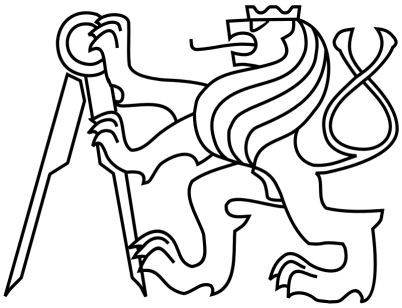
České vysoké učení technické v Praze
Fakulta jaderná a fyzikálně inženýrská

Obor: Matematické inženýrství
Zaměření: Aplikované matematicko-stochastické metody



Strojové učení pro klasifikaci zdrojů spojité akustické emise

Machine Learning for Classification of Continuous Acoustic Emission Sources

BAKALÁŘSKÁ PRÁCE

Vypracovala: **Tereza Fucsiková**
Vedoucí práce: **Ing. Milan Chlada, Ph.D.**
Konzultant: **Ing. Josef Krofta, Ph.D.**
Akademický rok: 2021/2022

České vysoké učení technické v Praze, Fakulta jaderná a fyzikálně inženýrská

Katedra: matematiky

Akademický rok: 2020/2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student:	Tereza Fucsiková
Studijní program:	Aplikace přírodních věd
Studijní obor:	Matematické inženýrství
Studijní zaměření:	Aplikované matematicko-stochastické metody
Název práce (česky):	Strojové učení pro klasifikaci zdrojů spojité akustické emise
Název práce (anglicky):	Machine Learning for Classification of Continuous Acoustic Emission Sources

Pokyny pro vypracování:

1) Rešerše základních principů strojového učení se zaměřením na rekurentní (RNN) a konvoluční (CNN) neuronové sítě.
2) Výběr softwarové platformy a implementace modelů RNN a CNN pomocí dostupných knihoven.
3) Rozvaha zpracování experimentálních měření a příprava tréninkových, validačních a testovacích množin adekvátního rozsahu.
4) Testování aplikace a optimalizace učení RNN, CNN a jejich kombinací na připravených datech.
5) Analýza výsledků klasifikace zdrojů spojité akustické emise pomocí implementovaných metod.

Doporučená literatura:

1) F. Chollet, Deep Learning with Python. Manning Publications Co., 2017.
2) S. Pal, A. Gulli, Deep Learning with Keras. Packt Publishing Ltd., 2017.
3) J. F. Kolen, S. C. Kremer, A Field Guide to Dynamical Recurrent Networks. Wiley-IEEE Press, 2001.
4) D. Eigen, J. Rolfe, R. Fergus, Y. LeCun, Understanding Deep Architectures using a Recursive Convolutional Network. arXiv:1312.1847, 2014.
5) J. L. Rose, Ultrasonic Waves in Solid Media. Cambridge University Press, 2004.

Jméno a pracoviště vedoucího bakalářské práce:

Ing. Milan Chlada, Ph.D.
Ústav termomechaniky AV ČR, v. v. i., Dolejškova 1402/5, 182 00 Praha 8

Jméno a pracoviště konzultanta:

Ing. Josef Krofta, Ph.D.
Ústav termomechaniky AV ČR, v. v. i., Dolejškova 1402/5, 182 00 Praha 8

Datum zadání bakalářské práce: 31.10.2020

Datum odevzdání bakalářské práce: 7.7.2021

Doba platnosti zadání je dva roky od data zadání.

V Praze dne 30.10.2020

.....
garant oboru

.....
vedoucí katedry

.....
děkan

Prohlášení

Prohlašuji, že jsem svou bakalářskou práci vypracovala samostatně a použila jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Praze dne ....

Tereza Fucsiková

Poděkování

Děkuji svému školiteli panu Ing. Milanu Chladovi, Ph.D. za odborné rady a čas, který mi věnoval. Dále děkuji rodině a přátelům, kteří mi byli vždy velkou oporou.

Tereza Fucsiková

Název práce:
Strojové učení pro klasifikaci zdrojů spojité akustické emise

Autor: Tereza Fucsiková

Obor: Matematické inženýrství
Zaměření: Aplikované matematicko-stochastické metody
Druh práce: Bakalářská práce

Vedoucí práce: Ing. Milan Chlada, Ph.D.

Konzultanti: Ing. Josef Krofta, Ph.D.

Abstrakt: Strojové učení je v současné době velmi populární oblastí umělé inteligence. Hluboké učení poté zpracovává vstupní data skrze vrstvy předávající datům smysluplnější reprezentace. Jedním z modelů, které hluboké učení k reprezentaci vrstev používá, jsou neuronové sítě. Konvoluční neuronové sítě (CNN) představují průlomovou architekturu pro rozpoznání obrazových dat, zároveň nabízejí 1D variantu pro časové řady. V práci jsou použity různé architektury CNN ke klasifikaci dat ultrazvukového signálu akustické emise, který je generován odkrajováním materiálu v důsledku vyvrtání díry vrtačkou. Zároveň je zde vytvořen programovací základ k jejich implementaci. Na vykreslených datech se nakonec diskutuje správnost fungování modelů.

Klíčová slova: strojové učení, hluboké učení, vrstevnatá neuronová síť, konvoluční neuronová síť, akustická emise, keras, tensorflow

Thesis title:

Machine Learning for Classification of Continuous Acoustic Emission Sources

Author: Tereza Fucsiková

Branch of study: Applied Mathematical Stochastic Methods
Kind of thesis: Bachelor thesis

Supervisor: Ing. Milan Chlada, Ph.D.

Consultants: Ing. Josef Krofta, Ph.D.

Abstract: Machine learning is currently a very popular field of artificial intelligence. Deep learning then processes the input data through layers imparting more meaningful representations to the data. One of the models that deep learning uses to represent layers is neural networks. Convolutional neural networks (CNNs) represent a breakthrough architecture for image recognition, while offering a 1D variant for time series. In this paper, different CNN architectures are used to classify ultrasonic acoustic emission signal data that is generated by material removal due to drilling a hole by a drill. Simultaneously, a fundamental code for their implementation is developed. Finally, the correct functioning of the models is discussed on the plotted data.

Key words: machine learning, deep learning, artificial neural network, convolution neural network, acustic emission, keras, tensorflow

Obsah

Úvod

Strojové učení usiluje o napodobení lidského uvažování. Člověk svými smysly umí velmi dobře rozpoznat různé objekty, zdroje zvuku aj. Důvodem je nashromáždění velkého množství informací o těchto datech za celý jeho život. Rozezná různé transformace objektů, jejich variace jako např. poškození, píseň za šumu z dobře nenaladěného rádia a mnoho dalších. Všechny tyto vjemy se uchovávají v mozku jako v největším učícím nástroji vůbec. Strojové učení pracuje s obrovským množstvím dat právě z důvodu, aby alespoň z části napodobil učící schopnost člověka.

Neuronové sítě zpracovávají data skrze vrstvy tvořící jejich různé reprezentace. Mají klíčovou učící schopnost, která je získána postupným upravováním parametrů na základě zpětné vazby o správnosti fungování sítě. Trénováním zlepšují jejich modely svou úspěšnost. Natrénovaný model lze poté pustit na neznámá data a získat tak odpovědi, které požadujeme. Výhodou je, že uložené natrénované modely lze použít i na data, která klasifikují zcela jiný problém a může se stát, že model bude prospěšný i pro tuto skupinu dat.

Pro správné fungování modelu je důležité nejen vyladit architekturu sítě, ale i její parametry. Programování neuronových sítí je o zkoušení nastavení různých parametrů, funkcí a metrik. Činíme tak dokud nám nějaká kombinace nezafunguje. Důležité je správně připravit data, správnou přípravou si lze ušetřit velké množství práce a času.

Tato práce pojednává o základech strojového učení a neuronových sítí. Zabývá se architekturami vrstevnatých sítí (ANN) a konvolučních sítí (CNN). Studuje přípravu dat, správné fungování sítě a řešení problémů s tím spojených. CNN nadále implementuje na experimentální měření zdroje spojité akustické emise.

Kapitola 1

Počátky neuronových sítí, jejich vývoj a současný význam

1.1 Umělá inteligence, strojové a hluboké učení

Vznik umělé inteligence (angl. *Artificial intelligence*, zkr. AI) se datuje již od roku 1950, kdy postupný rozvoj oboru počítačových věd zpřístupnil otázku, zda jsou počítače schopné "myslet". A. M. Turing ve své přelomové publikaci [1] z tohoto roku otázku strojového myšlení rozebírá a přichází k závěru, že stroje opravdu mohou uvažovat. Impulsem k zamyšlení pro něj bylo vyjádření Lady Lovelace k analytickému stroji z roku 1843 [2], které ve své publikaci cituje [9]. Tato otázka nás ovšem provází až dodnes a je základním kamenem, na kterém umělá inteligence buduje své další struktury.

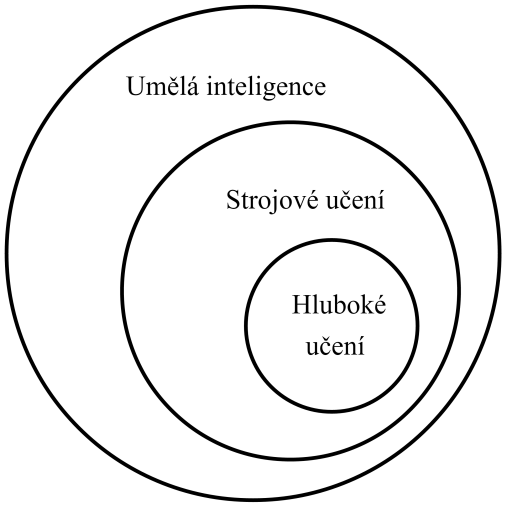
Definice umělé inteligence by mohla znít následovně: *snaha automatizovat intelektuální úlohy prováděné lidmi*. Umělou inteligencí rozumíme hlavní odvětví, které zahrnuje mimo jiné i strojové a hluboké učení, kterými se budeme v této práci zabývat. Propojení těchto tří odvětví znázorňuje obrázek 1.1.

Od řešení logických problémů, kterými jsou například šachové partie, ušla umělá inteligence dlouhou cestu až k řešení praktičtějších problémů, tzv. "fuzzy problems", které jsou bližší lidskému uvažování. Pod nimi si můžeme představit rozpoznávání obrazu, hlasu či jazykové překlady. Tento přístup vytvořil pod názvem *strojové učení* nový podobor umělé inteligence.

Strojové učení (angl. *Machine learning*, zkr. ML) se snaží docílit toho, aby byl počítač schopný naučit se sám, jak zadanou úlohu provést, a to při pouhém pohledu na data. Existují dva různé přístupy k programování. Klasické programování je založeno na přímém diktování pravidel, kterými se má program řídit za dosažením požadovaných výsledků. Přičemž musíme uvést veškeré možnosti a situace, které mohou v průběhu nastat. Strojové učení má však konstrukci odlišnou. Vstupní

[1] A. M. Turing, Počítačové stroje a inteligence (angl. *Computing Machinery and Intelligence*)

[2] "Analytický stroj nemá žádnou snahu cokoli vytvořit. Může dělat pouze to, u čeho víme, jak nařídit, aby to fungovalo... Jeho doménou je pomoci zpřístupnit to, s čím jsme se již seznámili."[1]



Obrázek 1.1: Souvislost hlavních odvětví

informací je již kategorizovaná sada dat, tedy známe vstup a požadovaný výstup (označení dat). Úkolem algoritmu je poté, aby se "naučil", jakým způsobem vstupní sadu dat zpracovat za účelem dosažení požadovaného výstupu. Vytvořená obecná pravidla pro automatizaci řešení jsou následně aplikována na nová neoznačená data určená k rozpoznání. Je důležité si uvědomit, že v případě strojového učení se jedná spíše o trénování, než-li vysloveně programování.

Systém mapování vstupních dat na známé výstupy se nazývá řízené učení (angl. *supervised learning*), též označováno jako *učení s učitelem*. Jedná se o jednu ze čtyř kategorií strojového učení. Řízené učení je nejběžnější a také předmětem zkoumání pro tuto práci. Dalšími kategoriemi jsou neřízené učení (angl. *unsupervised learning*), samořízené učení (angl. *self-supervised learning*) a posilované učení (angl. *reinforcement learning*). U neřízeného učení nejsou známa výstupní data, jedná se spíše o hledání zajímavých reprezentací dat skrze vrstvy sítě. Řízené učení, kde označení dat nebylo provedeno mechanicky člověkem, ale bylo vygenerováno ze vstupních dat (zpravidla za pomoci heuristického algoritmu), se nazývá samořízené učení. Nakonec při posilovaném učení se snaží neuronová síť z dostupných informací o svém prostředí na toto prostředí reagovat a maximalizovat tak nějaký zisk. Dobře přístupnou představou je realizace herních akcí za dosažením vysokého skóre. Zajímavou aplikací posilovaného učení pak může být řízení automobilů.

Nedílnou součástí strojového učení je vypořádat se s obrovským množstvím dat. Pokud například usilujeme o rozpoznání objektu na obrázku, pouze jeden vstupní obrázek nám zdaleka nebude stačit. Naopak se bavíme v řádech tisíců či milionů vstupních obrázků, pokud má být naše úspěšnost rozpoznání vysoká. Existuje ale pár tipů, jak vygenerovat další data z dat původních a datovou sadu rozšířit.

Co si představit pod výše zmíněným pojmem učení? Po obdržení výstupních (skutečných) dat je potřeba určit způsob ověření jejich odlišnosti od dat požadovaných, která jsou jedním ze vstupních parametrů a máme je tedy k dispozici. Informace o tom, jak se data vzájemně liší, je poté využita k úpravě algoritmu a zajištění jeho

správného fungování. Učením se tedy snažíme minimalizovat rozdíl skutečných a požadovaných výstupů. Jinými slovy se jedná o proces automatického vyhledávání užitečné reprezentace vstupních dat, která se přibližuje dostupnému výstupu. Činí tak za pomoci zpětnovazebního signálu.

Nyní již máme potřebný aparát k pochopení, co si pod pojmem *hluboké učení* představit. Hluboké učení (angl. *Deep learning*, zkr. DL) zpracovává vstupní data skrze vrstvy, které umožňují jejich stále smyslupnější reprezentaci. Hloubka modelu nám říká, kolik vrstev je použito. Pro lepší představu můžeme vrstvy chápat jako určité filtry vstupních dat, které tato data transformují za účelem získání požadovaného výsledku. *Neuronové sítě* jsou poté modely, jež hluboké učení používá k reprezentaci těchto vrstev.

1.2 Neuronové sítě

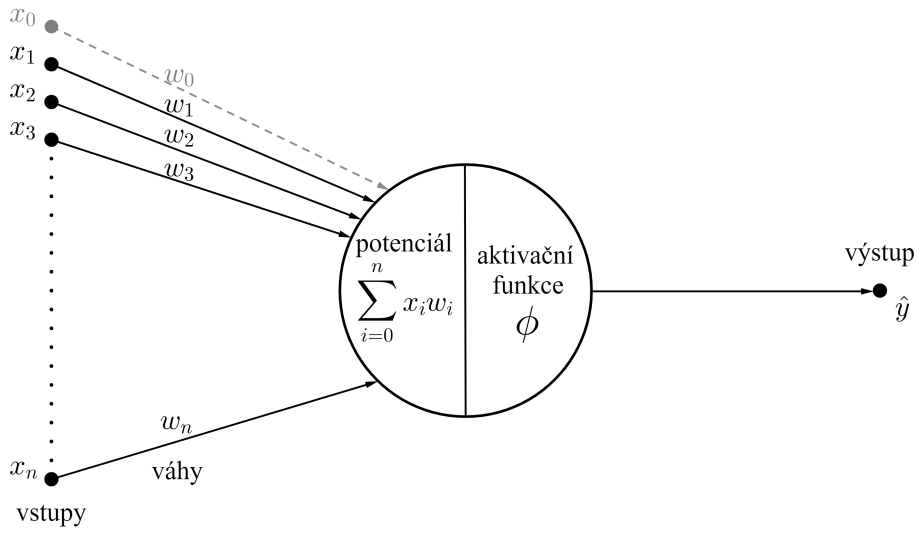
Termín neuronová síť (angl. *Neural network*, zkr. NN) se odkazuje na neurobiologii a je inspirován neurony v mozku. Lidský mozek je nejmocnějším učícím nástrojem na světě a zmíněná inspirace pramení ze snahy o napodobení jeho učící schopnosti. Samotný neuron je stejně jako v mozku, tak ve strojovém učení, bezvýznamný. Významu nabývá teprve ve vzájemné spolupráci s dalšími neurony. Odtud vzniká pojem neuronová síť.

1.2.1 Neuron

Základním stavebním prvkem neuronové sítě jsou vzájemně propojené neurony. S opětovnou inspirací v neurobilogii je propojení mezi neurony označováno jako *synapse*. Na obrázku 1.2 je znázorněn neuron strojového učení přijímající synapsemi vstupní hodnoty a jeho výstup. Vstupy i výstupy nazýváme opět neurony.

Každé synapsi je přazena *váha*. Vahami se modeluje míra propustnosti informace jednotlivých neuronů, čímž je jim připsána určitá míra důležitosti. V našem případě váhy $\{w_1, ..., w_n\}$ udávají význam vstupním neuronům $\{x_1, ..., x_n\}$ a podle velikosti tohoto významu jsou jejich informace předávány dále. Váhy jsou klíčovým pojmem pro učení neuronové sítě, tedy pro proces, při kterém se snažíme postupně adaptovat váhy tak, abychom se co nejvíce přiblížili požadovanému výsledku. Nejprve jsou vahám přiřazeny náhodné hodnoty generující náhodné transformace dat. Získáváme tak velký prvotní rozdíl mezi výstupy, který je následnou adaptací usměrňován. K nalezení uspokojivých vah z hlediska tolerance nastavené chyby, nebo-li k trénování neuronové sítě, budeme využívat *metody největšího spádu* a *algoritmu zpětného šíření chyby*, kterými se zabývají kapitoly 2.2 a 2.3.

Neuron má kromě výše zmíněných vstupů jeden speciální x_0, který je na obrázku 1.2 vykreslen šedou barvou. Vstup x_0 má konstantní hodnotu 1 a je mu přidělena váha w_0, též označována jako b z anglického *bias*. Český ekvivalent *práh* se užívá pro jeho zápornou hodnotu $\theta = -b$ (angl. *treshold*). Vážený součet vstupních hodnot $\boldsymbol{x} = (1, x_1, ..., x_n) \in \mathbb{R}^{n+1}$ a vektoru vah $\boldsymbol{w} = (b, w_1, ..., w_n) \in \mathbb{R}^{n+1}$ se nazývá



Obrázek 1.2: Matematický model neuronu

potenciál neuronu $\mu \in \mathbb{R}$ a platí pro něj vztah

$$\mu = \sum_{i=1}^{n} x_i w_i + b = \sum_{i=0}^{n} x_i w_i. \tag{1.1}$$

Po aplikaci tzv. *aktivační* (přenosové) *funkce* $\phi : \mathbb{R} \to \mathbb{R}$ dostaneme skutečný výstup, nebo-li predikci, $\hat{y} \in \mathbb{R}$ s předpisem

$$\hat{y} = \phi(\sum_{i=0}^{n} x_i w_i). \tag{1.2}$$

Aktivačními funkcemi se podrobněji zabývá podkapitola 2.1.

1.3 Rozvoj hlubokého učení

Nabízí se otázka, jak je možné, že rozmach a popularita neuronových sítí vzrostla až ve 21. století, ačkoliv již od roku 1989 máme zpřístupněny architektury konvolučních neuronových sítí a od roku 1997 algoritmus LSTM pro rekurentní sítě [1]. Odpověď na tuto otázku je prostá, rozmach byl pozastaven z důvodu omezené kapacity úložišť a nízké výkonnosti procesorů.

Strojové učení pracuje s obrovským množstvím dat, na kterém je také založeno. Představme si pevný disk z roku 1956, který náležel prvnímu "SUPER" počítači s názvem *305 RAMAC*. Tento pevný disk vážil tunu a jeho úložiště dat činilo pouhých 5MB. Zároveň je třeba zmínit cenovou dostupnost, která byla také velkým omezením, a to 2 500 dolarů za pronájem pevného disku na měsíc. Ani pevný disk z roku 1980 by nám nestačil, ačkoliv lze zaznamenat určitý pokrok. Úložiště dat bylo

zdvojnásobeno na 10MB a pořizovací cena činila 3 495 dolarů. V dnešní době nám však 10MB vystačí na uschování pouze jedné fotky. Konečně se však dostáváme k exponenciálnímu vzrůstu pokroku objemu pevného disku. Za 37 let jsme již měli dostupný disk *micro SD* s velikostí 256 GB za pouhých 150 dolarů. V současnosti však míříme stále výš a jako úložiště nám má posloužit DNA. Pokud by informace mohly být na sebe nabaleny tak hustě jako jsou v genech bakterie Escherichia coli, data na celém světě by se vešla do 1kg DNA [2].

Na vzestupu neuronových sítí měl vysoký podíl rozvoj internetu v důsledku vysoké poptávky herního trhu. Tento rozmach vedl k vývoji vysoce výkonných grafických čipů. Zároveň internet umožnil distribuci datových sad v obrovském měřítku. O šíření znalostí ze světa neuronových sítí se postaraly také soutěže na portálu *Kaggle* s odměnami pro nejlepší vytvořené architektury. Soutěživost a motivace v odměnách posunula neuronové sítě mnohem dál. Se zvýšenou popularitou strojového učení se zvyšovala poptávka velkých firem jako jsou Google, Microsoft a Facebook, které investovaly do vývoje velké sumy peněz. Podle studií inzerovaných v časopise TIMES jsme v roce 2015 překonali myšlení mozku myši a toho lidského bychom měli dosáhnout podle předpovědi již v roce 2023. Dovršení znalostí představuje rok 2045, ve kterém se očekává překonání myšlení všech lidských mozků dohromady.

Kapitola 2

Vrstevnaté neuronové sítě (ANN)

Vrstevnatá neuronová síť (angl. *Artificial neural network*, zkr. ANN) představuje základní model, z něhož vycházejí složitější architektury neuronových sítí. Skládá se ze vstupní vrstvy, několika skrytých a vrstvy výstupní. Vstupní informace se šíří sítí postupně po jednotlivých vrstvách a není tedy možné mezi nimi přeskakovat. Obrázek 2.1 představuje schéma vrstevnaté neuronové sítě o n skrytých vrstvách. Fiktivní vstupy sítě jsou pro přehlednost uvedeny šedou barvou ve schématu 2.2 doplňujícím zmíněný obrázek 2.1. Výstupy sítě $\{\hat{y}_1, ..., \hat{y}_p\} \in \mathbb{R}$, kde $p \in \mathbb{N}$, se poté vypočítají jako

$$\hat{y}_j = \phi^{(n)}(\sum_{k=1}^{m_n} x_k^{(n)} w_{kj}^{(n)} + b_j^{(n)}) = \phi^{(n)}(\sum_{k=0}^{m_n} x_k^{(n)} w_{kj}^{(n)}) \overset{ozn.}{=} \phi^{(n)}(\mu_j^{(n)}) \tag{2.1}$$

pro $j \in \{1, ..., p\}$, kde $\phi^{(n)} : \mathbb{R} \to \mathbb{R}$ je aktivační funkce n-té vrstvy[1], $\{1, x_1^{(n)}, ..., x_{m_n}^{(n)}\} \in \mathbb{R}$ jsou neurony v n-té vrstvě a písmena $\{b_j^{(n)}, w_{1j}^{(n)}, ..., w_{m_n j}^{(n)}\} \in \mathbb{R}$ označují váhy určující propustnost informací z neuronů poslední skryté vrstvy do neuronů vrstvy výstupní. Neurony n-té vrstvy se vypočítají iteračním způsobem pomocí neuronů předchozích vrstev a jejich vah pro $i \in \{1, ..., n\}$, kde i indexuje jednotlivé vrstvy jako

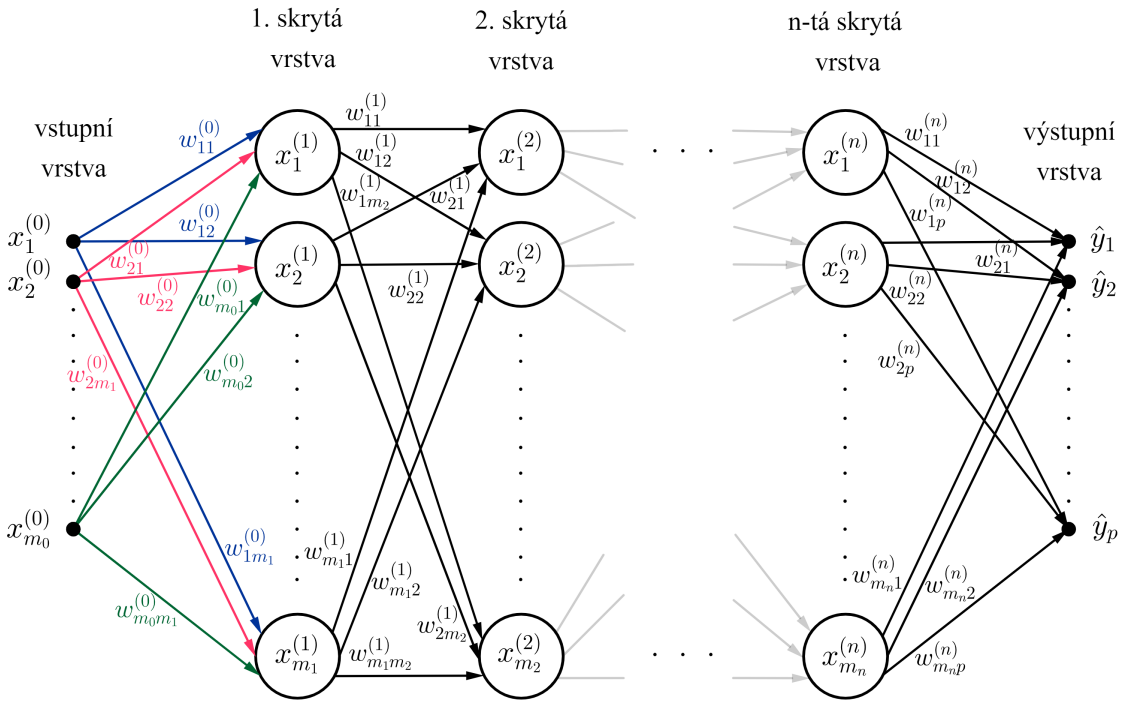
$$x_k^{(i)} = \phi^{(i-1)}(\sum_{l=0}^{m_{i-1}} x_l^{(i-1)} w_{lk}^{(i-1)}) \overset{ozn.}{=} \phi^{(i-1)}(\mu_j^{(i-1)}) \tag{2.2}$$

pro $k \in \{0, 1, ..., m_i\}$. Symboly $\{\phi^{(0)}, ..., \phi^{(n-1)}\} : \mathbb{R} \to \mathbb{R}$ označujeme aktivační funkce vstupní vrstvy a prvních $(n-1)$ vrstev výstupních[2], stejný princip indexace pro vrstvy je poté použit i pro vektory neuronů $\{x_k^{(0)}, ..., x_k^{(n)}\} \in \mathbb{R}$, ke kterým přiřazujeme vektory vah $\{w_{lk}^{(1)}, ..., w_{lk}^{(n)}\} \in \mathbb{R}$.

Vstupní vrstva (angl. *input layer*, její hodnoty angl. *samples* nebo *inputs*) je tvořena neurony, které distribuují vstupní informaci do 1. skryté vrstvy. Aby neuronová

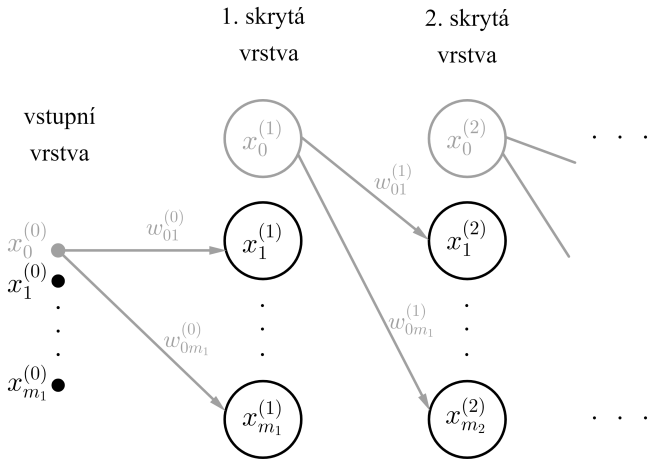
[1] Obecně pro vektor $\hat{\boldsymbol{y}} = (\hat{y}_1, ..., \hat{y}_p) \in \mathbb{R}^p$ dostaneme zobrazení $\phi^{(n)} : \mathbb{R}^p \to \mathbb{R}^p$.

[2] Obecně pro vektor $\boldsymbol{x}^{(i)} = (x_0^{(i)}, x_1^{(i)}, ..., x_{m_i}^{(i)}) \in \mathbb{R}^{m_i}$ dostaneme zobrazení $\phi^{(i-1)} : \mathbb{R}^{m_i} \to \mathbb{R}^{m_i}$ pro $i \in \{1, ..., n\}$, tedy pro $\boldsymbol{x} = (\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(n)})$ platí $\boldsymbol{\phi} = (\phi^{(0)}, ..., \phi^{(n-1)}) : \mathbb{R}^{m_n} \to \mathbb{R}^{m_n}$.



Obrázek 2.1: Schéma vrstevnaté neuronové sítě

síť správně fungovala, je důležité umožnit jí vstupní hodnoty snáze zpracovat. K tomuto účelu se využívají *standardizace* a *normalizace* popsané v kapitole 4.1.



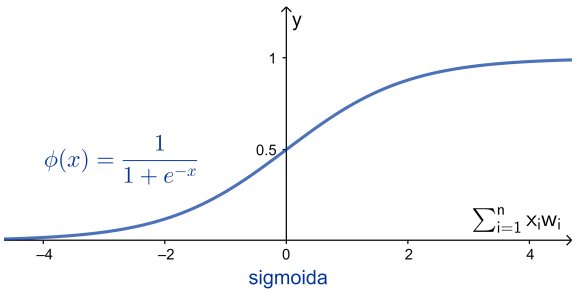
Obrázek 2.2: Fiktivní vstupy sítě

Reprezentace vrstvy výstupní (angl. *output layer*, její hodnoty angl. *outputs* nebo *predictions*) může být binární, spojitá (např. cena) či kategorická. V prvních dvou případech se jedná o jeden výstup, poslední má pak výstupů, nebo-li kategorií (tříd), více. Spojitý výstup nazýváme *regrese* a kategorický *klasifikace*. Schéma 2.1 má binární výstup, pokud $p = 1$, kategorický pro $p > 1$ a regresi tvoří v případě, že výstupem je spojitý vektor $\hat{\boldsymbol{y}} = (\hat{y}_1, ..., \hat{y}_p) \in \mathbb{R}^p$, tzv. vektorová regrese, nebo spojitá skalární hodnota $\hat{y} \in \mathbb{R}$ pro skalární regresi (např. predikce cen nemovitostí).

2.1 Aktivační funkce

Aktivační funkce nám umožňují získat hodnotu součtu $\sum_{i=1}^{n} x_i w_i$ v námi zvoleném intervalu. Existuje široké spektrum takových funkcí s různými vlastnostmi. Jejich společným předpokladem je nelinearita. Operace sčítání a násobení tenzorů, které modelují výstupy vrstev, jsou lineární operace. Aplikací nelineární funkce se vrstva naučí mimo lineárních transformací i ty nelineární a zpřístupní tak větší prostor hypotéz. Dalším z předpokladů může být spojitá diferencovatelnost funkce, která je vyžadována u jedné z možností minimalizace ztrátové funkce, viz kapitola 2.3.

Pro vrstvu výstupní se obvykle používá *sigmoida* (obr.2.3), jejíž výstupem je číslo v intervalu 0 až 1 označující ve většině případů pravděpodobnost. Běžně je aplikována při problému binární klasifikace.



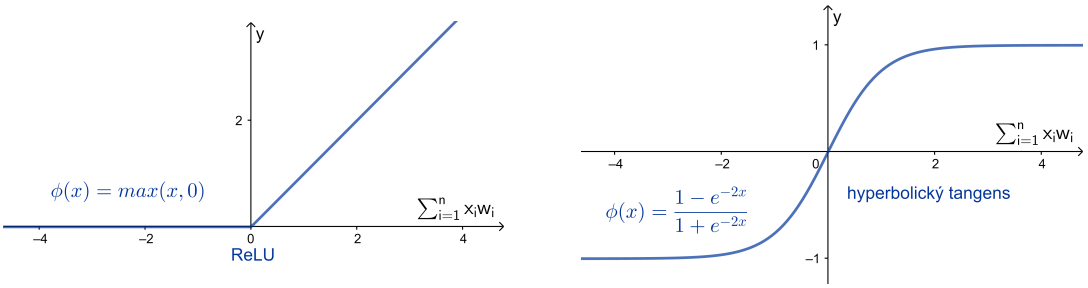
Obrázek 2.3: Aktivační funkce používaná ve výstupní vrstvě

Pro klasifikaci do více tříd pak poslouží funkce *softmax* s předpisem

$$\phi(x_i) = \frac{e^{x_i}}{\sum_{j=0}^{n} e^{x_j}}, \tag{2.3}$$

která obdobně jako funkce sigmoida udává pravděpodobnost. Tedy zajišťuje, že součet hodnot pro jednotlivé kategorie je roven 1. S narůstající hodnotou x roste i pravděpodobnost.

Na skryté vrstvy lze aplikovat hyperbolický tangens, který má podobnou strukturu jako sigmoida, jejich rozdílem je obor hodnot. Oproti sigmoidě tedy dokáže lépe reprezentovat záporné hodnoty. V současnosti je však populární volbou pro skryté vrstvy funkce ReLU (zkratka pro angl. *Rectifier Linear Unit*), která vykazuje pozoruhodné výsledky, viz studie v článku [3]. ReLu není spojitě diferencovatelná v 0. V praxi jednoduše poslouží $\phi'(0) = 0$ jako dodatečný předpoklad. Obě funkce jsou vykresleny na obrázku 2.4.



Obrázek 2.4: Aktivační funkce používané ve skrytých vrstvách

2.2 Učení vrstevnatých sítí

K učení neuronové sítě je nejprve potřeba porovnat odlišnost skutečného výstupu (angl. *target*) $\hat{\boldsymbol{y}}$ a požadovaného (angl. *prediction*) výstupu $\boldsymbol{y}$. Za tímto účelem je definována tzv. ztrátová funkce (angl. *cost function* nebo *loss function*), která mapuje, jak dobře síť funguje. Jelikož zachycuje způsobenou chybu v predikci, cílem je tuto funkci minimalizovat se záměrem přiblížení výstupů až na předepsanou hodnotu. Jinými slovy to znamená analyticky zjistit kombinaci hodnot vah, které poskytují nejmenší možnou ztrátu [1]. Nejpoužívanějšími ztrátovými funkcemi jsou:

1. **Střední kvadratická chyba** (angl. *mean squared error*, zkr. MSE)

$$C_{MSE} = \frac{1}{2} \sum_{j=1}^{p} (\hat{y}_j - y_j)^2. \tag{2.4}$$

2. **Křížová entropie** (angl. *Cross-Entropy*)

$$C_{CE} = -\sum_{j=1}^{p} y_j \log \hat{y}_j. \tag{2.5}$$

3. **Binární křížová entropie** (angl. *Binary Cross-Entropy*)

$$C_{BCE} = -\sum_{j=1}^{p} [y_j \log \hat{y}_j + (1 - y_j) \log(1 - \hat{y}_j)]. \tag{2.6}$$

Proměnná p udává počet tříd. Střední kvadratická chyba se obvykle používá pro regresi ($p = 1$), křížové entropie pak pro klasifikaci. V binárním problému máme 2 třídy ($p = 2$). Informace o odlišnosti výstupů se poté vrací neuronovou sítí zpátky, kde se na základě této zpětné vazby upraví hodnoty vah.

Minimalizace ztrátové funkce (chyby) obvyklým způsobem, tedy zkoušením různých možností nastavení jednotlivých vah a hledáním minima funkce pro tyto výstupy, je ve většině případů neefektivní. Mohla by dobře fungovat při optimalizaci pouze jedné váhy. Avšak vzrůstem počtu synapsí a vah s nimi, se zvětšuje i počet dimenzí. Rozměrový problém řeší optimalizační metoda tzv. *metoda největšího spádu*

(angl. *gradient descent*, zkr. GD) umožňující limitním přechodem přes iterace dosáhnout lokálního minima. V každé iteraci je sestrojen tečný vektor ke grafu ztrátové funkce v bodě se souřadnicemi aktuálních vah. Přizpůsobení vah probíhá v záporném směru gradientu[3], čímž je zajištěno dosažení zmiňovaného lokálního minima, tj. minimalizace ztrátové funkce. Iterace s krokem k mají následující předpis:

$$w_{ij}^{(l)}(k+1) = w_{ij}^{(l)}(k) - \alpha \frac{\partial C(W(k))}{\partial w_{ij}^{(l)}}, \tag{2.7}$$

kde $W = \{W^{(0)}, ..., W^{(n)}\} = \{w_{ij}^{(0)}, ..., w_{ij}^{(n)}\}$ označuje všechny váhy sítě. Parametr učení α (angl. *learning rate*) určuje rychlost učení, tj. rychlost adaptace vah. Ačkoliv s větší hodnotou α konverguje ztrátová funkce k minimu rychleji, může dojít k "přeskočení" lokálního minima důsledkem příliš velkého adaptačního kroku. Naopak moc malé hodnoty α patrně vedou k nalezení uspokojivých vah, ale za cenu velmi velké časové náročnosti. Přičemž malé kroky navíc mohou dospět k "ukotvení" hodnot v lokálním minimu funkce a zamezení nalezení minima globálního. Parametr je tedy potřeba správně naladit a za účelem nalezení globálního minima umožnit jeho průběžnou aktualizaci založenou na předchozích hodnotách gradientu. Efektivní metodou vyčíslení hodnot gradientu je *algoritmus zpětného šíření chyby*, který využívá předchozích výsledků a snižuje tak složitost výpočtu. Důležitým předpokladem pro jeho použití je diferencovatelnost ztrátové funkce.

2.3 Algoritmus zpětného šíření chyby

Robustnost algoritmu zpětného šíření chyby (angl. *backpropagation*, zkr. backprop, BP) je viditelná z jeho matematického zápisu. Nejprve vypočteme gradient ztrátové funkce derivacemi podle vah synapsí vedoucích do výstupní vrstvy $w_{rs}^{(n)}$[6]. Tato volba prvního kroku bude objasněna dalšími výpočty. Využijeme vztahů střední kvadratické chyby (2.4) a výstupu ANN (2.1). Ztrátová funkce není přímou funkcí od $w_{rs}^{(n)}$, proto ji budeme derivovat jako funkci složenou

$$\frac{\partial C_{MSE}}{\partial w_{rs}^{(n)}} = \frac{\partial C_{MSE}}{\partial \hat{y}_s} \frac{\partial \hat{y}_s}{\partial \mu_s^{(n)}} \frac{\partial \mu_s^{(n)}}{\partial w_{rs}^{(n)}}. \tag{2.8}$$

Derivací dostaneme vztah, jehož dílčí výsledek si označíme jako $\delta_s^{(n)}$

$$(\hat{y}_s - y_s)(\phi^{(n)}(\mu_s^{(n)}))' x_r^{(n)} \overset{ozn.}{=} \delta_s^{(n)} x_r^{(n)}. \tag{2.9}$$

Pokračujeme dále v opačném chodu ANN, tj. derivací podle vah $w_{rs}^{(n-1)}$, přičemž využijeme vztahu (2.2) pro výstupy neuronů skrytých vrstev. Opět derivujeme složenou

[3]Gradient je tečný vektor vyjadřující největší růst. V našem případě udává záporně vzatý gradient největší spád.

funkci, ale zde musíme vzít v úvahu, že neurony n-té skryté vrstvy ovlivňují neurony ve výstupní vrstvě. Proto je potřeba do výpočtu zařadit sumu, která prochází indexy neuronů vrstvy výstupní

$$\frac{\partial C_{MSE}}{\partial w_{rs}^{(n-1)}} = \sum_{j=1}^{p} \frac{\partial C_{MSE}}{\partial \hat{y}_j} \frac{\partial \hat{y}_j}{\partial \mu_j^{(n)}} \frac{\partial \mu_j^{(n)}}{\partial w_{rs}^{(n-1)}}. \tag{2.10}$$

Výpočtem derivace prvních dvou členů a rozepsáním třetího jako derivaci složené funkce obdržíme výraz

$$\sum_{j=1}^{p} (\hat{y}_j - y_j)(\phi^{(n)}(\mu_j^{(n)}))' \frac{\partial \mu_j^{(n)}}{\partial x_s^{(n)}} \frac{\partial x_s^{(n)}}{\partial w_{rs}^{(n-1)}}, \tag{2.11}$$

který po derivaci zbývajících členů vypadá následovně

$$\sum_{j=1}^{p} (\hat{y}_j - y_j)(\phi^{(n)}(\mu_j^{(n)}))' w_{sj}^{(n)} (\phi^{(n-1)}(\mu_s^{(n-1)}))' x_r^{(n-1)}. \tag{2.12}$$

Ve vztahu rozeznáváme známou část z předchozího výsledku (2.9), kterou jsme si označili $\delta_s^{(n)}$, v našem případě $s = j$. Nakonec si opět označíme dílčí výsledek jako $\delta_s^{(n-1)}$ a získáme

$$\sum_{j=1}^{p} \delta_j^{(n)} w_{sj}^{(n)} (\phi^{(n-1)}(\mu_s^{(n-1)}))' x_r^{(n-1)} \overset{ozn.}{=} \delta_s^{(n-1)} x_r^{(n-1)}. \tag{2.13}$$

Postupujeme sítí dále směrem na začátek a derivujeme podle vah $w_{rs}^{(n-2)}$. Obdobně jako v předchozím případu (2.10) dostaneme

$$\frac{\partial C_{MSE}}{\partial w_{rs}^{(n-2)}} = \sum_{j=1}^{p} \frac{\partial C_{MSE}}{\partial \hat{y}_j} \frac{\partial \hat{y}_j}{\partial \mu_j^{(n)}} \frac{\partial \mu_j^{(n)}}{\partial w_{rs}^{(n-2)}}, \tag{2.14}$$

přičemž závislost (n-1)-ní skryté vrstvy vyjádříme pomocí sumy jako

$$\sum_{j=1}^{p} \Big((\hat{y}_j - y_j)(\phi^{(n)}(\mu_j^{(n)}))' \sum_{i=1}^{m_n} \frac{\partial \mu_j^{(n)}}{\partial x_i^{(n)}} \frac{\partial x_i^{(n)}}{\partial x_s^{(n-1)}} \frac{\partial x_s^{(n-1)}}{\partial w_{rs}^{(n-2)}} \Big), \tag{2.15}$$

kde jsme navíc vyčíslili první dvě derivace a poslední rozepsali jako derivaci složené funkce. Ve vypočtených derivacích opět vidíme vztah $\delta_s^{(n)}$ pro $s = j$. Derivováním zbylých členů dospějeme k vyjádření

$$\sum_{j=1}^{p} \Big(\delta_j^{(n)} \sum_{i=1}^{m_n} w_{ij}^{(n)} (\phi^{(n-1)}(\mu_i^{(n-1)}))' w_{si}^{(n-1)} (\phi^{(n-2)}(\mu_s^{(n-2)}))' x_r^{(n-2)} \Big). \tag{2.16}$$

Následné přeuspořádání členů vede k nalezení vztahu pro $\delta_s^{(n-1)}$, kde $s = i$ (viz (2.13)). Označením dílčího výsledku vznikne finální tvar

$$\sum_{i=1}^{m_n} \delta_i^{(n-1)} w_{si}^{(n-1)} (\phi^{(n-2)}(\mu_s^{(n-2)}))' x_r^{(n-2)} \overset{ozn.}{=} \delta_s^{(n-2)} x_r^{(n-2)}. \tag{2.17}$$

Obdobným způsobem se spočítají další derivace. Ve výsledcích (2.9), (2.13) a (2.17) nacházíme stejný vzor opakování, ze kterého dostáváme obecný vzorec pro výpočet derivací tvaru

$$\frac{\partial C_{MSE}}{\partial w_{rs}^{(l)}} = \delta_s^{(l)} x_r^{(l)}, \tag{2.18}$$

kde pro $l \in \{0, .., (n-1)\}$ platí

$$\delta_s^{(l)} = (\phi^{(l)}(\mu_s^{(l)}))' \sum_{i=1}^{m_n} \delta_i^{(l+1)} w_{si}^{(l+1)} \tag{2.19}$$

a speciálně pro $l = n$ dostaneme

$$\delta_s^{(l)} = (\hat{y}_s - y_s)(\phi^{(n)}(\mu_s^{(n)}))'. \tag{2.20}$$

Po spočtení první složky gradientu jsme se při výpočtech dalších složek vždy odkazovali na předchozí výsledky, konkrétně na již známé členy $\delta_s^{(l)}$. Z tohoto důvodu bylo zvoleno pořadí vyčíslení složek od poslední vrstvy, odtud také vznikl název algoritmu *zpětné šíření chyby*. Navíc obtížnost počítání derivací směrem k vstupní vrstvě stoupala braním v úvahu ovlivnění chybové funkce dalšími vrstvami. Ulehčení zahrnutím dříve vypočtených částí vyčíslení značně usnadnilo. Získali jsme efektivní postup spočtení gradientu ztrátové funkce pro metodu největšího spádu a následnou adaptaci vah[6].

Za podmínky, že se jedná o jednodimenzionální prostor s konvexní ztrátovou funkcí, je lokální minimum i tím globálním a metoda největšího spádu správně minimalizuje ztrátovou funkci. V případě použití jiné než konvexní ztrátové funkce, nebo práce v multidimenzionálním prostoru, je vhodné učinit stochastickou aproximaci skutečného gradientu (angl. *stochastic gradient descent*, zkr. SGD). Důvodem je velká výpočetní náročnost přímého výpočtu. Narozdíl od metody největšího spádu, která nejprve bere všechna vstupní data a až poté upravuje váhy, se zde náhodně [4] vybere podmnožina vstupů, pro kterou jsou váhy upravovány zvlášť.

[4] Odtud slovo "stochastický" v názvu metody.

Kapitola 3

Konvoluční neuronové sítě (CNN) pro časové řady

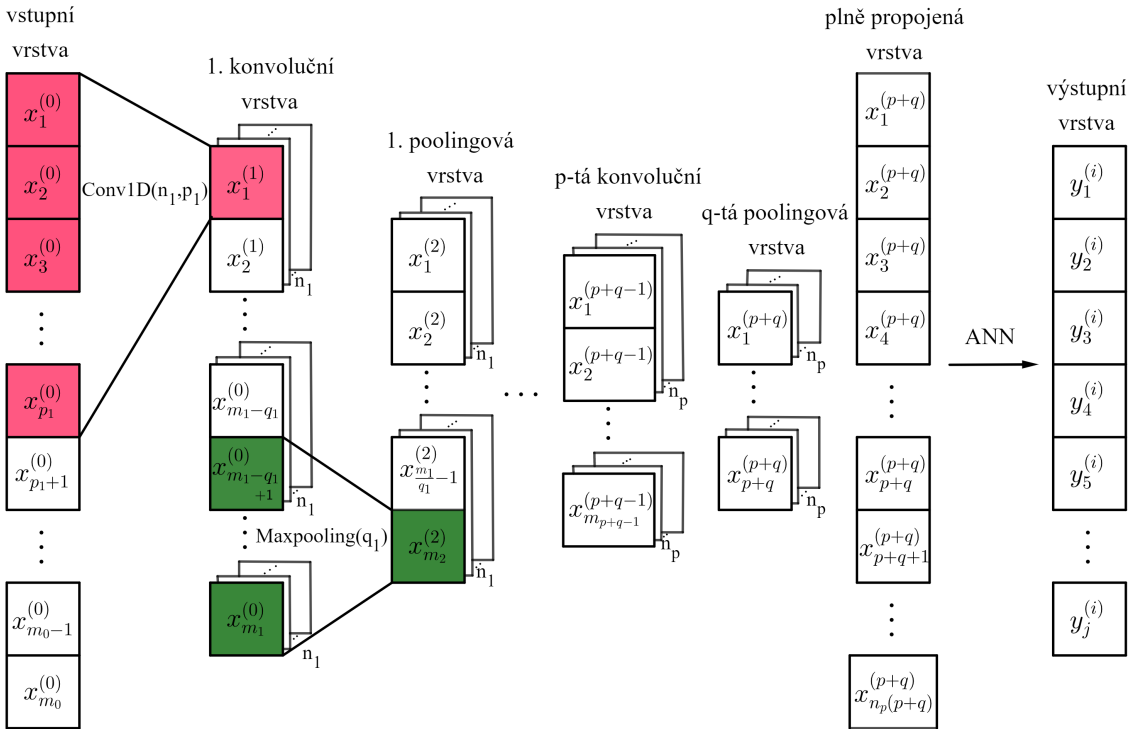
Konvoluční neuronové sítě (angl. *Convolutional neural networks*, zkr. CNN) usilují o napodobení další průlomové vlastnostni lidského mozku pro rozpoznávání daných objektů. Konkrétně se jedná o jeho schopnost nalézt určité rysy či charakteristiky, které jsou pro daný objekt typické a na základě této znalosti je mozek schopen objekt kategorizovat a rozpoznat.

Konvoluční sítě byly původně vyvinuty pro rozpoznání objektů na obrázcích. Jejich průlomovou vlastností pro tuto kategorii klasifikace je schopnost invariance vůči rotaci či translaci objektu[5]. Neřeší totiž rozpoznání obrazu jako globální problém, ale soustředí se pouze na lokální vzory objektu na něm. Nepracuje tedy se všemi pixely obrázku, ale rozděluje ho podle lokálních charakteristik objektu. Další důležitou vlastností je schopnost naučit se prostorové hierarchie vzorů [1]. To znamená, že se po vrstvách učí nejprve malé lokální vzory a poté pokračuje těmi většími (skládáním menších vzorů). Všechny tyto vlastnosti vizuální svět má a vyžaduje je při rozpoznávání. Výhodou vlastnosti invariance je schopnost trénování na relativně malé množině dat z důvodu efektivity rozpoznání lokálních příznaků.

Struktury CNN se dají také implementovat na klasifikaci časových řad a sekvencí. Činí se tak buď tvorbou spektogramů[1], nebo redukcí operací jednotlivých vrstev CNN na 1D problém. Schéma CNN pro 1D vstup, časovou řadu, vyjadřuje obrázek 3.1.

Existují 3 základní typy vrstev CNN: konvoluční vrstva, operace sdružování dle maxima (také poolingová vrtsva, odvozeno od angl. *pooling*) a plně propojená vrstva. Schéma CNN se skládá ze série konvolučních a poolingových vrstev, které zakončuje plně propojená vrstva. Vektor plně propojené vrstvy je vstupem do klasické ANN známé z předchozí kapitoly. Matematické operace ve vrstvách jsou podrobněji rozebrány v následujících podkapitolách.

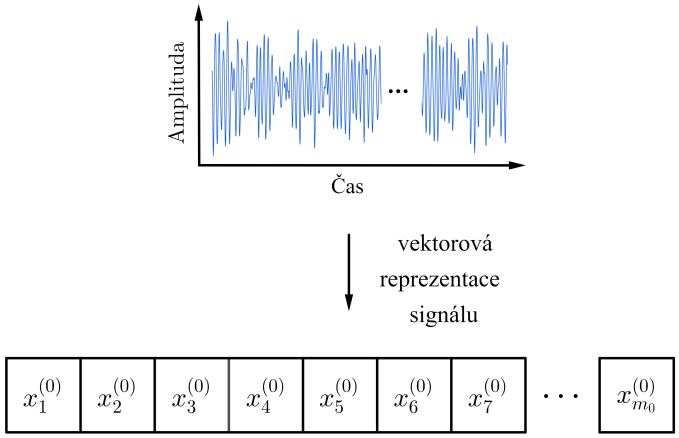
[1] Skeptogram je obrazové vyjádření signálu.



Obrázek 3.1: Zpracování 1D sekvence pomocí architektury CNN

3.1 Konvoluční vrstva

Vstupní data prochází nejprve tzv. konvoluční vrstvou (angl. *convolutional layer*). Vstupy tvoří časové řady reprezentující signál, tedy vektory obsahující reálná čísla (viz obr. 3.2). Úvodní schéma 3.1 ukazuje zpracování jedné sekvence, datová sada má však několik časových řad pro jednotlivá měření. Řešením je zmultiplikování do dimenze počtu sekvencí.



Obrázek 3.2: Zpracování signálu

Konvoluce je matematická operace dvou funkcí, která ukazuje, jak jedna funkce mo-

difikuje tvar té druhé. V CNN se využívá diskrétní konvoluce, lze vyjádřit iteracemi pro $i \in \{1, ..., (m_j - p_{j+1} + 1)\}$ ve tvaru

$$(x^{(j)} * k)(i) = \sum_{l=1}^{p_{j+1}} x^{(j)}(i + l - 1)k(l), \tag{3.1}$$

kde $x^{(j)}$ je vstupní signál pro j-tou vrstvu délky m_j a symbol k označuje tzv. *konvoluční jádro* délky p_{j+1}[11]. Označení navazuje na schéma 3.1, přičemž operace konvoluce je v něm znázorněna růžovou barvou. Konvoluční jádro (angl. *kernel*) si lze představit jako filtr, který postupně transformuje data signálu. Transformace probíhá posouváním okna filtru přes vstupní signál. Tím vzniká mapa příznaků (angl. *feature map*). Mapy příznaků tvoří různé transformace, které slouží např. pro detekci hran, zaostření nebo rozmazání. Počet filtrů poté určuje hloubku mapy příznaků, na obrázku 3.1 označované jako n_j, kde $j \in \hat{p}$ udává pořadí dané konvoluční vrtsvy. Hloubka map příznaků má v síti obvykle[2] rostoucí tendenci, naopak velikost map příznaků se postupně snižuje[10].

Výhoda použití 1D konvoluční vrstvy pro zpracování časových řad oproti vytvoření spektogramů a aplikaci 2D CNN spočívá ve zpřístupnění složitějších architektur díky sníženému počtu dimenzí.

3.2 Operace sdružování dle maxima

Operace sdružování dle maxima (angl. *max-pooling*) prochází signál posouváním okna obdobně jako u konvoluční vrstvy, viz zelená část na obrázku 3.1. Nyní však už není k dispozici konvoluční jádro a okno slouží pro rozlišení částí vstupního signálu, u kterých se bere maximální hodnota. Konkrétně pro schéma 3.1 mají výstupy j-té vrstvy tvar

$$x_k^{(j)} = max(x_{(k-1)q_l+1}^{(j-1)}, ..., x_{kq_l}^{(j-1)}), \tag{3.2}$$

kde $k \in \hat{m}_2$, $kq_l \in \hat{m}_1$ a q_1 je parametr operace sdružování dle maxima, tj. velikost okna procházejícího části signálu, přičemž $l \in \hat{q}$ vyjadřuje pořadí vrstvy poolingu.

3.3 Zploštění

Zploštění (angl. *flattening*) transformuje hodnoty tenzoru tvaru $(p + q, n_p)$ do 1D vektoru délky $n_p(p + q)$, viz obr. 3.1. Takový vektor je vstupem do klasické ANN, která byla představena v kapitole 2. Transformace probíhá složením vektorů délky $p + q$.

[2]Záleží na nastavení velikosti hloubky uživatelem.

Kapitola 4

Programové rozhraní

4.1 Příprava dat

Před zavedením dat do neuronové sítě je potřeba vstupní data a skutečné výstupy připravit (angl. *preprocessing*), aby byla pro neuronovou síť lépe zpracovatelná. Prvními kroky je provést vektorizaci a klasifikaci dat. Vektorizace převádí vstupní data na tenzory datového typu `float`[1]. K vytvořenému tenzoru vstupních dat je potřeba vytvořit tenzor jejich cílů, tedy tenzor označení dat podle příslušných tříd. Tento proces se nazývá klasifikace dat.

Obecně není dobré dávat neuronové síti příliš velké hodnoty dat, které se výrazně liší od počátečního nastavení vah sítě. Příčinou může být narušení konvergence vah k minimální ztrátové funkci. K tomuto účelu se zavádí *normalizace* dat, která převede hodnoty do intervalu $[0, 1]$ nebo $[-1, 1]$. Podobně by data neměla být heterogenní, tj. rozsah hodnot mezi třídami by neměl být odlišný. Řešením je *standardizace* převádějící data různých tříd do zhruba stejného rozsahu. Dosáhne tak nastavením průměru všech dat na hodnotu 0 a směrodatnou odchylku na 1.

4.1.1 Rozdělení datové sady

Posledním krokem přípravy dat před vstupem do neuronové sítě je jejich rozdělení na trénovací a testovací množiny. Obvykle se data dělí na 80% trénovacích a zbylých 20% testovacích. Na trénovacích datech se neuronová síť učí. Testovací data jsou poté puštěna na již naučenou síť. Je důležité si uvědomit, že testovací data síť doposud neviděla. Důvodem pro takové rozdělení je zjistit, jak správně síť funguje na neznámých datech. Výkonnost neuronové sítě má tendenci se na trénovacích datech neustále zlepšovat a přizpůsobovat se charakteristickým znakům těchto dat. Problém nastává, když se přizpůsobí do takové míry, že již nedá prostor charakteru nových dat. Takovému stavu naučené sítě se říká *přeučení* a je podrobněji rozepsán v následující kapitole 4.2. Rozdělením dat se tedy snažíme předejít sklonu sítě se přeučovat.

[1] `Float` je označení v programovacích jazycích pro datový typ reprezentující reálná čísla.

Po rozdělení dat na trénovací a testovací množiny, je dobré vytvořit třetí *validační* množinu dat. Vytváří se oddělením několika dat z trénovací množiny. Na validačních datech, která předtím nebyla síti zpřístupněna, lze během trénování pozorovat ztrátu a správnost fungování sítě. Vyhodnocení modelu na validačních datech se použije jako zpětnovazební signál sítě, pomocí kterého se ladí parametry jako počet a velikost vrstev během samotného trénování sítě. Část informace o charakteru validačních dat se během trénování sítě dostane do modelu a sklon k přeučování sítě hrozí i u této množiny dat. Z tohoto důvodu byla na začátku oddělena testovací data, která se pouští na již natrénovaný model sítě.

Nevýhodou rozdělení vstupních dat na tři množiny je ztráta počtu celkového množství dat, když přihlédneme k již známé skutečnosti, že dostatečně velké množství dat je pro strojové učení velmi klíčové. Pro shrnutí funkcí jednotlivých množin je trénovací množina dat určená k trénování sítě, na validační množině dat se ověřuje model sítě a zkoušení modelu se provádí na testovacích datech.

4.2 Přeučení a nedoučení

Přeučení (angl. *overfitting*) nastává u trénovacích a validačních množin dat. Přeučená neuronová síť podává lepší výsledky na naučených datech než na doposud neznámých testovacích. Tento problém nastává při každém učení a je potřeba naučit se s ním vypořádat. Klíčem k úspěchu je správně vybalancovat optimalizaci s generalizací.

Cílem optimalizace je vyladit model sítě a jeho parametry na trénovacích datech tak, abychom dosáhli nejlepších možných výsledků. Generalizace poté mapuje chování natrénované sítě na datech, která nikdy předtím neviděla. Zpočátku jsou při trénování optimalizace s generalizací v korelaci, tj. se zmenšováním ztráty na trénovacích datech je zmenšována ztráta i na těch testovacích. Taková síť je prozatím nedoučená (angl. *underfit*), protože se stále ještě učí charakteristické vzory a výsledky zlepšuje. Po několika iteracích síť dospěje k bodu, kdy se na trénovacích datech výsledky stále zlepšují, ale validační množina dat naopak vykazuje stále horší výsledky a začne se přeučovat. Učí se vzory specifické pro trénovací sadu dat, které jsou pro testovací data vedlejší. Ideální model sítě je na hranici podučení a přeučení.

Existuje několik možností, jak přeučení neuronové sítě zabránit. Jednou z nich je mít dostatečně velký dataset, aby se již při trénování sítě vyfiltrovaly pouze důležité vzory a těm zavádějícím nebyl dán prostor. Další možností je snížit velikost modelu, tj. počet vrstev a jejich skrytých jednotek. Sníží se tak počet učebních parametrů a předejde se učení perfektních vzorů. Počet učebních parametrů zároveň určuje kapacitu modelu, a pokud tato kapacita bude příliš malá na ukládání dat, nebude se síť učit správně. Řešením je vybalancovat tyto stavy nalezením ideální velikosti modelu. Obecným postupem pro takové nalezení je začít trénovat s menším počtem učebních parametrů a postupně tento počet zvyšovat podle pohybu validační ztráty.

Zmírnit sklon k přeučení sítě lze také váhovou regularizací, tj. zahrnutím ztrátové funkce napojené na hodnoty vah po dobu trénování sítě. Regularizátor se používá ve vrstvách a za každý koeficient vah ve vrstvě přidá malou hodnotu ke ztrátové

funkci. Čím jsou hodnoty vah větší, tím se zvětšuje i velikost ztrátové funkce. Cílem je tedy předat vahám malé hodnoty, které zaručí jejich pravidelnou distribuci v síti.

V neposlední řadě slouží k zamezení přeučení sítě tzv. výpadek (angl. *dropout*). Jde o náhodné vynulování části výstupů pro jednotlivé vrstvy v průběhu trénování sítě. Obvykle je vynulováno 20-50% příznaků. Podíl vynulovaných výstupů se nazývá míra výpadku. Tímto podílem jsou následně při testování sítě vynásobeny výstupy za účelem snížení jejich hodnot. Jsou tak vyrovnány výstupům z trénování sítě se způsobeným výpadkem. Zavedením takového šumu do výstupních hodnot vrstvy se dají rozdělit vzory nevýznamných náhodných událostí.

Posledním příkladem zmírnění přeučení, využívaným především pro CNN, je rozšíření dat (angl. *data augmentation*), též známé pod názvem augmentace. Z existujících dat je vygenerováno další množství trénovacích příznaků. Rozšíření dat probíhá pomocí náhodných transformací jako je náhodné otáčení, vertikální či horizontální posunutí, přiblížení, překlopení a jiné. V této práci bude augmentace využita v podobě generování nových vektorů dat metodou překrývání. Síť by neměla vidět stejné příklady dat dvakrát, ačkoliv jsou nově vygenerovaná data s originálními úzce svázána.

4.3 Programovací prostředí

Programovací jazyk zvolený pro tuto práci je *Python* s volbou vhodného frameworku pro podporu v programování[2]. První verze jednoho z nejpopulárnějších a nejrychleji se rozvíjejících frameworků pro hluboké učení *Keras*, který je v práci používán, vyšla v březnu roku 2015. Keras byl původně vyvinut pro výzkumníky s cílem umožnit jim rychlé experimentování [1]. Nyní je distribuován pod licencí MIT a může být tak používán i pro komerční účely. Zvlášť jeho vestavěná podpora pro konvoluční neuronové sítě, rekurentní neuronové sítě a jejich kombinace nalezne v této práci uplatnění. Keras je používán u společností Google, Netflix, Uber, CERN, Yelp, Square a dalších [1]. Matematické pozadí neuronových sítí jsou především tenzorové operace. Proto je jako backendový engine použit *Tensorflow* vyvinutý společností Google. Výhodou je vzájemná podpora programů Keras a Tensorflow. Python je importuje následovně

```
import tensorflow as tf
from tensorflow import keras.
```

4.4 Použité knihovny a funkce

Následující přehled knihoven a funkcí úzce souvisí konkrétně s touto prací. Jsou zde uvedeny pouze argumenty funkcí používané pro tuto práci. K zjištění ostatních argumentů lze nahlédnout do dokumentace Keras na odkazu `https://keras.io/`.

Pro usnadnění práce s velkými tenzory dat je dobré použít knihovnu `numpy`, která

[2]Podpora zahrnutím příslušných knihoven s některými již naimplementovanými funkcemi.

má již vestavěné základní operace jako je např. sčítání po prvcích. Importuje se příkazem

```
import numpy as np.
```

Model neuronové sítě je definován pomocí třídy `Sequential`, která se používá pro lineární množiny vrstev umožňujících pouze jeden vstup a jeden výstup. Přidávání vrstev do modelu zajišťuje metoda `add` třídy `layers`. K vytvoření modelu a přidávání vrstev slouží řádky

```
from keras.models import Sequential
from keras import layers
model = Sequential()
Sequential.add(layer),
```

kde je nejprve potřeba importovat potřebné třídy z frameworku Keras.

Vrstva ANN, tzv. hluboká vrstva (angl. *dense layer*) se volá příkazem `dense`. Jako první argument má počet skrytých jednotek vrstvy (angl. *units*), které udávají počet dimenzí reprezentačního prostoru. S větším množstvím skrytých jednotek je síť schopna naučit se komplexnější reprezentace, ovšem za cenu vyšší výpočetní náročnosti a sklonu k naučení se nežádoucích vzorů vedoucích k přeučení sítě. Počet jednotek u výstupní vrstvy je roven počtu kategorií. V dalším kroku se volí aktivační funkce vrstvy a posledním speciálním argumentem je specifikace velikosti vstupních dat (angl. *input shape*). Argument je speciální, protože ho lze použít pouze pro první vrstvu v modelu. Zbylé vrstvy si vygenerují tvar vstupu z vrstev předchozích.

Konvoluční vrstva pro 1D sekvence je aplikována prostřednictvím funkce `Conv1D` zpracovávající vstupní data tvaru (příklady, čas, funkce). V našem případě je první hodnotou počet vektorů časových řad, poté jejich délka a nakonec přidaná dimenze 1 (viz experimentální část). Tvar vstupních dat požaduje stejně jako u ANN pouze první konvoluční vrstva. Dalšími argumenty vrstvy jsou hloubka mapy příznaků (angl. *filters*), velikost konvolučního jádra (angl. *kernel size*) a aktivační funkce.

Funkce `Maxpooling1D` zpracovává operaci sdružování dle maxima pro sekvenční data. Jejím jediným argumentem je velikost okna procházející vstupy, ze kterých bere pouze maximální hodnotu. Místo transformace dat na 1D výstup metodou zploštění (angl. *flatten*) budeme používat operaci globálního sdružování dle maxima (angl. *global max-pooling*) s příkazem `GlobalMaxPooling1D` . Velikost okna je nyní celý vstupní vektor a počítá se tedy jediná maximální hodnota pro každý vektor hloubky sítě[8]. Důvodem užití této vrstvy oproti zploštění je značná redukce velikosti dat, lze tedy s nimi pracovat efektivněji. Předvedené vrstvy sítí jsou volány jako

```
from keras import layers
layers.Dense(units, activation=None, input_shape=None)
layers.Conv1D(filters, kernel_size, activation = None, input_shape =
None)
layers.Maxpooling1D(pool_size)
layers.GlobalMaxPooling1D()
model.summary().
```

V kompilačním kroku je konfigurován model pro proces učení sítě. Volí se zde optimalizátor, ztrátová funkce a příslušné metriky, které budou předmětem pozorování. Optimalizátor specifikuje konkrétní volbu stochastického gradientního sestupu (SGD). V souvislosti s touto prací je vybrán optimalizátor `RMSprop`, nebo-li stochastický gradientní sestup s hybností. Kompilaci zpracovává funkce `compile`. Trénování sítě je spuštěno metodou `fit`. Metoda si bere vstupní data a určujeme u ní velikost dávek (angl. *batch size*), počet epoch a podíl oddělení validační množiny dat z dat trénovacích (angl. *validation split*). Programují se následovně

```
from keras.optimizers import RMSprop
model.compile(optimizer = RMSprop(), loss = None, metrics = None)
model.fit(input, output, batch_size = None, epoch = 1,
validation_split = 0.0).
```

Uložením metody trénování sítě do proměnné `history` lze po natrénování vygenerovat hodnoty dat z jednotlivých epoch. Tato zpětná vazba je důležitá pro vykreslení grafů ztrát a úspěšností, pomocí kterých můžeme zhodnotit správnost fungování sítě. Pro uložení modelu a způsob generování jeho hodnot (trénovací a validační ztráta a jejich přesnost) se použijí příkazy

```
history = model.fit(input, output, batch_size = None, epoch = 1,
validation_split = 0.0)
history_dict = history.history
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
acc = history_dict['acc']
val_acc = history_dict['val_acc'].
```

Kapitola 5

Experimentální část

V předchozích kapitolách jsme si vybudovali základní teorii, kterou nyní převedeme do praxe. Než se však dostaneme k popisu experimentálního měření, zpracování dat a následnému učení neuronů, uveďme pojem akustické emise.

Akustickou emisí nazýváme elastické napěťové vlny generované dynamickým uvolněním mechanického napětí uvnitř materiálu tělesa nebo procesem působícím vznik elastických napěťových vln na povrchu tělesa [4]. Jinými slovy se snažíme detekovat akustické jevy doprovázející plastickou deformaci materiálu, jeho porušení, únik média pod tlakem a další. Činíme tak skrze výše zmíněné elastické napěťové vlny, které se šíří tělesem od zdroje deformace ke snímači. K měření dat budeme využívat metody akustické emise, jež zahrnuje samotnou detekci signálu akustické emise, jeho elektronické zpracování a následné vyhodnocení parametrů.

Signály akustické emise rozdělujeme na praskavé a spojité. Rozlišujeme je pomocí tzv. "hitů" detekovaného signálu, které jsou příčinou rázové vlny jevu způsobujícího vznik akustické emise. Praskavý signál akustické emise je tvořen posloupností dílčích časově separovaných hitů poté spojitý posloupností na sebe nahuštěných hitů, které se vzájemně překrývají a tvoří tak akustický šum.

Běžně se k měření akustické emise používá více snímačů, umožňují nám získat další informace o zdroji. Například rozdíly časového zpoždění detekce signálu zaznamenané různými snímači nám dávají informaci o místu původu zdroje. Odlišná vzdálenost snímačů od zdroje také způsobuje detekci různorodé intenzity signálu a jeho "ostrosti" či "rozmazanosti". Tyto pojmy úzce souvisí s tzv. *disperzí*, kdy se vlny vysokých frekvencí šíří pomaleji než vlny frekvencí nízkých. Porovnáním těchto rozdílů dostaneme informaci o charakteru šíření akustické emise v tělese. Rozšíření množství dat měřením z různých úhlů pohledu umožňuje určitou flexibilitu v následném rozpoznávání neuronovou sítí a nazýváme jej *augmentace*.

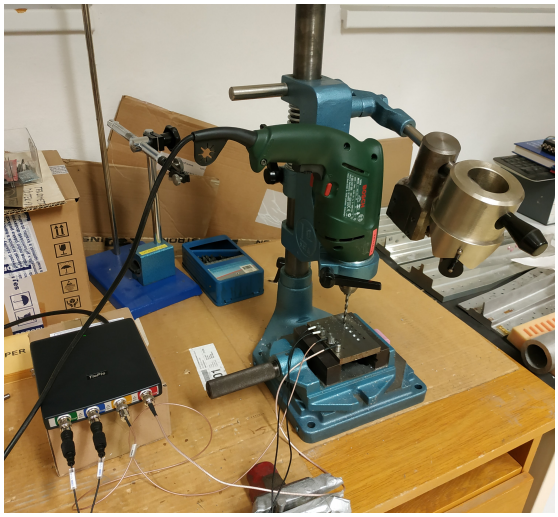
Kromě časových charakteristik zdrojů akustické emise máme i charakteristiky frekvenční. V případě zdrojů akustické emise hovoříme o frekvenčně širokopásmovém signálu. Nezískáme tedy pouze jednu dominantní frekvenci, ale širokopásmově vybudíme všechny. S touto problematikou souvisí volba frekvenčního pásma měření akustické emise. Volbou chceme eliminovat nízkofrekvenční působení vzdálených rušivých zdrojů a příliš vysoké frekvence způsobující potlačení citlivosti detekce zdrojů

akustické emise.

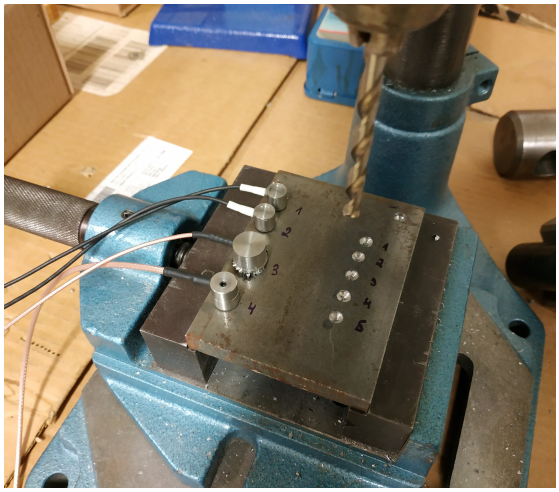
5.1 Provedení experimentu

V experimentu sledujeme ultrazvukový signál, který je generován odkrajováním materiálu v důsledku procesu vyvrtání díry vrtačkou. Cílem je postupné zaznamenávání tohoto signálu po stále intenzivnějším tupení vrtáku. Při vrtání dochází ke tření dvou nerovných ploch, kdy splynutím signálů jednotlivých poprasknutí (rázových poskoků v důsledku kontaktu výstupků) vzniká déle trvající signál charakteru laviny poprasknutí. Očekáváme, že ostrý vrták bude odkrajovat materiál po tenkých vrstvách a jeho elastické vlny se tak budou lišit od toho tupého, který by měl odkrajovat materiál hrubě. Stejný přítlak nám zajistí co největší amplitudovou stabilitu signálu. Předpoklady experimentu nám zaručují problematiku spojité akustické emise. V následujících řádcích si popíšeme aparaturu a postup experimentu.

Vrtačku firmy *BOSCH* jsme umístili do stojanu, který umožňoval nastavit v každém z měření zatížení 3kg, síla přítlaku tedy činila 30N (viz obr. 5.1). Při každém měření bylo do ocelové destičky o rozměrech 70 x 93 x 10 mm vyvrtáno 5 děr (viz obr. 5.2), přičemž doba vrtání jedné díry byla 5 s. První měření se provedlo s ostrým vrtákem, který byl následně v pěti krocích více otupen. K tupení jsme využívali brusného kamene na obrázku 5.3, ten byl přikládán k aktivnímu vrtáku za požadovaným účelem jeho otupení. Výsledkem je šest měření prováděných na pěti různých místech (pro 5 děr).



Obrázek 5.1: Uspořádání experimentu

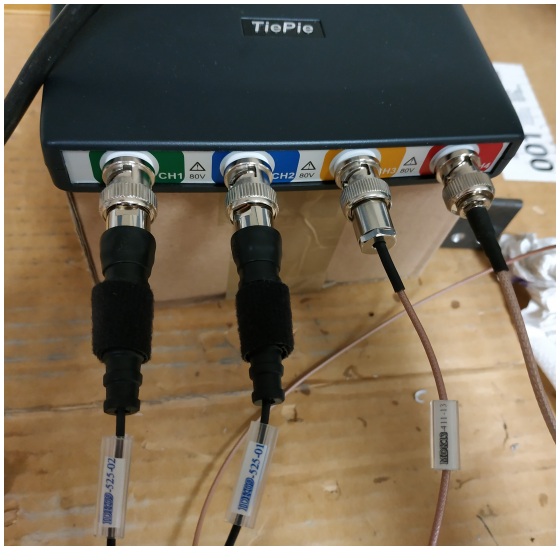


Obrázek 5.2: Ocelová deska osazená snímači

Ultrazvukový signál akustické emise byl naměřen USB osciloskopem *Handyscope HS6 DIFF* firmy TiePie z obrázku 5.4, na který byly napojeny tři různé typy snímačů: nerezový s korundovou keramickou dotykovou plochou *IDK09-525-02*, magnetický *MDK13-411-13*, oba od firmy Dakel, a nerezový *UTS-400-BNC* firmy BOTEG. Celkově však byly na destičku různou akustickou vazbou připevněny snímače

čtyři. První dva kanály osciloskopu (označené ch-1 a ch-2) byly obsazeny dvěma stejnými keramickými snímači s vazbou lepením kyanoakrylátem, následoval magnetický snímač, jehož vazbou byla silikonová vazelína a poslední nerezový jsme opět lepili. Snímače ukazuje obrázek 5.2. Každý snímač "slyší" zvuk s jiným frekvenčním zbarvením, tudíž nám snímače poskytly další čtyři sady různých dat. Důvodem pro vrtání na více místech a pro použití různých snímačů je pokrytí výše zmíněné *augmentace*.

Obrázek 5.3: Vrták a tupící brousek



Obrázek 5.4: Osciloskop s obsazenými kanály různými snímači

Vstupní rozsah osciloskopu byl nastaven na 400 mV. Rezonanční frekvence použitých snímačů se pohybují v pásmu 100-300 kHz. Vzorkovací frekvence definující počet vzorků za jednotku času byla 3.125 MHz. Jak je patrné z rezonančního pásma snímačů, vzorkovací frekvence je větší než dvojnásobek maximální frekvence měřeného signálu, tudíž je splněn Nyquistův teorém a lze předpokládat zamezení aliasing efektu [1]. Odfiltrování nízkých frekvencí (šumu) jsme provedli pomocí frekvenčního filtru s názvem *horní propust* (high-pass filter, ve zkratce HPF) se strmostí 40 dB a dělící frekvencí 120 kHz. Dále jsme za účelem zlepšení přesnosti měření nastavili rozlišení osciloskopu na 16 bitů. Osciloskop přenesl data do počítače přes USB rozhraní, přičemž data byla uložena v binárním formátu na Disk PC. Celkově máme k dispozici dataset o šesti měřeních, prováděných na pěti místech a zachycovaných čtyřmi snímači. Sestavená neuronová síť má za úkol rozeznat míru otupení vrtáku.

[1] Tento efekt způsobuje ztrátu informace z důvodu špatně zvoleného vzorkování.

5.2 Aplikace implementovaných metod na získaná data

V experimentální části bylo provedeno 6 pozorování vrtání do kovové desky, která se lišila opotřebením vrtáku (nový vrták, po 1. otupení,..., po 5. otupení). Získáváme 6 jednorozměrných časových řad

$$\vec{X^i} = [X_1, X_2, ..., X_{T_i}] \in \mathbb{R} \tag{5.1}$$

pro $i \in \hat{6}$, $\hat{6} = \{0, 1, 2, 3, 4, 5\}$, označující míru otupení vrtáku a délku jednotlivých řad T_i. Zahrnutí augmentace v experimentální části nám rozšíří dimenzi časových řad pozorování o data naměřená čtyřmi snímači v pěti odlišných vzdálenostech od nich. Každé pozorování $i \in \hat{6}$ má tak 20-ti rozměrnou časovou řadu

$$\mathbf{X}^i = [\mathbf{X}_1, \mathbf{X}_2, ..., \mathbf{X}_{T_i}] \in \mathbb{R}^{20 \times T_i}. \tag{5.2}$$

Výsledná datová sada připravena pro programovací část je tvaru

$$\mathbf{D} = [(\mathbf{X}^1, \mathbf{Y}^1), (\mathbf{X}^2, \mathbf{Y}^2), ..., (\mathbf{X}^6, \mathbf{Y}^6)] \tag{5.3}$$

skládající se z dvojic $(\mathbf{X}^i, \mathbf{Y}^i)$, kde $\mathbf{X}^i \in \mathbb{R}^{20 \times T_i}$ je 20-ti rozměrná časová řada, které odpovídá označení (angl. *label*) $\mathbf{Y}^i \in \mathbb{R}^{20}$, kde $i \in \hat{6}$. Načtení vstupních časových řad, jejich zpracování a architektura neuronové sítě jsou předmětem následujících kapitol.

5.2.1 Načtení dat

K načtení vstupních dat uložených v binárním formátu v Pythonu zavoláme knihovnu `numpy`. Tato knihovna umožňuje snadné načtení binárního formátu následujícím příkazem

```
import numpy as np
f = open(file_name, 'rb')
data = np.fromfile (f, dtype = 'float32'),
```

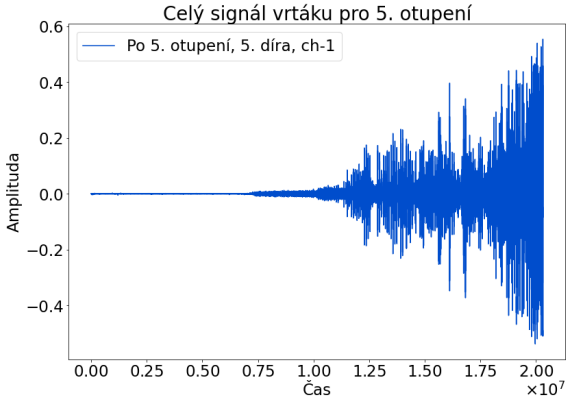
kde `file_name` je cesta k souboru a `rb` značí, že načtený soubor bude mít binární formát. Typ načtených dat je nastaven na `float32` za účelem jejich následného rychlejšího zpracování. Načtením dat obdržíme vektory reálných čísel mezi hodnotami -0.5 a 0.5 reprezentující časové řady. Nejprve zpracujeme pouze dvě datové sady pozorování z původních šesti s větším rozdílem v otupení vrtáku. Důvodem této volby je potenciálně větší šance na úspěch a rychlejší práce s menším množstvím dat.

Abychom získali potřebné informace o podobě dat, je důležitým krokem jejich vykreslení. Zjistíme tak, že signály pro 5. otupení nejsou v důsledku zahlcení počítače v době měření celé a mají pouze část použitelných hodnot, viz obr. 5.5. Naší volbou

pozorování dat k prvnímu zpracování bude tedy nulté a čtvrté otupení vrtáku, signály, jejichž hodnoty by se od sebe měly podle našeho očekávání značně lišit. Datová sada je po předchozích úvahách tvaru

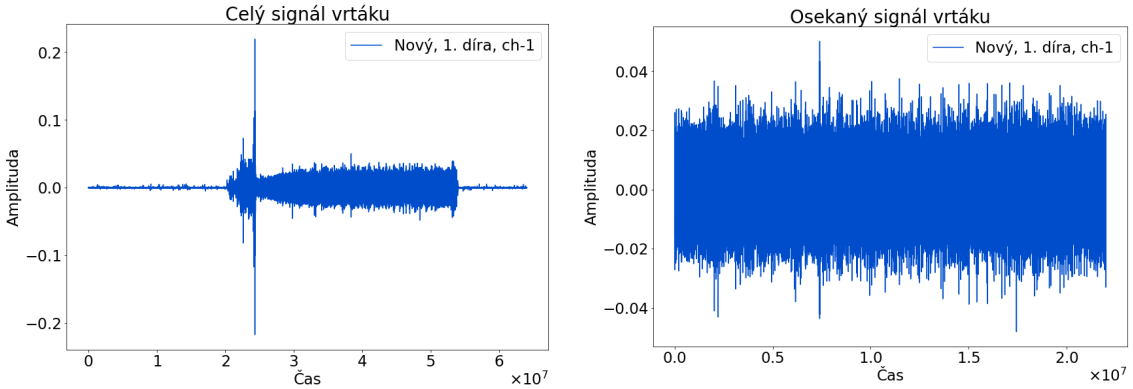
$$\mathbf{D} = [(\mathbf{X}^1, \mathbf{Y}^1), (\mathbf{X}^4, \mathbf{Y}^4)], \tag{5.4}$$

kde $\mathbf{X^i} \in \mathbb{R}^{20\times 22mil.}$ a $Y^i \in \mathbb{R}^{20}$ pro $i \in \{1, 4\}$. Průměrná délka časových řad je 50×10^6, ale vykreslením amplitudy v čase odhalíme, že proces zachycování ultrazvukových signálů probíhal i nějaký čas před a po samotném vrtání, tj. musíme odstranit začátky a konce signálů neposkytující žádnou informaci o vrtání. Neupravené signály by mohly zahrnovat zavádějící hodnoty. Příklady takové úpravy dat u obou zvolených datových sad ukazují obrázky 5.6 a 5.7. Oba grafy poskytlo měření na první díře zachycené prvním snímačem. Současným sjednocením délky řad podle nejkratšího pozorování získáme společnou délku pro všechny signály T = 22mil.

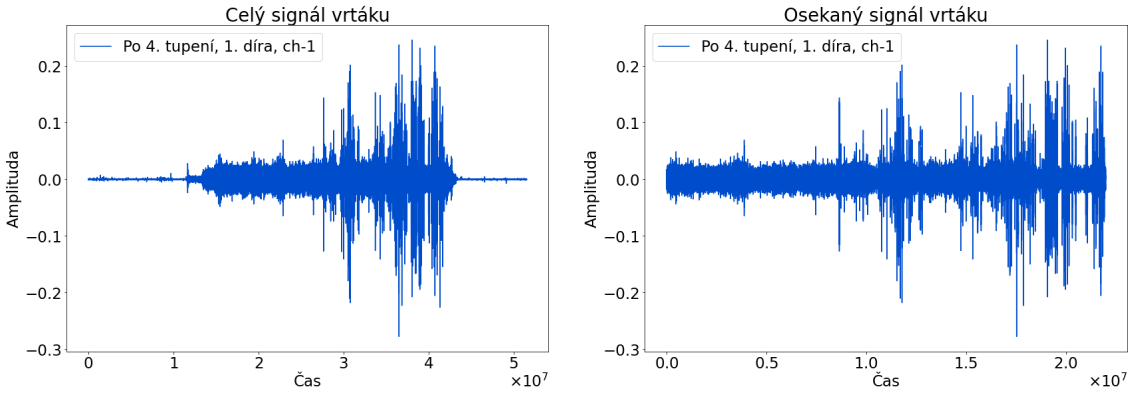


Obrázek 5.5: Signál, u kterého se během měření zaznamenala pouze jeho část

Pro zajímavost si vykreslíme, jaký je rozdíl mezi signály zaznamenanými různými snímači. Konkrétně data nového vrtáku a pro změnu na 2. díře, viz obr. 5.8. Vidíme, že data prvních tří snímačů se velmi podobají, čtvrtý má podobný charakter, ale už se poměrně odlišuje. Důvodem může být jiný typ vrtáku nebo větší vzdálenost od místa, které je předmětem pozorování. Jako další a poslední porovnání vykreslíme



Obrázek 5.6: Signál nového vrtáku zachycen 1. snímačem na 1. díře před a po osekání

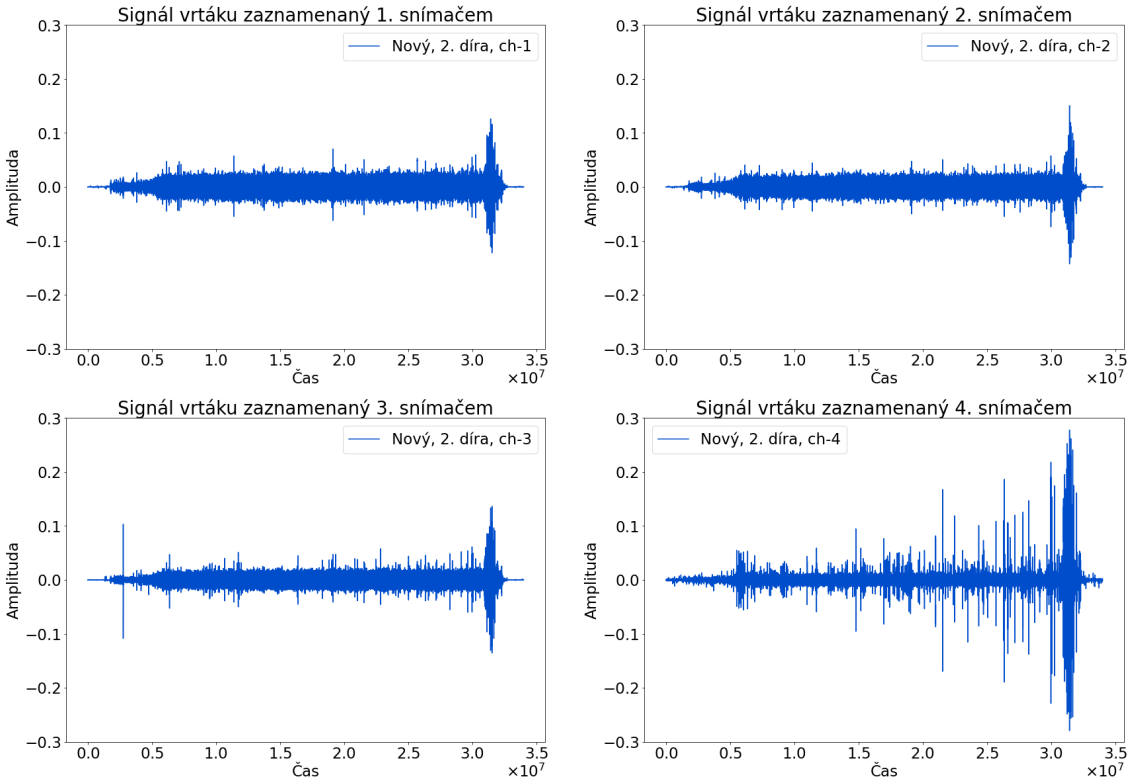


Obrázek 5.7: Signál vrtáku po 4. otupení zachycen 1. snímačem na 1. díře před a po osekání

data všech míst vrtání pro stejný stupeň otupení 4 a shodný 2. snímač. Rozdíly signálů lišících se v místě vrtání jsou na obrázku 5.9. Při pozorování výsledných grafů zjišťujeme, že amplitudy se liší především u 2. a 4. díry vrtání. Jejich hodnoty jsou docela malé oproti měření na ostatních místech. Takové odchylky mohly nastat opět z důvodu různé vzdálenosti od snímače či opětovným tupením vrtáku v důsledku vrtání dalších děr. Úkolem neuronové sítě je vyhledat v signálech charakteristický vzor a zároveň by měla být schopna nalézt spojitost mezi těmito daty navzdory různým měřením a s jejich odlišným zaznamenáváním.

Abychom zajistili dostatečné množství dat a lépe pokryly charakteristiky signálu, vytvoříme z původních vektorů časových řad délky 22mil. vzájemně se překrývající vektory. Konkrétní příklad generování vektorů demonstruje obrázek 5.10, na kterém je modrou barvou znázorněna část signálu nového vrtáku (délky 15 000) a ostatními barvami jeho rozdělení. Metoda překrývání bere nejprve z modrého signálu úsek zafixované délky 6 000 (signál zelené barvy), poté je vytvořen signál růžové barvy, který je složen z konce zeleného signálu délky 2 500 a nově si bere další část modrého signálu délky 3 500. Získáváme tak signály délky 6 000 s překrytím 2 500. Další signály vytváříme obdobným způsobem. Velikost generovaných signálů a jejich překrytí jsou v celé práci počítány tak, aby vyšel tento způsob dělení pozorování s délkou 22mil. až do konce a nemuseli jsme řešit případné doplňování posledního kratšího signálu. Počet vygenerovaných signálů z jedné časové řady je 6 285. Finální matice $\mathbf{X}^i$ má po zahrnutí měření na pěti místech a zachycované čtyřmi snímači rozměr 125 680×6 000 pro $i \in \{1, 4\}$.

Zaveďme pro další účely označení pro délku generovaných vektorů jako *length* a velikost překrytí M. Generování vektorů překrýváním se dá naprogramovat následujícím způsobem



Obrázek 5.8: Rozdíl ve zaznamenávání signálu nového vrtáku různými snímači na 1. díře

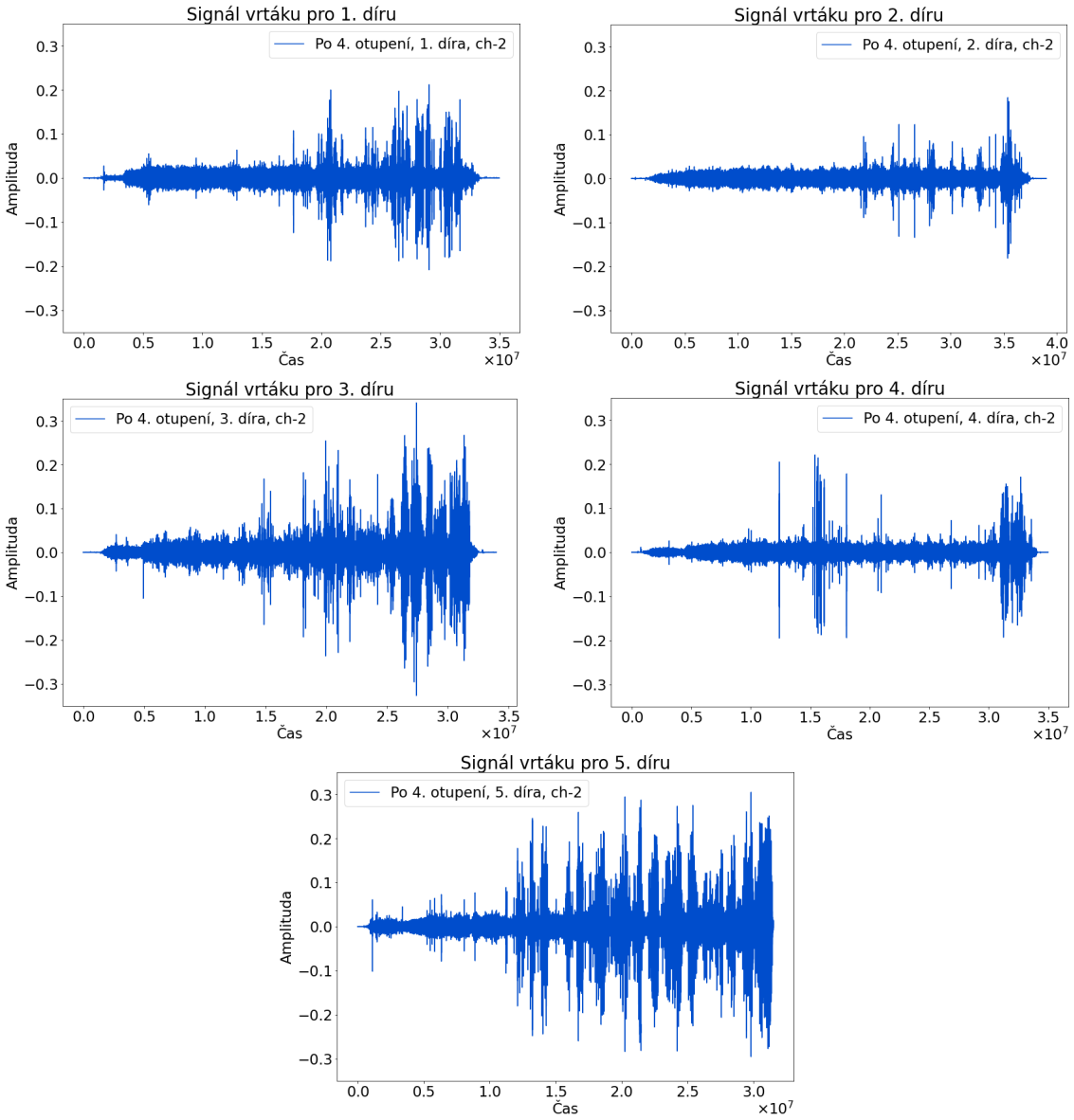
```
b = np.array(data[i1:i2])
l = length - M
x = np.zeros(((int((i2 - i1 - length)/l)+1), length))

x[0,:] = b[0:length]
for z in range(l,int(i2 - i1 - length)+1,l):
  x[int(z/l),:] = b[z:(z+length)],
```

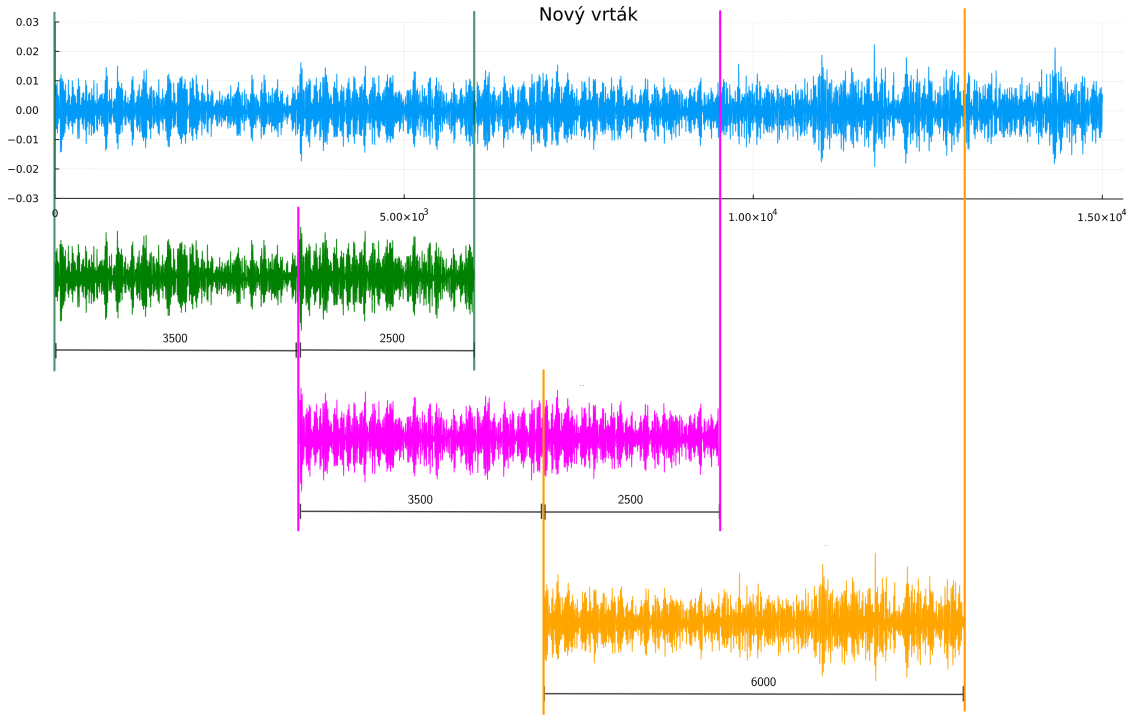
kde nejprve definujeme proměnné b a l a vytvoříme matici x, kterou naplníme nulami. Proměnná b je `numpy` pole hodnot načtených dat v intervalu $[i_1, i_2]$, můžeme tak načíst libovolný úsek časových řad. Označení délky dat braných z původního signálu po vygenerování prvního vektoru je l. Poté do 0-tého[2] řádku matice x dáme prvních *length* hodnot z původního vektoru délky 22mil. Další řádky plníme obdobně za pomoci for cyklu s krokem l. Výsledkem je matice $\mathbf{X}$ s rozměry $(i_2 - i_1 - length)/l) \times length$, kde $i_2 - i_1 = T = 22mil$. Úpravy během generování vektorů převedly typ hodnot výsledné matice na `float64`. Pomocí příkazu `astype` je přetypujeme zpět na `float32`, aby se nám s daty lépe pracovalo.

K vytvořené matici $\mathbf{X}$ sestrojíme vektor skalárů $\mathbf{Y}$, které označují řádky matice $\mathbf{X}$ tvořící vygenerované signály. Jelikož matice $\mathbf{X}$ je vytvořena z úseků stejného signálu, naplníme vektor totožnými hodnotami označení a jeho délka bude počtem

[2]Python indexuje od 0.



Obrázek 5.9: Rozdíly signálů po 4. otupení lišících se místem vrtání zaznamenávané 2. snímačem



Obrázek 5.10: Ukázka generování vektorů metodou překrývání

vygenerovaných vektorů $(T - length)/l$. Finální vstupní matice $\mathbf{X} = [\mathbf{X}^1, \mathbf{X}^2] \in 2(T-length)/l) \times length$ vzniká složením submatic pro jednotlivá pozorování. Vektor požadovaných hodnot výstupů je složením vektorů označení jednotlivých měření. Příkaz pro skládání `numpy` polí je `concatenate`. Máme tak připravenou datovou sadu pro dvě pozorování, na které budeme ladit architektury neuronových sítí. Kompletní datovou sadu pustíme až na vyladěnou architekturu.

5.2.2 Příprava dat

Předběžná příprava dat byla již provedena při jejich načítání odstraněním nežádoucích hodnot signálu a rozšířením dat (augmentace) metodou překrývání. Zároveň se na data umíme dívat z různých úhlů díky měření různými snímači a v odlišné vzdálenosti od nich. Další část příprav proběhla při samotném provádění experimentu odfiltrováním šumu, nastavením vhodných frekvencí, seřízením rozlišení osciloskopu aj. Hodnoty signálů byly centrovány kolem nuly a žádná z hodnot nepřekročila rozsah [-0.5,0.5]. Data tedy máme již od začátku velmi dobře připravená. Normalizovat data není potřeba, hodnoty jsou v potřebném rozsahu. Data jsou zároveň heterogenní, tudíž ani standardizaci dat nebude třeba dělat.

Rovnou tedy přejdeme k bodu rozdělení dat na trénovací, testovací a validační množiny. Trénovací a testovací data rozdělíme funkcí `train_test_split`. Výchozí nastavení funkce zahrnuje rozdělení na 75% trénovacích dat a 25% testovacích, tyto hodnoty nám vyhovují a nastavení ponecháme. Validační množinu lze oddělit přímo při trénování modelu jako argument funkce `model.fit`. V práci je tato hodnota nastavena na 0.2, tedy z trénovací množiny je odděleno 20% dat pro validaci.

Poslední přípravou před sestavením modelu je zajištění správného rozměru vstupních dat. Vstupem do konvolučních vrstev jsou 3D tenzory, je tedy potřeba přidat jednu dimenzi s hodnotou 1 funkcí `reshape`. Nyní jsou data připravena pro vstup do neuronových sítí.

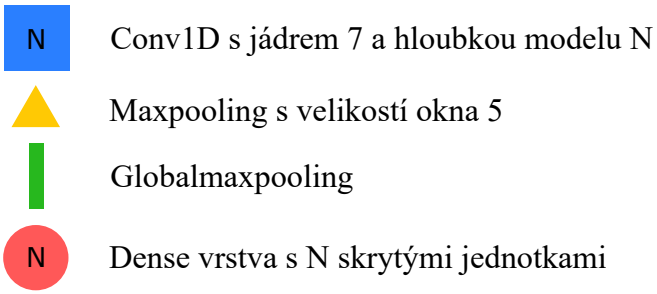
5.3 Analýza výsledků

Ačkoliv je velkým lákadlem pustit se do náročných architektur sítí a nastavovat vysoké hodnoty u parametrů, často zde platí, že v jednoduchosti je síla. Složité architektury mohou mít velmi vysokou přesnost hned po první epoše, záhy však dochází k velmi rychlému přeučení. Pustíme se tedy do jednodušších architektur, které jak koneckonců uvidíte, nevykazují vůbec špatné výsledky.

Budeme pouštět 4 odlišně vygenerované sady dat na 2 modely CNN a porovnávat jejich výsledky. Parametry pro sady dat jsou v tabulce 5.1. Legenda pro vizualizaci modelů je obrázkem 5.11.

Označení	Délka vektorů	Překrývání vektorů
L10_M2500	10 000	2 500
L10_M5000	10 000	5 000
L15_M6206	15 000	6 206
L20_M2500	20 000	2 500

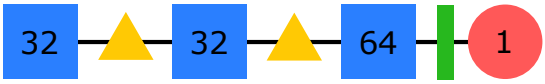
Tabulka 5.1: Tabulka čtyř sad vygenerovaných dat



Obrázek 5.11: Legenda pro značení architektur CNN

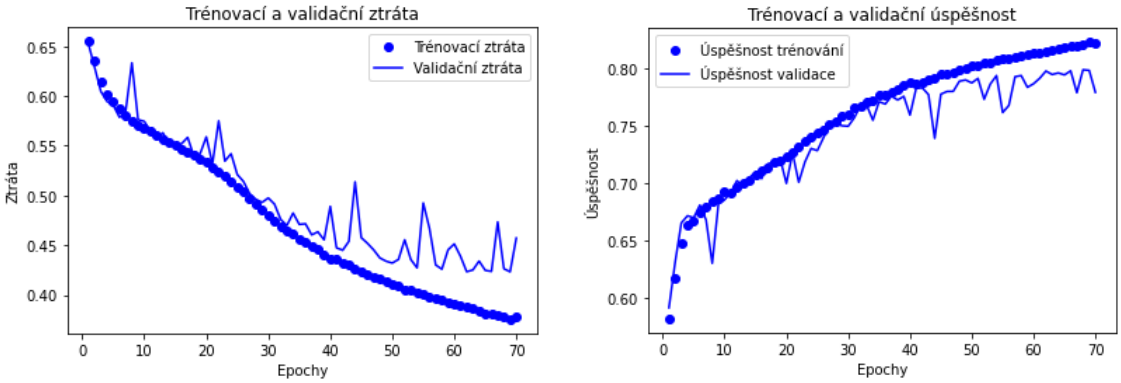
Prvním modelem je CNN na obrázku 5.12. Hustě propojená síť na konci modelu má pouze jednu skrytou jednotku. Jedná se o poslední výstupní vrstvu, kterou je binární hodnota. Máme jen dvě třídy klasifikace (nulté a 4. otupení), jde tedy o případ binární klasifikace se ztrátovou funkcí binární křížová entropie. Aktivační funkce výstupu je sigmoida.

Na první sadě dat s délkou vygenerovaných vektorů 10 000 a překrytím 2 500 klesla trénovací ztráta pod hodnotu 0.4 a úspěšnost trénování překročila 80%, viz obr. 5.13. Validační hodnoty se po 40. epoše začínají od těch trénovacích oddalovat. To značí



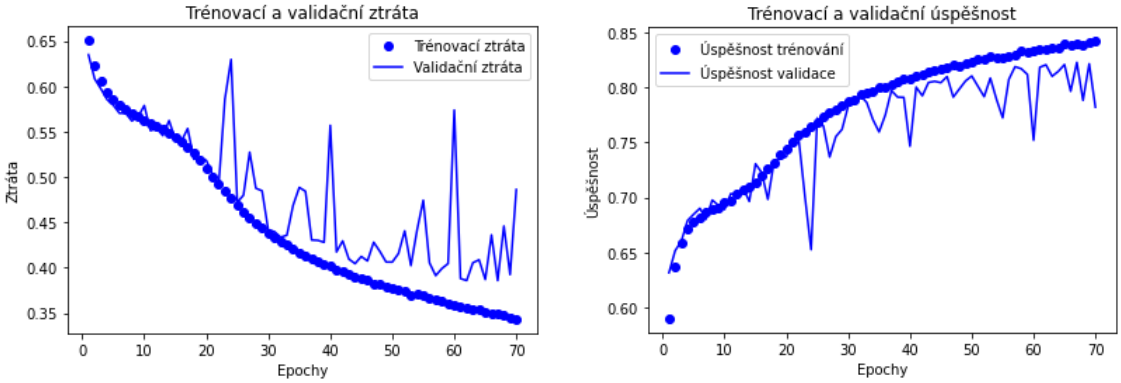
Obrázek 5.12: Model 1

přeučení sítě, protože trénovací úspěšnost převažuje nad tou validační. Využijeme však naučeného modelu do 40. epochy, který ještě není přeučen a ve kterém jsme si udělali "checkpoint". Spustíme síť na jednu epochu tentokrát s testovacími daty, které síť nezná. Dostáváme ztrátu 0.4705 a úspěšnost 77.12%.



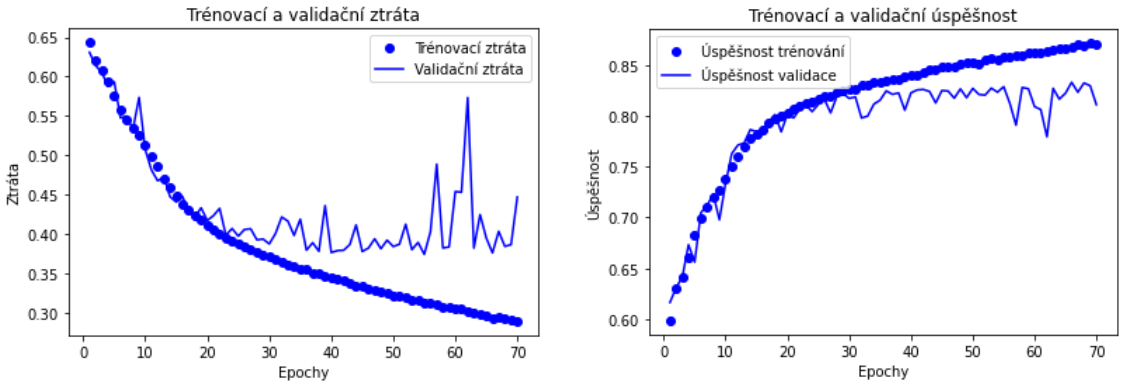
Obrázek 5.13: Trénovací a validační ztráta s úspěšností L10_M2500

Trénovací a validační ztráta s úspěšností pro vektory délky 10 000 s překrýváním 5 000 je vykreslena v grafech na obrázku 5.14. Oproti předchozí sadě vidíme značně větší oscilace dat pro validaci. Model nefunguje úplně podle představ, ale stále se drží do 45. epochy u dat trénovacích. Po spuštění modelu zkráceného o 25 epoch získáme ztrátu 0.4196 a úspěšnost 80.32%.



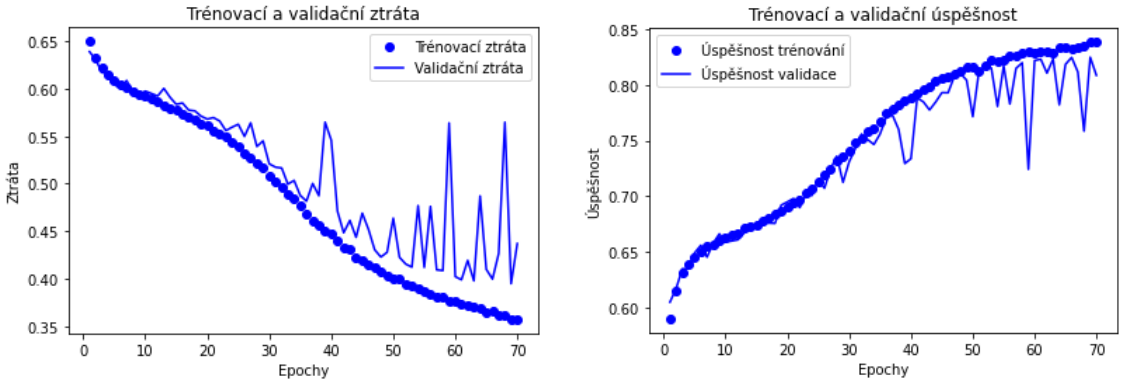
Obrázek 5.14: Trénovací a validační ztráta s úspěšností L10_M5000

Obrázek 5.15 vypovídající o modelu sady vektorů délky 15 000 s překrývající částí 6206 vykazuje jasné výsledky. Přímo vidíme, že 30 epocha je hraniční a testovací sada na těchto datech dosáhla přesnosti 81.2% a ztráta klesla na 0.4053.



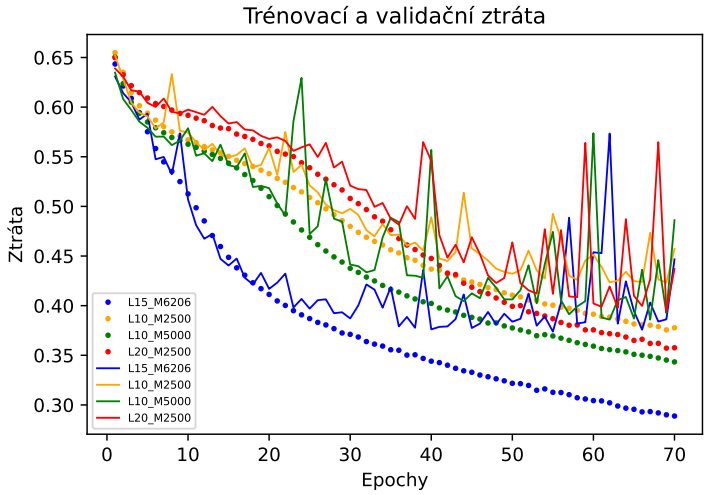
Obrázek 5.15: Trénovací a validační ztráta s úspěšností L15_M6206

Poslední sadu mapují grafy na obrázku 5.16. Vstupem byly vektory délky 20 000 s překrytím 2 500. Nastávají větší oscilace u validace, ale ne do takové míry jako v modelu L10_M5000. Poslední epochou před přeučením sítě by mohla být 45., na tomto modelu ukázala testovací data ztrátu 0.4519 a úspěšnost 79.36%.

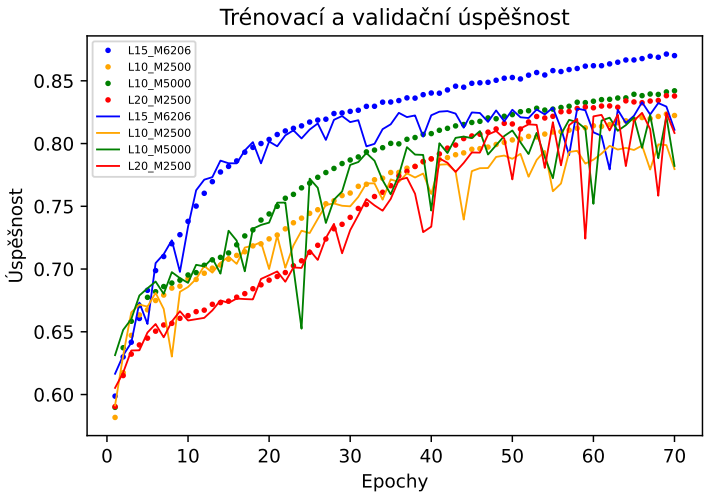


Obrázek 5.16: Trénovací a validační ztráta s úspěšností L20_M2500

V tuto chvíli zbývá porovnat grafy prvního modelu na sadách dat vygenerovaných odlišným způsobem. Pro přehlednost jsou grafy 5.17 a 5.18 odděleny. Vidíme, že hodnoty validací u ztráty vykazují pro všechny sady dat od 50. epochy velmi podobné hodnoty. Při testování sítě měla nejmenší ztrátu sada L15_M6206 a největší naopak L10_M2500. Vykreslený obrázek potvrzuje otestování sítě. Úspěšnost všech hodnot na grafu 5.18 se časem opět sloučí, nás ale zajímají hodnoty před oddělením trénovacích a validačních dat. Z grafu je patrné, že největší úspěšnost má modrá křivka sady L15_M6206. Nejmenší pak žlutá sada L10_M2500. Tyto hodnoty sedí s testováním dat i s faktem, že grafy ztrát a úspěšností se jistým způsobem prolínají.

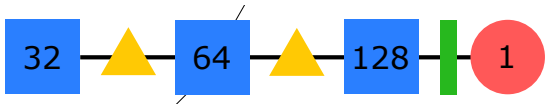


Obrázek 5.17: Porovnání ztráty různých podob dat pro stejný model 1



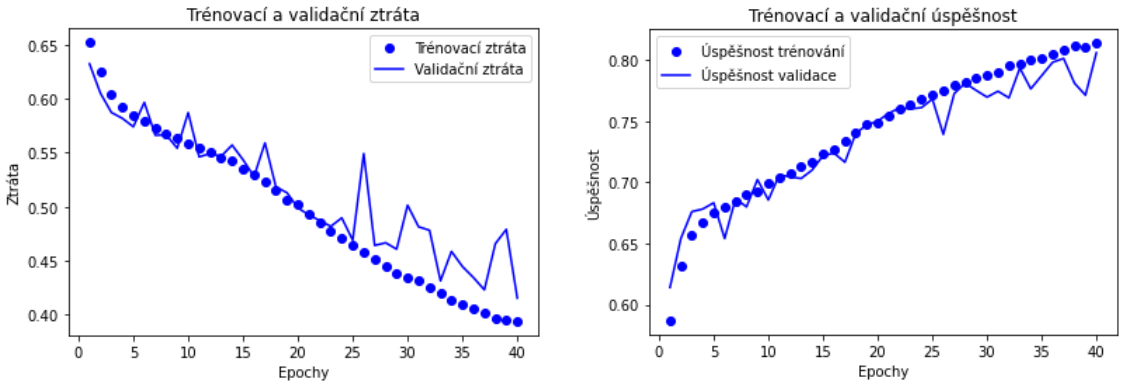
Obrázek 5.18: Porovnání úspěšnosti různých podob dat pro stejný model 1

Obrázek 5.19 ukazuje architekturu druhého modelu. Ten je podobný tomu prvnímu, ale je pozměněn počet skrytých jednotek. Opět pro tento model vykreslíme grafy, které poté okomentujeme. Tentokrát je navíc snížený počet epoch na 40 z důvodu vyšší počítačové náročnosti modelu.



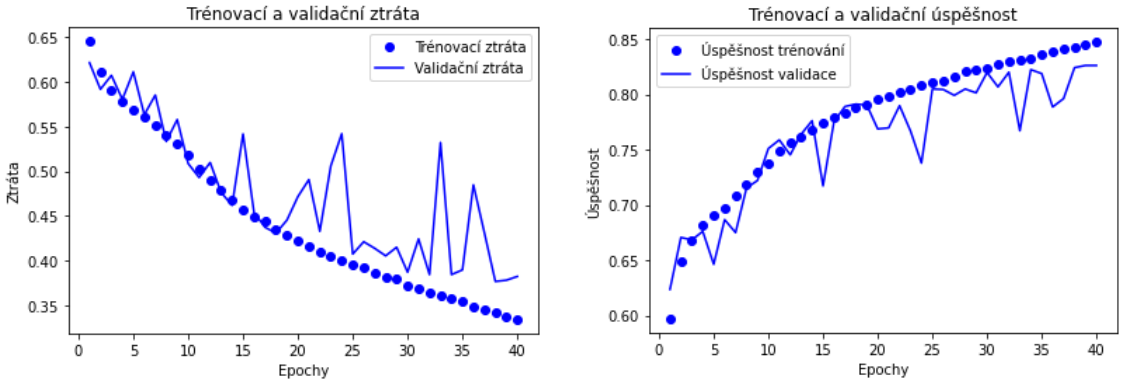
Obrázek 5.19: Model 2

První sada L10_M2500 vykazuje na obrázku 5.20 velmi dobré výsledky. Trénovací a validační data na grafech se příliš nevzdalují. Výsledkem testování modelu do 35. epochy jsou hodnoty 0.4563 pro ztrátu a 78.24% pro úspěšnost.



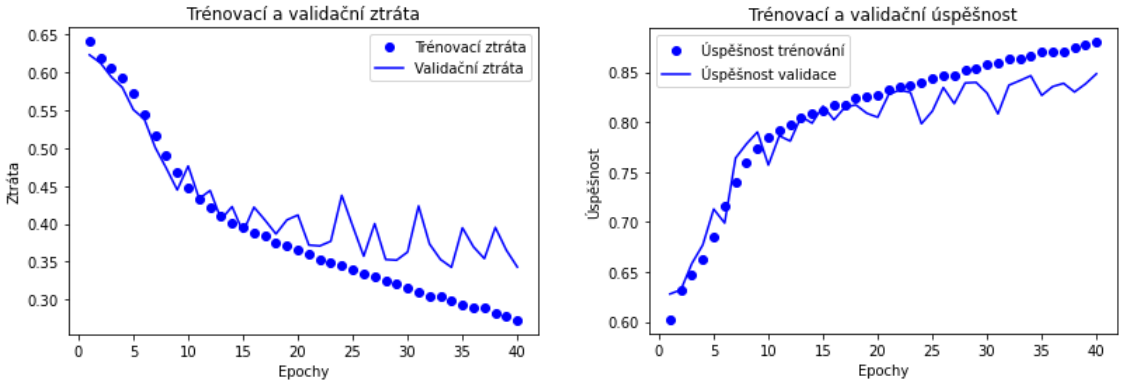
Obrázek 5.20: Trénovací a validační ztráta s úspěšností L10_M2500

U sady L10_M5000 validace na grafech 5.21 velmi osciluje, úspěšnost ale dosahuje vysokých hodnot. Neznámá testovací data mají na modelu do 30. epochy úspěšnost 80.6% a ztrátu 0.4133.



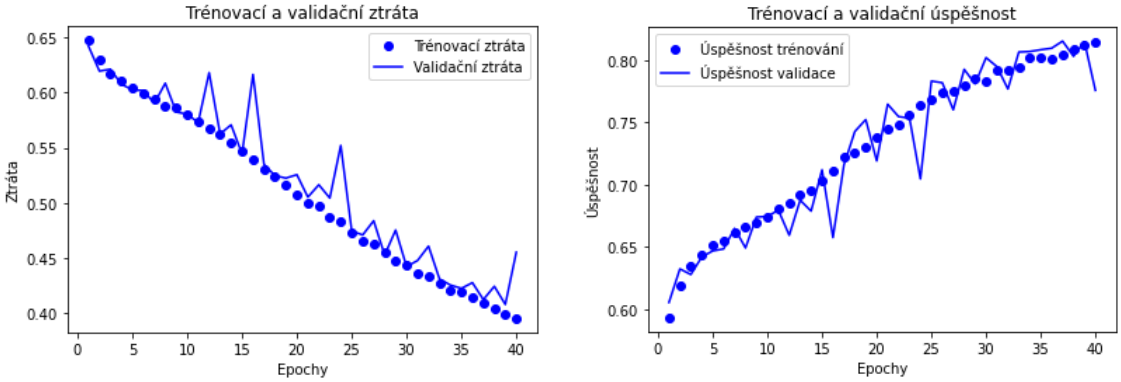
Obrázek 5.21: Trénovací a validační ztráta s úspěšností L10_M5000

Pro předposlední datovou sadu L15_M6206 jsou vykresleny hodnoty ztráty a úspěšnosti na obrázku 5.22. Je vidět, že data se oddělují velmi brzy, ale když přihlédneme k úspěšnosti, nejsou to vůbec špatné hodnoty. Testovací sada dává pro model do 30. epochy výsledky 0.4053 pro ztrátu a 81.2% pro úspěšnost.



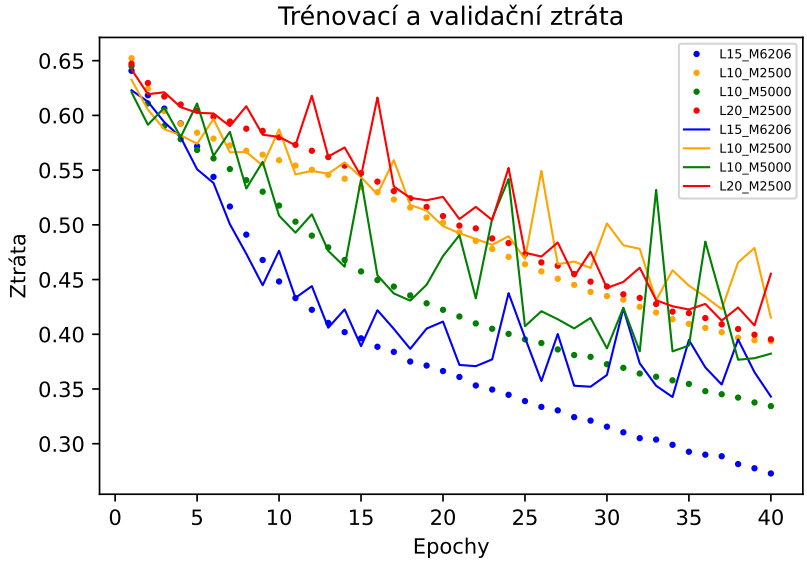
Obrázek 5.22: Trénovací a validační ztráta s úspěšností L15_M6206

Na obrázku 5.23 jsou hodnoty trénování a validace sady L20_M2500 stále v kontaktu, vyzkoušíme testovací data na modelu do 35. epoch. Ztráta činí 0.4541 a úspěšnost 78.83%.

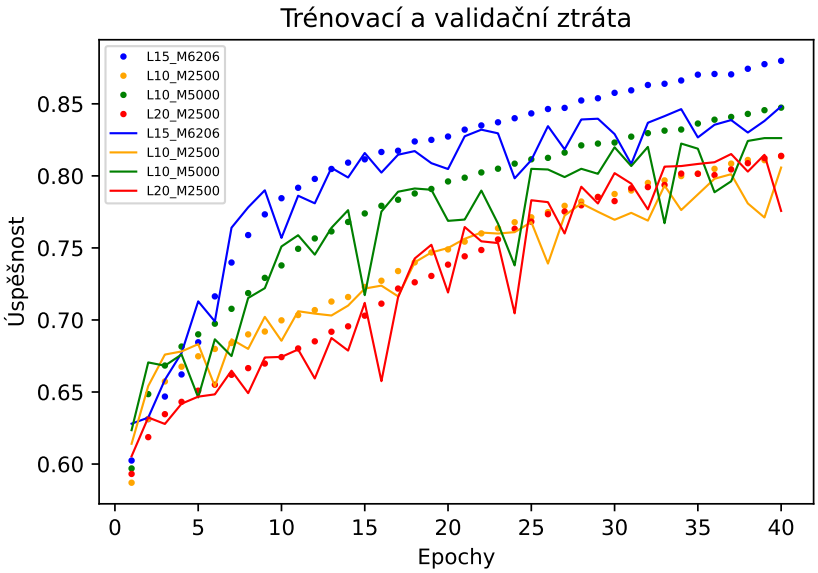


Obrázek 5.23: Trénovací a validační ztráta s úspěšností L20_M2500

Hodnoty ztrát a úspěšností pro všechny sady dat trénované na 2. modelu jsou vyobrazeny na grafech 5.24 a 5.25. Ztráta i úspěšnost sady L15_M6206 vykazuje opět nejlepší výsledky a sada L10_M2500 opět nejhorší. Tyto domněnky z grafu potvrdilo i testování na neznámé sadě testovacích dat. Oba modely tedy vykazují velmi podobné hodnoty. Výsledky jednotlivých sad dat nejsou závislé na volbě modelu.



Obrázek 5.24: Porovnání ztráty různých podob dat pro stejný model 2



Obrázek 5.25: Porovnání úspěšnosti různých podob dat pro stejný model 2

Nakonec byla zpracována pouze data pro nulté a čtvrté otupení vrtáku. Všechna data dohromady byla příliš velká a program se nepovedl spustit.

Závěr

Cílem této práce bylo seznámit se s architekturami konvolučních (CNN) a rekurentních (RNN) neuronových sítí, implementace jejich modelů pomocí dostupných knihoven a jejich aplikaci na naměřené ultrazvukové signály procesu vyvrtání díry vrtačkou do ocelové desky. Byla sledována míra otupení vrtáku. Zpracovány byly nakonec pouze sítě konvoluční. Důvodem byla nevhodná datová sada pro práci s typem sítí jako jsou RNN. Dlouhé úseky signálů v datové sadě jsou pro RNN velmi náročné. Pojednání o této problematice zpracovává článek [7] Zkoušela jsem programovat jejich architektury na CPU, avšak počítač takovou náročnost neutáhl.

Naopak použití 1D konvolučních sítí pro časové řady přineslo slibné výsledky. Nejvyšší testovací přesnost modelu byla dosažena na sadě vektorů délky 15 000 s překrytím 6 206 a činila 81.2%. Oba modely vyzkoušené v práci přinesly úplně stejný výsledek a i u ostatních sad dat vykazovaly podobné výsledky. První model obsahoval 3 konvoluční vrstvy s velikostí konvolučního jádra 7 a hloubkou map příznaků po řadě 32, 32, 64. Mezi konvolučními vrstvami byly vrstvy poolingové s velikostí okna 5. Přípravu do ANN modelu zajišťovala operace globálního sdružování dle maxima. Ve druhém modelu byla pozměněna hloubka map příznaků na 32, 64, 128.

Dosažené výsledky by mohly nalézt uplatnění v sériové výrobě při sledování efektivity strojů. Neuronové sítě jsou však natolik mocným nástrojem, že model by se nemusel omezovat na dosavadní typ vstupních dat, ale mohl by být vyzkoušen na datech zcela odlišných. Sledováním zdrojů akustické emise lze předejít např. opotřebení materiálu, které by mohlo vést k nežádoucím jevům. Dále například k detekci unikání plynu malým otvorem. Určitě je v této oblasti široká škála uplatnění neuronových sítí.

Literatura

[1] F. Chollet. *Deep Learning with Python*. Manning Publications Company, Shelter Island, 2017.

[2] A. Extance. How DNA could store all the world's data. *Nature*, 537:22–24, 2016.

[3] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. volume 15, pages 315–323, 2011.

[4] B. Kopec. *Nedestruktivní zkoušení materiálů a konstrukcí*. ČNDT, Brno, 2008.

[5] A. Krizhevsky, I. Sutskever, and G. E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[6] Y. A. LeCun, L. Bottou, Genevieve B. Orr, and Klaus-Robert Müller. *Efficient BackProp*, pages 9–48. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[7] T. K. Rusch and S. Mishra. Unicornn: A recurrent model for learning very long time dependencies. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 9168–9178. PMLR, 18–24 Jul 2021.

[8] D. Scherer, A. Müller, and S. Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. In Konstantinos Diamantaras, Wlodek Duch, and L. S. Iliadis, editors, *Artificial Neural Networks – ICANN 2010*, pages 92–101, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[9] A. M. Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950.

[10] J. Wu. Introduction to convolutional neural networks. *National Key Lab for Novel Software Technology. Nanjing University. China*, 5(23):495, 2017.

[11] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. 2013.