Základní rekurentní vrstva je definována rovnicí:

$$\mathbf{h}_t = f(\mathbf{W}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{b}) \tag{3.15}$$

kde:

- $\mathbf{h}_t \in \mathbb{R}^{n_h}$ je vektor skrytého stavu v čase t
- $\mathbf{x}_t \in \mathbb{R}^{n_x}$ je vstupní vektor v čase t
- $\mathbf{W} \in \mathbb{R}^{n_h \times n_x}$ je matice vah pro transformaci vstupu
- $\mathbf{U} \in \mathbb{R}^{n_h \times n_h}$ je matice rekurentních vah
- $\mathbf{b} \in \mathbb{R}^{n_h}$ je vektor biasů
- n_h je dimenze skrytého stavu
- n_x je dimenze vstupního vektoru

LSTM a GRU

Long Short-Term Memory (LSTM), původně navržený Hochreiterem a Schmidhuberem [13], a Gated Recurrent Unit (GRU) jsou pokročilé varianty RNN řešící problém mizejícího gradientu pomocí bránových mechanismů.

LSTM používá tři brány pro řízení toku informací:

$$\begin{aligned} \mathbf{f}_t &= \sigma(\mathbf{W}_f\mathbf{x}_t + \mathbf{U}_f\mathbf{h}_{t-1} + \mathbf{b}_f) \\ \mathbf{i}_t &= \sigma(\mathbf{W}_i\mathbf{x}_t + \mathbf{U}_i\mathbf{h}_{t-1} + \mathbf{b}_i) \\ \mathbf{o}_t &= \sigma(\mathbf{W}_o\mathbf{x}_t + \mathbf{U}_o\mathbf{h}_{t-1} + \mathbf{b}_o) \end{aligned} \tag{3.16}$$

3.4.4 Moderní architektury a trendy

S rostoucí výpočetní silou a množstvím dostupných dat se vyvíjely stále sofistikovanější architektury neuronových sítí. Tyto moderní přístupy řeší specifické problémy a omezení klasických architektur.

Transformery

Transformery [14] představují revoluci v zpracování sekvencí díky mechanismu self-attention:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{3.17}$$

Klíčové vlastnosti:

- Paralelní zpracování sekvencí
- Schopnost modelovat dlouhodobé závislosti
- Škálovatelnost na velmi dlouhé sekvence

Hybridní architektury

Moderní trend směřuje k kombinování různých architektonických prvků:

- CNN-LSTM pro zpracování video sekvencí
- Transformer-CNN pro analýzu obrazu
- Attention-augmented CNN pro vylepšené zpracování obrazu

3.4.5 Výběr architektury pro konkrétní úlohy

Volba vhodné architektury závisí na několika faktorech:

- **Povaha dat:**
  - Obrázky → CNN
  - Sekvence → RNN/Transformer
  - Tabulková data → MLP
- **Velikost datasetu:**
  - Malé datasety → jednodušší architektury
  - Velké datasety → komplexnější modely
- **Výpočetní omezení:**

- Omezené zdroje → efektivní architektury
- Dostupné GPU → paralelizovatelné modely

- **Požadavky na interpretovatelnost:**
  - Vyšší nároky → jednodušší architektury
  - Nižší nároky → komplexnější modely

3.5 Proces učení neuronových sítí

Proces učení neuronových sítí představuje komplexní optimalizační úlohu, jejímž cílem je nalézt takové hodnoty parametrů sítě (vah a biasů), které minimalizují chybu na trénovacích datech [1]. Na rozdíl od tradičních algoritmů, kde je chování explicitně naprogramováno, neuronové sítě své chování odvozují ze zkušenosti získané během trénování [2].

3.5.1 Principy strojového učení v kontextu neuronových sítí

Učení neuronových sítí lze kategorizovat do několika základních paradigmat [4]:

Učení s učitelem (Supervised Learning): Nejčastější forma učení, při které síť dostává páry vstupních dat a požadovaných výstupů. Matematicky lze tento proces vyjádřit pomocí chybové funkce:

$$E = \frac{1}{N} \sum_{i=1}^{N} L(\mathbf{y}_i, \hat{\mathbf{y}}_i) \tag{3.18}$$

kde:

- E reprezentuje průměrnou chybu na trénovací množině
- N je počet trénovacích vzorků
- $\mathbf{y}_i$ je požadovaný (cílový) výstup
- $\hat{\mathbf{y}}_i$ je skutečný výstup sítě
- L je zvolená ztrátová funkce

Učení bez učitele (Unsupervised Learning): V tomto případě síť dostává pouze vstupní data a musí v nich sama identifikovat významné vzory nebo strukturu. Tento typ učení je často využíván pro:

- Redukci dimenzionality dat
- Shlukovou analýzu

- Detekci anomálií
- Extrakci příznaků

Posilované učení (Reinforcement Learning): Kombinuje prvky obou předchozích přístupů - síť se učí na základě zpětné vazby ve formě odměn nebo trestů za své akce.

3.5.2 Optimalizační algoritmy

Gradient Descent

Gradient descent představuje fundamentální algoritmus pro trénování neuronových sítí [8]. Jeho princip spočívá v iterativní úpravě parametrů sítě ve směru záporného gradientu chybové funkce:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \nabla E(\mathbf{w}_t) \tag{3.19}$$

kde:

- $\mathbf{w}_t$ jsou parametry sítě v čase t
- η je learning rate (rychlost učení)
- $\nabla E(\mathbf{w}_t)$ je gradient chybové funkce

Existují tři hlavní varianty gradient descent algoritmu:
1. **Batch Gradient Descent:** - využívá celou trénovací množinu pro jeden update - přesný, ale výpočetně náročný - vhodný pro malé datasety
2. **Stochastic Gradient Descent (SGD):** - aktualizuje parametry po každém vzorku - rychlejší konvergence, ale s větší variancí - může pomoci překonat lokální minima
3. **Mini-batch Gradient Descent:** - kompromis mezi předchozími přístupy - využívá podmnožinu trénovacích dat - v praxi nejčastěji používaná varianta

3.5.3 Zpětná propagace chyby

Algoritmus zpětného šíření chyby (backpropagation) představuje efektivní metodu pro výpočet gradientů v neuronových sítích [8]. Vychází ze základního principu aktualizace vah představeného v sekci 1.3, který lze zapsat jako:

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} \tag{3.20}$$

Pro efektivní implementaci v moderních neuronových sítích se tento princip rozšiřuje do maticové formy. Proces probíhá ve dvou fázích:

Dopředná fáze:

- Vstupní data jsou propagována sítí vpřed
- Pro každou vrstvu l počítáme:

$$\mathbf{h}^{(l)} = f(\mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}) \tag{3.21}$$

  kde $\mathbf{h}^{(l)}$ jsou aktivace v l-té vrstvě, $\mathbf{W}^{(l)}$ jsou váhy, $\mathbf{b}^{(l)}$ jsou biasy a f je aktivační funkce.

- Ukládáme mezivýsledky pro zpětný průchod

Zpětná fáze: Výpočet gradientu využívá řetězové pravidlo. Pro váhy v l-té vrstvě lze gradient v maticové formě vyjádřit jako:

$$\frac{\partial E}{\partial \mathbf{W}^{(l)}} = \boldsymbol{\delta}^{(l)}(\mathbf{h}^{(l-1)})^T \tag{3.22}$$

kde:

- $\boldsymbol{\delta}^{(l)} \in \mathbb{R}^{n_l}$ je vektor chyb pro neurony v l-té vrstvě
- $\mathbf{h}^{(l-1)} \in \mathbb{R}^{n_{l-1}}$ je vektor aktivací předchozí vrstvy
- $\frac{\partial E}{\partial \mathbf{W}^{(l)}} \in \mathbb{R}^{n_l \times n_{l-1}}$ je gradient chyby vzhledem k vahám

Chybový vektor $\boldsymbol{\delta}^{(l)}$ se počítá rekurzivně od výstupní vrstvy zpět. Pro výstupní vrstvu L platí:

$$\boldsymbol{\delta}^{(L)} = \frac{\partial E}{\partial \mathbf{h}^{(L)}} \odot f'(\mathbf{z}^{(L)}) \tag{3.23}$$

Pro skryté vrstvy se chyba propaguje zpětně:

$$\boldsymbol{\delta}^{(l)} = ((\mathbf{W}^{(l+1)})^T \boldsymbol{\delta}^{(l+1)}) \odot f'(\mathbf{z}^{(l)}) \tag{3.24}$$

kde $\odot$ značí prvkové násobení a $\mathbf{z}^{(l)}$ je vstup do aktivační funkce v l-té vrstvě.

Tato maticová formulace umožňuje efektivní implementaci na moderním hardware a je základem pro trénování hlubokých neuronových sítí.

3.5.4 Regularizační techniky

Regularizace představuje klíčový nástroj pro prevenci přetrénování sítě. Mezi hlavní regularizační techniky patří:

L2 regularizace: Přidává k chybové funkci penalizační člen úměrný čtverci vah:

$$E_{\text{reg}} = E + \frac{\lambda}{2}\sum_{w} w^2 \tag{3.25}$$

L1 regularizace: Používá absolutní hodnotu vah:

$$E_{\text{reg}} = E + \lambda \sum_{w} |w| \tag{3.26}$$

kde λ je regularizační parametr kontrolující sílu regularizace.

Dropout: Dropout, představený Srivastava et al. [15], představuje modernější přístup k regularizaci. Během trénování náhodně deaktivuje část neuronů:

$$\tilde{\mathbf{h}} = \mathbf{m} \odot \mathbf{h} \tag{3.27}$$

kde:

- $\mathbf{h}$ je vektor aktivací
- $\mathbf{m}$ je binární maska s pravděpodobností p pro hodnotu 0
- $\odot$ značí prvkové násobení

3.5.5 Praktické aspekty trénování

Volba hyperparametrů představuje kritický aspekt trénování neuronových sítí [16]. Mezi klíčové hyperparametry ovlivňující proces učení patří:

- **Learning rate** [17]:
  - Příliš vysoká → nestabilní trénování
  - Příliš nízká → pomalá konvergence
  - Optimální hodnota závisí na datové sadě a architektuře
- **Batch size** [16]:
  - Ovlivňuje stabilitu a rychlost učení
  - Typické hodnoty: 32, 64, 128, 256
  - Limitováno dostupnou pamětí GPU
- **Počet epoch** [15]:
  - Určuje celkovou dobu trénování
  - Příliš mnoho → přetrénování
  - Příliš málo → nedostatečné naučení

Monitorování průběhu učení: pro efektivní trénování je klíčové sledovat několik metrik:

- **Trénovací chyba:**
  - Měří výkon na trénovací množině

- Měla by postupně klesat

- **Validační chyba:**
  - Měří generalizaci na neviděných datech
  - Slouží k detekci přetrénování

- **Gradient norm:**
  - Indikuje stabilitu učení
  - Pomáhá identifikovat problémy s učením

3.6 Praktické aspekty implementace

Pro úspěšnou implementaci neuronových sítí je klíčové porozumět několika praktickým aspektům, které významně ovlivňují jejich výkon a efektivitu učení. Tyto aspekty zahrnují správnou inicializaci vah, volbu optimalizačního algoritmu a techniky normalizace dat.

3.6.1 Inicializace vah

Správná inicializace vah představuje jeden z kritických faktorů pro úspěšné trénování hlubokých neuronových sítí [18]. Tento aspekt byl dlouhou dobu přehlížen, dokud výzkum neukázal jeho zásadní vliv na stabilitu a rychlost učení. Nevhodná inicializace může vést ke dvěma hlavním problémům:

- Mizející gradient (vanishing gradient) - kdy se gradient během zpětného šíření stává příliš malý

- Explodující gradient (exploding gradient) - kdy gradient naopak nekontrolovaně narůstá

Pro řešení těchto problémů byly vyvinuty specializované inicializační metody. Xavier (nebo Glorot) inicializace, představená v práci Glorot a Bengio [18], byla navržena specificky pro sigmoidální aktivační funkce a tanh. Tato metoda bere v úvahu počet vstupních a výstupních neuronů pro zachování rozptylu aktivací napříč vrstvami:

$$W \sim \mathcal{N}\left(0, \sqrt{\frac{2}{n_{\text{in}} + n_{\text{out}}}}\right) \tag{3.28}$$

kde:

- n_{in} je počet vstupních neuronů

- n_{out} je počet výstupních neuronů

- $\mathcal{N}(\mu, \sigma)$ označuje normální rozdělení

Princip této inicializace spočívá v udržení rozptylu gradientů přibližně konstantní při průchodu sítí, což významně zlepšuje konvergenci učení, jak ukázali Glorot a Bengio [18].

Pro sítě využívající ReLU aktivační funkce byla později navržena He inicializace [11], která lépe zohledňuje asymetrickou povahu ReLU funkce:

$$W \sim \mathcal{N}\left(0, \sqrt{\frac{2}{n_{\text{in}}}}\right) \tag{3.29}$$

Tato inicializace je specificky navržena pro ReLU aktivační funkce a jejich varianty. Zohledňuje fakt, že ReLU propouští pouze kladné hodnoty, což efektivně znamená, že pouze polovina neuronů je v průměru aktivní.

3.6.2 Adam optimalizátor

Optimalizace je klíčovým prvkem trénování neuronových sítí. Adam (Adaptive Moment Estimation) optimalizátor, představený Kingmou a Ba [17], kombinuje výhody dvou populárních metod - RMSprop a momentum optimalizace. Autoři ukazují, že Adam překonává ostatní optimalizační algoritmy v mnoha praktických aplikacích díky své schopnosti efektivně přizpůsobovat learning rate pro každý parametr sítě. Adam adaptivně upravuje learning rate pro každý parametr sítě na základě odhadů prvního a druhého momentu gradientu. Jeho matematický popis je následující:

$$\begin{aligned}
\mathbf{m}_t &= \beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\mathbf{g}_t \\
\mathbf{v}_t &= \beta_2 \mathbf{v}_{t-1} + (1-\beta_2)\mathbf{g}_t^2 \\
\hat{\mathbf{m}}_t &= \frac{\mathbf{m}_t}{1-\beta_1^t} \\
\hat{\mathbf{v}}_t &= \frac{\mathbf{v}_t}{1-\beta_2^t} \\
\mathbf{w}_{t+1} &= \mathbf{w}_t - \frac{\eta}{\sqrt{\hat{\mathbf{v}}_t} + \epsilon}\hat{\mathbf{m}}_t
\end{aligned} \tag{3.30}$$

kde:

- $\mathbf{m}_t, \mathbf{v}_t$ jsou odhady prvního a druhého momentu gradientu
- β_1, β_2 jsou hyperparametry pro výpočet exponenciálních průměrů (doporučené hodnoty jsou $\beta_1 = 0.9$ a $\beta_2 = 0.999$)
- η je základní learning rate (doporučená hodnota je $\eta = 0.001$)
- ϵ je malá konstanta pro numerickou stabilitu (typicky 10^{-8})

Kingma a Ba [17] demonstrovali několik klíčových vlastností Adam optimalizátoru:

- **Adaptivní learning rate** pro každý parametr zvlášť
- **Momentum** implementované pomocí exponenciálních klouzavých průměrů gradientů
- **Korekce bias** pro přesnější odhady momentů, zejména během počátečních iterací
- **Invariance vůči škálování gradientů**, což znamená, že algoritmus funguje dobře i bez nutnosti pečlivého nastavení hyperparametrů

Empirické výsledky ukazují, že Adam je obzvláště efektivní v následujících situacích [17]:

- Problémy s velkými datasety a velkým počtem parametrů
- Nestacionární cílové funkce
- Řídké gradienty a zašuměná data
- Trénování hlubokých neuronových sítí s různými architekturami

3.6.3 Batch normalizace

Batch normalizace, představená Ioffe a Szegedy [16], představuje významný pokrok v oblasti trénování hlubokých neuronových sítí. Tato technika řeší problém tzv. „internal covariate shift“ - jevu, kdy se distribuce aktivací v průběhu trénování mění, což zpomaluje konvergenci a může vést k nestabilitě učení. Proces batch normalizace lze popsat následujícími rovnicemi:

$$\begin{aligned} \mu_B &= \frac{1}{m}\sum_{i=1}^{m} x_i \\ \sigma_B^2 &= \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2 \\ \hat{x}_i &= \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \\ y_i &= \gamma\hat{x}_i + \beta \end{aligned} \tag{3.31}$$

kde:

- x_i jsou vstupní hodnoty v rámci mini-batche
- μ_B je průměr mini-batche
- σ_B^2 je rozptyl mini-batche
- γ, β jsou trénovatelné parametry pro škálování a posun

- ϵ je malá konstanta pro numerickou stabilitu

Podle [16], batch normalizace přináší několik významných výhod:

Zrychlení trénování:

- Umožňuje použití vyšších learning rates
- Snižuje počet epoch potřebných ke konvergenci
- Redukuje závislost na přesné inicializaci vah

Regularizační efekt:

- Přidává šum do procesu učení
- Působí jako lehká forma regularizace
- Může redukovat potřebu použití dropout vrstev

Stabilizace trénování:

- Redukuje problém internal covariate shift
- Zlepšuje tok gradientů sítí
- Umožňuje trénování hlubších architektur

3.7 Vyhodnocování výkonu

Vyhodnocování výkonu neuronových sítí představuje kritickou součást jejich vývoje a nasazení. Podle Powers [19] správná volba a interpretace metrik je klíčová pro porozumění skutečné efektivitě modelu a jeho praktické využitelnosti. Sokolova a Lapalme [20] zdůrazňují důležitost použití komplexní sady metrik, protože každá z nich zachycuje jiný aspekt výkonu modelu.

3.7.1 Metriky pro klasifikaci

Pro klasifikační úlohy se používá několik základních metrik, které společně poskytují komplexní pohled na výkon modelu. Powers [19] a Sokolova et al. [20] zdůrazňují důležitost použití více metrik současně, protože každá metrika zachycuje jiné aspekty výkonu modelu.

Accuracy (Přesnost) je nejzákladnější metrikou, která udává poměr úspěšně klasifikovaných případů k celkovému počtu případů [19]:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{3.32}$$

Tato metrika je snadno interpretovatelná, ale může být zavádějící v případě nevyvážených tříd. Proto se často používají další, specifičtější metriky.

Precision (Preciznost) měří, jaký podíl pozitivních predikcí byl skutečně správný:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{3.33}$$

Podle [1], tato metrika je zvláště důležitá v případech, kdy falešně pozitivní predikce jsou nákladné nebo nebezpečné (například v medicínské diagnostice).

Recall (Úplnost) udává, jaký podíl skutečně pozitivních případů byl správně identifikován:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3.34}$$

Haykin [1] zdůrazňuje význam této metriky v aplikacích, kde je kritické nepromeškání pozitivních případů (například detekce rakoviny). **F1-skóre** představuje harmonický průměr precision a recall:

$$\text{F1-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.35}$$

Tato metrika poskytuje vyvážený pohled na výkon modelu, zejména v případech nevyvážených datasetů. Pro správnou interpretaci těchto metrik je klíčové porozumět základním pojmům:

- **True Positives (TP)** - správně identifikované pozitivní případy
- **True Negatives (TN)** - správně identifikované negativní případy
- **False Positives (FP)** - nesprávně identifikované pozitivní případy (Typ I chyba)
- **False Negatives (FN)** - nesprávně identifikované negativní případy (Typ II chyba)

3.7.2 Křivky a vizualizace

Pro komplexnější analýzu výkonu modelu se často používají různé typy křivek a vizualizací:

- **ROC křivka** (Receiver Operating Characteristic) - zobrazuje vztah mezi true positive rate a false positive rate
- **PR křivka** (Precision-Recall) - užitečná zejména pro nevyvážené datasety
- **Confusion matrix** - poskytuje detailní pohled na distribuci predikcí napříč třídami
- **Learning curves** - zobrazují vývoj chyby během trénování na trénovací a validační sadě

3.7.3 Validační strategie

Pro získání spolehlivých odhadů výkonu modelu je klíčová volba správné validační strategie [20, 19]. V praxi se využívá několik základních přístupů:

- **Hold-out validace** - rozdělení dat na trénovací, validační a testovací sadu [19]
- **K-fold cross-validace** - rozdělení dat na K částí a postupné trénování na K-1 částech [20]
- **Stratifikovaná cross-validace** - zachovává distribuci tříd v jednotlivých foldech [20]
- **Časová validace** - specifická pro časové řady, kde testovací data následují po trénovacích [19]

3.8 Současné trendy a výzvy

Oblast neuronových sítí čelí několika významným výzvám, které současně představují příležitosti pro další výzkum a vývoj. Tyto výzvy týkají nejen technických aspektů, ale i praktických omezení při nasazení modelů v reálných podmínkách.

3.8.1 Energetická efektivita

Energetická náročnost trénování a provozování hlubokých neuronových sítí představuje jeden z nejvýznamnějších problémů současnosti. Strubell et al. [21] dokumentují dramatický nárůst energetické náročnosti moderních neuronových sítí a upozorňují na environmentální dopady jejich trénování. Energetickou náročnost výpočtu lze vyjádřit pomocí následujícího modelu:

$$E_{\text{total}} = P_{\text{comp}} \cdot t_{\text{comp}} + P_{\text{mem}} \cdot t_{\text{mem}} \tag{3.36}$$

kde:

- E_{total} reprezentuje celkovou spotřebovanou energii
- P_{comp} je výkon spotřebovaný na výpočetní operace
- t_{comp} představuje čas strávený výpočty
- P_{mem} je výkon spotřebovaný na přístupy do paměti
- t_{mem} označuje čas strávený přístupy do paměti

Existuje několik přístupů k řešení energetické náročnosti:

Architektonická optimalizace:

- Návrh energeticky efektivních architektur
- Redukce počtu parametrů
- Optimalizace využití paměti

Hardwarová akcelerace:

- Specializované čipy pro neuronové sítě
- Optimalizace výpočetních jednotek
- Efektivní správa paměti

Algoritmické optimalizace:

- Pruning (prořezávání) sítí
- Kvantizace vah
- Distilace znalostí

3.8.2 Kvantizace modelů

Kvantizace představuje významnou techniku pro kompresi a optimalizaci neuronových sítí, která je zvláště důležitá pro nasazení na embedded systémech a mobilních zařízeních [22]. Proces kvantizace transformuje původní hodnoty vah s plovoucí desetinnou čárkou na celočíselné hodnoty s nižší bitovou přesností:

$$w_q = \text{round}\left(\frac{w - w_{\min}}{w_{\max} - w_{\min}} \cdot (2^n - 1)\right) \tag{3.37}$$

kde:

- w_q je kvantizovaná hodnota váhy
- w reprezentuje původní hodnotu váhy
- $w_{\min}, w_{\max}$ jsou minimální a maximální hodnoty vah
- n označuje počet bitů pro reprezentaci hodnoty

Jacob et al. [23] identifikují několik klíčových výhod kvantizace:

Redukce paměťových nároků:

- Menší velikost modelu
- Efektivnější využití cache paměti
- Nižší energetická náročnost

Zrychlení inference:

- Rychlejší aritmetické operace
- Efektivnější využití hardware
- Možnost specializované hardwarové akcelerace

Energetická efektivita:

- Nižší spotřeba při výpočtech
- Redukce paměťových přístupů
- Celkově nižší energetická náročnost

3.8.3 Interpretovatelnost neuronových sítí

Problém interpretovatelnosti neuronových sítí je jednou z klíčových výzev v oblasti umělé inteligence. Ribeiro et al. [24] představili metodu LIME pro vysvětlování predikcí libovolného klasifikátoru. Pro analýzu chování sítě byla vyvinuta řada technik, přičemž jednou z nejvýznamnějších je metoda saliency map [25]:

$$\text{Saliency Map} = \left| \frac{\partial \mathbf{y}}{\partial \mathbf{x}} \right| \tag{3.38}$$

kde:

- $\mathbf{y}$ představuje výstup sítě
- $\mathbf{x}$ je vstupní obraz
- $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$ reprezentuje gradient výstupu vzhledem ke vstupu

V oblasti interpretace neuronových sítí bylo vyvinuto několik hlavních přístupů [24, 25]:

Vizualizační techniky: [25]

- aktivační mapy
- feature vizualizace
- attention mapy
- gradient-based saliency mapy

Post-hoc vysvětlení: [24]

- LIME (Local Interpretable Model-agnostic Explanations)
- SHAP (SHapley Additive exPlanations)
- analýza důležitosti příznaků

Interpretovatelné architektury:

- attention mechanismy
- capsule networks
- decision trees z neuronových sítí

3.8.4 Kontinuální učení

Kontinuální učení představuje jeden z klíčových směrů vývoje v oblasti neuronových sítí, který se zaměřuje na schopnost systému průběžně získávat nové znalosti při zachování již naučených dovedností [26]. Tento přístup se snaží překonat fundamentální omezení tradičního jednorázového trénování a přiblížit se způsobu, jakým se učí biologické systémy.

Zásadním problémem v oblasti kontinuálního učení je fenomén tzv. katastrofického zapomínání (catastrophic forgetting), kdy neuronová síť při učení nových úloh ztrácí schopnost řešit úlohy předchozí [27]. Tento jev představuje významnou překážku pro vytvoření skutečně adaptivních systémů umělé inteligence.

Pro řešení problému katastrofického zapomínání byla navržena řada přístupů, mezi nimiž vyniká metoda elastické váhové regularizace (Elastic Weight Consolidation, EWC) představená Kirkpatrickem et al. [27]. Tato technika zavádí dodatečný regularizační člen do chybové funkce:

$$L = L_{\text{current}} + \lambda \sum_i F_i (w_i - w_i^*)^2 \tag{3.39}$$

kde L_{current} reprezentuje chybu na současné úloze, w_i jsou aktuální váhy sítě, w_i^* představují váhy optimalizované pro předchozí úlohy, F_i je Fisher informační matice indikující důležitost jednotlivých vah a λ je regularizační parametr určující sílu zachování předchozích znalostí.

Chen a Liu [26] identifikují tři hlavní výzvy v oblasti kontinuálního učení:

Správa paměti: Efektivní správa paměťových zdrojů představuje kritický aspekt kontinuálního učení [26]:

- Selektivní ukládání klíčových vzorů a příkladů
- Optimální vyvažování mezi uchováním starých a získáváním nových znalostí
- Dynamická adaptace kapacity modelu podle komplexity úloh

Transfer znalostí: Přenos znalostí mezi různými doménami a úlohami vyžaduje sofistikované mechanismy [28]:

- Identifikace a extrakce přenositelných znalostí

- Adaptace naučených reprezentací na nové domény
- Efektivní kombinace znalostí z různých zdrojů

Meta-učení: Hospedales et al. [28] zdůrazňují význam meta-učení pro kontinuální adaptaci:

- Vývoj strategií pro efektivní učení nových konceptů
- Automatická adaptace hyperparametrů a architektury
- Optimalizace procesu učení na základě předchozích zkušeností

Tyto výzvy společně formují komplexní problém, jehož řešení je klíčové pro vývoj skutečně adaptivních systémů umělé inteligence. Současný výzkum se zaměřuje na kombinaci různých přístupů, včetně neuroplasticity inspirované biologickými systémy, adaptivních architektur a pokročilých optimalizačních technik [28].

3.9 Budoucí směry vývoje

Budoucí směry vývoje neuronových sítí se formují na základě současných výzev a nových technologických možností. Vývoj ubírá několika hlavními směry, které kombinují poznatky z neurověd, kvantové fyziky a klasického strojového učení.

3.9.1 Neuromorphic Computing

Neuromorphic computing představuje revoluci v oblasti výpočetní techniky, která překonává omezení von Neumannovy architektury implementací biologicky inspirovaných mechanismů zpracování informací [29]. Tento přístup využívá synaptic memristors a synaptic transistors - nanoelektronická zařízení, která dokáží napodobit funkci biologických synapsí lidského mozku.

Klíčovým konceptem je modelování synaptického proudu:

$$I_{\text{syn}}(t) = \sum_i w_i \sum_j \alpha(t - t_j^i) \tag{3.40}$$

kde:

- $I_{\text{syn}}(t)$ představuje synaptický proud v čase t
- w_i jsou synaptické váhy
- $\alpha(t)$ je funkce popisující tvar postsynaptického proudu
- t_j^i jsou časy příchodu spike signálů

Na základě nejnovějších výzkumů [29] přináší neuromorphic computing několik zásadních výhod:

Energetická efektivita:

- Ultra-nízká spotřeba energie v řádu femtojoulů na synaptickou událost
- Vysokorychlostní paralelní zpracování dat
- Překonání von Neumannova bottlenecku díky sloučení paměti a výpočetní jednotky

Biologická věrnost:

- Přímá emulace synaptické plasticity pomocí memristivních zařízení
- Schopnost implementace jak krátkodobé (STP), tak dlouhodobé (LTP) plasticity
- Biologicky věrné zpracování temporální informace

Současný výzkum se zaměřuje zejména na vývoj nových materiálů pro synaptic devices, včetně kvantových teček (QDs), jednorozměrných nanostruktur, dvourozměrných nanomateriálů a trojrozměrných architektur [29]. Tyto materiály umožňují vytvoření zařízení s lepší stabilitou, nižší energetickou náročností a vyšší hustotou integrace.

3.9.2 Kvantové neuronové sítě

Kvantové neuronové sítě představují průnik mezi kvantovým počítáním a neuronovými sítěmi [30]. Základním stavebním prvkem je kvantový neuron, který je popsán unitární transformací:

$$|\psi_{\text{out}}\rangle = \hat{U}(\mathbf{w})|\psi_{\text{in}}\rangle \tag{3.41}$$

kde:

- $|\psi_{\text{in}}\rangle$ je vstupní kvantový stav
- $|\psi_{\text{out}}\rangle$ je výstupní kvantový stav
- $U(\mathbf{w})$ je unitární operátor závislý na vahách

Peters et al. [30] identifikují několik potenciálních výhod kvantových neuronových sítí:

Výpočetní výhody:

- kvantový paralelismus
- efektivnější zpracování určitých typů úloh

- nové možnosti optimalizace

Nové architektury:

- hybridní kvantově-klasické modely
- kvantové konvoluční sítě
- kvantové rekurentní sítě

3.9.3 Hybridní architektury

Hybridní architektury kombinují neuronové sítě s jinými přístupy strojového učení nebo klasickými algoritmy [3]. Základní model můžeme vyjádřit jako:

$$f_{\text{hybrid}}(\mathbf{x}) = \alpha f_{\text{NN}}(\mathbf{x}) + (1 - \alpha) f_{\text{other}}(\mathbf{x}) \tag{3.42}$$

kde:

- f_{NN} je výstup neuronové sítě
- f_{other} je výstup alternativního modelu
- α je váhový koeficient

Literatura identifikuje několik klíčových směrů vývoje hybridních architektur [4]:

Kombinace s klasickými algoritmy:

- integrace expertních systémů
- hybridní optimalizační metody
- kombinace s statistickými modely

Multi-modální systémy:

- kombinace různých typů vstupů
- integrace různých reprezentací dat
- fúze různých zdrojů informací

Adaptivní architektury:

- dynamické přepínání mezi modely
- kontextově závislé rozhodování
- automatická volba optimální strategie

Tyto směry vývoje naznačují, že budoucnost neuronových sítí bude charakterizována větší integrací s jinými přístupy a technologiemi, přičemž důraz bude kladen na efektivitu, interpretovatelnost a praktickou použitelnost v reálných aplikacích [4].

3.10 Závěr

Neuronové sítě představují dynamicky se rozvíjející oblast výzkumu a vývoje v rámci umělé inteligence. Tato kapitola představila komplexní pohled na základní principy, matematické modely a praktické aspekty implementace neuronových sítí. Z prezentovaného materiálu vyplývá několik klíčových poznatků.

Za prvé, matematický model umělého neuronu úspěšně abstrahuje základní principy biologických neuronů do formy vhodné pro výpočetní zpracování. Tento model, ačkoliv značně zjednodušený oproti své biologické předloze, poskytuje dostatečnou flexibilitu a výpočetní kapacitu pro řešení široké škály praktických problémů [1].

Za druhé, vývoj algoritmů učení, zejména algoritmu zpětného šíření chyby, umožnil efektivní trénování vícevrstvých neuronových sítí [8]. Tento pokrok otevřel cestu k řešení komplexních úloh v oblasti počítačového vidění, zpracování přirozeného jazyka a dalších oblastech.

Za třetí, moderní architektury neuronových sítí, jako jsou konvoluční a rekurentní sítě, demonstrují schopnost adaptace základních principů na specifické typy úloh. Tyto specializované architektury významně rozšiřují aplikační možnosti neuronových sítí [12].

Současně je třeba zdůraznit, že oblast neuronových sítí čelí několika významným výzvám. Mezi nejvýznamnější patří:

- Energetická náročnost trénování a provozu komplexních modelů
- Potřeba větší interpretovatelnosti a vysvětlitelnosti modelů
- Omezení plynoucí z požadavků na velké množství trénovacích dat

Budoucí vývoj v oblasti neuronových sítí bude pravděpodobně směřovat k řešení těchto výzev prostřednictvím nových architektur, efektivnějších algoritmů učení a hybridních přístupů kombinujících neuronové sítě s jinými metodami umělé inteligence [2].

Význam neuronových sítí pro budoucnost umělé inteligence nelze podceňovat. Jejich schopnost učit se z dat a adaptovat se na nové úlohy je činí klíčovým nástrojem pro řešení komplexních problémů v mnoha oblastech lidské činnosti. S pokračujícím výzkumem a vývojem lze očekávat další rozšíření jejich možností a aplikačních oblastí.

V kontextu této práce představují neuronové sítě teoretický základ pro praktickou realizaci popsanou v následujících kapitolách. Pochopení základních principů a matematických modelů je nezbytné pro efektivní návrh a realizaci konkrétních řešení využívajících neuronové sítě.

Kapitola 4

Resonite a vizuální programování

Tato kapitola popisuje platformu Resonite se zaměřením na její systém vizuálního programování ProtoFlux. Kromě představení samotné platformy a jejích základních funkcí se kapitola detailně věnuje principům vizuálního programování, které budou následně využity pro implementaci vizualizace neuronových sítí. Důraz je kladen na pochopení možností a omezení systému ProtoFlux, který představuje klíčový nástroj pro realizaci interaktivních 3D aplikací v prostředí Resonite [31].

4.1 Resonite

Resonite [31] je sociální platforma pro virtuální realitu vyvinutá studiem Yellow Dog Man Studios, která klade důraz na kreativitu, kolaboraci a vizuální programování. Na rozdíl od tradičních vývojových prostředí pro VR aplikace umožňuje Resonite vytvářet a programovat objekty přímo ve virtuálním prostředí. Výhodou platformy je její dostupnost jak pro VR headsety, tak pro klasické počítače v desktop módu, což významně usnadňuje proces vývoje a testování aplikací.

4.1.1 Základní koncepty

Resonite organizuje virtuální objekty pomocí hierarchické struktury slotů, které představují základní stavební bloky pro organizaci scény. Každý slot může obsahovat další sloty nebo komponenty, čímž vzniká přehledná stromová struktura virtuálního světa. Komponenty definují vlastnosti a chování objektů, přičemž systém podporuje širokou škálu funkcí od základních vizuálních prvků až po komplexní interaktivní chování.

4.1.2 Asset systém

Resonite poskytuje robustní systém pro práci s různými typy assetů, který je klíčový pro vývoj komplexních aplikací. Tento systém umožňuje import, správu a dynamické načítání různých typů médií a 3D objektů. Podporované jsou standardní formáty 3D modelů, textury, audio soubory i video obsah. Zvláštní pozornost je věnována optimalizaci assetů pro VR prostředí, kde je kritická rychlost načítání a efektivní správa paměti.

Důležitou součástí asset systému je také možnost sdílení a znovupoužití vytvořených objektů napříč různými virtuálními světy. Tento přístup podporuje kolaborativní aspekt platformy a umožňuje vytvářet rozsáhlé knihovny znovupoužitelných komponent.

4.2 ProtoFlux - systém vizuálního programování

ProtoFlux představuje jádro vývojového prostředí Resonite a nabízí přístup k vizuálnímu programování v prostředí virtuální reality. Na rozdíl od tradičních textových programovacích jazyků využívá ProtoFlux grafické reprezentace programových struktur, které lze manipulovat přímo v 3D prostoru. Tento přístup činí programování intuitivnějším a přístupnějším i pro uživatele bez rozsáhlých programátorských zkušeností.

4.2.1 Architektura ProtoFluxu

Základním stavebním kamenem ProtoFluxu jsou uzly (nodes), které reprezentují různé programové elementy - od jednoduchých matematických operací přes práci s 3D transformacemi až po komplexní logické struktury. Uzly jsou propojeny pomocí vazeb (connections), které definují tok dat a řízení v programu [31].

Systém rozlišuje několik základních typů uzlů:

- **Výpočetní uzly** provádějí matematické a logické operace
- **Stavové uzly** uchovávají data a stav aplikace
- **Událostní uzly** reagují na interakce a systémové události
- **Řídící uzly** implementují programové struktury jako podmínky a cykly

Každý uzel může mít několik typů portů:

- **Vstupní porty** přijímají data pro zpracování
- **Výstupní porty** poskytují výsledky operací
- **Impulsní porty** slouží k řízení toku programu
- **Konfigurační porty** umožňují nastavení parametrů uzlu

4.2.2 Datový tok a synchronizace

ProtoFlux implementuje model datového toku (dataflow), kde se změny hodnot automaticky propagují systémem a vyvolávají přepočet závislých uzlů. Tento přístup zajišťuje konzistenci dat a zjednodušuje vývoj reaktivních aplikací. Systém také poskytuje mechanismy pro řízení toku výpočtů pomocí impulsů, které umožňují explicitní kontrolu nad časováním operací [32].

4.2.3 Události a interaktivita

Významnou součástí systému ProtoFlux je práce s událostmi, které umožňují reagovat na interakce uživatelů a systémové změny. Události v ProtoFluxu lze rozdělit do několika kategorií:

Uživatelské události zahrnují přímé interakce s VR kontrolery nebo myší a klávesnicí v desktop módu. Systém dokáže rozpoznávat širokou škálu gest a pohybů, což umožňuje vytvářet přirozené a intuitivní uživatelské rozhraní. Zpracování těchto událostí je klíčové pro vytváření interaktivních aplikací ve virtuální realitě.

Systémové události souvisejí s životním cyklem objektů a aplikace. Patří mezi ně například události inicializace, aktualizace a destrukce objektů. Tyto události jsou důležité pro správnou správu zdrojů a synchronizaci stavu aplikace.

4.2.4 Datové typy a konverze

ProtoFlux podporuje širokou škálu datových typů, které jsou optimalizovány pro práci ve virtuální realitě.

Základní datové typy zahrnují běžné programovací typy jako jsou čísla (celá čísla a čísla s plovoucí desetinnou čárkou), logické hodnoty a textové řetězce. Tyto typy jsou doplněny o specializované typy pro práci s 3D grafikou, jako jsou vektory, matice a quaterniony.

Systém také poskytuje pokročilé datové struktury včetně polí, seznamů a slovníků. Zvláštní pozornost je věnována typům pro práci s 3D transformacemi a prostorovými vztahy, které jsou klíčové pro vývoj VR aplikací.

4.2.5 Optimalizace a výkon

ProtoFlux implementuje několik optimalizačních strategií pro zajištění plynulého běhu aplikací ve virtuální realitě:

Lazy evaluation zajišťuje, že výpočty jsou prováděny pouze když jsou skutečně potřeba. Tento přístup je zvláště důležitý pro složité výpočetní grafy, ve kterých by naivní implementace mohla vést k výkonnostním problémům.

Systém také poskytuje mechanismy pro cachování výsledků a minimalizaci přepočtů v situacích, kdy se vstupní hodnoty nemění. Tato optimalizace je

kritická pro udržení stabilního snímkového kmitočtu, který je ve VR aplikacích zásadní pro komfort uživatelů.

4.3 Praktické aspekty vývoje v ProtoFluxu

Vývoj aplikací v prostředí ProtoFlux vyžaduje specifický přístup, který se liší od tradičního programování. Vývojáři musí zvládnout několik klíčových konceptů a technik, které jsou specifické pro vizuální programování ve virtuální realitě.

4.3.1 Správa verzí a kolaborace

Resonite poskytuje základní nástroje pro správu verzí a kolaborativní vývoj:

Systém zálohování umožňuje ukládat změny v projektu a pokud je vytvořená kopie i návrat k předchozím verzím. Tento mechanismus je důležitý pro experimentování s různými přístupy k implementaci.

Multi-user editing umožňuje více vývojářům pracovat současně na stejném projektu. Systém zajišťuje synchronizaci změn a řešení konfliktů v reálném čase.

4.3.2 Integrace s externími systémy

ProtoFlux poskytuje několik mechanismů pro komunikaci s externími systémy a službami [32]:

HTTP komunikace umožňuje interakci s webovými službami a API. Systém poskytuje specializované uzly pro vytváření HTTP požadavků a zpracování odpovědí, včetně podpory pro různé metody (GET, POST).

WebSocket komunikace je podporována pro aplikace vyžadující real-time komunikaci. Tato funkcionalita je zvláště užitečná pro vytváření interaktivních aplikací s přímou zpětnou vazbou.

File system integrace umožňuje číst a zapisovat soubory, což je důležité pro ukládání a načítání konfiguračních dat nebo assetů.

4.4 Limitace a výzvy

Při vývoji v prostředí ProtoFlux je třeba brát v úvahu několik omezení a výzev:

4.4.1 Výkonnostní omezení

Vizuální programování může vést k výkonnostním problémům při práci s komplexními systémy. Velké množství uzlů a propojení může způsobit zpomalení systému, zejména v případech, kdy není správně implementována optimalizace výpočtů.

4.4.2 Správa komplexity

S rostoucí složitostí aplikace se může stát vizuální reprezentace programu nepřehlednou. Je důležité používat správné návrhové vzory a organizační principy pro udržení čitelnosti a udržovatelnosti kódu.

4.4.3 Vývojové nástroje

Ačkoliv ProtoFlux poskytuje pokročilé možnosti pro vývoj aplikací, některé běžné vývojářské nástroje (jako pokročilý profiling nebo automatizované testování) nejsou v současné době podporovány.

4.5 Obdobné projekty

V současné době existuje v prostředí Resonite pouze jediný projekt, zabývající se implementací neuronových sítí. Jedná se o reimplementaci klasické arkádové hry Breakout od Atari, která využívá neuronovou síť pro řízení pohybu herní platformy viz obr 4.1. Tento projekt demonstruje praktické využití strojového učení v interaktivním prostředí virtuální reality.

Obrázek 4.1 Implementace hry Pong s využitím neuronové sítě pro ovládání herní platformy.

Implementace hry využívá jednoduchý model neuronové sítě, který na základě pozice míčku predikuje optimální pohyb platformy. Tento svět představuje zajímavý příklad aplikace strojového učení v herním prostředí a demonstruje potenciál využití neuronových sítí v kontextu virtuální reality. Nízký počet podobných projektů v Resonite současně naznačuje, že oblast neuronových sítí v tomto prostředí představuje příležitost pro další výzkum a vývoj.

4.6 Závěr

ProtoFlux představuje přístup k vývoji aplikací ve virtuální realitě, který kombinuje intuitivnost vizuálního programování s možnostmi moderních vývojových nástrojů. Systém je zvláště vhodný pro rychlý vývoj prototypů a interaktivních aplikací, kde je důležitá přímá vizuální zpětná vazba.

Pochopení principů práce v ProtoFluxu je klíčové pro efektivní implementaci vizualizace neuronových sítí v prostředí virtuální reality. Systém poskytuje potřebné nástroje pro vytváření interaktivních 3D reprezentací a zpracování datových struktur.

V kontextu této práce tvoří ProtoFlux základní platformu pro implementaci vizualizace neuronových sítí, která bude detailněji popsána v následujících kapitolách. Jeho možnosti v oblasti vizuálního programování a 3D manipulace umožňují vytvořit intuitivní a interaktivní reprezentaci neuronových sítí ve virtuálním prostředí.

Kapitola 5

Webová aplikace

Tato kapitola popisuje technickou implementaci distribuovaného systému pro vizualizaci a konfiguraci neuronových sítí. Platforma využívá mikroservisní architekturu postavenou na technologiích React a FastAPI, přičemž jednotlivé komponenty jsou nasazeny v prostředí Google Cloud Platform. Systém implementuje techniky vizualizace neuronových sítí v reálném čase, včetně interaktivní manipulace s váhami a přenosu dat mezi jednotlivými vrstvami aplikace. Kapitola detailně rozebírá architekturu systému, specifické implementační přístupy jednotlivých komponent a matematické principy využité při realizaci výpočetního jádra neuronové sítě.

5.1 Architektura systému

Implementovaný systém využívá moderní distribuovanou architekturu založenou na principech událostmi řízeného návrhu (event-driven design) a reaktivního programování. Zvolená architektura umožňuje efektivní škálování jednotlivých komponent a zajišťuje vysokou dostupnost celého systému. Aplikace je rozdělena do tří hlavních domén: prezentační vrstvy implementované v prostředí Next.js, aplikační vrstvy realizované pomocí FastAPI frameworku a perzistentní vrstvy využívající služby Google Cloud Storage.

5.1.1 Principy architektonického návrhu

Při návrhu systému byly implementovány následující architektonické principy, které zajišťují dlouhodobou udržitelnost a rozšiřitelnost aplikace:

Doménově orientovaná architektura: Systém je rozdělen do jasně oddělených domén, kde každá doména zapouzdřuje specifickou funkčnost. Prezentační doména se zaměřuje výhradně na vizualizaci a interakci s uživatelem, aplikační doména implementuje business logiku a výpočetní algoritmy, zatímco perzistentní doména zajišťuje správu dat a jejich dlouhodobé ukládání.

Reaktivní programování: Řešení využívá reaktivní přístup v obou hlavních komponentách systému. Na frontendu je použit React s hooks API, zatímco backend implementuje asynchronní zpracování pomocí FastAPI. Toto řešení bylo zvoleno pro:

- Zajištění plynulé uživatelské zkušenosti při práci s interaktivními vizualizacemi
- Efektivní zpracování asynchronních operací při trénování neuronových sítí
- Minimalizaci latence při aktualizacích uživatelského rozhraní
- Optimalizaci využití systémových prostředků při náročných výpočtech

Mikroservisní dekompozice: Systém je rozdělen do dvou hlavních mikroservis - frontend služby realizované v Next.js a backend služby postavené na FastAPI. Toto rozdělení bylo zvoleno především kvůli:

- Možnosti nezávislého škálování jednotlivých komponent podle aktuální zátěže
- Zjednodušení vývoje a údržby díky jasně definovaným rozhraním mezi službami
- Možnosti využití různých technologií optimalizovaných pro konkrétní úlohy
- Lepší izolaci případných chyb a zvýšení celkové robustnosti systému

5.1.2 Implementace cloudové infrastruktury

Systém je nasazen v prostředí Google Cloud Platform, které poskytuje komplexní sadu služeb pro provoz distribuovaných aplikací. Řešení využívá následující klíčové komponenty:

Kontejnerová orchestrace: Aplikační komponenty jsou nasazeny pomocí služby Cloud Run, která implementuje serverless přístup ke spouštění kontejnerizovaných aplikací. Tato služba automaticky škáluje počet instancí podle aktuálního zatížení a zajišťuje vysokou dostupnost systému.

Perzistentní úložiště: Pro ukládání konfigurací neuronových sítí a tréninkových dat je implementováno řešení postavené na službě Cloud Storage. Tato služba poskytuje geograficky redundantní úložiště s vysokou dostupností.

Continuous Deployment: Systém implementuje automatizované nasazování pomocí služby Cloud Build. Tento proces zahrnuje sestavení Docker image, provedení automatizovaných testů a nasazení aplikace do produkčního prostředí.

5.1.3 Kontejnerizace aplikačních komponent

Implementace využívá kontejnerizaci pro zajištění konzistentního běhového prostředí a zjednodušení procesu nasazení. Kontejnerizace je realizována pomocí vícevrstvého přístupu, který optimalizuje velikost výsledných image a zrychluje proces sestavení.

5.2 Implementace backendu

Backend systému realizuje aplikační vrstvu postavenou na frameworku FastAPI. Tato technologie byla zvolena především kvůli její schopnosti efektivně zpracovávat asynchronní požadavky a poskytovat vysoký výkon při práci s konfiguracemi neuronových sítí. Framework také nabízí integrovanou podporu pro OpenAPI specifikaci, což významně usnadňuje dokumentaci a testování API rozhraní.

5.2.1 Datové modely

Jádro systému je postaveno na typovaných datových modelech implementovaných pomocí knihovny Pydantic. Tato knihovna zajišťuje automatickou validaci dat a serializaci objektů. Hlavní datový model NetworkConfiguration zapouzdřuje kompletní konfiguraci neuronové sítě včetně její architektury, vah a tréninkových parametrů. Implementované datové modely zajišťují typovou bezpečnost a automatickou validaci vstupních dat. Každý model obsahuje podrobné validační pravidla, která kontrolují konzistenci dat a dodržování stanovených omezení. Systém například automaticky validuje strukturu neuronové sítě, konzistenci počtu vah mezi vrstvami, platnost hodnot hyperparametrů a kompatibilitu vstupních a výstupních dimenzí.

5.2.2 Perzistentní úložiště

Pro ukládání konfigurací neuronových sítí systém využívá službu Google Cloud Storage. Řešení zahrnuje specializovanou třídu StorageManager, která zapouzdřuje veškerou logiku pro práci s cloud úložištěm. Třída implementuje asynchronní metody pro ukládání a načítání konfigurací s důrazem na efektivitu a spolehlivost. Systém využívá dedikovaný bucket neural-network-config pro ukládání dat. Každá konfigurace je uložena jako samostatný JSON objekt s kompresí gzip pro minimalizaci přenosových dat. Implementace zahrnuje také systém cachování pro optimalizaci často načítaných konfigurací a automatické čištění zastaralých dat.

5.2.3 API Endpointy

Backend systém implementuje REST API rozhraní, jehož klíčovým endpointem je /config s volitelným parametrem format. Ačkoliv endpoint podporuje čtyři různé formáty odpovědí (json, csv, tsv a simple) pro případné rozšíření aplikace, v produkčním nasazení se využívá především formát simple, který byl optimalizován pro integraci s Resonite. Formát simple byl vyvinut na základě testování různých způsobů reprezentace dat v prostředí virtuální reality. Tento formát využívá kompaktní serializaci s oddělovači pro efektivní přenos konfigurace neuronové sítě včetně vah, aktivačních funkcí a výstupních souřadnic.

5.2.4 Optimalizace výkonu

Backend implementuje několik optimalizací pro zajištění efektivního zpracování požadavků:

- **Asynchronní zpracování** - Systém využívá asynchronní zpracování I/O operací prostřednictvím FastAPI frameworku při práci s Cloud Storage, což umožňuje efektivní obsluhu více současných požadavků.
- **Efektivní serializace** - Pro různé formáty výstupu (JSON, CSV, TSV, simple) jsou implementovány optimalizované serializační mechanismy.
- **Error handling** - Systém implementuje robustní zpracování chyb s automatickým logováním pro zajištění spolehlivosti služby.

5.2.5 Zabezpečení a monitoring

Backend implementuje základní mechanismy pro monitoring a zabezpečení:

- **Logging** - Systém využívá standardní Python logging modul pro zaznamenávání klíčových operací a chyb:
  - Logování přístupů k API endpointům
  - Zaznamenávání chyb při práci s Cloud Storage
  - Základní diagnostické informace pro debugování
- **CORS a zabezpečení** - Aplikace implementuje základní CORS (Cross-Origin Resource Sharing) konfiguraci a využívá zabezpečení poskytované platformou Cloud Run:
  - Povolení cross-origin požadavků pro webovou aplikaci
  - HTTPS zabezpečení zajištěné službou Cloud Run
  - Automatická správa SSL certifikátů

5.3 Implementace frontendu

Frontend aplikace představuje systém postavený na frameworku Next.js s TypeScriptem, který realizuje komplexní vizualizace neuronových sítí a techniky interaktivní manipulace s jejich parametry. Architektura je navržena s důrazem na modularitu a uživatelskou přívětivost, přičemž využívá moderní přístupy k vývoji webových aplikací včetně reaktivního programování a komponentové architektury.

5.3.1 Architektura komponent

Systém implementuje hierarchickou strukturu komponent, kde každá komponenta má jasně definovanou zodpovědnost a rozhraní. Centrálním prvkem je komponenta NetworkLayout, která řídí celkové rozvržení aplikace a implementuje přepínání mezi konfigurační a dokumentační částí systému. Tato komponenta využívá systém podmíněného renderování založený na React hooks pro efektivní správu stavu aplikace. Architektura je organizována do několika klíčových vrstev:

- **Prezentační vrstva** zahrnuje komponenty pro vizualizaci sítě a uživatelské rozhraní
- **Logická vrstva** implementuje výpočetní jádro neuronové sítě a správu stavu
- **Datová vrstva** zajišťuje komunikaci s backend API a perzistenci dat

Pro zajištění konzistentního vzhledu a chování je realizován vlastní systém UI komponent postavený na knihovně shadcn/ui, který poskytuje sadu znovupoužitelných prvků optimalizovaných pro práci s neuronovými sítěmi.

5.3.2 State Management

Systém realizuje přístup ke správě stavu pomocí React hooks, přičemž klíčovou roli hraje hook useNetworkConfiguration. Tento hook zapouzdřuje veškerou logiku související s konfigurací a stavem neuronové sítě, včetně:

- Správy konfigurace sítě (vrstvy, váhy, biasy)
- Řízení procesu trénování a optimalizace
- Synchronizace stavu mezi komponentami
- Persistence stavu a komunikace s backend API

Hook implementuje komplexní systém pro správu side-effects pomocí useEffect, který zajišťuje:

- Automatické načtení konfigurace při inicializaci
- Aktualizaci vizualizací při změnách konfigurace
- Správu životního cyklu trénování
- Optimalizaci výkonu pomocí debouncing a throttling

Pro zajištění konzistence dat napříč aplikací systém využívá immutable přístup ke správě stavu:

- Veškeré změny stavu jsou prováděny pomocí atomických operací
- Každá aktualizace vytváří novou instanci stavu
- Implementace využívá strukturální sdílení pro optimalizaci paměti
- Změny stavu jsou propagovány pomocí reactive patterns

5.3.3 Implementace neuronové sítě

Tato subsekce popisuje realizaci interaktivní neuronové sítě v prostředí webového prohlížeče, která tvoří jádro systému pro vizualizaci a experimentování s neuronovými sítěmi. Implementace je realizována v TypeScriptu s využitím React frameworku a je optimalizována pro real-time interakci s uživatelem. Systém umožňuje dynamickou konfiguraci architektury sítě, vizualizaci procesu učení a okamžitou zpětnou vazbu při změnách parametrů. Zvláštní důraz je kladen na vizuální aspekt a intuitivní reprezentaci transformací mezi dvourozměrným a trojrozměrným prostorem, což umožňuje uživatelům lépe porozumět chování neuronové sítě.

Základní architektura a konfigurace

Systém implementuje flexibilní architekturu neuronové sítě, která je plně konfigurovatelná prostřednictvím uživatelského rozhraní. Základní omezení architektury vycházejí z požadavků na vizualizaci:

- Vstupní vrstva je fixně nastavena na 2 neurony pro zpracování 2D vstupních bodů
- Výstupní vrstva obsahuje vždy 3 neurony pro generování bodů v 3D prostoru
- Počet skrytých vrstev je volitelný v rozsahu 0-3 (celkem se vstupní a výstupní vrstvout tedy 2-5 vrstev)
- Každá skrytá vrstva může obsahovat 1-10 neuronů

- Aktivační funkce je konfigurovatelná v uživatelském rozhraní, přičemž zvolená aktivační funkce je aplikována na všechny vrstvy sítě kromě výstupní vrstvy, která používá lineární aktivaci pro zachování rozsahu výstupních hodnot

Konfigurace sítě je implementována v komponentě NetworkControls, která zajišťuje:

- Validaci architektury a parametrů sítě
- Dynamickou aktualizaci vizualizace při změnách konfigurace
- Persistence stavu sítě pomocí React hooks
- Komunikaci s backend API pro ukládání a načítání konfigurací

Výpočetní jádro a proces trénování

Jádro systému implementuje komplexní proces trénování a evaluace neuronové sítě. Centrálními komponentami jsou dopředná propagace (forward propagation) pro výpočet výstupů a zpětná propagace chyby (backpropagation) pro aktualizaci vah. Systém zpracovává data v mini-batches pro optimální využití paměti a výpočetního výkonu.

Pro každou epochu trénování systém provádí následující kroky:

- Náhodné zamíchání trénovacích dat pro lepší generalizaci
- Rozdělení dat do mini-batches definované velikosti (1-1000)
- Pro každý batch:
  - Dopředná propagace pro výpočet predikcí
  - Výpočet chyby pomocí kombinované loss funkce
  - Zpětná propagace pro výpočet gradientů
  - Aktualizace vah a biasů pomocí optimalizačního algoritmu
- Aktualizace vizualizace každých 5 epoch

Dopředná propagace je realizována jako sekvence maticových operací dle definice uvedené v sekci Zpětná propagace chyby(3.21):

Systém podporuje čtyři typy aktivačních funkcí:

- ReLU: $f(x) = \max(0, x)$ - výchozí volba pro většinu aplikací
- Sigmoid: $f(x) = \frac{1}{1+e^{-x}}$ - vhodná pro binární klasifikace
- Tanh: $f(x) = \tanh(x)$ - alternativa k sigmoidu s výstupem v rozsahu [-1,1]
- Linear: $f(x) = x$ - používá se hlavně v poslední vrstvě pro regresi

Pro trénování sítě je realizována komplexní chybová funkce kombinující tři složky:

$$E_{\text{total}} = (1 - \alpha)E_{\text{MSE}} + \alpha E_{\text{geometric}} + \lambda E_{\text{reg}} \tag{5.1}$$

kde E_{MSE} je standardní střední kvadratická chyba, $E_{\text{geometric}}$ je geometrický člen zachovávající prostorové vztahy a E_{reg} je L2 regularizace.

Systém trénování a optimalizace

Pro trénování sítě je implementován pokročilý systém optimalizace založený na mini-batch stochastickém gradientním sestupu. Systém obsahuje několik klíčových optimalizačních mechanismů:

Adaptivní learning rate: Learning rate se dynamicky upravuje během trénování podle vzorce:

$$\eta_t = \eta_0 \cdot 0.998^t \tag{5.2}$$

kde η_0 je počáteční learning rate z rozsahu $[0.0001, 1.0]$. Toto exponenciální snižování pomáhá zajistit konvergenci při zachování rychlého učení v počátečních fázích.

Momentum s adaptivním koeficientem: Pro zlepšení konvergence systém používá momentum s proměnným koeficientem:

$$m_t = m_{\text{base}} + (1 - m_{\text{base}}) \cdot \frac{t}{T} \tag{5.3}$$

kde $m_{\text{base}} = 0.9$ a T je celkový počet epoch. Aktualizace vah pak probíhá podle:

$$v_t = m_t v_{t-1} + \eta_t \nabla E_t \tag{5.4}$$

$$w_t = w_{t-1} - v_t \tag{5.5}$$

Systém obsahuje následující klíčové vlastnosti:

- Velikost batch je konfigurovatelná v rozsahu 1-1000 (výchozí 32)
- Learning rate je adaptivní s exponenciálním poklesem
- Implementováno momentum s proměnným koeficientem
- Použita L2 regularizace pro prevenci přetrénování

Pro numerickou stabilitu jsou implementovány ochranné mechanismy:

- Gradient clipping pro prevenci explodujícího gradientu
- Dynamická úprava learning rate při detekci nestability
- Early stopping při dosažení dostatečně nízké chyby

Generátory trénovacích dat a transformace

Systém realizuje specializované generátory trénovacích dat pro testování různých aspektů neuronové sítě. Pro každou variantu vstupních dat je implementována odpovídající cílová transformace, která definuje požadovaný 3D výstup sítě. Počet generovaných vzorků je konfigurovatelný uživatelem v rozsahu 10-1000 bodů, s výchozí hodnotou 100 bodů.

Kruhový vzor a hemisféra: Tento generátor vytváří body v kruhovém uspořádání s nelineární distribucí hustoty bodů. Body jsou generovány pomocí polárních souřadnic:

$$\begin{aligned} x &= r\cos(\theta) + \epsilon_r\cos(\theta) \\ y &= r\sin(\theta) + \epsilon_r\sin(\theta) \end{aligned} \tag{5.6}$$

kde:

- $r = t^{0.7}$ pro $t \in [0, 1]$, což vytváří vyšší hustotu bodů blíže k okraji kruhu
- $\theta \in [0, 2\pi]$ je náhodný úhel
- $\epsilon_r = 0.03 \cdot \text{noise} \cdot \text{randUniform}(-1, 1)$ je radiální šum

Cílová transformace převádí 2D kruhový vzor na povrch polokoule:

$$z = \cos(r\pi), \quad \text{kde } r = \sqrt{x^2 + y^2} \tag{5.7}$$

Tato transformace vytváří hladký přechod od maximální výšky ve středu k nulové výšce na okraji kruhu. Body blízko středu jsou transformovány na vrchol polokoule, zatímco body na okraji kruhu zůstávají v nulové výšce.

Spirálový vzor a zvlněný povrch: Systém generuje spirálový vzor pomocí parametrických rovnic:

$$\begin{aligned} x &= r \cdot \sin(t) + \epsilon_x \\ y &= r \cdot \cos(t) + \epsilon_y \end{aligned} \tag{5.8}$$

kde:

- $r = i/n$ pro náhodně vybrané $i \in [0, n-1]$, kde $n = 100$ určuje jemnost vzorkování
- $t = 1.75 \cdot i/n \cdot 2\pi + d$ je úhlový parametr, kde faktor 1.75 určuje počet závitů spirály
- $d \in \{0, \pi\}$ je náhodně volený fázový posun vytvářející dvě vzájemně posunuté spirály
- ϵ_x, ϵ_y reprezentují náhodný šum z intervalu $[-0.1, 0.1]$ škálovaný parametrem šumu pro větší variabilitu vzoru

Cílová transformace vytváří komplexní zvlněný povrch:

$$z = \sin(\theta \cdot 3 + r \cdot 4), \quad \text{kde } \theta = \arctan 2(y, x) \tag{5.9}$$

Tato transformace produkuje povrch s periodickými vlnami, kde θ určuje úhlovou pozici vlny a r moduluje její frekvenci v závislosti na vzdálenosti od středu. Kombinace těchto složek vytváří charakteristický spirálovitý vzor ve výškové mapě.

Gaussovské shluky a vícevrcholový povrch: Body jsou generovány ve čtyřech shlucích pomocí Box-Mullerovy transformace:

$$\begin{aligned} x &= \mu_x + \sigma\sqrt{-2\ln(U_1)}\cos(2\pi U_2) \\ y &= \mu_y + \sigma\sqrt{-2\ln(U_1)}\sin(2\pi U_2) \end{aligned} \tag{5.10}$$

kde:

- (μ_x, μ_y) jsou centra shluků v bodech $\{(-0.5, -0.5), (0.5, -0.5), (-0.5, 0.5), (0.5, 0.5)\}$
- $\sigma = 0.2 + 0.3 \cdot \text{noise}$ je směrodatná odchylka určující rozptyl bodů
- $U_1, U_2 \sim \mathcal{U}(0, 1)$ jsou nezávislé rovnoměrně rozdělené náhodné veličiny

Cílová transformace vytváří povrch složený ze čtyř gaussovských kopců s různými výškami:

$$z = \sum_{i=1}^{4} h_i \exp(-3\sqrt{(x - x_i)^2 + (y - y_i)^2}) \tag{5.11}$$

kde hodnoty $h_i \in \{1.0, -1.0, 0.5, -0.5\}$ určují výšky jednotlivých vrcholů. Každý shluk vstupních bodů je transformován na 3D kopec s unikátní výškou, přičemž výška klesá exponenciálně se vzdáleností od centra kopce.

XOR vzor a binární povrch: Body jsou generovány ve čtyřech kvadrantech 2D prostoru s minimálním odstupem padding od os:

$$(x, y) = \begin{cases} (\text{randUnif}(-1, -p), \text{randUnif}(-1, -p)) & \text{III. kvadrant} \\ (\text{randUnif}(p, 1), \text{randUnif}(p, 1)) & \text{I. kvadrant} \\ (\text{randUnif}(-1, -p), \text{randUnif}(p, 1)) & \text{II. kvadrant} \\ (\text{randUnif}(p, 1), \text{randUnif}(-1, -p)) & \text{IV. kvadrant} \end{cases} \tag{5.12}$$

kde p je hodnota padding určující minimální vzdálenost bodů od os. Cílová výška je určena znaménkem součinu souřadnic:

$$z = \text{sign}(xy) \tag{5.13}$$

Tento vzor je důležitý pro testování schopnosti sítě naučit se ostré přechody a nelineární klasifikační hranice. Body v prvním a třetím kvadrantu (kde x a y mají stejné znaménko) jsou transformovány na výšku +1, zatímco body v druhém a čtvrtém kvadrantu (kde x a y mají opačná znaménka) jsou transformovány na výšku -1, což vytváří charakteristický binární 3D povrch.

5.3.4 Vizualizace neuronových sítí

Tato sekce detailně popisuje implementaci tří klíčových komponent pro vizualizaci neuronových sítí: vizualizaci architektury sítě, 2D vizualizaci vstupního datasetu a 3D vizualizaci výstupu sítě. Každá komponenta využívá specifické technologie a přístupy pro dosažení efektivní a interaktivní vizuální reprezentace. Systém je navržen s důrazem na modularitu a uživatelskou přívětivost, přičemž jednotlivé komponenty jsou integrovány do komplexního řešení prostřednictvím komponenty NetworkControls.

Vizualizace architektury neuronové sítě

Komponenta NetworkVisualization realizuje interaktivní vizuální reprezentaci struktury neuronové sítě prostřednictvím React a SVG. Systém dynamicky generuje a aktualizuje vizualizaci na základě konfigurace sítě, využívá efektivní správu stavu pomocí React hooks a implementuje pokročilé interaktivní funkce pro manipulaci s parametry sítě viz obr. 5.1.

Základní architektura komponenty zahrnuje systém dynamického výpočtu pozic a propojení jednotlivých elementů. Pro každou vrstvu neuronové sítě systém vypočítává optimální vertikální rozmístění neuronů podle následujícího principu:

$$y = \text{startY} + \text{neuronIndex} \cdot \text{verticalSpacing} \tag{5.14}$$

kde vertikální mezera je dynamicky počítána na základě počtu neuronů v největší vrstvě:

$$\text{verticalSpacing} = \min\left(\frac{\text{svgHeight} - 0.2 \cdot \text{svgHeight}}{\text{maxNeuronsInLayer}}, 0.1 \cdot \text{svgHeight}\right) \tag{5.15}$$

Pro horizontální pozicování vrstev systém realizuje rovnoměrné rozložení s dynamickým výpočtem mezer:

$$x = \text{startX} + \text{layerIndex} \cdot \text{layerSpacing} \tag{5.16}$$

kde layerSpacing je odvozeno od celkové šířky SVG a počtu vrstev:

$$\text{layerSpacing} = \frac{\text{svgWidth} - 2 \cdot \text{horizontalPadding}}{\text{maxLayers} - 1} \tag{5.17}$$

Klíčovým prvkem vizualizace je systém reprezentace vah mezi neurony pomocí zakřivených spojnic. Pro vytvoření vizuálně přehledných propojení systém využívá kvadratické Bézierovy křivky, kde krajní neurony mají největší zakřivení, aby vizualizace byla co nejpřehlednější. Kontrolní body křivek jsou vypočítávány s ohledem na pozice propojovaných neuronů:

$$\text{controlX} = \frac{\text{startX} + \text{endX}}{2} \tag{5.18}$$

$$\text{controlY} = \frac{\text{startY} + \text{endY}}{2} + \text{curveOffset} \tag{5.19}$$

kde curveOffset je dynamicky počítán na základě relativních pozic neuronů a celkové architektury sítě. Tento offset zajišťuje, že křivky se vzájemně minimálně překrývají a vytváří vizuálně příjemný tok informací sítí.

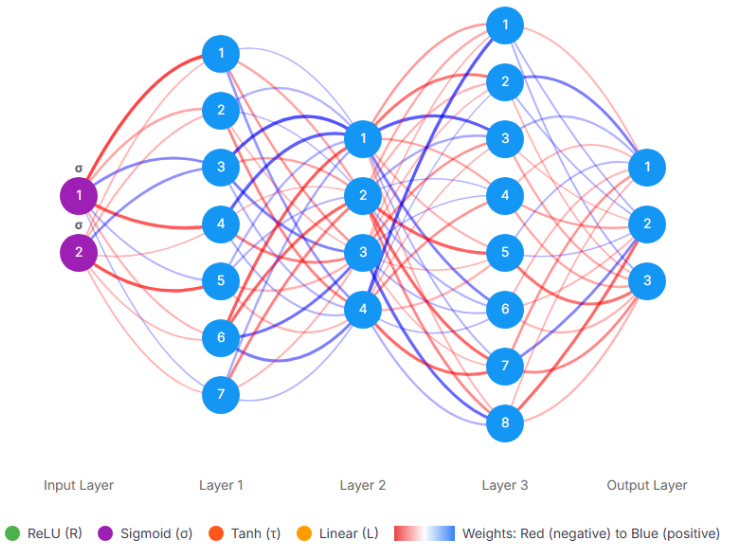
Pro vizuální reprezentaci vah systém implementuje systém barevného kódování a mapování tloušťky čar. Barva spojnice je určena funkcí getWeightColor, která realizuje následující logiku:

- Pro kladné váhy: odstíny modré s intenzitou úměrnou velikosti váhy
- Pro záporné váhy: odstíny červené s intenzitou úměrnou absolutní hodnotě váhy
- Průhlednost je dynamicky upravována pro lepší čitelnost při překryvu spojnic

Tloušťka spojnice je určena funkcí getWeightWidth:

$$\text{width} = \min(\max(|w| \cdot 2 + 1.5, 1.5), 4) \tag{5.20}$$

kde w je hodnota váhy. Toto mapování zajišťuje, že i velmi malé váhy jsou viditelné, zatímco velké váhy nepůsobí rušivě viz obr. 5.1.

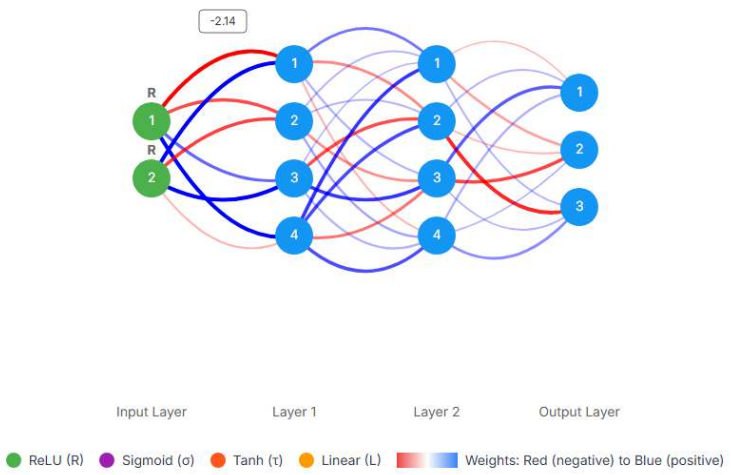


Obrázek 5.1 Ukázka vizualizace komplexnější neuronové sítě s více vrstvami.

Komponenta implementuje sofistikovaný systém interaktivity, který zahrnuje:

- Detekci hoveru nad spojnicemi s vizualizací přesné hodnoty vah viz obr. 5.2
- Editaci vah přímo v prostředí vizualizace viz obr. 5.3
- Dynamickou aktualizaci vizuálních elementů při změnách konfigurace
- Responsivní přizpůsobení velikosti a rozložení při změnách velikosti okna

Pro editaci vah systém implementuje specializovaný editor, který se zobrazuje jako překryvné okno na pozici kliknutí. Editor poskytuje přímý vstup pro numerickou hodnotu váhy s okamžitou vizuální zpětnou vazbou při změnách.

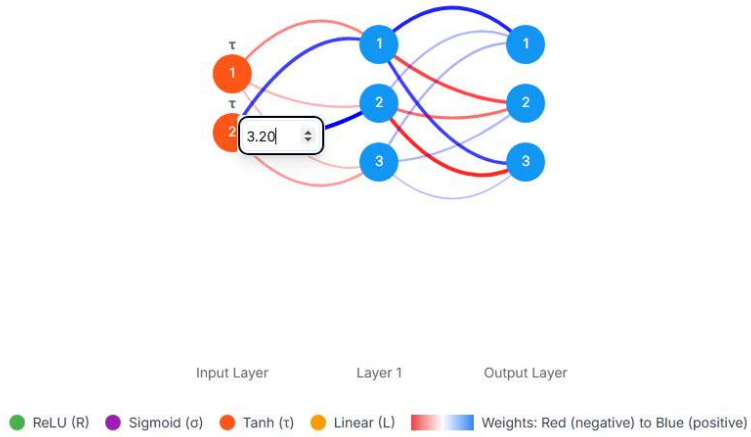


Obrázek 5.2 Ukázka vizualizace neuronové sítě s myší na spojnici zobrazující konkrétní váhu.

2D vizualizace vstupního datasetu

Komponenta NetworkVisualization2D poskytuje komplexní řešení pro vizualizaci vstupních dat ve dvojrozměrném prostoru viz obr. 5.4. Řešení je postaveno na knihovně Three.js, která umožňuje efektivní vykreslování a manipulaci s velkým množstvím bodů. Komponenta implementuje následující klíčové funkce:

- Automatické škálování zobrazení podle rozsahu vstupních dat
- Interaktivní ovládání pomocí OrbitControls s omezenou rotací pro 2D pohled
- Dynamický systém vykreslování bodů s optimalizací výkonu



Obrázek 5.3 Ukázka úpravy vah ve vizualizaci neuronové sítě.

- Responsivní přizpůsobení velikosti vizualizace

Systém implementuje sofistikovaný mechanismus pro inicializaci a správu Three.js scény. Při inicializaci je vytvořena ortografická kamera, která je optimální pro 2D zobrazení:

$$\text{camera} = \text{OrthographicCamera}(-\text{aspect} \cdot \text{zoom}, \text{aspect} \cdot \text{zoom}, \text{zoom}, -\text{zoom}) \tag{5.21}$$

kde aspect je poměr stran kontejneru a zoom je faktor přiblížení nastavený na hodnotu 2 pro optimální počáteční zobrazení.

Pro vytvoření referenčního rámce systém implementuje mřížku pomocí GridHelper, která je rotována do horizontální roviny:

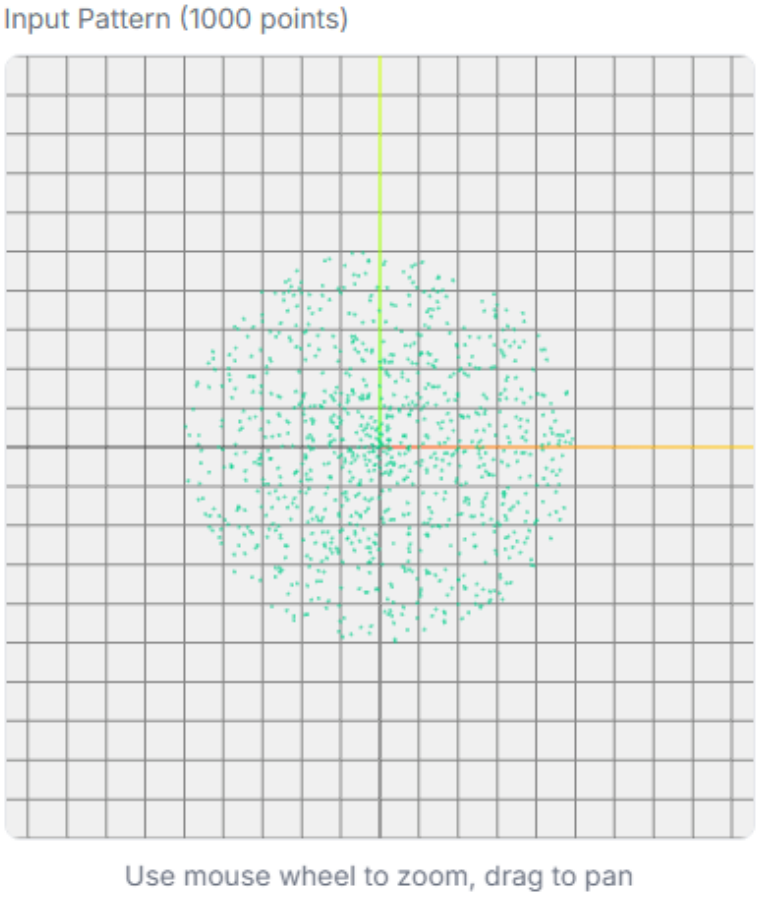
$$\text{gridHelper.rotation.x} = \frac{\pi}{2} \tag{5.22}$$

Body datasetu jsou vykreslovány pomocí optimalizované geometrie:

$$\text{geometry} = \text{CircleGeometry}(0.01, 16) \tag{5.23}$$

s materiálem definovaným následujícími parametry:

- Barva: 0x00cc88 (zeleno-modrá)
- Průhlednost: 0.6
- Aktivované blending pro správné zobrazení překrývajících se bodů



Obrázek 5.4 Ukázka 2D vizualizace vstupního datasetu.

Komponenta implementuje komplexní systém pro správu životního cyklu a optimalizaci výkonu:

- Automatické čištění prostředků při unmount komponenty
- Efektivní správa paměti s okamžitým uvolňováním nepoužívaných objektů
- Optimalizované překreslování scény pouze při změnách
- Adaptivní systém kvality vykreslování podle výkonu zařízení

3D vizualizace výstupu neuronové sítě

Komponenta NetworkVisualization3D implementuje plnohodnotnou 3D vizualizaci výstupních dat neuronové sítě s možností současného zobrazení predikovaných a cílových hodnot viz obr. 5.5. Systém je postaven na knihovně Three.js a realizuje pokročilé techniky pro efektivní vykreslování a interakci s 3D scénou.

Inicializace scény zahrnuje sofistikovaný výpočet optimálního pohledu na základě rozsahu dat:

$$\text{distance} = \text{radius} \cdot 3 \tag{5.24}$$

$$\text{camera.position} = (\text{centerX} + \text{distance}, \text{centerY} + \text{distance}, \text{centerZ} + \text{distance}) \tag{5.25}$$

kde radius je vypočítán jako:

$$\text{radius} = \max(\max(x) - \min(x), \max(y) - \min(y), \max(z) - \min(z))/2 \tag{5.26}$$

Základní vlastnosti vizualizace zahrnují:

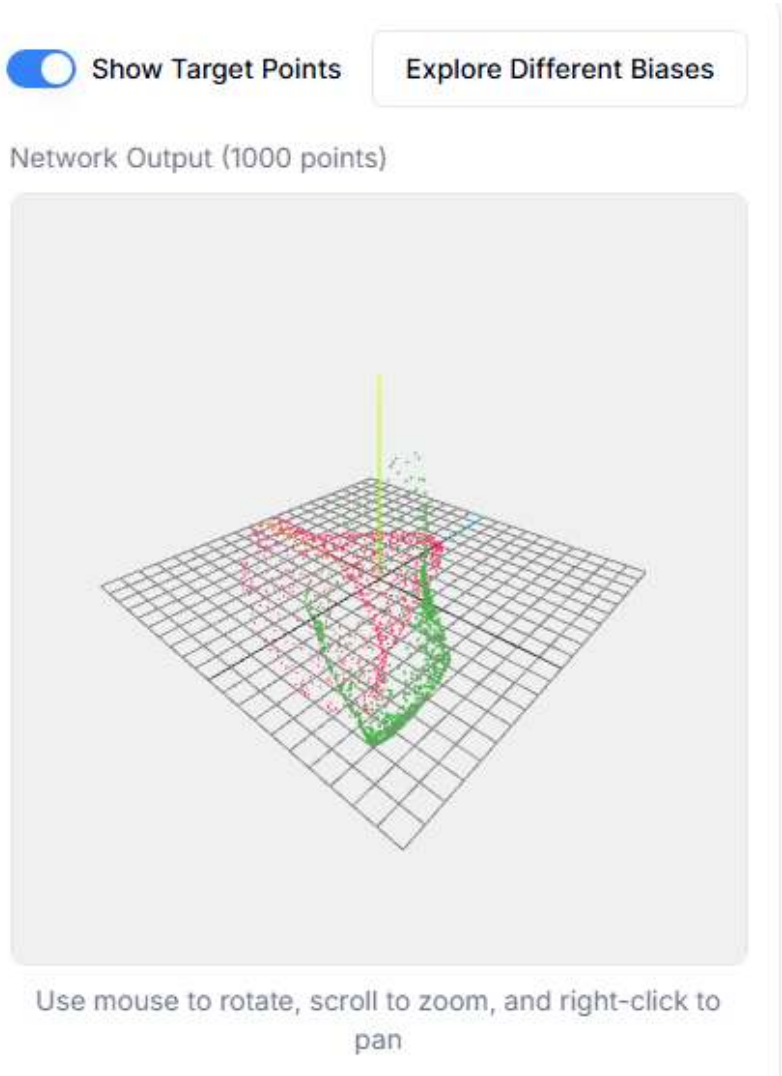
- Průhlednost bodů (0.8) pro lepší vizualizaci překryvů
- Přizpůsobení velikosti bodů vzdálenosti od kamery
- Optimalizované ukládání pozic pomocí Float32Array

Pro orientaci v prostoru systém využívá:

- GridHelper pro vizualizaci referenční mřížky
- AxesHelper pro zobrazení os souřadnicového systému
- Automatické přizpůsobení pohledu rozsahu dat

Implementace zahrnuje základní responsivní funkcionalitu:

- Přizpůsobení velikosti při změně rozměrů okna
- Aktualizace pohledu kamery při resize událostech
- Zachování poměrů stran při změnách velikosti



Obrázek 5.5 Ukázka 3D vizualizace výstupního datasetu.

5.3.5 Uživatelské rozhraní aplikace

Uživatelské rozhraní aplikace je navrženo s důrazem na intuitivnost a efektivitu práce s neuronovými sítěmi. Implementace využívá moderní komponenty z knihovny shadcn/ui, která poskytuje konzistentní vzhled a chování napříč celou aplikací. Rozhraní je rozděleno do několika logických sekcí, které uživateli umožňují postupnou konfiguraci a trénování neuronové sítě.

Hlavní komponenty rozhraní

Centrálním prvkem aplikace je komponenta NetworkLayout, která implementuje dvě hlavní zobrazení:

- **Konfigurační pohled** - poskytuje komplexní rozhraní pro nastavení a trénování sítě viz obr. 5.6
- **Dokumentační pohled** - nabízí podrobný průvodce funkcemi a možnostmi aplikace viz obr. 5.7

Konfigurační pohled je dále rozdělen do několika sekcí:

- **Architektura sítě** - umožňuje definovat počet a velikost vrstev
- **Trénovací parametry** - nastavení learning rate, epochs, batch size a L2
- **Vizualizace** - interaktivní zobrazení struktury sítě a výsledků trénování
- **Dataset** - výběr a konfigurace vstupních dat

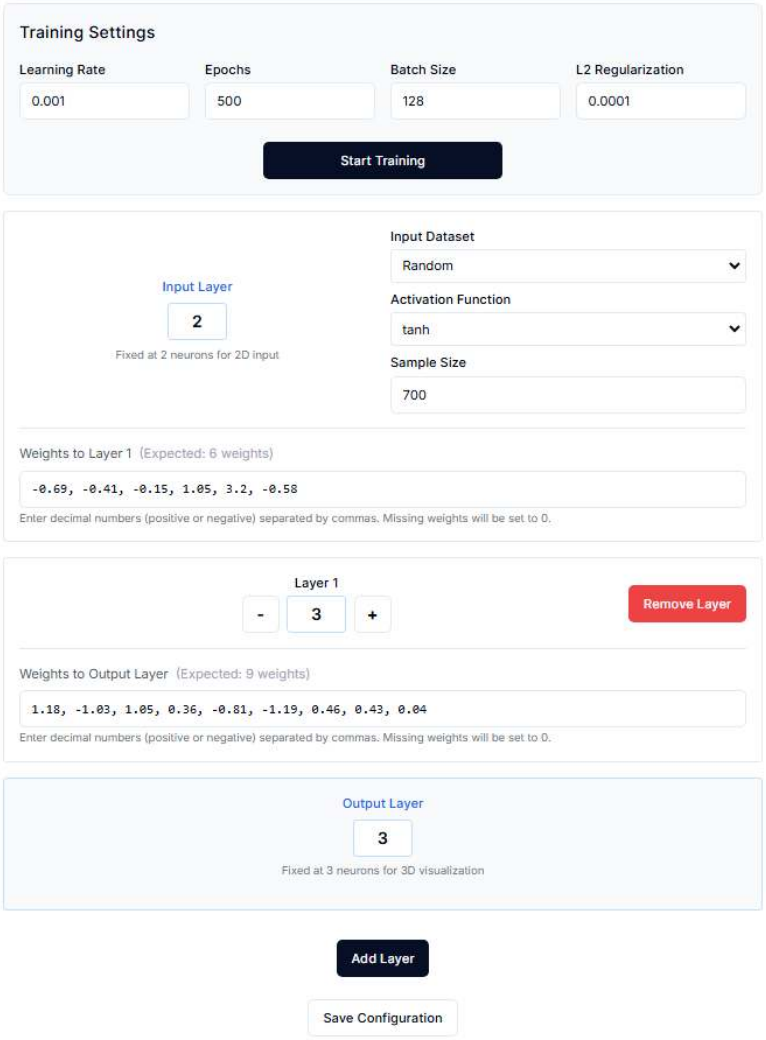
Responsivní design

Aplikace implementuje plně responsivní design pro optimální zobrazení na různých zařízeních:

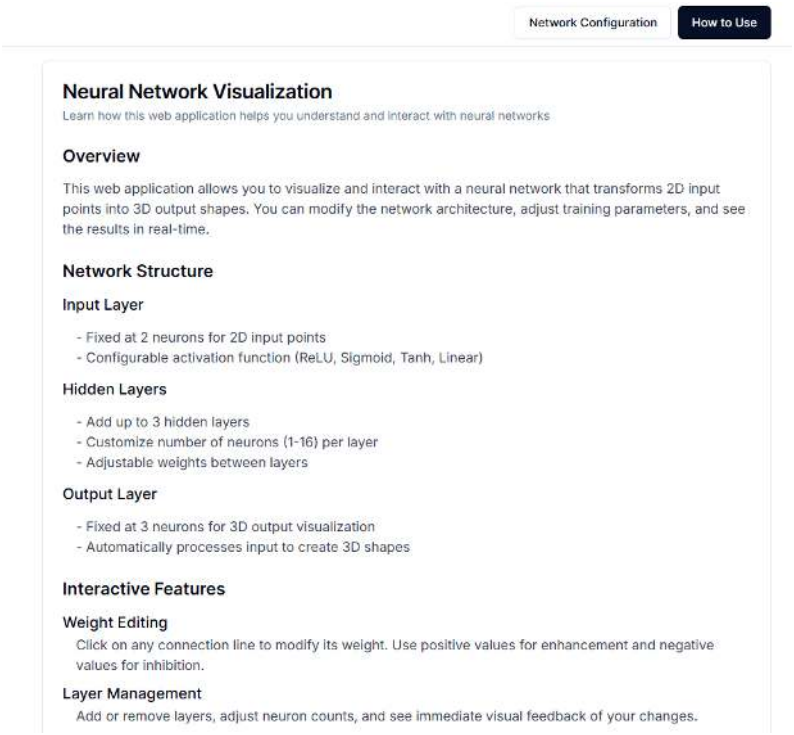
- Adaptivní layout přizpůsobující se velikosti obrazovky
- Optimalizované ovládací prvky pro dotykové obrazovky
- Přizpůsobení velikosti vizualizací dostupnému prostoru

5.4 Nasazení aplikace

Webová aplikace je nasazena na platformě Google Cloud Platform (GCP) s využitím služby Cloud Run, která poskytuje škálovatelné a bezpečné prostředí pro běh kontejnerizovaných aplikací. Systém využívá několik klíčových služeb GCP pro zajištění vysoké dostupnosti a efektivního provozu.



Obrázek 5.6 Náhled na úpravu konfigurace sítě.



Obrázek 5.7 Náhled stránky popisující webovou stránku.

5.4.1 Volba cloudové platformy

Google Cloud Platform byla zvolena z několika důvodů:

- **Serverless architektura** - Cloud Run umožňuje provoz aplikace bez nutnosti správy serverové infrastruktury
- **Automatické škálování** - systém automaticky přizpůsobuje počet instancí podle aktuálního zatížení
- **Pay-per-use model** - platba pouze za skutečně využité výpočetní prostředky
- **Integrované služby** - snadná integrace s dalšími službami GCP jako Cloud Storage a Cloud Build

5.4.2 Cloud Run implementace

Aplikace je rozdělena do dvou hlavních služeb běžících na Cloud Run:

- **neural-network-backend** - FastAPI backend služba
- **neural-network-frontend** - Next.js frontend aplikace

Každá služba je nakonfigurována s následujícími parametry:

- Automatické škálování od 0 do n instancí podle zátěže
- Přidělení výpočetních prostředků podle potřeby
- Monitoring a logování pro sledování výkonu
- HTTPS zabezpečení s automatickou správou certifikátů

5.4.3 Doménová konfigurace

Aplikace je dostupná pod doménou neural-network-visualizer.com, která byla zakoupená přes Google Cloud a je spravována pomocí Cloud DNS. Implementace zahrnuje:

- Konfiguraci DNS záznamů pro směrování provozu
- SSL certifikáty pro zabezpečenou komunikaci
- CDN pro optimalizaci doručování statického obsahu
- Monitoring dostupnosti a výkonu domény

5.4.4 Continuous Deployment

Systém využívá Cloud Build pro automatizované nasazování:

- Automatické spuštění buildu při změnách v repozitáři
- Sestavení Docker image pro každou službu
- Spuštění automatizovaných testů
- Nasazení na Cloud Run při úspěšném průběhu testů

5.4.5 Monitoring a škálování

Pro zajištění optimálního výkonu systém implementuje:

- Automatické škálování na základě vytížení CPU a počtu požadavků
- Monitoring klíčových metrik pomocí Cloud Monitoring
- Alerting pro včasnou detekci problémů
- Automatické zotavení při selhání instancí

5.5 Závěr

Tato kapitola představila implementaci webové aplikace pro vizualizaci a konfiguraci neuronových sítí. Systém byl navržen s důrazem na modularitu, škálovatelnost a uživatelskou přívětivost. Realizace využívá moderní technologie a postupy pro vytvoření robustního a efektivního řešení.

Kapitola 6

Aplikace ve virtuální realitě

Tato kapitola popisuje implementaci systému pro vizualizaci neuronové sítě v prostředí virtuální reality Resonite. Představuje komplexní řešení využívající vizuální programování ProtoFlux pro vytvoření interaktivní trojrozměrné reprezentace neuronové sítě, včetně vizualizace její architektury, vah a výstupních dat. Dále popisuje samotné uživatelské rozhraní, s kterým přijde uživatel do kontaktu. Svět je dostupný v Resonite v nabídce světů s názvem ***Neural Network Visualizer****.*

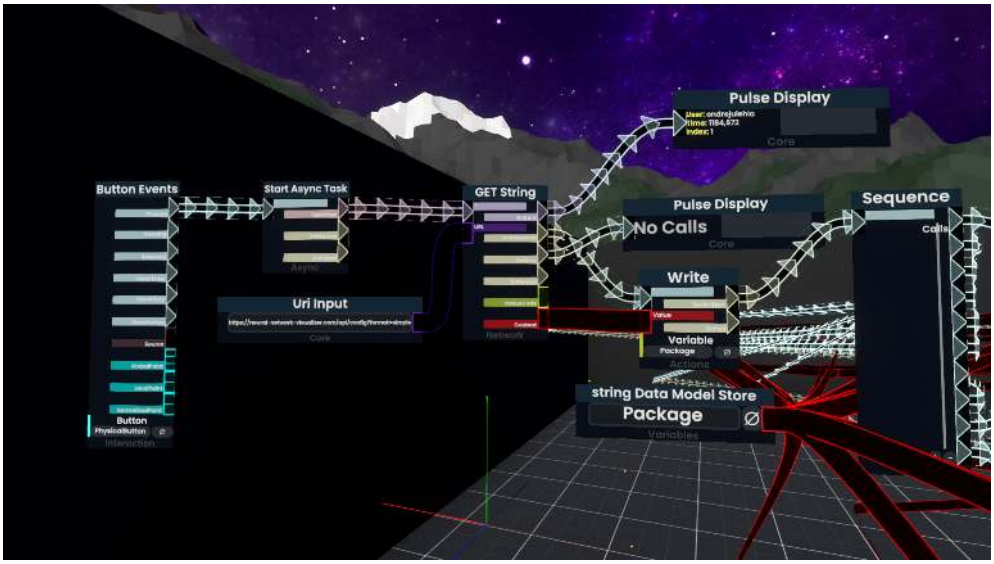
6.1 Implementace komunikace a zpracování dat

Základním stavebním kamenem implementace je komunikace s backend API, která je realizována pomocí HTTP GET požadavku na endpoint /api/config?format=simple. Tento proces je iniciován pomocí jediného tlačítka, které spouští celou sekvenci operací:

- **Button Events** komponenta zachycuje interakci uživatele, v tomto případě na tlačítku
- **Start Async Task** zajišťuje asynchronní zpracování HTTP požadavku
- **GET String** realizuje samotnou HTTP komunikaci
- **Write** komponenta ukládá získaná data do proměnné Package

Proces komunikace a uložení dat je zobrazen na obrázku 6.1.

Server vrací data v optimalizovaném formátu, který je speciálně navržen pro efektivní zpracování v prostředí ProtoFlux. Tento formát používá oddělovač '/' pro jednotlivé sekce dat, což umožňuje snadné parsování pomocí stringových operací. Pro testování byl využit tento formát, jelikož pro standartní formáty jako například JSON, by bylo nutné vytvořit Parser, neboť žádná implementace v současné době v Resonite neexistuje.



Obrázek 6.1 Implementace HTTP komunikace

6.1.1 Parsování dat

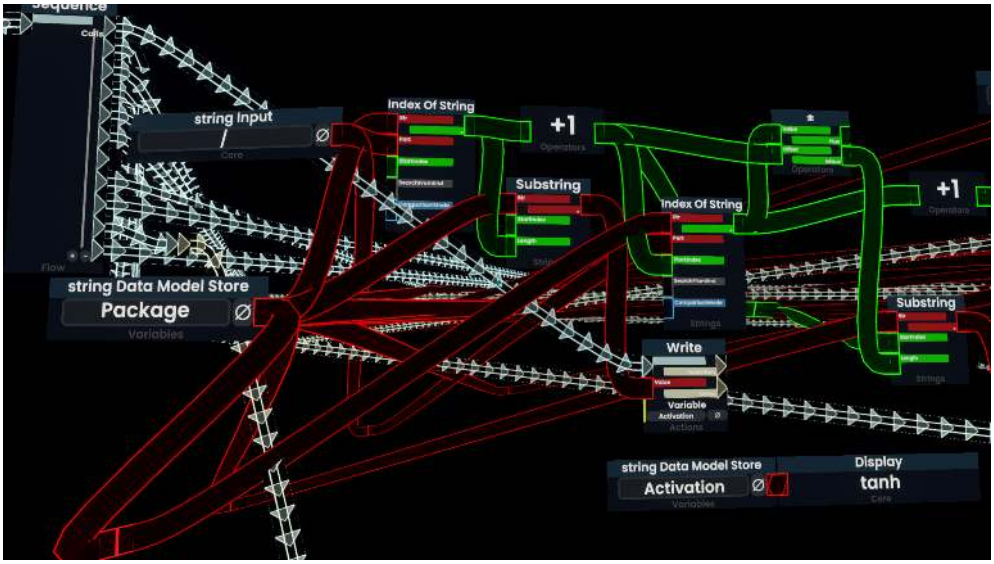
Pro zpracování získaných dat systém implementuje delimited format parser využívající několik klíčových komponent ProtoFluxu:

- **Index Of String** komponenta vyhledává pozice oddělovačů '/' v payloadu
- **Substring** extrahuje jednotlivé části dat mezi oddělovači
- **Write** ukládá zpracované hodnoty do příslušných proměnných

Proces parsování je realizován postupně pro všechny části konfigurace neuronové sítě:

- Aktivační funkce
- Počet vrstev neuronové sítě
- Počet neuronů v jednotlivých vrstvách
- Váhy mezi neurony
- Počet koordinát
- 3D koordináty pro vizualizaci výstupů
- 3D koordináty pro vizualizaci cílového tvaru

Proces parsování proměnných je znázorněn na obrázku 6.2 na příkladu extrakce aktivační funkce.



Obrázek 6.2 Proces parsování jedné z proměnných (Aktivační funkce).

6.2 Generování vizualizace sítě

Proces generování vizuální reprezentace neuronové sítě představuje systém vzájemně propojených komponent v ProtoFluxu. Tento proces lze rozdělit do několika klíčových fází, které společně vytváří kompletní vizualizaci architektury sítě.

6.2.1 Inicializace procesu

Systém začíná inicializací základních proměnných pomocí komponenty String Data Model Store, která uchovává klíčové informace o struktuře sítě:

- **NumOfNeuronsInEachLayer** - počet neuronů v jednotlivých vrstvách
- **NumOfLayers** - celkový počet vrstev sítě
- **LastIndex** - index poslední vrstvy pro řízení procesu generování

6.2.2 Řízení generování vrstev

Pro systematické vytváření jednotlivých vrstev sítě systém implementuje řídící logiku pomocí následujících komponent:

- **For** - komponenta řídí iteraci přes jednotlivé vrstvy
- **Parse Int** - převádí textové hodnoty na číselné pro další zpracování
- **ChangeableSource** - zajišťuje dynamické generování vizuálních elementů

- **Write Dynamic Int** - ukládá průběžné stavy procesu

Pro každou vrstvu systém provádí následující operace:

1. Načtení počtu neuronů pro aktuální vrstvu
2. Vytvoření instance vrstvy pomocí Duplicate Slot
3. Nastavení pozice vrstvy v prostoru
4. Propojení s okolními vrstvami

6.2.3 Zpracování vah a propojení

Systém implementuje mechanismus pro vytváření vizuálních reprezentací vah mezi neurony:

- **Index Of String** - komponenta identifikuje relevantní váhy pro aktuální spojení
- **Substring** - extrahuje konkrétní hodnoty vah
- **Parse Float** - převádí textové hodnoty vah na číselné hodnoty
- **Write Dynamic Slot** - aktualizuje vizuální reprezentaci vah

Pro každé propojení mezi vrstvami systém:

1. Vypočítá počet potřebných spojení
2. Vytvoří vizuální reprezentace pomocí Arrow Visual
3. Zajistí správné umístění v prostoru

6.2.4 Optimalizace pozic

Pro zajištění optimální vizuální reprezentace systém implementuje algoritmus pro rozmístění neuronů a jejich spojení:

- **Global Index of Weight** - určuje pozice jednotlivých spojení
- **Current Layer Slot** - spravuje aktuální kontext vrstvy
- **Set Parent** - zajišťuje správnou hierarchii objektů
- **Point Visual** - komponenta řídí pozicování v 3D prostoru

Tento systém zajišťuje, že:

1. Vrstvy jsou rovnoměrně rozmístěny v prostoru

2. Spojení mezi neurony se nekříží více, než je nezbytné
3. Vizualizace zůstává přehledná i při větším počtu neuronů
4. Uživatel může snadno sledovat tok informací sítí

Implementace systému pro vizualizaci neuronové sítě je zobrazena na obrázku 6.3.

Celý tento proces vytváří interaktivní 3D reprezentaci neuronové sítě, která věrně odráží její architekturu a umožňuje uživatelům intuitivně porozumět její struktuře a fungování. Vizualizace je dynamicky aktualizována při změnách konfigurace sítě, což poskytuje okamžitou zpětnou vazbu o provedených úpravách.

Obrázek 6.3 Implementace systému pro vizualizaci neuronové sítě.

6.3 Generování vizualizace výstupních dat

Systém implementuje specializovaný mechanismus pro vizualizaci výstupních dat neuronové sítě v trojrozměrném prostoru. Tento proces začíná parsováním koordinátů z textového formátu a končí vytvořením interaktivní vizuální reprezentace v prostředí virtuální reality. Proces zpracování koordinátů je realizován v několika navazujících krocích. Nejprve systém zpracuje celkový počet koordinátů, který určuje rozsah následné iterace. Pro každý bod pak systém provádí komplexní sekvenci operací: První fáze zahrnuje extrakci jednotlivých složek koordinátů pomocí řetězce komponent pro zpracování řetězců. Systém využívá Index Of String komponenty pro identifikaci oddělovačů mezi hodnotami a Substring komponenty pro izolaci samotných číselných hodnot. Následně jsou tyto hodnoty převedeny z textové reprezentace na číselné hodnoty pomocí Parse Float komponent. Pro organizaci trojrozměrných dat systém implementuje Pack xyz komponentu, která sdružuje jednotlivé složky koordinátů do jediné struktury. Tato komponenta zajišťuje konzistentní práci s prostoro-

vými daty a zjednodušuje následné operace s body v prostoru. Proces vytváření vizuální reprezentace je řízen pomocí následujících klíčových komponent:

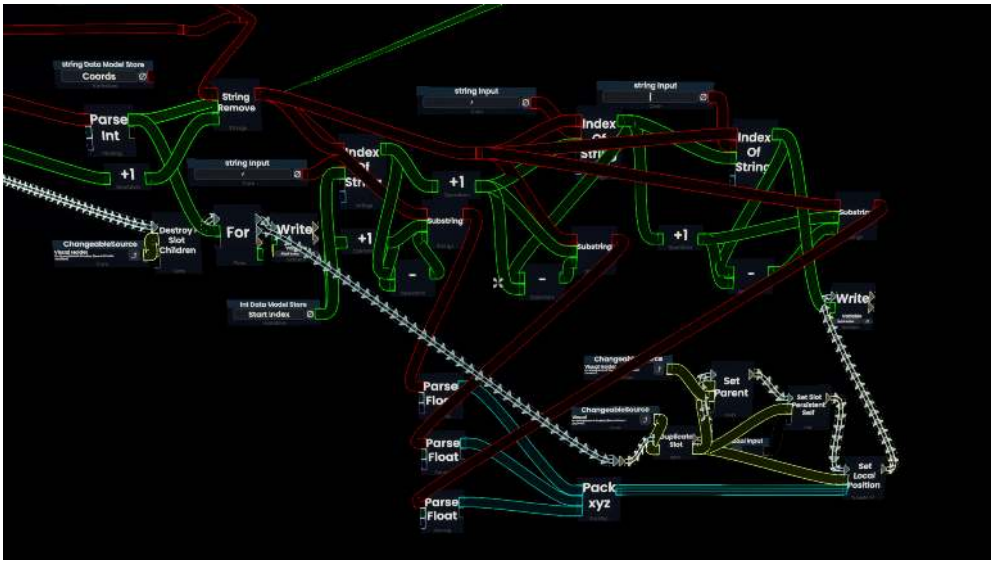
- **ChangeableSource Visual Holder** - zajišťuje správu vizuálních elementů
- **Duplicate Slot** - vytváří instance vizuálních reprezentací bodů
- **Set Parent** - komponenta organizuje hierarchickou strukturu vizuálních elementů
- **Set Local Position** - určuje přesnou pozici každého bodu v prostoru

Systém také implementuje optimalizační mechanismy pro efektivní správu velkého množství bodů:

- Dávkové zpracování koordinátů pro minimalizaci režie
- Dynamická správa instancí vizuálních elementů
- Optimalizované přepočítávání pozic při změnách dat

Proces implementace vizualizace výstupních dat je znázorněn na obrázku 6.4.

Výsledkem tohoto procesu je interaktivní trojrozměrná vizualizace, která přesně reprezentuje výstupní data neuronové sítě. Uživatelé mohou s touto vizualizací přímo interagovat ve virtuálním prostředí, prohlížet si ji z různých úhlů a zkoumat vztahy mezi jednotlivými body v prostoru.



Obrázek 6.4 Implementace systému pro vizualizaci výstupních dat.

6.4 Generování vizualizace cílových dat

Systém implementuje paralelní proces vizualizace pro cílové (target) výstupy neuronové sítě, který sdílí značnou část architektury s vizualizací aktuálních výstupů. Tento přístup zajišťuje konzistentní reprezentaci dat a umožňuje přímé vizuální porovnání mezi aktuálními a cílovými výstupy sítě. Proces zpracování cílových koordinátů následuje obdobnou strukturu jako zpracování výstupních dat. Systém nejprve načte celkový počet koordinátů, který určuje rozsah zpracování. Následně pro každý koordinát provádí sekvenci transformací z textové reprezentace do trojrozměrného prostoru. Zpracování dat je realizováno pomocí řetězce komponent, které zajišťují následující operace:

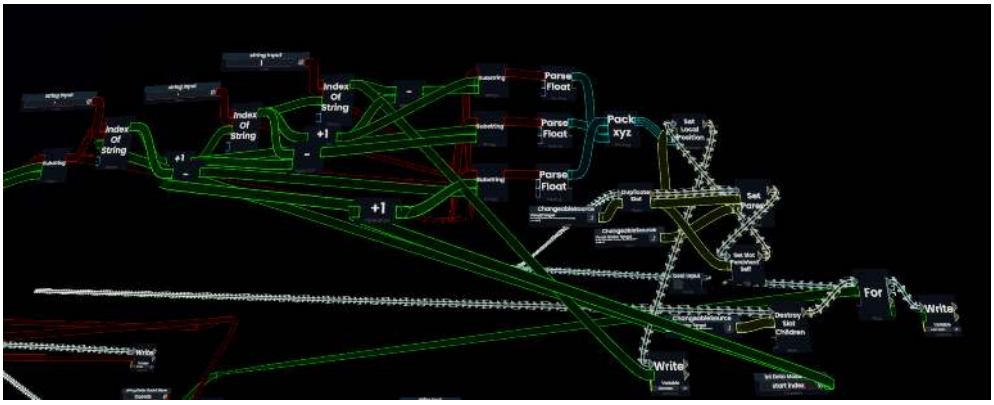
- Index Of String komponenty identifikují oddělovače mezi hodnotami koordinátů
- Substring komponenty izolují číselné hodnoty z textových řetězců
- Parse Float komponenty převádějí textové reprezentace na číselné hodnoty
- Pack xyz komponenta sdružuje jednotlivé složky koordinátů do ucelené prostorové reprezentace

Pro vytváření vizuální reprezentace cílových bodů systém využívá stejnou sadu komponent jako u aktuálních výstupů, zahrnující ChangeableSource, Duplicate Slot a Set Parent komponenty. Tento přístup zajišťuje konzistentní vizuální reprezentaci obou sad dat a umožňuje jejich snadné porovnání v prostoru. Klíčovým rozdílem oproti vizualizaci aktuálních výstupů je barevné odlišení cílových bodů, které uživatelům umožňuje rychle rozlišit mezi aktuálními a cílovými hodnotami. Toto barevné rozlišení poskytuje okamžitou vizuální zpětnou vazbu o kvalitě výstupu neuronové sítě a míře její konvergence k požadovaným hodnotám. Výsledkem procesu je interaktivní trojrozměrná vizualizace, která uživatelům umožňuje:

- Prohlížet cílové hodnoty v kontextu aktuálních výstupů
- Analyzovat odchylky mezi aktuálními a cílovými hodnotami
- Sledovat proces učení neuronové sítě v reálném čase
- Identifikovat oblasti, kde síť dosahuje dobré nebo nedostatečné aproximace cílových hodnot

Řešení systému pro vizualizaci cílových dat je zobrazena na obrázku 6.5.

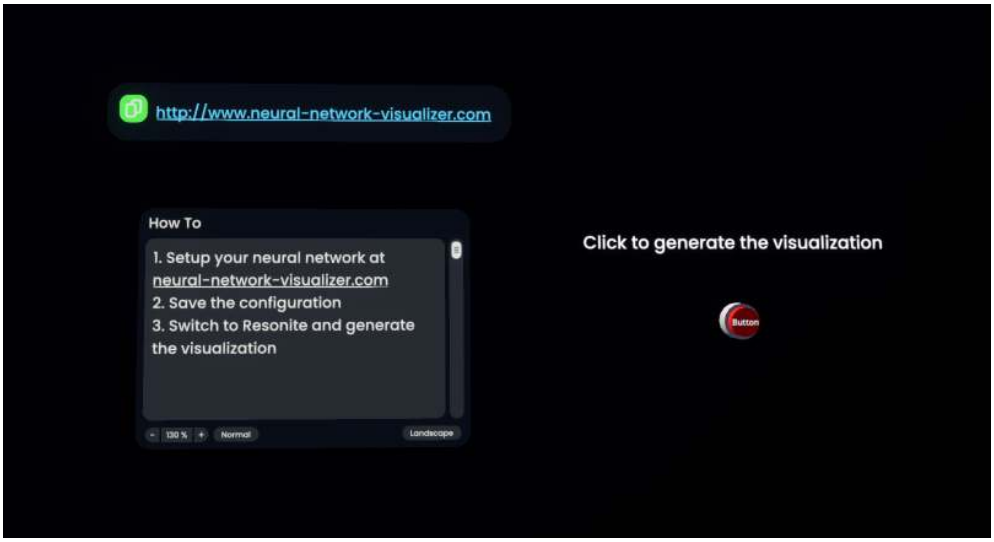
Tato duální vizualizace představuje efektivní nástroj pro evaluaci výkonu neuronové sítě a poskytuje intuitivní způsob porozumění jejímu chování v kontextu řešené úlohy, vizualizace zároveň slouží k otestování.



Obrázek 6.5 Implementace systému pro vizualizaci cílových dat.

6.5 Uživatelské rozhraní virtuálního prostředí

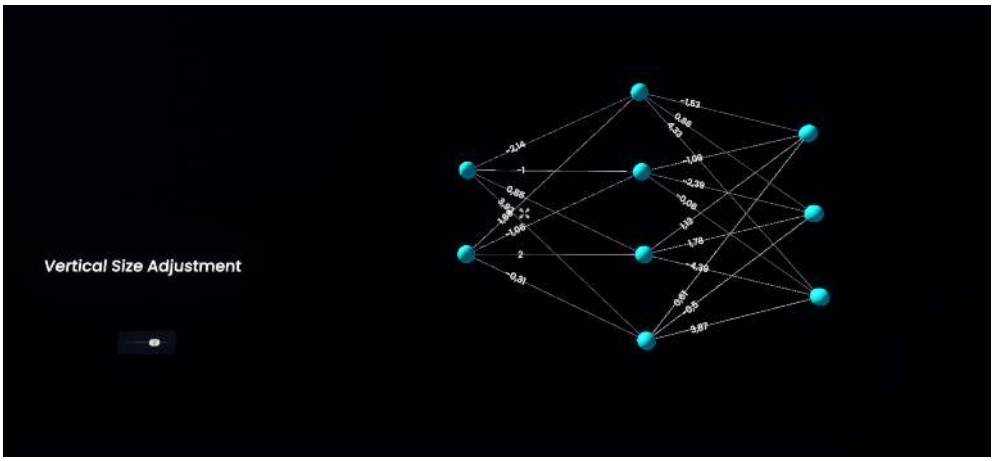
Virtuální prostředí pro vizualizaci neuronových sítí je navrženo s důrazem na intuitivní ovládání a přehlednou prezentaci dat. Vstupním bodem systému je webová stránka neural-network-visualizer.com, kde uživatelé mohou konfigurovat parametry neuronové sítě. Lze se na ní dostat přímo přes odkaz ve virtuálním světě. Po nastavení konfigurace se uživatelé přepnou zpět do prostředí Resonite, kde mohou pomocí jediného tlačítka viz obr. 6.6 vygenerovat kompletní vizualizaci sítě a jejích výstupů viz obr. 6.7. Svět obsahuje jednoduchý informační panel s instrukcemi pro použití systému.



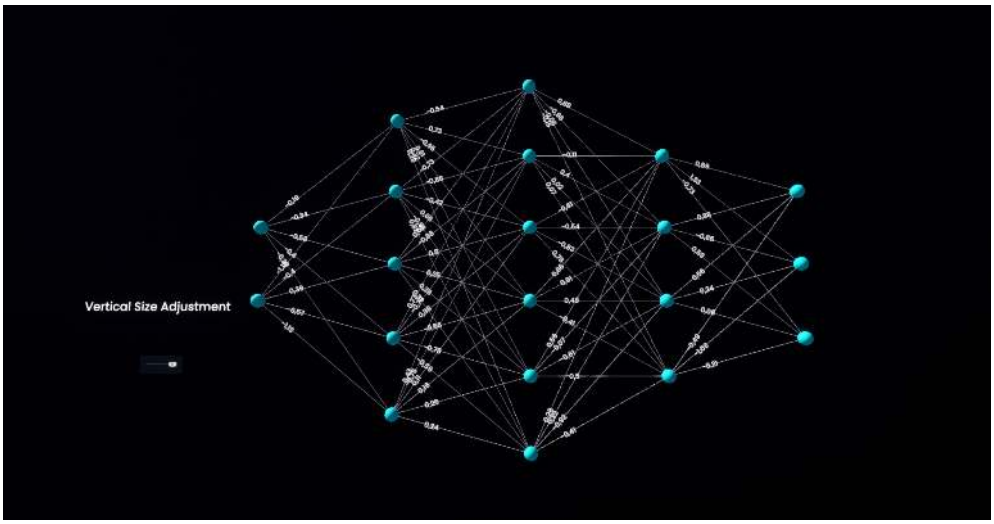
Obrázek 6.6 Tlačítko pro vygenerování vizualizací.

Po stisknutí tlačítka pro generování se vytvoří vizualizace neuronové sítě, která přesně odpovídá konfiguraci nastavené ve webovém rozhraní viz obr. 6.7.

Síť je zobrazena v prostoru s jasně viditelnými neurony, jejich propojeními a vahami. Pro optimální zobrazení různých architektur sítě je implementováno vertikální přizpůsobení velikosti, které umožňuje upravit měřítko vizualizace podle potřeby viz obr. 6.8.



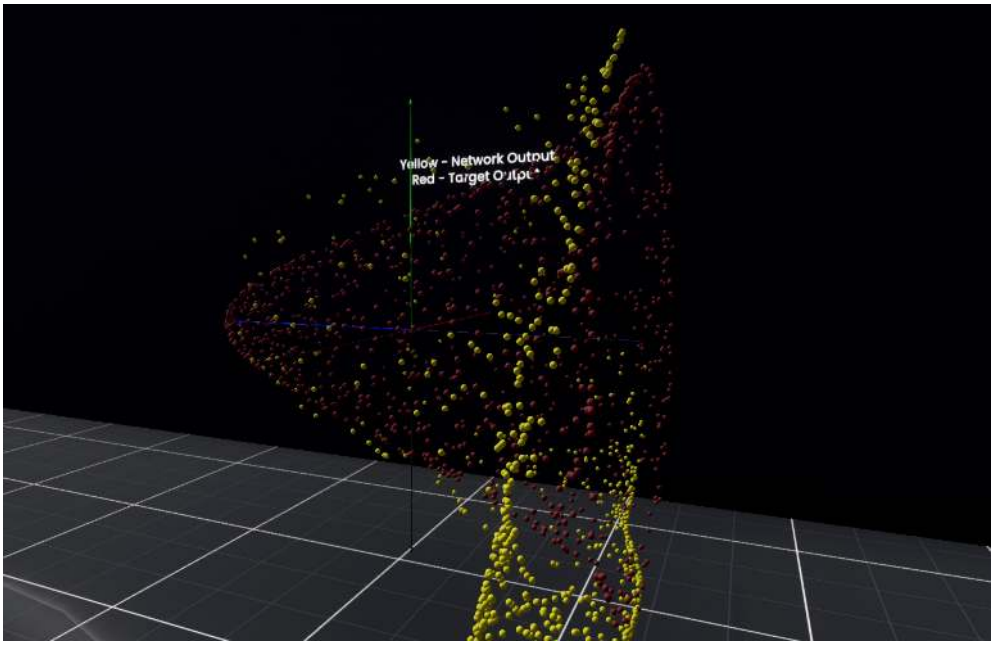
Obrázek 6.7 Vizualizace neuronové sítě.



Obrázek 6.8 Vizualizace neuronové sítě s více vrstvami a neurony s upraveným veritkálním měřítkem.

Systém dále implementuje sofistikovanou vizualizaci výstupních dat neuronové sítě. V trojrozměrném prostoru jsou současně zobrazeny jak aktuální výstupy sítě (žlutě), tak cílové hodnoty (červeně), což umožňuje okamžité vizuální porovnání výsledků viz obr. 6.9 a obr. 6.10. Toto barevné rozlišení poskytuje intuitivní způsob hodnocení přesnosti sítě.

Dále bylo pro svět vytvořeno pozadí scény v podobě low-poly horského pro-



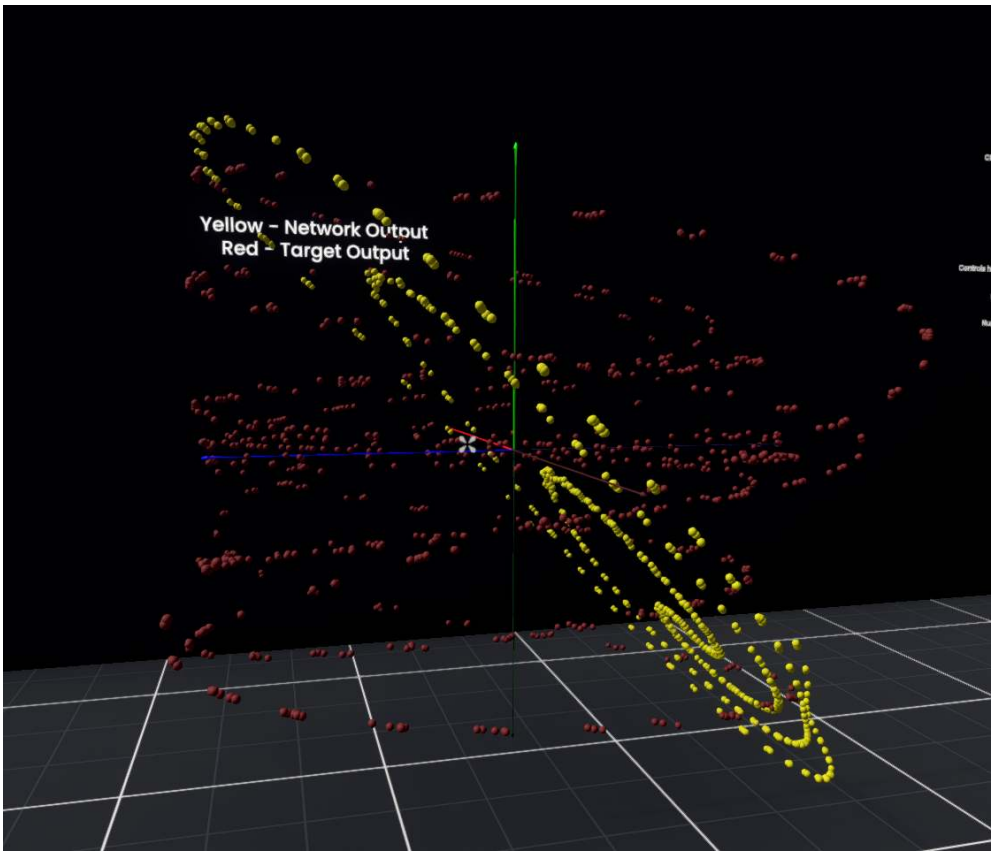
Obrázek 6.9 Vizualizace výsledku a cílových hodnot.

středí viz obr. 6.11. Toto prostředí, vymodelované v Blenderu, pomocí pluginu Landscape Mesh, umožňuje vygenerovat terény. Následně u modelu došlo ke snížení počtu vrcholů, aby byl co nejméně náročný na renderování ve světě. Nakonec byly přidány materiály na povrch objektu, které terénu dodaly reálnější vzhled.

Pro přesnou orientaci v prostoru u vizualizací výstupních a cílových byly přímo v Resonite implementovány osy souřadnicového systému viz obr. 6.12. Tyto osy jsou reprezentovány barevně odlišenými přímkami, které poskytují jasný referenční rámec pro pozicování s vizualizovanými daty.

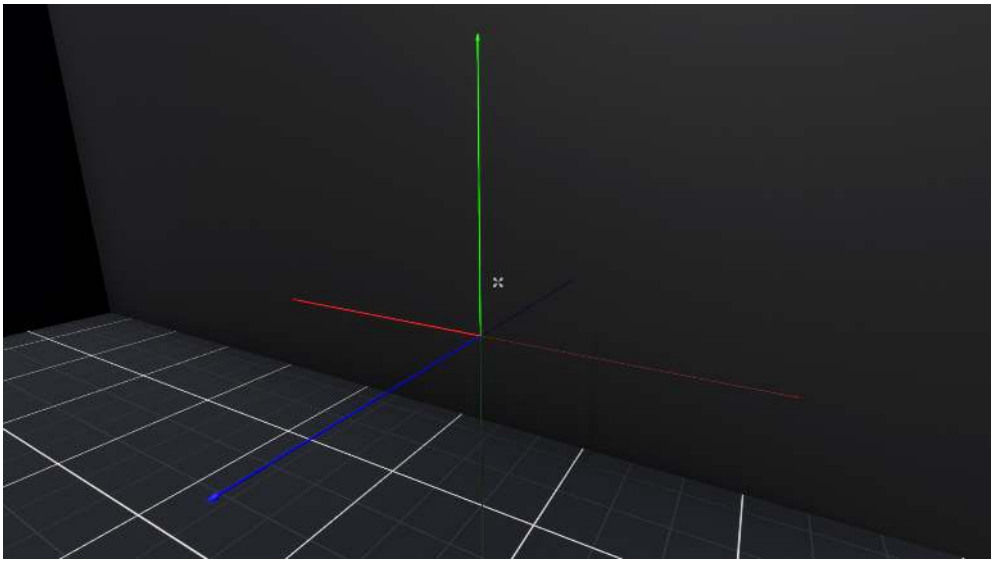
6.6 Závěr

Tato kapitola představila implementaci systému pro vizualizaci neuronových sítí v prostředí virtuální reality Resonite pomocí virtualního programování ProtoFlux. Implementace zahrnovala několik klíčových komponent, od komunikace s backend API přes zpracování dat až po samotnou vizualizaci. Dále popisuje uživatelské rozhraní světa, které uživateli umožňuje vytvářet a interagovat s neuronovými sítěmi.



Obrázek 6.10 Další vizualizace výsledku a cílových hodnot.

Obrázek 6.11 Low-poly horské prostředí použité jako pozadí světa.



Obrázek 6.12 Souřadnicový systém

Kapitola 7

Testování implementace

Tato kapitola se zaměřuje na testování funkčnosti webové aplikace a vizualizací v prostředí Resonite. Cílem testování je ověřit, zda implementovaný systém splňuje všechny stanovené požadavky a poskytuje uživatelům očekávané výsledky. Proces testování probíhá ve dvou hlavních fázích - nejprve se testuje funkcionalita webové aplikace pro konfiguraci neuronových sítí a následně se ověřuje správnost vizualizací generovaných v Resonite porovnáním s webovou aplikací.

7.1 Testování webové aplikace

Webová aplikace slouží jako centrální bod pro konfiguraci parametrů neuronových sítí před jejich vizualizací ve virtuálním prostředí Resonite. Pro zajištění spolehlivosti celého systému je kritické důkladně otestovat všechny aspekty této aplikace.

7.1.1 Testovací scénáře

Pro komplexní otestování webové aplikace byla připravena sada testovacích scénářů, které pokrývají všechny klíčové funkce a možné uživatelské interakce. Tyto scénáře zahrnují:

1. Vytvoření nové konfigurace neuronové sítě s různými parametry (počet vrstev, počet neuronů, aktivační funkce, datové sady)
2. Úprava existující konfigurace a ověření správnosti provedených změn
3. Validace vstupních hodnot a ošetření chybových stavů (např. záporný počet neuronů, neplatná aktivační funkce)
4. Testování responzivity uživatelského rozhraní na různých rozlišeních obrazovky

5. Ověření správnosti komunikace s backend API a ukládání konfigurací
6. Testování výkonnosti aplikace při práci s většími datovými sadami a komplexními konfiguracemi sítí

Každý scénář obsahuje podrobný popis kroků, očekávané výsledky a kritéria pro vyhodnocení úspěšnosti. Během testování jsou pečlivě zaznamenávány všechny nalezené chyby a nedostatky, které jsou následně analyzovány a opraveny.

7.1.2 Průběh a výsledky testování

Testování webové aplikace probíhalo iterativním způsobem, kdy byly postupně procházeny jednotlivé scénáře a ověřována funkčnost všech komponent systému. V rámci testování bylo identifikováno několik drobných chyb v uživatelském rozhraní a ve validaci vstupů, které byly následně opraveny. Zásadním zjištěním bylo nekonzistentní chování aplikace při práci s velmi velkými datovými sadami, kde docházelo k výraznému zpomalení odezvy a občasným selháním při ukládání konfigurací. Tento problém byl vyřešen optimalizací algoritmu pro zpracování dat a implementací dávkového ukládání. Po aplikaci všech nezbytných oprav a vylepšení byla provedena kompletní regrese testovacích scénářů, aby bylo ověřeno, že změny neměly negativní dopad na ostatní funkce systému. Výsledkem tohoto testování je funkční a optimalizovaná webová aplikace, která splňuje stanovené požadavky a je připravena pro nasazení do produkčního prostředí.

7.2 Testování vizualizací v Resonite

Druhou klíčovou fází testování je ověření správnosti a efektivity vizualizací neuronových sítí generovaných v prostředí Resonite. Cílem je zajistit, že vizualizace přesně odrážejí konfiguraci definovanou ve webové aplikaci a poskytují uživatelům hodnotný náhled na strukturu a chování sítě.

7.2.1 Příprava testovacích dat

Pro důkladné otestování vizualizací byla připravena sada předem definovaných konfigurací neuronových sítí s různými parametry a datovými sadami. Tyto konfigurace pokrývají spektrum možných architektur a výpočetních scénářů, včetně:

1. Sítí s různým počtem skrytých vrstev a neuronů
2. Různých aktivačních funkcí (ReLU, sigmoid, tanh)
3. Datových sad s různými charakteristikami (kruhový, spirálový, XOR vzor)

4. Sítí s různou komplexitou vah a propojení

Pro každou konfiguraci byly také definovány očekávané výstupy a referenční vizualizace, vůči kterým budou porovnávány výsledky generované v Resonite.

7.2.2 Generování a vyhodnocení vizualizací

Proces testování vizualizací probíhal následovně:

1. Ve webové aplikaci byly postupně načítány připravené konfigurace neuronových sítí.
2. Pro každou konfiguraci byla pomocí tlačítka v Resonite vygenerována odpovídající vizualizace.
3. Vygenerovaná vizualizace byla porovnána s očekávaným výstupem a referenční vizualizací.
4. Byly zaznamenány všechny odchylky a anomálie ve vizualizaci, jako jsou chybějící nebo nesprávně umístěné neurony, chybná propojení nebo nekonzistence s konfigurací.

Během testování bylo odhaleno několik drobných chyb v algoritmu pro generování vizualizací, například nesprávné umístění neuronů v prostoru při určitých konfiguracích nebo chybějící propojení mezi vrstvami. Tyto chyby byly následně opraveny úpravou příslušných komponent v ProtoFluxu. Důležitým aspektem testování byla také validace výstupních dat neuronové sítě ve 3D prostoru. Pro každou testovací konfiguraci bylo ověřeno, že vizualizované výstupy odpovídají očekávaným hodnotám a že jsou správně reprezentovány v kontextu cílových hodnot.

7.2.3 Optimalizace výkonu

Součástí testování vizualizací byla i analýza výkonnosti a identifikace potenciálních problémů s odezvou systému. Zátěžovými testy byla simulována práce s rozsáhlými datovými sadami a komplexními konfiguracemi neuronových sítí. Na základě výsledků těchto testů bylo identifikováno několik úzkých hrdel v algoritmu generování vizualizací, zejména při práci s velkým počtem neuronů a propojení. Tyto problémy byly řešeny optimalizací klíčových komponent v ProtoFluxu, využitím efektivnějších datových struktur. Po aplikaci těchto optimalizací bylo dosaženo zlepšení výkonu a schopnosti systému zvládat i velmi komplexní konfigurace neuronových sítí bez negativního dopadu na uživatelskou zkušenost.

7.3 Případová studie: Vizualizace kruhových dat

V rámci testování byla provedena detailní případová studie zaměřená na zpracování kruhového vzoru dat s využitím specifické architektury neuronové sítě. Tato studie demonstruje kompletní workflow systému od konfigurace sítě až po vizualizaci výsledků v prostředí virtuální reality, jak je dokumentováno na obrázcích 7.2 až 7.5.

7.3.1 Konfigurace testovací sítě

Pro testování byla zvolena konfigurace neuronové sítě zobrazená na obrázku 7.2. Tato konfigurace zahrnuje:

- Vstupní vrstva: 2 neurony (fixní pro 2D vstupní data)
- Skrytá vrstva: 4 neurony s aktivační funkcí tanh
- Výstupní vrstva: 3 neurony (fixní pro 3D vizualizaci)
- Trénovací parametry:
  - Learning rate: 0.01
  - Epochs: 500
  - Batch size: 32
  - L2 regularizace: 0.00001
- Velikost vzorku: 1000 bodů
- Typ datasetu: Circle (kruhový vzor)

Váhy mezi vrstvami byly inicializovány s konkrétními hodnotami pro zajištění reprodukovatelnosti výsledků, jak je detailně zobrazeno na obrázku 7.1. Mezi vstupní vrstvou a skrytou vrstvou byly nastaveny váhy v rozsahu od -4.3 do 1.23, zatímco mezi skrytou a výstupní vrstvou byly váhy v rozsahu od -3.94 do 3.52.

7.3.2 Validace vizualizací

Provedené testování potvrdilo přesnou korespondenci mezi vizualizacemi ve webové aplikaci a v prostředí Resonite. Jak je patrné na obrázcích 7.2 a 7.4, reprezentace struktury sítě zůstává konzistentní napříč oběma platformami. Testování potvrdilo několik klíčových aspektů implementace:

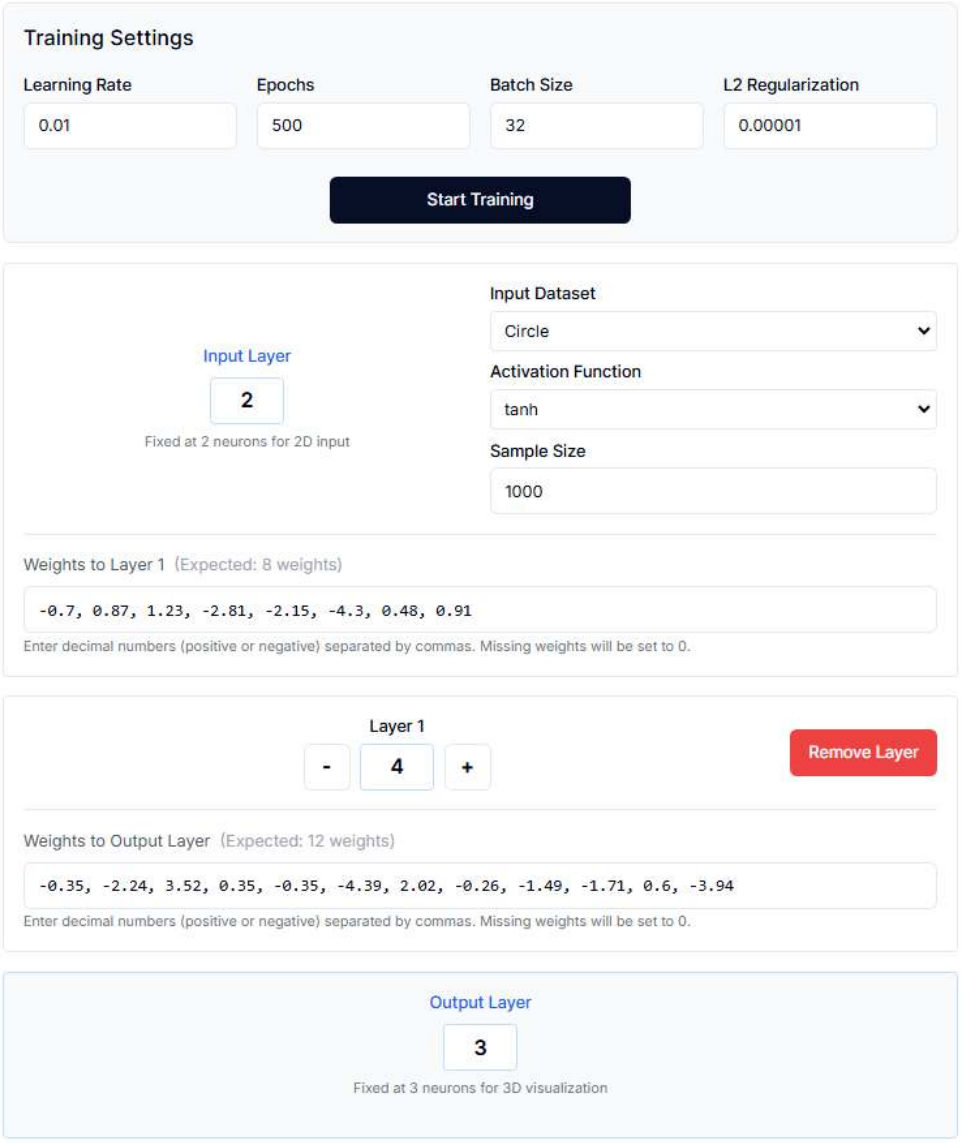
- **Struktura sítě:** Vizualizace v obou prostředích přesně zobrazily konfigurovanou architekturu se dvěma vstupními neurony, čtyřmi skrytými neurony a třemi výstupními neurony, jak dokumentují obrázky 7.2 a 7.4.

- **Váhy spojení:** Hodnoty vah byly korektně přeneseny a vizualizovány v Resonite (obrázek 7.4), včetně jejich přesných numerických hodnot z webové aplikace zobrazené na obrázku 7.1.
- **3D výstup:** Porovnání výstupních vizualizací mezi webovou aplikací (obrázek 7.3) a prostředím Resonite (obrázek 7.5) potvrzuje konzistentní reprezentaci dat. V obou případech jsou výstupní body zobrazeny zeleně ve webové aplikaci a žlutě v Resonite a cílové hodnoty červeně, což umožňuje přímé vizuální porovnání výsledků.

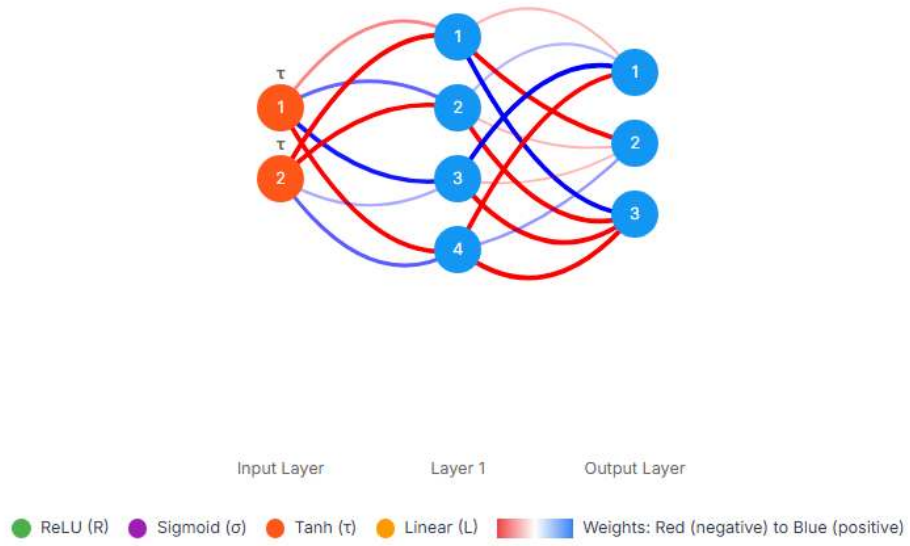
Toto porovnání vizualizací mezi webovou aplikací a prostředím Resonite demonstruje úspěšnou implementaci systému pro přenos a zobrazení konfigurace neuronové sítě ve virtuální realitě. Konzistence vizualizací napříč platformami potvrzuje správnou funkčnost celého řešení od konfigurace přes přenos dat až po jejich trojrozměrnou reprezentaci.

7.4 Závěr testování

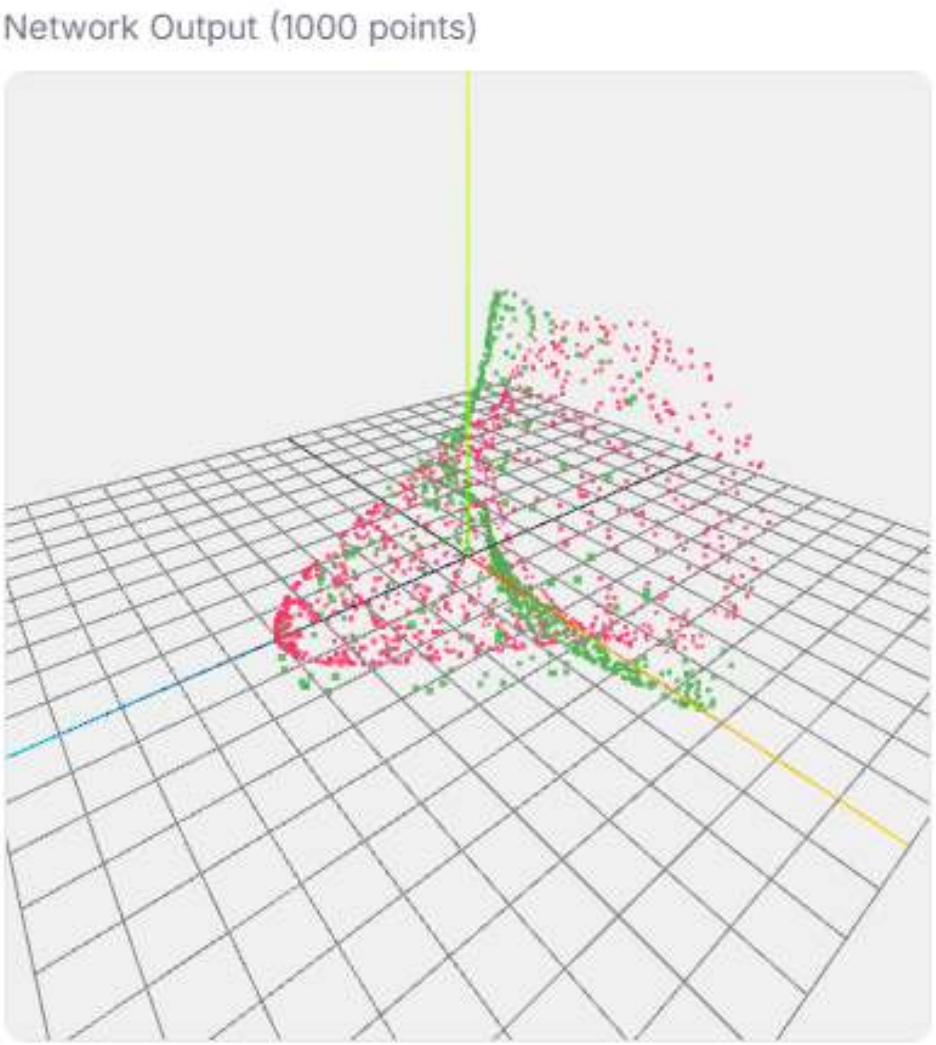
Komplexní testování webové aplikace a vizualizací v Resonite potvrdilo, že implementovaný systém splňuje všechny stanovené požadavky a je připraven pro nasazení a praktické využití. Webová aplikace poskytuje intuitivní a spolehlivé rozhraní pro konfiguraci neuronových sítí s robustní validací vstupů a efektivním zpracováním dat. Všechny klíčové funkce aplikace byly důkladně otestovány a optimalizovány pro zajištění vysoké kvality uživatelské zkušenosti. Vizualizace generované v prostředí Resonite věrně odrážejí konfigurace definované ve webové aplikaci a poskytují uživatelům hodnotný náhled na strukturu a chování neuronových sítí. Proces generování vizualizací byl optimalizován pro zajištění vysokého výkonu i při práci s komplexními konfiguracemi a rozsáhlými datovými sadami. Případné chyby a nedostatky odhalené během testování byly analyzovány a opraveny, což zajišťuje spolehlivost a robustnost celého systému. Na základě výsledků testování lze konstatovat, že vyvinutý systém je připraven sloužit svému účelu - poskytnout uživatelům efektivní nástroj pro konfiguraci a vizualizaci neuronových sítí v prostředí virtuální reality.



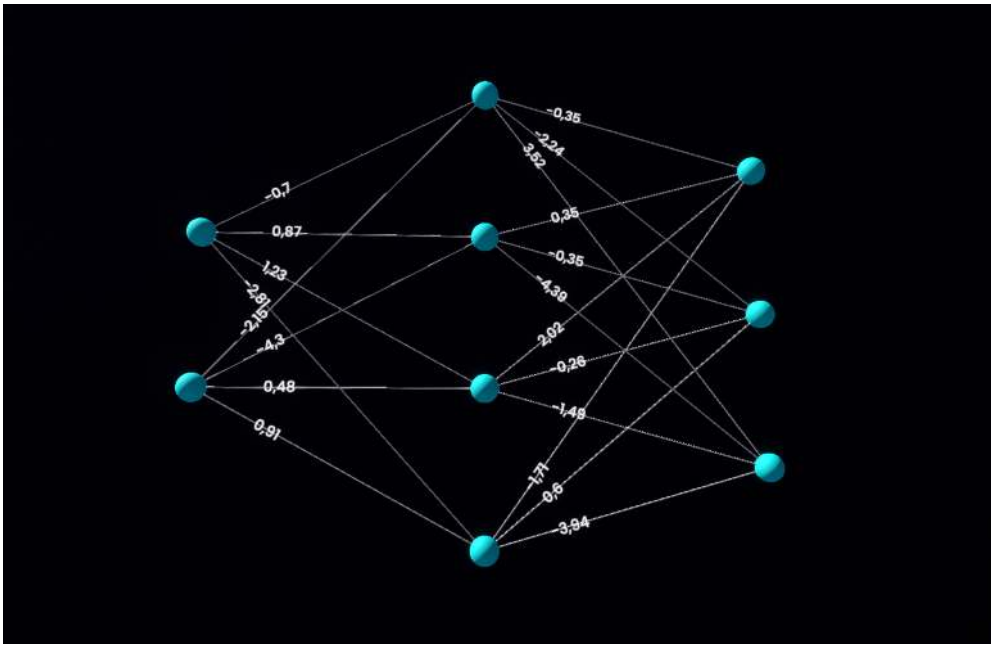
Obrázek 7.1 Váhy neuroové sítě v aplikaci.



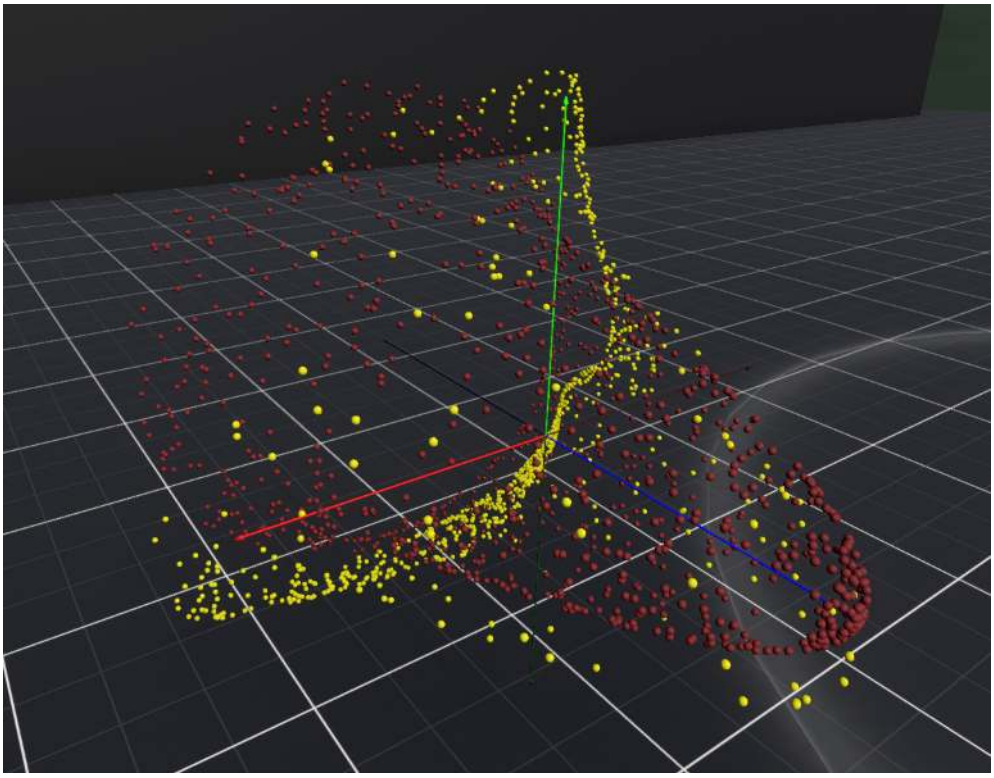
Obrázek 7.2 Neuronová síť ve webové aplikaci.



Obrázek 7.3 Vizualizace výstupu neuronové sítě ve webové aplikaci.



Obrázek 7.4 Vizualizace neuronové sítě s váhami v Resonite.



Obrázek 7.5 Vizualizace výstupu neuronové sítě v Resonite

Kapitola 8
Závěr

Tato diplomová práce se zabývala návrhem a řešením systému pro vizualizaci neuronových sítí ve virtuální realitě. Systém kombinuje webovou aplikaci pro konfiguraci parametrů neuronové sítě s interaktivním 3D prostředím v platformě Resonite, což umožňuje uživatelům intuitivně zkoumat a porozumět struktuře a chování neuronových sítí.

V rámci práce byl vytvořen systém zahrnující tři hlavní komponenty. První komponenta představuje webovou aplikaci implementovanou pomocí technologií Next.js a FastAPI, která poskytuje uživatelské rozhraní pro konfiguraci neuronových sítí. Aplikace umožňuje definovat architekturu sítě, včetně počtu vrstev, neuronů, aktivačních funkcí a tréninkových parametrů. Implementace zahrnuje také sofistikovaný systém pro trénování sítí a vizualizaci jejich výstupů v reálném čase.

Druhou klíčovou komponentou je backend systém postavený na FastAPI, který zajišťuje perzistenci dat a komunikaci mezi webovou aplikací a virtuálním prostředím. Backend implementuje optimalizovaný formát pro přenos dat a využívá služby Google Cloud Platform pro škálovatelné nasazení.

Třetí komponenta představuje implementaci v prostředí Resonite, která využívá systém vizuálního programování ProtoFlux pro vytvoření interaktivních 3D vizualizací neuronových sítí. Tato část systému umožňuje uživatelům prozkoumat strukturu sítě a její výstupy v imerzivním prostředí virtuální reality.

Během implementace systému jsem získal cenné zkušenosti s vizuálním programováním v prostředí Resonite.

Zároveň jsem se setkal s několika výzvami a omezeními:

- Při rostoucí komplexitě programu se stává vizuální reprezentace nepřehlednou - nutnost omezení maximalního počtu neuronů na vrstvu
- Nutnost předchozí znalosti dostupných ProtoFlux nodů a jejich funkcí
- Omezené možnosti debugování a testování

- Absence pokročilých vývojářských nástrojů

Samotná implementace v Resonite mi zabrala přibližně 80 hodin práce, což zahrnovalo studium dokumentace, experimentování s různými přístupy k vizualizaci a optimalizaci výsledného řešení. Přestože dokumentace na Resonite Wiki poskytuje užitečné informace o dostupných komponentách, praktické zvládnutí systému ProtoFlux vyžaduje značné úsilí a experimentování.

Co se týče výkonnosti systému, webová aplikace je optimalizována pro efektivní zpracování dat. Nicméně při trénování komplexnějších sítí s velkými batch sizes nebo vysokým počtem epoch může doba výpočtu dosáhnout až jedné minuty. Toto omezení je dáno především výpočetní náročností trénování neuronových sítí v prostředí webového prohlížeče.

Pro budoucí rozvoj projektu plánuji několik vylepšení:

- Implementace podpory pro vlastní datové sady s definovaným formátem
- Rozšíření možností vizualizace v prostředí Resonite
- Optimalizace výkonu při trénování komplexních sítí
- Implementace nových typů neuronových sítí a architektur
- Rozšíření možností interakce s modelem ve virtuální realitě

Vyvinutý systém představuje nástroj pro vizualizaci a studium neuronových sítí ve virtuální realitě. Kombinace webové aplikace pro konfiguraci a 3D vizualizace v Resonite vytváří prostředí, které může významně usnadnit pochopení principů fungování neuronových sítí. Přestože současná implementace má určitá omezení, poskytuje solidní základ pro další rozvoj a vylepšování funkcionalit systému.

Bibliografie

1. HAYKIN, Simon. *Neural Networks and Learning Machines*. 3. vyd. Upper Saddle River, NJ: Pearson, 2009. ISBN 978-0131471399.
2. GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. *Deep Learning*. MIT Press, 2016. ISBN 978-0262035613. Dostupné také z: https://www.deeplearningbook.org.
3. SCHMIDHUBER, Jürgen. Deep Learning in Neural Networks: An Overview. *Neural Networks*. 2015, roč. 61, s. 85–117. Dostupné z DOI: 10.1016/j.neunet.2014.09.003.
4. LECUN, Yann; BENGIO, Yoshua; HINTON, Geoffrey. Deep Learning. *Nature*. 2015, roč. 521, č. 7553, s. 436–444. Dostupné z DOI: 10.1038/nature14539.
5. FAUSETT, Laurene. *Fundamentals of Neural Networks: Architectures, Algorithms, and Applications*. Englewood Cliffs, NJ: Prentice-Hall, 1994. ISBN 978-0133341867.
6. ROSENBLATT, Frank. The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain. In: 1958, sv. 65, s. 386–408. Č. 6.
7. MINSKY, Marvin; PAPERT, Seymour. *Perceptrons: An Introduction to Computational Geometry*. Cambridge, MA: MIT Press, 1969.
8. RUMELHART, David E.; HINTON, Geoffrey E.; WILLIAMS, Ronald J. Learning Representations by Back-propagating Errors. *Nature*. 1986, roč. 323, s. 533–536. Dostupné z DOI: 10.1038/323533a0.
9. HEBB, Donald O. *The Organization of Behavior: A Neuropsychological Theory*. New York: Wiley, 1949.
10. NAIR, Vinod; HINTON, Geoffrey E. Rectified Linear Units Improve Restricted Boltzmann Machines. In: *International Conference on Machine Learning*. 2010. Dostupné také z: https://api.semanticscholar.org/CorpusID:15539264.

11. DATTA, Leonid. *A Survey on Activation Functions and their relation with Xavier and He Normal Initialization*. 2020. Dostupné z arXiv: 2004.06632 [cs.NE].

12. KRIZHEVSKY, Alex; SUTSKEVER, Ilya; HINTON, Geoffrey E. ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems*. 2012, roč. 25, s. 1097–1105.

13. HOCHREITER, Sepp; SCHMIDHUBER, Jürgen. Long Short-term Memory. *Neural computation*. 1997, roč. 9, s. 1735–80. Dostupné z DOI: 10.1162/neco.1997.9.8.1735.

14. VASWANI, Ashish; SHAZEER, Noam; PARMAR, Niki; USZKOREIT, Jakob; JONES, Llion; GOMEZ, Aidan N.; KAISER, Lukasz; POLOSUKHIN, Illia. *Attention Is All You Need*. 2023. Dostupné z arXiv: 1706.03762 [cs.CL].

15. SRIVASTAVA, Nitish; HINTON, Geoffrey; KRIZHEVSKY, Alex; SUTSKEVER, Ilya; SALAKHUTDINOV, Ruslan. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* 2014, roč. 15, č. 1, s. 1929–1958. ISSN 1532-4435.

16. IOFFE, Sergey; SZEGEDY, Christian. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *CoRR*. 2015, roč. abs/1502.03167. Dostupné z arXiv: 1502.03167.

17. KINGMA, Diederik P; BA, Jimmy. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*. 2014.

18. GLOROT, Xavier; BENGIO, Yoshua. Understanding the difficulty of training deep feedforward neural networks. In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010, s. 249–256.

19. POWERS, David M. W. Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. *CoRR*. 2020. Dostupné z eprint: 2010.16061.

20. SOKOLOVA, Marina; LAPALME, Guy. A systematic analysis of performance measures for classification tasks. *Information Processing & Management*. 2009, roč. 45, s. 427–437. Dostupné z DOI: 10.1016/j.ipm.2009.03.002.

21. STRUBELL, Emma; GANESH, Ananya; MCCALLUM, Andrew. Energy and policy considerations for deep learning in NLP. 2019. Dostupné také z: https://arxiv.org/abs/1906.02243.

22. DALLY, William J; TURAKHIA, Yatish; HAN, Song. High-performance hardware for machine learning. *Computing in Science & Engineering*. 2015, roč. 17, č. 5, s. 31–38.

23. BENOIT JACOB Skirmantas Kligys, Bo Chen. *Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference*. 2017. Dostupné z arXiv: 1712.05877 [cs.LG].

24. RIBEIRO, Marco Tulio; SINGH, Sameer; GUESTRIN, Carlos. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. *CoRR*. 2016, roč. abs/1602.04938. Dostupné z arXiv: 1602.04938.

25. SIMONYAN, Karen; VEDALDI, Andrea; ZISSERMAN, Andrew. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*. 2014. Dostupné také z: https://arxiv.org/abs/1312.6034.

26. CHEN, Zhiyuan; LIU, Bing. *Lifelong Machine Learning, Second Edition*. Morgan & Claypool Publishers, 2018. Synthesis Lectures on Artificial Intelligence and Machine Learning. ISBN 1681733021.

27. KIRKPATRICK, James; PASCANU, Razvan; RABINOWITZ, Neil; VENESS, Joel; DESJARDINS, Guillaume; RUSU, Andrei A; MILAN, Kieran; QUAN, John; RAMALHO, Tiago; GRABSKA-BARWINSKA, Agnieszka et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*. 2017, roč. 114, č. 13, s. 3521–3526. ISSN 1091-6490. Dostupné z DOI: 10.1073/pnas.1611835114.

28. HOSPEDALES, Timothy; ANTONIOU, Antreas; MICAELLI, Paul; STORKEY, Amos. Meta-Learning in Neural Networks: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2021, roč. 44, č. 9, s. 5149–5169. Dostupné z DOI: 10.1109/TPAMI.2021.3079209.

29. SUN, Bai; GUO, Tao; ZHOU, Guangdong; RANJAN, Shubham; JIAO, Yixuan; WEI, Lan; ZHOU, Y. Norman; WU, Yimin A. Synaptic devices based neuromorphic computing applications in artificial intelligence. *Materials Today Physics*. 2021, roč. 18, s. 100393. ISSN 2542-5293. Dostupné z DOI: https://doi.org/10.1016/j.mtphys.2021.100393.

30. PETERS, Johannes; LEHNERT, Michael; KLCO, Natalie et al. Quantum Neural Networks - A practical guide. *arXiv preprint arXiv:2205.08154*. 2022. Dostupné také z: https://arxiv.org/abs/2205.08154.

31. YELLOW DOG MAN STUDIOS. *Resonite - Social VR Platform* [https://wiki.resonite.com]. 2024.

32. YELLOW DOG MAN STUDIOS. *ProtoFlux Documentation* [https://wiki.resonite.com/ProtoFlux]. 2024.

Obsah příloh