

Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

FIIT-136227-71997

Bc. Dávid Beňo

AUTOMATIZOVANÉ VYHODNOCOVANIE
NEUROFYZIOLOGICKÝCH SIGNÁLOV

Diplomová práca

Študijný program: Internetové technológie

Študijný odbor: Počítačové inžinierstvo

Miesto vypracovania: Ústav počítačového inžinierstva a aplikovej informatiky, FIIT STU ,
Bratislava; Ústav elektroniky a fotoniky, FEI STU, Bratislava

Vedúci práce: Ing. Lukáš Kohútka, PhD.

apríl 2019

Zadanie diplomovej práce

Meno študenta: Bc. Dávid Beňo

Študijný program: Internetové technológie

Študijný odbor: Počítačové inžinierstvo

Názov práce: Automatizované vyhodnocovanie neurofyziologických signálov

Samostatnou výskumnou a vývojovou činnosťou v rámci predmetov Diplomový projekt I, II, III vypracujte diplomovú prácu na tému, vyjadrenú vyššie uvedeným názvom tak, aby ste dosiahli tieto ciele:

Všeobecný cieľ:

Vypracovaním diplomovej práce preukážte, ako ste si osvojili metódy a postupy riešenia relatívne rozsiahlych projektov, schopnosť samostatne a tvorivo riešiť zložité úlohy aj výskumného charakteru v súlade so súčasnými metódami a postupmi študovaného odboru využívanými v príslušnej oblasti a schopnosť samostatne, tvorivo a kriticky pristupovať k analýze možných riešení a k tvorbe modelov.

Špecifický cieľ:

Vytvorte riešenie, zodpovedajúce návrhu textu zadania, ktorý je prílohou tohto zadania.

Návrh bližšie opisuje tému vyjadrenú názvom. Tento opis je záväzný, má však rámcový charakter, aby vznikol dostatočný priestor pre Vašu tvorivosť.

Riadte sa pokynmi Vášho vedúceho.

Pokiaľ v priebehu riešenia, opierajúc sa o hlbšie poznanie súčasného stavu v príslušnej oblasti alebo o priebežné výsledky Vášho riešenia alebo o iné závažné skutočnosti, dospejete spoločne s Vašim vedúcim k presvedčeniu, že niečo v texte zadania a/alebo v názve by sa malo zmeniť, navrhnite zmenu. Zmena je spravidla možná len pri dosiahnutí kontrolného bodu.

Miesto vypracovania: Ústav počítačového inžinierstva a aplikovanej informatiky, FIIT STU, Bratislava

Vedúci práce: Ing. Lukáš Kohútka

Termíny odovzdania:

podľa harmonogramu štúdia platného pre semester, v ktorom máte príslušný predmet (Diplomový projekt I, II, III) absolvovať podľa Vášho študijného plánu

Predmety odovzdania:

V každom predmete dokument podľa pokynov na www.fiit.stuba.sk v časti:
home > Informácie o > štúdiu > organizácia štúdia > diplomový projekt

SLOVENSKÁ TECHNICKÁ UNIVERZITA
V BRATISLAVE

Fakulta informatiky a informačných technológií
Ilkovičova 2, 842 16 Bratislava 4

1



Ing. Katarína Jelemenská, PhD.
riadička ústavu

V Bratislave dňa 12. februára 2018

Návrh zadania diplomovej práce

Finálna verzia do diplomovej práce¹

Študent:

Meno, priezvisko, tituly: Dávid Beňo, Bc.
Študijný program: Internetové technológie
Kontakt: dawid.beno@gmail.com

Výskumník:

Meno, priezvisko, tituly: Lukáš Kohútka, Ing.

Projekt:

Názov: Automatizované vyhodnocovanie neurofyziológických signálov
Názov v angličtine: Automated evaluation of neurophysiological signals
Miesto vypracovania: Ústav počítačového inžinierstva a aplikovanej informatiky, FIIT STU
Oblast' problematiky: Peroperačný neurofyziológický monitoring

Text návrhu zadania²

Zameranie diplomovej práce je z oblasti medicíny, konkrétnie peroperačného neuromonitoringu. Monitorovanie funkčnosti vybranej časti nervovej sústavy je veľmi dôležité v rámci neurochirurgických ale aj mnohých iných operácií, počas ktorých hrozí nenávratné poškodenie danej časti nervovej sústavy človeka. V súčasnosti výsledky týchto neuromonitorovacích vyšetrení musí analyzovať lekár. Táto skutočnosť je motiváciou pre návrh a implementáciu softvéru, ktorý dokáže výsledky neuromonitoringu automaticky vyhodnotiť, a tak zefektívniť prácu lekára, čím napomôže eliminovať chybovosť človeka pri tak závažných úknoch ako je operácia. Analyzujte existujúce systémy, ktoré sú v súčasnosti používané na peroperačný neuromonitoring na Slovensku, a spôsob ich použitia na vyhodnotenie výsledných neurofyziológických údajov vygenerovaných neuromonitorovacím systémom pre vybraný typ ochorenia. Taktiež analyzujte existujúce spôsoby vyhodnocovania neurofyziológických údajov. Na základe analýzy navrhnite a implementujte v jazyku C vlastný prototyp softvérového nástroja, ktorý dostane na vstupe neurofyziológické údaje od neuromonitorovacieho systému, a ako výstup automaticky vygeneruje hodnotu miery korektnej funkčnosti monitorovaného tkaniva. Overte funkčnosť vášho riešenia simuláciou vyhodnotenia výsledkov vyšetrenia na reálnych údajoch poskytnutých vo forme súboru a zhodnoťte efektivitu vytvoreného softvéru.

¹ Vytláčiť obojstranne na jeden list papiera

² 150-200 slov (1200-1700 znakov), ktoré opisujú výskumný problém v kontexte súčasného stavu vrátane motivácie a smerov riešenia

Literatúra³

- [HUSAIN, 2008] HUSAIN, A. M.: A Practical Approach to Neurophysiologic Intraoperative Monitoring, 2008, ISBN 9781933864, 1933864095.
- [AANEM, 2011] AANEM: Recommended Policy for Electrodiagnostic Medicine American Association of Neuromuscular & Electrodiagnostic Medicine, 2011, https://www.aanem.org/getmedia/3275d71c-81dc-4b23-96a7-03173ecf8446/Recommended_Policy_EDX_Medicine_062810.pdf.

Vyššie je uvedený návrh diplomového projektu, ktorý vypracoval(a) Bc. Dávid Beňo, konzultoval(a) a osvojil(a) si ho Ing. Lukáš Kohútka a súhlasi, že bude takýto projekt viesť v prípade, že bude pridelený tomuto študentovi.

V Bratislave dňa 9.2.2018


Beňo

Podpis študenta


Kohútka

Podpis výskumníka

Vyjadrenie garanta predmetov Diplomový projekt I, II, III

Návrh zadania schválený: áno / nie⁴

Dňa: 16.2.2018


Kohútka

Podpis garanta predmetov

³ 2 vedecké zdroje, každý v samostatnej rubrike a s údajmi zodpovedajúcimi bibliografickým odkazom podľa normy STN ISO 690, ktoré sa viažu k téme zadania a preukazujú výskumnú povahu problému a jeho aktuálnosť (uveďte všetky potrebné údaje na identifikáciu zdroja, pričom uprednostnite vedecké príspevky v časopisoch a medzinárodných konferenciách)

⁴ Nehodiace sa prečiarknite

Čestné prehlásenie

Čestne prehlasujem, že som celú bakalársku prácu vypracoval samostatne pod odborným vedením vedúceho bakalárskej práce, s využitím získaných poznatkov a na základe uvedených zdrojov.

V Bratislave, apríl 2019

.....
Bc. Dávid Beňo

Pod'akovanie

Chcel by som sa pod'akovat' svojmu vedúcemu práce, *Ing. Lukášovi Kohútkovi*, PhD., za odborné vedenie mojej diplomovej práce a takisto za mnohé cenné rady, podporu a ochotu pomôcť.

Ďalej by som sa chcel pod'akovat' pani doktorke *MUDr. Miroslave Kohútkovej*, za odbornú pomoc pri medicínskych témach a jej radách pri smerovaní projektu.

Vďaka tiež patrí celej rodine a priateľom, ktorí ma počas celého štúdia podporovali a pomáhali mi.

Anotácia

Slovenská technická univerzita v Bratislave

FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

Študijný program: Internetové technológie

Autor: Bc. Dávid Beňo

Diplomová práca: Automatizované vyhodnocovanie neurofyziológických signálov

Vedúci diplomovej práce: Ing. Lukáš Kohútka, PhD.

apríl 2019

Cieľom tejto diplomovej práce bolo vyvinúť program, ktorý zabezpečí pomoc lekárovi pri vyhodnocovaní výsledkov neuromonitoringu. Program analyzuje dát a upozorní lekára na možné komplikácie pri operácii.

V tejto diplomovej práci je opísané existujúce riešenie, ktoré sa využíva v praxi v nemocniacích. Tiež boli opísané typy a vzory neurosignálov, s ktorými sa lekár v praxi stretáva. Na základe analýzy sme vytvorili generátor, ktorý generuje prúd dát s podobnými vzormi a tým simuluje reálne meranie pri vyšetrení pacienta. Vytvorené dátá následne analyzujeme na dvoch úrovniach. Prvá statický vyhľadáva hodnoty, ktoré prekročia stanovenú hodnotu a tým určuje miesta, kde sa pravdepodobne nachádzajú simulované vlny neurosignálov. Druhá úroveň pomocou natrénovaného modelu neurónovej siete vyhodnotí o aký typ impulzu sa jedná.

Annotation

Slovak University of Technology Bratislava

FACULTY OF INFORMATICS AND INFORMATION TECHNOLOGIES

Degree course: Internet technologies

Author: Bc. Dávid Beňo

Master's Thesis: Automated evaluation of neurophysiological signals

Supervisor: Ing. Lukáš Kohútka, PhD.

2019, april

The aim of this diploma thesis was to develop a program that will help the physicians to evaluate the results of neuromonitoring. The program analyzes the data and alerts the doctor to possible complications of surgery.

This thesis describes the existing solution, which is used in practice in hospitals. The types and patterns of neurosignals encountered in practice have also been described. Based on the analysis, we have created a generator that generates a stream of data with similar patterns, thereby simulating a real measurement when examining a patient. We then analyze the generated data on two levels. The first statically searches for values that exceed a specified threshold, and thus determines where simulated waves of neurosignals are likely to be found. The second level, using a trained model of neural network, evaluates what type of impulse it is.

1.	ÚVOD	1
1.1.	POUŽITÉ SKRATKY A POJMY	1
1.2.	ZADANIE	2
2.	ANALÝZA PROBLÉMU	3
2.1.	DIAGNÓZY	3
2.1.1.	<i>Spasticita</i>	4
2.1.2.	<i>Spina bifida</i>	4
2.2.	LIEČEBNÁ METÓDA - SELEKTÍVNA DORZÁLNA RIZOTÓMIA.....	5
2.3.	NEUROMONITORING.....	5
2.3.1.	<i>Intraoperačný neuromonitoring</i>	6
2.3.2.	<i>Elektromyografia</i>	7
2.3.3.	<i>Faktory ovplyvňujúce záznam neuromonitoringu</i>	9
2.3.4.	<i>Evokované potenciály</i>	11
2.4.	ZÍSKAVANIE MONITOROVACÍCH DÁT	12
2.4.1.	<i>Diferenčný zosilňovač</i>	12
2.4.2.	<i>Analógová filtrácia</i>	12
2.4.3.	<i>Analógovo-digitálna konverzia</i>	13
2.4.4.	<i>Digitálne vyhľadenie signálu</i>	13
2.5.	METÓDY VYHODNOCOVANIA SIGNÁLU.....	13
2.5.1.	<i>Atribúty signálu</i>	13
2.5.2.	<i>Klasifikácia tvaru impulzu</i>	14
2.5.3.	<i>Neurónové siete</i>	17
2.5.4.	<i>OpenNN</i>	21
2.5.5.	<i>Genann</i>	22
2.6.	VYHODNOTENIE ANALÝZY.....	22
3.	OPIS RIEŠENIA	23
3.1.	ŠPECIFIKÁCIA POŽIADAVIEK.....	23
3.2.	NÁVRH RIEŠENIA	23
3.2.1.	<i>Formát dát</i>	24
3.2.2.	<i>Platforma</i>	24
3.2.3.	<i>Architektúra aplikácie</i>	24
3.2.4.	<i>Generátor signálu</i>	24
3.2.5.	<i>Vyhľadávanie vzorov v dátach</i>	25
3.3.	ZMENY V NÁVRHU RIEŠENIA	26
3.3.1.	<i>Rozšírenie architektúry aplikácie</i>	26
3.3.2.	<i>Rozšírenie generátoru dát</i>	26
3.4.	IMPLEMENTÁCIA	27
3.4.1.	<i>Analyzátor signálov</i>	27
3.4.2.	<i>Generátor signálov</i>	33
3.5.	OVERENIE RIEŠENIA	37
3.5.1.	<i>Vytvorenie tréningového datasetu</i>	37
3.5.2.	<i>Scenár testovania</i>	37
3.5.3.	<i>Rozdielne topológie</i>	38
3.5.4.	<i>Rozdielne datasety</i>	42
3.5.5.	<i>Analýza reálneho merania</i>	43
4.	ZHODNOTENIE	44
5.	TECHNICKÁ DOKUMENTÁCIA	46
5.1.	DOKUMENTÁCIA IMPLEMENTÁCIE.....	46
5.1.1.	<i>Main.cpp</i>	47

5.1.2.	<i>Waveinfo</i>	49
5.1.3.	<i>Ann</i>	52
5.1.4.	<i>Bluetoothreceiver</i>	59
5.1.5.	<i>Measurement</i>	65
5.1.6.	<i>Utils</i>	73
5.1.7.	<i>Generator.py</i>	75
5.2.	POUŽIVATEĽSKÁ PRÍRUČKA	79
5.2.1.	<i>Použitie analyzátoru</i>	80
5.2.2.	<i>Použitie generátoru</i>	82
6.	ZOZNAM POUŽITEJ LITERATÚRY	84
PRÍLOHA A	PLÁN PRÁCE NA DIPLOMOVEJ PRÁCI 1	I
PRÍLOHA B	PLÁN PRÁCE NA DIPLOMOVEJ PRÁCI 2	II
PRÍLOHA C	PLÁN PRÁCE NA DIPLOMOVEJ PRÁCI 3	III
PRÍLOHA D	ZHODNOTENIE PLNENIA PRÁCE V JEDNOTLIVÝCH ETAPÁCH	IV
PRÍLOHA E	OPIS DIGITÁLNEJ ČASTI PRÁCE	V

1. Úvod

Monitorovanie pacientov s neurologickým ochorením v kritickom stave sa výraznejšie rozšírilo počas niekoľkých posledných desaťročí. V čase, kym ešte neboli neuromonitorovacie techniky dostupné, sa lekári počas starostlivosti o týchto kriticky chorých pacientov spoliehali najmä na klinický obraz stavu pacienta, fyzikálne vyšetrenia, prípadne ďalšie diagnostické metódy. Avšak existujú rôzne ochorenia, pri ktorých nie sú prítomné klinické príznaky, alebo sú to príznaky nešpecifické. Zavedenie neuromonitoringu malo obrovský prínos pri samotnej diagnostike, kedy môžu byť zachytené aj najmenšie zmeny v neurologickom obraze, pri znížení výskytu iatrogénnych komplikácií operácií, pomáhajú pri určení prognózy a pri sledovaní odozvy na liečbu.

Všeobecne je potrebné vybrať najvhodnejšiu metódu, ktorou bude možné zhodnotiť stav nervovej, a prípadne aj svalovej sústavy daného pacienta. Na získanie výsledkov sa využívajú elektronické zariadenia, ktoré snímajú rozdiely elektrických potenciálov, získavajú analógový signál, ktorý je možné spracovať a previesť na signál digitálny. Tieto informácie sa následne spracovávajú softvérovými nástrojmi. Celý proces, ktorý musí prebehnúť až po získanie nameraných hodnôt v grafickej podobe, je po technologickej stránke veľmi komplexný a presahuje rozsah jednej diplomovej práce. V rámci tejto diplomovej práce sa teda zameriame na automatické vyhodnotenie signálov, ktoré získame. Výsledky modulu, ktorý vytvoríme v rámci tejto práce následne odošleme modulu, ktorý ich vizualizuje [1,3].

1.1. Použité skratky a pojmy

DMO – Detská mozgová obrna

EEG - Elektroencefalografia

EMG - Elektromiografia

SDR - Selektívna dorzálna rizotómia

ITB - Inratekálna aplikácia baklofénu

BP – Baklofénová pumpa

IONM - Intraoperačný monitoring nervovej funkcie

SEP - Somatosenzorické evokované potenciály

MEP - Motorické evokované potenciály

NIRS - Blízko-infračervená spektroskopia

MLP – multi layer perceptron (viacvrstvový perceptrón)

1.2. Zadanie

Zameranie diplomovej práce je z oblasti medicíny, konkrétnie peroperačného neuromonitoringu. Monitorovanie funkčnosti vybranej časti nervovej sústavy je veľmi dôležité v rámci neurochirurgických ale aj mnohých iných operácií, počas ktorých hrozí nenávratné poškodenie danej časti nervovej sústavy človeka. V súčasnosti výsledky týchto neuromonitorovacích vyšetrení musí analyzovať lekár. Táto skutočnosť je motiváciou pre návrh a implementáciu softvéru, ktorý dokáže výsledky neuromonitoringu automaticky vyhodnotiť, a tak zefektívniť prácu lekára, čím napomôže eliminovať chybovosť človeka pri tak závažných úkonoch ako je operácia. Analyzujte existujúce systémy, ktoré sú v súčasnosti používané na peroperačný neuromonitoring na Slovensku, a spôsob ich použitia na vyhodnotenie výsledných neurofyziológických údajov vygenerovaných neuromonitorovacím systémom pre vybraný typ ochorenia. Taktiež analyzujte existujúce spôsoby vyhodnocovania neurofyziológických údajov. Na základe analýzy navrhnite a implementujte v jazyku C vlastný prototyp softvérového nástroja, ktorý dostane na vstupe neurofyziológické údaje od neuromonitorovacieho systému, a ako výstup automaticky vygeneruje hodnotu miery korektnej funkčnosti monitorovaného tkaniva. Overte funkčnosť vášho riešenia simuláciou vyhodnotenia výsledkov vyšetrenia na reálnych údajoch poskytnutých vo forme súboru a zhodnot'te efektivitu vytvoreného softvéru.

2. Analýza problému

2.1. Diagnózy

Detská mozgová obrna je dlhodobé ochorenie spojené s mnohými ťažkosťami zdravotného, pedagogického, sociálneho a spoločenského charakteru. Starostlivosť o dieťa s DMO musí byť komplexná, vyžaduje si vytvorenie multidisciplinárneho tímu viacerých odborníkov, ktorého neoddeliteľnou súčasťou je najmä rodina. Detská mozgová obrna (DMO) predstavuje skupinu porúch ovplyvňujúcich schopnosť pohybovať sa. Je zapríčinená poškodením mozgu počas tehotenstva, pôrodu alebo krátka po narodení dieťaťa [4].

Spastická forma DMO

Táto forma DMO sa vyskytuje najčastejšie a je pre ňu charakteristická svalová spasticita. Svaly postihnuté týmto ochorením majú trvalé zvýšené napätie – sú stiahnuté a tuhé na dotyk. Spastická DMO sa ďalej rozdeľuje:

1. *Spastická diparéza* má najčastejšie zastúpenie s pomedzi všetkých spastických foriem. Je postihnutím hlavnej sivej mozgovej kôry najčastejšie sa vyskytuje u predčasne narodených detí v dôsledku popôrodných komplikácií ako je krvácanie do mozgu či nedokyslienie, v dôsledku ktorého odumierajú nervové bunky nachádzajúce sa v sivej kôre. Menej vyvinutá ostáva dolná polovica tela počas rastu. Dochádza k zvýšeniu svalového tonusu – napäťa na dolných končatinách. V stoji sú nohy stočené dovnútra a stoj je typizovaný na prstoch nôh.
2. *Spastická hemiparéza* je postihnutie jednej hemisféry mozgu. Jedná sa o postihnutie jednej celej polovice tela, ktoré je sprevádzané jednostrannou poruchou hybnosti najčastejšie spastickej formy. Pridružená je mentálna retardácia ktorá má spojitosť s epilepsiou a trpí ňou 50% hemiparetických pacientov.
3. *Spastická triparetická forma* je uvedená ako samostatné ochorenie detskej mozgovej obrny. Len tretina týchto detí ma normálny intelekt a polovica trpí epilepsiou. Ochorenie sprevádzza ťažké postihnutie, ktoré sa z terapeutického hľadiska zvláda len veľmi ťažko.
4. *Bilaterálna spastická hemiparéza* - často ide o najťažšiu a najzávažnejšiu formu detskej mozgovej obrny, pri ktorej bývajú poškodené obidve hemisféry mozgu [3].

Nespastické formy DMO

Pri tejto forme býva svalový tonus znížený. Môže sa vyskytovať v týchto typoch.

1. Hypotonická forma - je typická znížením svalového tonusu. Dieťa je chabé, prítomné býva oslabenie nielen svalových ale aj klíbnych štruktúr v zmysle hypermobility. Popisuje taktiež prítomnosť asynergie ako poruchu koordinácie a nevôle svalových

skupín skoordinovať sa. Tým mizne aj dôslednosť cielených pohybov. Spomalený celkový psychomotorický vývin.

2. Dyskinetická forma sa prejavuje patologicky rýchlym a neovládateľným striedaním napäťia a uvoľnenia svalstva v postihnutých oblastiach. U dieťaťa pozorujeme mimovoľné pohyby, zášklby tváre, nezrozumiteľná a pomalá reč, poruchy sluchu. Intelekt je na dobrej úrovni [5].

2.1.1. Spasticita

Spasticita je závažný prejav ochorení centrálneho nervového systému, ktorý spôsobuje výrazné zhoršenie kvality života pacienta. Môže nastať nielen po cievnej mozkovej príhode, ale aj po úraze, v dôsledku zápalu, nádoru alebo degeneratívneho procesu ako je DMO. Okrem zvýšenej svalovej aktivity sa vyznačuje tiež skrátením svalu a čiastočnou stratou hybnosti. Miera spasticity nie je vždy rovnaká, kolíše od ľahkej svalovej stuhnutosti až po ťažké a bolestivé stavby. Mení sa tiež podľa aktuálnej kondície, únavy, zhoršíť ju môže veľké horúco alebo naopak zima, infekčné ochorenie alebo tesné oblečenie. V miernejších formách sa spasticita prejavuje ako svalové napätie, ktoré neustáva, alebo ako mimovoľné pohyby končatín, ktoré sa môžu objavovať aj v noci. Môže byť aj bolestivá [5].

2.1.2. Spina bifida

Spina bifida - doslova preložené - znamená „rozštiepená chrbtica“. Ide o vývojovú anomáliu v oblasti chrbtice a miechy. Pretože sa chrbtica a miecha vyvíja z neurálnej rúry, hovorí sa aj o defekte neurálnej rúry. Za normálnych okolností splynú obe časti oblúka chrbtice do jedného kruhu, ktorý ohraničuje stavcový otvor. V stavcovom otvore leží miecha, obklopená miechovými plenami. Pri spina bifida chýba uzáver oblúka jedného alebo viacerých stavcov. Touto štrbinou sa môžu časti miechy a nervy vačkovito vyklenúť.

Liečba závisí od závažnosti poruchy chrbtice a miechy. V niektorých prípadoch si mierna forma nevyžaduje žiadnu liečbu. Spina bifida aperta však môže znamenať ťažké postihnutie. V takýchto prípadoch sa odporúča chirurgický zákrok do 48 hodín od pôrodu. Veľkým prínosom môže byť fyzioterapia, ako aj ortopedické pomôcky (napríklad ortézy), ktoré pomáhajú zmierňovať dopad deformácií kĺbov v dôsledku spina bifida [6].

2.2. Liečebná metóda - Selektívna dorzálna rizotómia

Medzi základné formy terapie a zlepšenia komfortu detí so spasticitou a ich rodín patria možnosti chirurgického ovplyvnenia spasticity. Dnes je chirurgická liečba spasticity u detí v mnohých krajinách benefitom pre pacienta, jeho rodinu a spoločnosť. Znižuje riziko komplikácií, zlepšuje komfort a prognózu sekundárnych komplikácií ochorenia, znižuje finančné zaťaženie a náklady spojené s liečbou jednak rodiny pacienta a jednak aj krajiny v ktorej žije. Inovatívnosť chirurgického riešenia preto poskytuje aj rozšírenie výberu terapie pre samotného pacienta a jeho rodinu. Deti, ktoré sú najvhodnejšie na SDR, sú vo veku od 3 do 10 rokov, ale môžu to byť aj staršie deti. Pacienti sú zvyčajne zapojení do aktívneho programu fyzickej terapie.

Pri výkone SDR sa jedná o redukciu spasticity parciálnym preťatím vlákien zadných senzitívnych miechových koreňov. Počas SDR sa využíva intraoperačný monitoring nervovej funkcie (IONM). Monitoring zahŕňa miechovú stimuláciu, snímanie elektrických potenciálov z vyšetrovaných svalových skupín a ich následné vyhodnotenie.[9]

2.3. Neuromonitoring

Elektrofiziologické monitorovanie alebo neuromonitoring sa používa počas chirurgického zákroku na posúdenie funkčnej integrity mozgu, mozkového kmeňa, miechy alebo periférnych nervov. Cieľom monitorovania je upozorniť chirurga a anesteziológa na hroziace zranenie, aby sa zabránilo trvalému poškodeniu. V niektorých prípadoch sa neuromonitoring používa na mapovanie oblastí nervového systému s cieľom viest' lekára. Neuromonitoring môže zahrňať zaznamenávanie spontánnej aktivity alebo vyvolanú odozvu na podnet (napríklad evokovaný potenciál). Stal sa bežným počas mnohých chirurgických zákrokov, často nahradzajúcich intraoperačné bdenie. Je vykonávaný špecializovaným tímom so špecifickými odbornými znalosťami v použitej technike [8].

Techniky neuromonitoringu

Podľa spôsobu merania evokovaných potenciálov, delíme tieto potenciály na:

- Jednotkové potenciály – Získavajú sa meraním jednotlivých vlákien nervových buniek alebo ich skupín. Merajú sa pomocou mikro-elektród, ktoré navyše ani nie sú v priamom kontakte s meraným vláknom. Takéto potenciály môžu predstavovať buď spontánnu aktivitu bez akejkoľvek stimulácie alebo môžu byť evokované externým stimulom [1, 2].
- Potenciály blízkeho poľa – Vytvorením sumy aktivít viacerých nervových buniek alebo vlákien, získame reprezentáciu potenciálu blízkeho poľa. Tento druh potenciálov, je už meraný

uložením elektród priamo na nerv alebo sval. Z toho vyplýva, že aj elektróda je väčšia ako tá, ktorá sa používa pri meraní jednotkových potenciálov [1, 2].

- Potenciály ďalekého poľa – Sú zmesou potenciálov, ktoré sú generované viacerými odlišnými nervovými štruktúrami, na rozdiel od potenciálov blízkeho poľa, ktoré odrážajú elektrickú aktivitu v konkrétnej nervovej štruktúre. Nie sú merané z elektród uložených priamo na nerve ale v určitej vzdialosti od nervovej štruktúry čím využívajú elektromagnetické polia a elektromagnetickú indukciu. V dôsledku tohto spôsobu merania a toho, že sú zložené z viacerých zdrojov signálu, sa potenciály ďalekého poľa vyznačujú menšou amplitúdou a teda je zložitejšie zobraziť ich priebeh [1, 2].

2.3.1. Intraoperačný neuromonitoring

Periprocedurálny neurofiziologický monitoring pacienta počas operácií (otvorených i endovaskulárnych) na aorte a prívodných mozgových artériach sa prvýkrát začal využívať okolo roku 1980. Odvtedy sa postupne rozšíril rozsah monitoringu a rastie aj počet operačných výkonov. Aktuálne trendy a rozvoj techniky a prístrojového vybavenia smerujú k multimodálnemu peroperačnému monitoringu pacienta zo strany neurológa i anestéziológa a tiež k procedúram „šitým na mieru“ konkrétneho pacienta [21]. Nároky kladené na optimálny monitoring sú jednoznačné – čo najmenšia záťaž pacienta a zdravotníckeho tímu a zaistenie čo najvyššej bezpečnosti výkonu s minimalizáciou rizík. Na druhej strane, rozvoj neuromonitoringu kladie vysoké nároky na medziodborovú spoluprácu všetkých zdravotníckych pracovníkov pred, pri i po výkone (zachytenie zmien neurologického stavu je možné len pri podrobnom zázname všetkých stavov). Intraoperatívny neurofiziologický monitoring je z hľadiska odbornosti celosvetovo najmä doménou neurológov a neurochirurgov so špecializovanou odbornosťou, no nevyhnutná je úzka spolupráca s anestéziológom a neurofiziologickými technikmi. V SR je doménou neurológov. Špecifická nadstavbová špecializácia, ktorá je bežná napríklad v USA, u nás zatiaľ nie je požadovaná. Najčastejšie sa peroperačný neuromonitoring využíva pri neurochirurgických (predovšetkým resekčných) výkonoch na mozgu a mieche. Z cievnych výkonov je vhodný najmä pri operáciách na aorte, pri komplikovaných karotických endarterektómiah v celkovej anestézii a pri ošetrení intrakraniálnych aneurysiem a artério-venóznych malformácií (chirurgicky i endovaskulárne). Cievne ochorenia miechy a operácie priamo na aorte sa vzhľadom na riziko poškodenia miechy peroperačne monitorujú formou multimodálneho sledovania zostupných (motorické EP) alebo vzostupných (senzitívne EP) miechových dráh a podrobnosti presahujú rozsah tohto článku [21].

Rozhodnutie o konkrétnom spôsobe monitoringu pacienta vychádza z individuálnej analýzy stanovenej diagnózy, anatomickej lokalizácie, anomálie cievneho zásobenia a jeho prínosom je sledovanie stavu a poskytovanie informácií, ktoré môžu pomôcť operatérovi pri rozhodovaní počas výkonu [1]. Zmyslom monitoringu v prípade endovaskulárnych výkonov na mozgových artériach je sledovanie funkcie mozgu so zameraním sa na úsek zásobený operovanou cievou (somatosenzorické evokované potenciály – SEP, motorické evokované potenciály – MEP,

elektroencefalografia – EEG, blízko-infračervená spektroskopia – NIRS), prípadne priame sledovanie prietoku v danej cieve (transkraniálna dopplerovská ultrazvuková sonografia – TCD). V posledných rokoch sa využíva celosvetovo (i u nás) pri endarterektómiách rutinne najmä NIRS, vzhľadom na jednoduchú, nezaťažujúcu aplikáciu, možnosť kontinuálneho monitoringu a nepotrebnosť špecializovaného personálu [21].

Pre prehľadnosť uvádzame i ďalšie modality monitoringu. EEG dokáže citlivou reagovať na zmeny perfúzie kortikálnych štruktúr (súvislejší pokles frekvencie, atenuácia amplitúdy až izoelektrická línia). Monitoring výhradne prostredníctvom EEG, vzhľadom na jeho záhyt predovšetkým kortikálnej aktivity, však nie je optimálny. Problematický je záhyt poškodenia subkortikálnych štruktúr, riziko nesprávnej interpretácie už preexistujúcich EEG zmien a taktiež nemožnosť záhytu menších (regionálnych) výpadkov perfúzie [21].

Monitoring formou sledovania evokovaných potenciálov (EP) analyzuje kortikálne potenciály vznikajúce pri elektrickej stimulácii periférnych nervov (somatosenzorické EP, prípadne periférne motorické prejavy pri externej kortikálnej stimulácii (motorické EP). Lepšie dokáže sledovať zmeny v perfúzii subkortikálnych štruktúr, vyžaduje však vysokú odbornú erudíciu hodnotiaceho neurológa. Transkraniálny doppler (TCD) môže sledovať rýchlosť prúdenia v a. cerebri media a detektovať prípadnú embolizáciu. Ako jediná forma neuromonitoringu nie je vhodný, keďže nesleduje funkčné cerebrálne zmeny, na druhej strane, môže dobre detegovať akútnu oklúziu prietoku, embolizáciu a tiež hyperperfúziu [21].



Obr. 1: Aplikácia, zobrazujúca graf pri intraoperačnom neuromonitoringu

2.3.2. Elektromyografia

Elektromyografia je vyšetrenie zachycujúce elektrickú aktivitu svalov. Overuje sa ich stav a správne nervové zásobovanie - inervácia. Vyšetrenie elektrickej aktivity svalov sa používa v diagnostike ochorení svalstva, pri postihnutí nervov a pri poruchách nervovo-svalového

prenosu. Pomáha aj pri lokalizácii postihnutia nervového systému. Vyšetrenie indikuje a vykonáva neurológ.

Každý zdravý človek ovláda prácu svojich svalov pomocou nervov, sval a jeho nervy spolu vytvárajú neoddeliteľný tím. Ich správna funkcia môže byť poškodená úrazom, nadmerným tlakom, zápalom, toxickejmi látkami alebo niektorou z vrodených vývojových chýb. EMG odhalí prítomnosť, umiestnenie a rozsah takéhoto poškodenia. Meranie elektrickej aktivity svalov vyšetrovaného pomáha poodhaliť príčinu ich oslabenia, ochrnutia alebo príčinu mimovoľných zášklbov svalstva.

Vyšetrenie sa vykonáva pomocou špeciálneho prístroja - elektromyografu, ktorý sa skladá z elektród, zosilňovača, vyhodnocovača a monitora. Bežne sa používajú dva druhy elektród, ktoré sa prikladajú na kožu alebo sa vpichujú pod kožu (majú tvar ihly).

Svaly ľudského tela fungujú na základe príkazov z centrálnej nervovej sústavy (mozgu a miechy), ktorá pomocou nervov im odovzdáva informácie. Nervy vedú signály, ktoré sa podobajú elektrickým impulzom. Keď sa signál - vzruch dostane až na miesto kontaktu so svalstvom, zmení sa elektrická rovnováha vo svalových bunkách a vznikne sťah svalstva, teda tým aj určitý pohyb časti tela.

Na vyšetrenie sa vždy používajú dve elektródy (stimulačná alebo vstupná elektróda, prijímacia elektróda), pomocou ktorých je možné sledovať zmeny elektrickej aktivity svalu pri činnosti. Do svalu je pomocou stimulačnej elektródy privádzajú slabé elektrické impulzy a prijímacia elektróda zachycuje odpoved' na toto elektrické podráždenie a rýchlosť vedenia vyvolaného vzruchu nervom. Výsledok sa zobrazuje na monitore ako krivka EMG [10].

EMG záznam

Počas elektromyografického vyšetrenia sa u pacienta stimulujú periférne nervy aby sa zistila ich funkčnosť. Nerv sa stimuluje pomocou elektród a impulz, ktorý prejde nervom sa zobrazí na grafe ako krivka. Pre tento postup sú potrebné minimálne dve elektródy, jedna vysielacia a jedna prijímacia. Hodnoty sa merajú v mikrovoltoch. Miesta, ku ktorým sú pripojené elektródy presne určí lekár. Týmto meraní môžeme zachytiť niekoľko typov vln, napríklad M vlny, F vlny a H reflexy [1, 17].

H – reflex

H-reflex alebo Hoffmannov reflex je reflexná reakcia svalov po elektrickej stimulácii senzorických vlákien. Môžeme ho snímať pri stimulácii až 20 svalov na tvári, horných aj dolných končatinách. Latencia od stimulu trvá približne 30ms. Pri silnejšej stimulácii sa môže vyskytnúť aj nález F-vlny, čím sa narušia výsledky. Jeho využitie spočíva hlavne vo vyhodnotení funkčnosti periférnych nervov, cez ktoré prechádzajú impulzy od miechy do končatín a opačne. Hodnoty H-reflexu závisia od sily elektrického stimulu [1, 6].

F – vlna

F-vlna je evokovaný výboj nadväzujúci na podaný stimul. Môže byť zaznamenaná z akéhokoľvek svalu na tele, ale avšak pre svoju latenciu býva na zázname skrytá za M-vlnou. Je to z toho dôvodu, že má na zázname natoľko malú amplitúdu. Fyziologická hodnota amplitúdy F-vlny je menej ako 5% maximálnej amplitúdy M-vlny. Ak jej hodnota vzrástá, je to znak spasticity. Z jej hodnoty je možné odvodiť rýchlosť, s akou sa impulz prenáša v danom rozsahu nervových dráh. Potrebné je poznať presnú dĺžku končatiny od miesta stimulácie po konkrétné miesto na chrbtici. Toto miesto na chrbtici je určené na základe vedomostí z anatómie [1, 6].

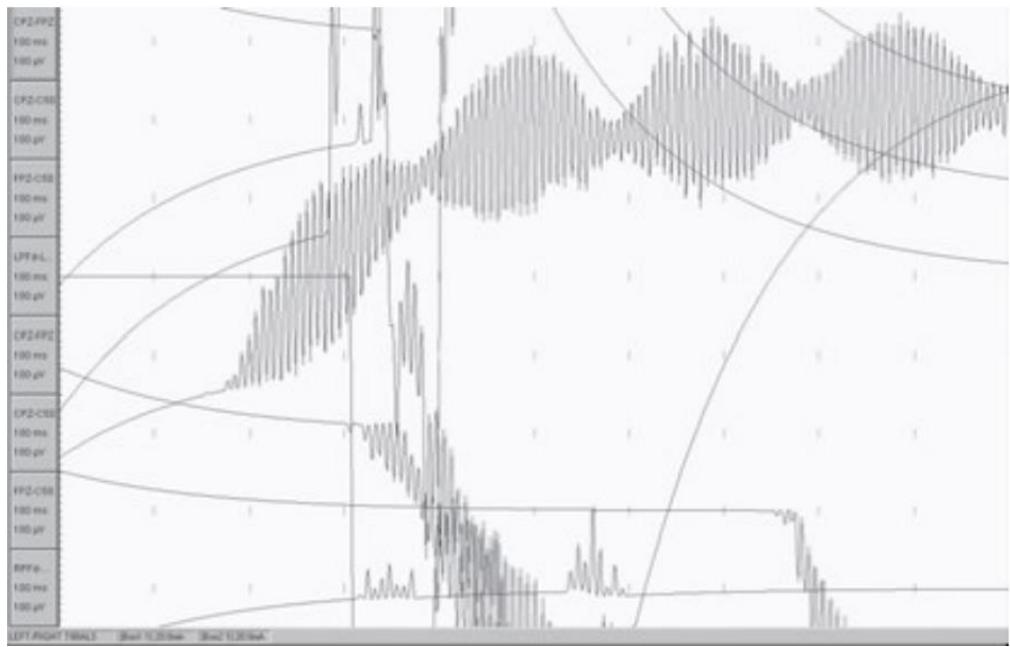
M – vlna

Je to akčný potenciál, spôsobený priamym vedením elektrického prúdu nervom. Tieto akčné potenciály sú vedené od miesta stimulácie k nervosvalovej platničke [1, 6].

2.3.3. Faktory ovplyvňujúce záznam neuromonitoringu

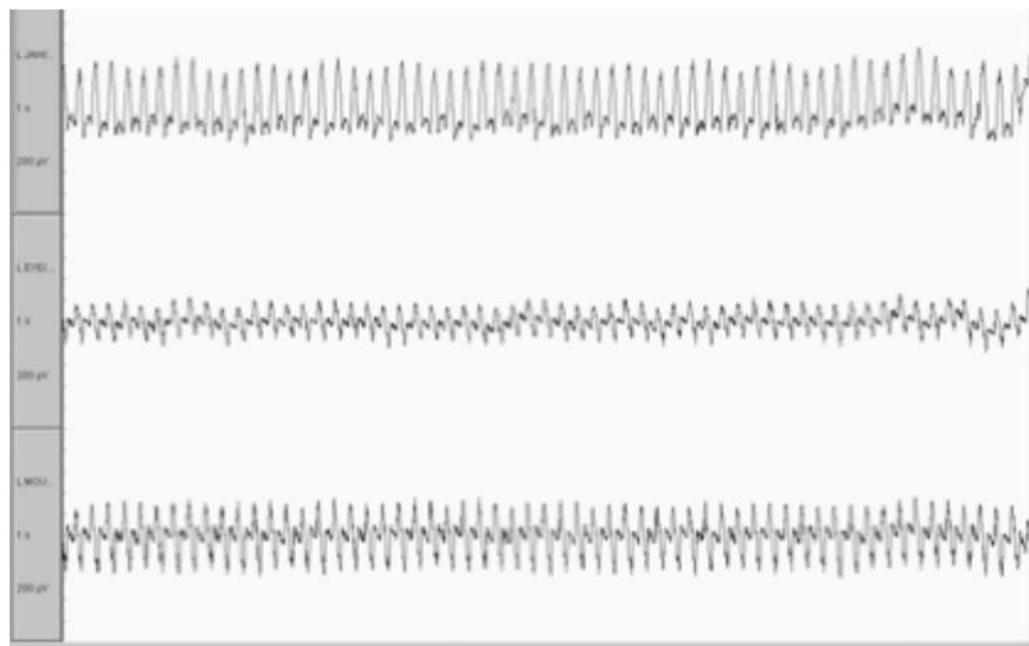
Na grafe neuromonitoringu sa okrem zachytených signálov nachádzajú aj tzv. *artefakty*. Je to zaznamenaná aktivita nervu, ktorá je ale spôsobená vonkajším prostredím. Narúša čistotu grafu, ktorý zobrazuje meranie a je nutné ich identifikovať a prípadne eliminovať.

Artefakty môžu vznikať pôsobením rôznych elektronických zariadení v miestnosti počas merania. Bežným dôvodom vzniku artefaktu na grafe je použitie elektrokauterizácie počas operácie. Kauterizácia sa môže využiť na elektrické rezanie kože alebo na spálenie krvácajúcich ciev, ktoré sú týmto spálením uzavreté, čím sa krvácanie zastaví. Príklad artefaktu spôsobeného kauterizáciou je zobrazený na obrázku 2 [1, 8].



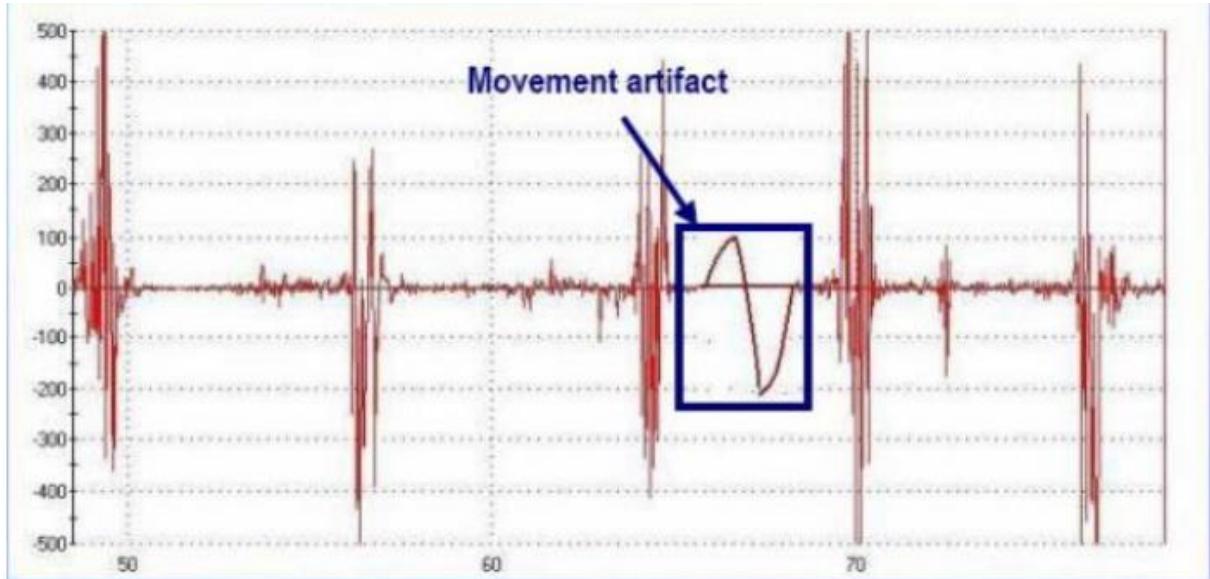
Obr. 2: Skreslenie neuromonitoringu elektrokauterizáciou. [6]

Ďalším z dôvodov, kvôli ktorému môžu vznikať artefakty je operačný stôl. Tie bývajú napájané elektrickým prúdom, pretože poskytujú polohovanie a napomáhajú k lepšej polohe pacienta. Pretože sú napájané z elektrickej zásuvky, môžu tiež produkovať artefakty, ktoré môžu skomplikovať spracovanie a zobrazenie evokovaných potenciálov. Na obrázku 3 je zobrazený takýto typ artefaktu [1, 8].



Obr. 3: Artefakt spôsobený operačným stolom. [6]

Je potrebné aby bol pacient počas vyšetrenia v relaxovanom stave a nehýbal sa vôbec alebo len minimálne aby bol záznam čo najčistejší a dostatočne presný. Keďže pri pohybe svalmi prechádza elektrický prúd, je každý pohyb zaznamenaný na grafe ako artefakt. Podobné artefakty môžu byť tiež spôsobené pohybom alebo uvoľnením elektród. Na obrázku 4 je zobrazený takýto typ artefaktu [1,8].



Obr. 4: Artefakt spôsobený pohybom. [9]

2.3.4. Evokované potenciály

Somatosenzorické evokované potenciály (SEP)

Sledovanie SEP sa uplatňuje predovšetkým pri operáciách miechy a periférnych nervov. Sleduje sa vedenie stimulov z periférie do centra. V chirurgii gliómov mozgu sa využívajú predovšetkým na lokalizáciu senzorimotorického kortexu (t. j. precentrálneho a postcentrálneho závitu). Využíva sa detekcia miesta „zvratu fázy“, t. j. inverzie polarity, pričom tento fenomén nie je doposiaľ úplne objasnený, predpokladá sa sumácia aktivity zo samostatných neurálnych generátorov, ktoré produkujú odlišné komponenty SEP v post- a precentrálnom suku. Registruje sa z povrchu mozgu subdurálne vloženými elektródami po stimulácii periférnych nervov. K zvratu fázy dochádza nad hranicou medzi gyrus precentralis a postcentralis. V porovnaní s inými metódami neuromonitoringu je táto metóda menej ovplyvniteľná anestetikami a je použiteľná aj u detí [3].

Motorické evokované potenciály (MEP)

Na rozdiel od SEP je pomocou MEP možné priamo monitorovať integritu kortikospinálnej dráhy, ich monitorovanie je však komplikovanejšie pre výrazné ovplyvnenie anestetikami. Pyramídovú dráhu je možné aktivovať elektrickou stimuláciou motorického kortexu, záznam

sa vykonáva distálne od miesta operácie z periférnych nervov alebo svalov (elektromyografické odpovede). Odpovede sú kontinuálne zaznamenávané počas resekcie v blízkosti motorických štruktúr, pokles amplitúdy MEP je predzvestou ich hroziaceho poškodenia. V niektorých prípadoch však nie je vylúčená možnosť irreverzibilného poškodenia subkortikálnych vláken predtým, ako je zmena zaznamenaná poklesom amplitúdy MEP [3].

2.4. Získavanie monitorovacích dát

V tejto kapitole je opísaný spôsob získavania analógového signálu a jeho následný prevod do digitálnej formy. Signál v digitálnej forme je pripravený na ďalšie spracovanie a vyhodnotenie. Analýze nameraného signálu sa venujeme v ďalších kapitolách. Na tomto mieste je však potrebné povedať, že už takto spracovaný signál je pripravený na zobrazenie a ponúka lekárovi prvotný pohľad na stav pacienta. Získavanie analógového signálu sa vykonáva snímaním rozdielov elektrických potenciálov [1].

2.4.1. Diferenčný zosilňovač

Diferenciálny zosilňovač je zosilňovač, ktorého výstupné napätie je funkciou rozdielu napäti na jeho dvoch vstupoch. Obvykle sa realizuje v integrovanej podobe, keďže je pre dobrú symetriu potrebné použiť vstupné tranzistory s rovnakými vlastnosťami. V tomto prípade sa v prvom kroku urobí normalizácia EMG signálu. Nasleduje zosilnenie tohto signálu. Ako vstup sa použijú meracie elektródy [1, 4].

2.4.2. Analógová filtrácia

Pred digitálnym spracovaním signálu sa využíva analógová filtrácia. Touto metódou sa odstraňujú extrémne výkyvy frekvencie signálu. Keďže rôzne svaly svojou činnosťou vytvárajú nízkofrekvenčné signály, je táto metóda použitá na ich odstránenie. Používa sa hornopriepustný filter. Medzi svaly, ktoré takéto signály produkujú patria napríklad srdce alebo dýchacie svalstvo. Naopak, pre odstránenie príliš vysokých frekvencií sa používa dolnopriepustný filter. Avšak pri použití tohto filtra je nutné dávať pozor na správne nastavenie, inak by sa mohli odfiltrovať aj sledované evokované potenciály. Medzi nežiaduce vysoké frekvencie patrí najmä elektronický šum. Existuje viacero typov šumov, pričom tie najvýznamnejšie sú: tepelný šum, kmitavý šum a výstrelkový (Shottkyho) šum. Frekvenčné pásmo odfiltrovaného signálu sa pohybuje zvyčajne v rozsahu od 200Hz do 1 až 2 kHz. Filtráciou by sa malo minimalizovať skreslovanie signálu javmi nezávislými od sledovaného javu [1, 4].

2.4.3. Analógovo-digitálna konverzia

Pri ďalšom kroku dochádza ku konverzii analógového signálu na digitálny. Na tento úkon sa využíva ADC – analógovo-digitálny konvertor. Je využitie nenájdeme len v medicíne pri tomto konkrétnom prípade ale v iných odvetviach, resp. zariadeniach ako napríklad mobilné telefóny, digitálne kamery, automatizované zberače dát v laboratóriách a pod. ADC prevádzka analógové hodnoty na digitálne na základe frekvencie vzorkovania. Je to časový úsek medzi dvoma zaznamenanými hodnotami. Hlavnou časťou prevodu je rozdelenie osi X (časovej osi) na kvantá – rovnomerné úseky a z nich sa vyberie jedna hodnota. Týmto spôsobom súčasťou stratíme veľa dát (a teoretickú možnosť nekonečného približovania) a spojité krivky sa prevedie len na množinu diskrétnych dát ale umožníme dnešným počítačom s týmito dátami plnohodnotne pracovať. Avšak pri tomto konkrétnom prípade je nutné aby vzorkovacia frekvencia ADC bola aspoň päťkrát väčšia ako medzná frekvencia dolnopriepustného filtra. Ak by to tak nebolo, môže sa stať, že evokovaný potenciál by bol skreslený alebo úplne vynechaný [1, 4].

2.4.4. Digitálne vyhľadenie signálu

Pre minimalizovanie amplitúdových rozdielov, ktoré sa nachádzajú v čase blízko seba, existuje niekoľko algoritmov. V tomto kroku sa použije aspoň jeden z nich [4].

2.5. Metódy vyhodnocovania signálu

Získaný signál je potrebné analyzovať a vyhodnotiť. Pre lepšiu prehľadnosť zobrazenia signálu lekárovi, je tiež vhodné signál upraviť a odstrániť artefakty, ktoré získaný obraz narúšajú. Pre takéto vyčistenie dát, je potrebné určiť atribúty, podľa ktorých sa budú dátá filtrovať.

2.5.1. Atribúty signálu

Nasleduje zoznam atribútov, ktoré je vhodné sledovať pri analýze signálu. Každý impulz v rámci merania sa vyznačuje istými atribútmi, ktorými je možné ho identifikovať. Na základe porovnania vzorového impulzu s nameraným impulzom vieme späť určiť funkčnosť meraného nervu alebo svalu. Tiež vieme na základe porovnania týchto atribútov vyfiltrovať tie časti dát z merania, ktoré nás zaujímajú.

Amplitúda - Amplitúda hovorí veľkosti napätia, ktoré daným nervom alebo svalom prešlo. Pri analýze evokovaných potenciálov, môže nastáť komplikácia práve pri určovaní amplitúdy, pretože je môžete byť nízka a teda v zázname ľahko rozpoznanateľná.

Šírka impulzu – Hovorí o časovom úseku, v ktorom sa nachádza priebeh daného impulzu.

Tvar impulzu – Hovorí o type impulzu. Zdravý nerv vytvára pri opakovanech stimuloch stále rovnaký tvar impulzu. Práve pri rôznych výchylkách tvaru, dokážeme odhaliť poruchu daného nervu.

Latencia – Čas od vyslania signálu z elektródy a kým druhá elektróda zachytí signál [6,7].

2.5.2. Klasifikácia tvaru impulzu

V tejto kapitole sú opísané metódy, ktoré je možné použiť pri analýze a vyhodnocovaní signálu. Tieto metódy sú určené všeobecne pre prácu s dátami, ktoré reprezentujú vektor alebo krivku. Pomocou týchto postupov je možné jednotlivé impulzy klasifikovať a na základe toho rozlíšiť, ktoré časti merania odfiltrovať a ktoré ponechať.

Klasifikácia je problém určenia, do ktorej triedy patrí nové pozorovanie, na základe trénovacieho súboru údajov obsahujúcich inštancie, ktorých členstvo v triede je už známe. K tomu máme k dispozícii trénovaciu množinu obsahujúcu pozorovania (dáta, inštancie), pre ktoré sú kategórie správne určené. Jednotlivé pozorovania sú analyzované do množiny kvantifikovateľných vlastností, známych ako nezávislé premenné, rysy, fíctury (features) apod. Tieto vlastnosti môžu byť kategoriálne (napr. "A", "B", "AB" alebo "O" pre krvné skupiny), ordinálne (napr. "veľký", "stredná" alebo "malý"), celočíselné (napr. počet výskytov slova v emaile) alebo reálne (napr. meranie krvného tlaku). Niektoré algoritmy pracujú iba s diskrétnymi hodnotami a požadujú, aby sa celočíselné alebo reálne dáta diskretizovali, tj. Previedli sa na kategórie obsahujúce podobné pozorovania (napr. "Menej ako 5", "medzi 5 a 10", "viac než 10"). Ako príklad problému klasifikácie je priradenie emailu do triedy "spam" alebo "nie-spam" alebo priradení diagnózy danému pacientovi, podľa jeho pozorovaných charakteristík (pohlavie, vek, krvný tlak, prítomnosť alebo absencia určitých symptómov, ...).

Algoritmus, ktorý implementuje klasifikáciu, sa nazýva klasifikátor. Tento termín sa používa aj pre matematickú funkciu, ktorá je implementovaná algoritmom a triedi vstupné dátu do tried.

V terminológii strojového učenia je klasifikácia považovaná za metódu učenia sa s učiteľom, to je učenie sa, pri ktorom je známa trénovacia množina správne klasifikovaných príkladov. Analogická metóda v učení bez učiteľa je známa ako klastrovanie a spočíva v spájaní dát do kategórií podľa nejakej miery vnútornej podobnosti (napr. Odvodené zo vzdialenosť medzi inštanciami, ktoré sú považované za vektory vo viac dimenzionálnom vektorovom priestore) [16].

KNN – K-nearest-neighbors

Algoritmus k-nearest neighbors je z hľadiska princípu jeden z najjednoduchších algoritmov strojového učenia. Tento algoritmus sa využíva na klasifikáciu. Algoritmus v tréningovej fáze najprv umiestní prvky tréningovej množiny do priestoru tak, aby prvky z rovnakých tried vytvárali zhluhy. Nasleduje fáza klasifikácie, kde je prvok, ktorý je potrebné klasifikovať umiestnený do priestoru podľa jeho vlastností. Potom sa v priestore nájde k najbližších prvkov

okolo klasifikovaného prvku. Následne je prvak klasifikovaný do triedy, do ktorej patrí väčšina k najbližším prvkov.

Číslo k musí byť prirodzené číslo. Pri klasifikačnom probléme s dvomi triedami musí byť k nepárne číslo, pretože by mohlo dôjsť pri klasifikácii k tomu, že by sa počet prvkov oboch tried vo vyhradenom priestore rovnal, a potom by algoritmus nedokázal určiť, ku ktorej triede klasifikovaný prvak pravdepodobnejšie bude patriť. Pre tento istý dôvod k nesmie byť ani násobok počtu tried [14].

SVM – support vector machines

Je to metóda strojového učenia s učiteľom, ktorá slúži na klasifikáciu a tiež regresnú analýzu. Princípom SVM je rozdelenie tréningových dát zakreslených v bodovom diagrame (angl. Scatter plot) na dve protiľahlé oblasti náležiace jednotlivým triedam dát. SVM priestor rozdeľuje tzv. nadrovinou (angl. Hyperplane). Nadrovinu tiež môže byť označená ako tzv. rozhodovacia hranica (angl. Decision boundary), ktorá oddeluje na bodovom diagrame dve triedy a určuje, ktoré body patria do ktorej triedy.

Pre optimálnu nadrovinu platí, že musí byť umiestnená v čo najväčšom odstupe (angl. Maximal margin) od krajných bodov, nazývaných podporné vektory (angl. Support vectors). Inými slovami, nadrovinu musí vytvárať okolo seba čo najväčšie pásmo bez bodov a musí byť v strede tohto pásma tak, aby okrajové body oboch rozdelených oblastí boli rovnako vzdialené. SVM sa delí na lineárnu a nelineárnu, a to podľa schopnosti rozdeliť priestor.

Lineárna SVM je variant tohto algoritmu, ktorý sa používa, keď sú dátá lineárne oddeliteľné, čiže oddeliteľné lineárnu nadrovinou. Pri lineárnej SVM algoritmus pracuje v pôvodnej dvojrozmernej rovine dát.

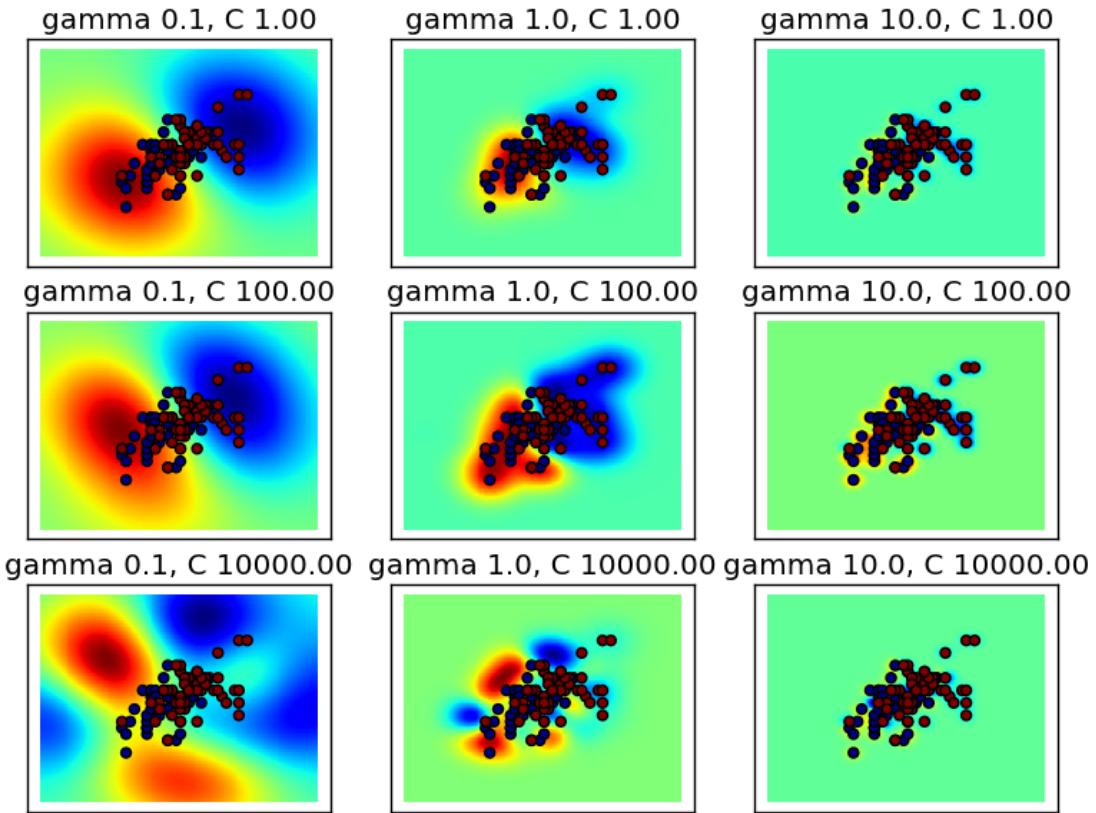
Nelineárna SVM sa používa, keď rovinu dát nie je možné rozdeliť lineárne. Tu sa uplatňuje funkcia zvaná jadrová transformácia (angl. *Kernel transformation*) alebo jadrový trik (angl. *Kernel trick*). Táto funkcia funguje tak, že lineárne nerozdeliteľnú dvojrozmerné množinu dát rozšíri o ďalšiu dimenziu, čím vzniká trojrozmerná rovina. Osi dvojrozmernej roviny x a y sú vlastnosti objektu, teda *features* a os z pri trojrozmernej rovine je výsledkom rovnice $z=x_1+y_2$. Táto trojrozmerná rovina je už potom lineárne rozdeliteľná.

Pri programovaní SVM klasifikátora sa používajú rôzne parametre. Medzi najdôležitejšie patria *jadro*, *gamma* a C [14].

Jadro alebo *kernel* je funkcia, ktorá je základom SVM a vykonáva jeho výpočty. Pre lineárnu SVM existuje len jedno lineárne jadro. Pre nelineárnu SVM je k dispozícii viacero jadier a prístupov k riešeniu. Medzi najpoužívanejšie patria RBF (*Radial basis function*) a polynomiálne jadro.

Gamma určuje ako ďaleko od nadroviny budú mať body vplyv na túto hranicu. Keď je gamma vysoká, nadrovinu ovplyvňujú len body, ktoré sú pri nej blízko a vzdialé body ignoruje. Pri vysokej hodnote sa môže začať nadrovinu zakrikovať a kl'ukatiť okolo jednotlivých dátových bodov. Ak je gamma nízka, nadrovinu ovplyvňujú aj vzdialé body. Tento parameter využíva len jadro RBF [13].

Parameter C určuje kompromis medzi rovnosťou nadroviny a počtom správne klasifikovaných tréningových bodov. Pri vyšších hodnotách C sa bude nadrovina zakrivovala a klúkať okolo jednotlivých dátových bodov, aby všetky zaradila do správnej oblasti. Týmto však klasifikátor stráca schopnosť zovšeobecňovať. Naopak pri nižších hodnotach bude nadrovina rovná, ale za cenu zopár nesprávne klasifikovaných dátových bodov. Na obrázku 5 vidíme vplyv týchto parametrov na jadro RBF.



Obr. 5: Vplyv gamma a parameter C na jadro RBF [13]

Neurónová sieť

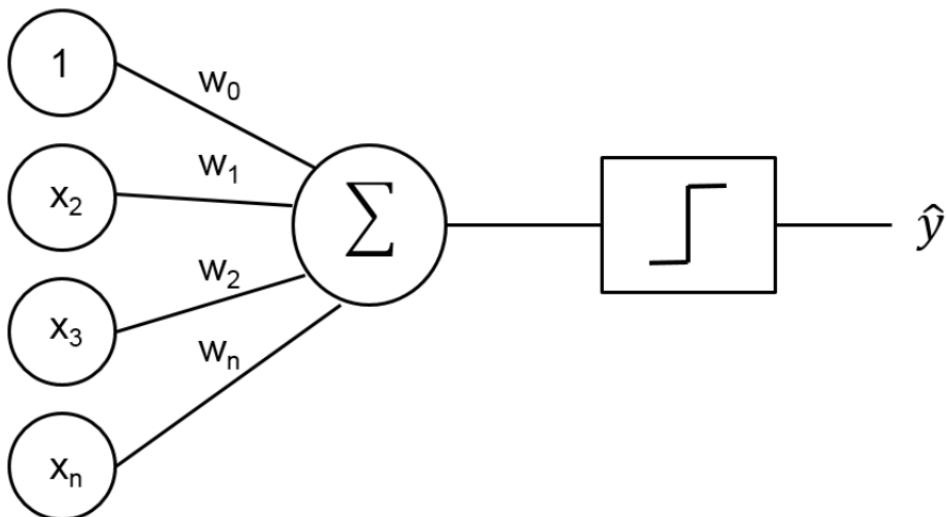
Neurónová sieť je jeden z modelov používaných v oblasti umelej inteligencie. Jej vzorom je správanie sa zodpovedajúcich biologických štruktúr – mozog človeka. Je to štruktúra určená pre distribuované paralelné spracovanie dát. Skladá sa z umelých (formálnych) neurónov, ktoré simulujú reálne biologické neuróny. Navzájom sú prepojené a prenášajú medzi sebou signály pomocou určitých prenosových funkcií. Neurón môže mať ľubovoľný počet vstupov ale iba jeden výstup. Neurónové siete sa používajú pre rozpoznávanie a aj kompresiu obrazu alebo zvukov, predpovedanie časových radoch a dokonca aj k filtrovaniu spamu. Využívajú sa tiež v oblasti medicíny, k skúmaniu fungovania nervových sústav živých organizmov. Napríklad perceptronová sieť vznikla ako simulácia fyziologickeho modelu rozpoznávania vzorov na sietniči ľudského oka [11].

Bootstrap aggregating

Bootstrap aggregating alebo bagging je množinový meta-algoritmus (angl. Ensemble meta-algorithm). Tento algoritmus nie je klasický algoritmus, ale klasické algoritmy spája a tým dosahuje lepšie výsledky. Najčastejšie sa používajú rozhodovacie stromy. Algoritmus si najprv vyžiada od každého podriadeného algoritmu predikciu. Každý z výsledkov má rovnakú váhu. Následne sa všetky predikcie spriemerujú a vytvorí sa jediná, ktorá je najpresnejšia zo všetkých.

2.5.3. Neurónové siete

Umelá neurónová sieť predstavuje sieť mnohých navzájom prepojených jednoduchých procesorov. Graf prepojení je zvyčajne označovaný ako topológia siete. Procesory sa potom nazývajú neuróny, pretože zjednodušene modelujú skutočné neuróny v ľudskej centrálnej nervovej sústave. Základom matematického modelu umelej neurónovej siete je formálny neurón označovaný ako *perceptron*. Perceptron rovnako ako biologický neurón má vstupy (v biológii nazývané dendrity). Na obrázku 6 je zobrazený perceptron [11].



Obr. 6: Perceptrón

Vstupy sú ohodnotené zodpovedajúcimi všeobecne reálnymi tzv. synaptickými váhami $w_0, w_1, w_2, \dots, w_n$, ktoré určujú priepustnosť vstupov. Nelineárny nárast výstupnej hodnoty y pri dosiahnutí prahovej hodnoty potenciálu h je daný *aktivačnou* funkciou.

Najjednoduchším typom aktivačnej funkcie je tzv. ostrá nelinearita (skoková funkcia), ktorá patrí do skupiny sigmoidných aktivačných funkcií. Aktivačná funkcia môže byť skoková (ostrá nelinearita), spojitá lineárne, štandardné (logistická) sigmoid alebo hyperbolický tangens [12].

Učenie neurónovej siete

Umelé neurónové siete majú dve pracovné fázy. Ide o fazu adaptívnu, v ktorej sa sieť učí, a fazu aktívnu, v ktorej vykonáva naučenú činnosť. Pri učení neurónovej siete dochádza k adaptácii siete na riešenie určitého problému. Učenie neurónovej siete je zvyčajne realizované

nastavovaním váh vstupov jednotlivých neurónov. Priamy výpočet synaptických váh neurónov je nevhodný a vo väčšine prípadov neuskutočniteľný, a preto nastupuje proces učenia. Rozlišujú sa dva typy učenia neurónových sietí. Jedná sa o tzv. učenie s učiteľom a učenia bez učiteľa [22].

Pri učení s učiteľom existuje nejaké vonkajšie kritérium, ktoré určuje, ktorý výstup je správny. V praxi sa to rieši tak, že je k dispozícii sada trénovacích príkladov, ku ktorým je známe riešenie. Tieto príklady sa postupne predkladajú na vstup neurónovej siete a porovnáva sa požadovaná odozva siete so skutočnou. Na základe odlišnosti skutočného výstupu od očakávaného sa potom pomocou spätej väzby upraví v neurónové sieti váhy. Metodiku úpravy váh určuje učiaci algoritmus. Je dokázané, že po vykonaní veľkého počtu učiacich sa krokov sa siet naučí poskytovať stabilný výstup ako reakciu na naučené vstupy. [11]

Pri type učenia bez učiteľa neexistuje žiadne vonkajšie kritérium správnosti. Algoritmus učenia je navrhnutý tak, že hľadá vo vstupných dátach určité vzorky sa spoločnými vlastnosťami. Tomuto typu učenia sa hovorí samoorganizácia. Do učenia bez učiteľa nie je zapojený žiadny vonkajší atribút a celé učenie je založené len na informáciách, ktoré samotná siet počas celého procesu získala. Učiaci proces môže prebiehať opakovane alebo jedno-rázovo. Neurónové siete sa môžu učiť nielen zmenou synaptických váh, ale aj prispôsobovaním prenosovej funkcie, počtu neurónov, prípadne aj zmenou topológie siete.[12]

Backpropagation – spätné učenie

Učiaci algoritmus Back-propagation je v podstate optimalizačný algoritmus, ktorý je schopný nájsť pre danú neurónovou siet a trénovaciu množinu váhové koeficienty a prahy. Pretože neuróny môžu mať rôznu štruktúru a rôzne výstupné funkcie, je treba najprv špecifikovať konkrétny typ siete, pre ktorú budú rovnice odvodené. Podstata algoritmu Back-propagation spočíva v hľadaní minima funkcie chyby E . [13, 24]

Chyba siete $E(w)$ je vzhľadom na tréningovú množinu definovaná ako súčet parciálnych chýb siete $E_l(w)$ vzhľadom k jednotlivým tréningovým vzorom a závisí od konfigurácie siete w :

$$E(\mathbf{w}) = \sum_{l=1}^q E_l(\mathbf{w}).$$

Parciálna chyba $E_l(w)$ siete je úmerná súčtu mocnín odchýlok skutočných hodnôt výstupu siete pre vstup l -tréningového vzoru od požadovaných hodnôt výstupov u tohto vzoru:

$$E_l(\mathbf{w}) = \frac{1}{2} \sum_{k \in Y} (y_k - t_k)^2$$

Cieľom adaptácie je minimalizácia chyby siete vo váhovom priestore. Vzhľadom k tomu, že chyba siete priamo závisí od komplikovanej nelineárnej zloženej funkciu viacvrstvovéej siete, predstavuje tento cieľ netriviálny optimalizačný problém. Pre jeho riešenie sa v základnom

modeli používa najjednoduchší variant - *gradientna metóda*, ktorá vyžaduje diferencovateľnosť chybovej funkcie. Chybová funkcia určuje chybu siete vzhľadom k pevnej tréningovej množine v závislosti na konfiguráciu siete. Pri adaptácii siete hľadáme takú konfiguráciu, pre ktorou je chybová funkcia minimálna [12, 24].

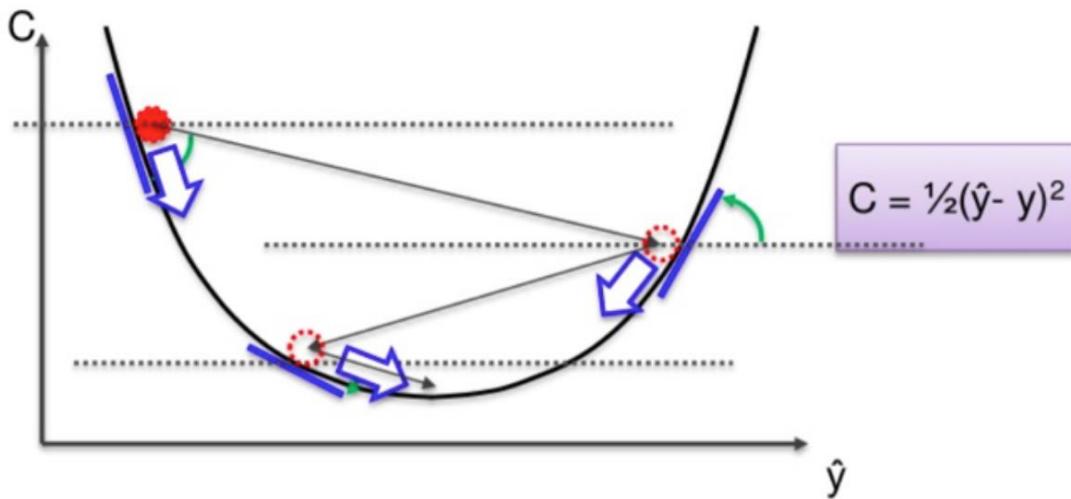
Gradientná metóda – gradient decent

Začneme s náhodne zvolenou konfiguráciou $w^{(0)}$, zodpovedajúca chyba siete od požadovanej funkcie bude pravdepodobne veľká. Pri adaptácii zostrojíme v tomto bode $w^{(0)}$ ku grafu chybovej funkcie točný vektor:

$$(\text{gradient}) \quad \frac{\partial E}{\partial \mathbf{w}}(\mathbf{w}^{(0)})$$

a posunieme sa v smere tohto vektora dole o ϵ . Pre dostatočne malé ϵ tak získame novú konfiguráciu $w^{(1)} = w^{(0)} + \Delta w^{(1)}$, pre ktorú je chybová funkcia menšia ako pre pôvodnú konfiguráciu $w^{(0)}$, tj. $E(w^{(0)}) \geq E(w^{(1)})$. Celý proces konštrukcie dotykového vektora opakujeme pre $w^{(1)}$ a získame tak $w^{(2)}$ také, že $E(w^{(0)}) \geq E(w^{(1)}) \geq E(w^{(2)})$ atď., až sa limitne dostaneme do lokálneho minima chybovej funkcie [25].

Hlavným problémom gradientnej metódy je, že ak už nájde lokálne minimum, potom toto minimum nemusí byť globálne. Uvedený postup adaptácie sa v takom minime zastaví (nulový gradient) a chyba siete sa už ďalej neznižuje. Na obrázku 7 je grafická reprezentácia tejto metódy [11, 25].

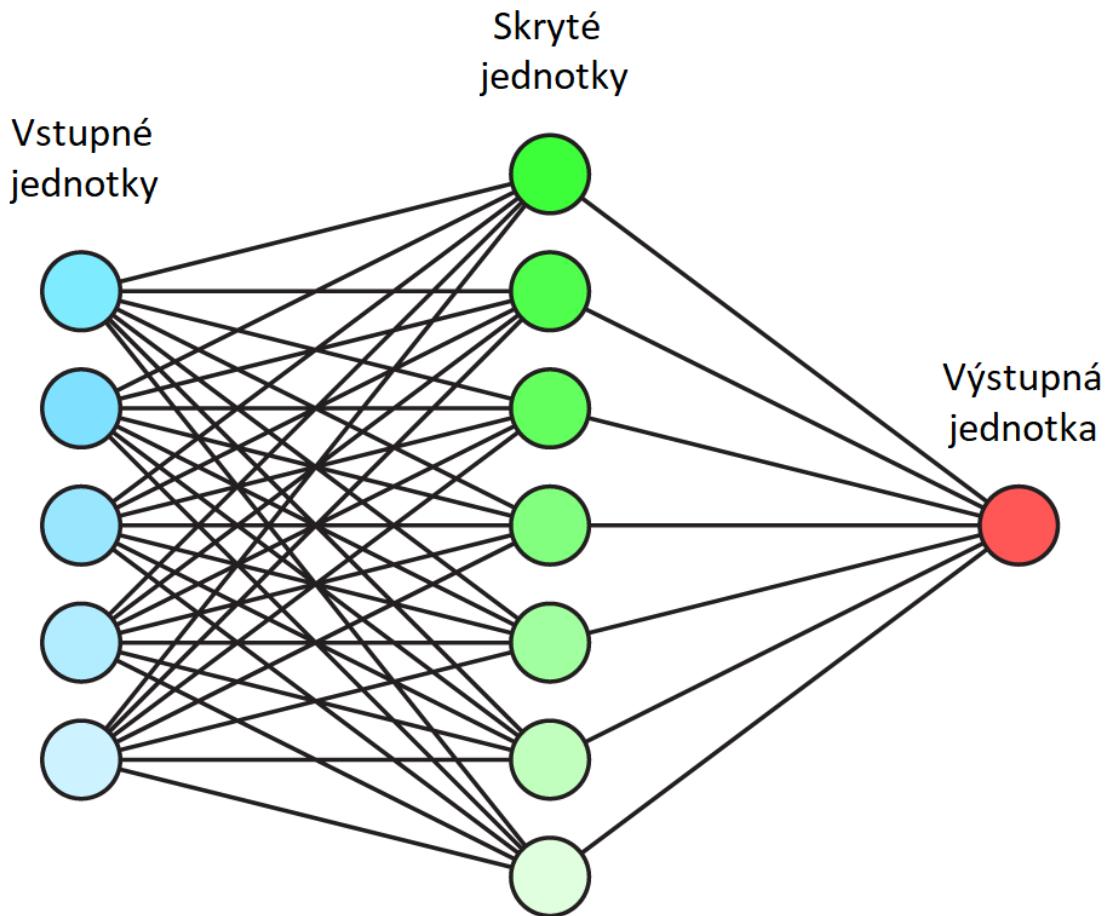


Obr. 7: Gradient decent [25]

Viacvrstvové siete

Medzi najrozšírenejšie spôsoby prepojenia spojitych perceptronov možno zaradiť tzv. vrstvené (viacvrstvové) neurónové siete, kde je neurónová sieť tvorená minimálne troma vrstvami neurónov. Jedná sa o vstupnú vrstvu, vnútorná (skrytú) vrstvu a výstupnú vrstvu. Medzi susednými dvoma vrstvami sa vždy nachádza tzv. úplné prepojenie neurónov, teda každý neurón nižšej vrstvy je prepojený so všetkými neuróny z vrstvy vyššej [25]. Vo vrstvenej perceptronovej sieti prebieha dopredné šírenie signálu (feedforward) týmto spôsobom:

- Neuróny vstupnej vrstvy sú excitované na zodpovedajúcu úroveň (v rozsahu 0 až 1).
- Excitácie sú potom pomocou väzieb privodené k nasledujúcej vrstve a upravené (zosilnené alebo zoslabené) pomocou synaptických váh.
- Každý neurón vyšej vrstvy vykoná sumár upravených signálov od neurónov nižšej vrstvy a je excitovaný na úroveň danú svojou aktivačnou funkciou.
- Tento proces prebieha cez všetky vnútorné vrstvy až k vrstve výstupnej, kde sa získajú excitačné stavy všetkých neurónov [14].



Obr. 8: 3-vrstvová neurónová sieť

Veľkým problémom modelu viacvrstvovej neurónovej siete s adaptačným algoritmom backpropagation je (okrem minimalizácie chybovej funkcie) voľba vhodnej topológie pre riešenie konkrétneho praktického problému. Zriedkakedy sú podrobnejšie známe vzťahy medzi vstupmi a výstupmi, ktoré by sa dali využiť pri návrhu špeciálnej architektúry. Väčšinou sa používa viacvrstvová topológia s jednou alebo dvoma vnútornými vrstvami a očakáva sa, že učiaci algoritmus backpropagation zovšeobecnení príslušné vzťahy z tréningovej množiny vo váhach jednotlivých spojov medzi neurónmi [15, 24].

Aj v tomto prípade je však potrebné vhodne voliť počty neurónov vo vnútorných vrstvách. Je zrejmé, že tento problém organizačnej dynamiky úzko súvisí s adaptáciou a generalizáciou neurónovej siete. Architektúra viacvrstvovej neurónovej siete (tj. určenie vhodného počtu vnútorných neurónov a ich spojenie), by mala zodpovedať zložitosti riešeného problému, teda počtu tréningových vzorov, ich vstupov a výstupov a štruktúre vzťahov, ktoré popisujú. Je zrejmé, že malá siet nemôže riešiť komplikovaný problém. Pri učení pomocou algoritmu backpropagation sa príliš malá siet zvyčajne zastaví v nejakom plynkom lokálnom minime a je potreba topológiu doplniť o ďalšie vnútorné neuróny, aby adaptácia mala väčší stupeň voľnosti. Na druhú stranu bohatá architektúra sice pri učení mnohokrát umožní nájsť globálne minimum chybovej funkcie, aj keď s väčším počtom váh rastie výpočtová náročnosť adaptácie. Avšak nájdená konfigurácia siete zvyčajne príliš zovšeobecňuje tréningové vzory vrátane ich nepresností a chýb a pre nenaучené vzory dáva chybné výsledky, tj. zle generalizuje [14].

Tomuto presnému zapamätaniu tréningovej množiny bez zovšeobecnenia zákonitostí v nej obsiahnutých, sa hovorí *overfitting* (Overfitting)[16]. Zdá sa teda, že existuje optimálna topológia, ktorá je na jednu stranu dostatočne bohatá, aby bola schopná riešiť daný problém, a na druhú stranu nie moc veľká, aby správne zovšeobecnila potrebné vzťahy medzi vstupmi a výstupmi. Existujú sice teoretické výsledky ohľadom horného odhadu počtu vnútorných neurónov postačujúcich pre realizáciu ľubovoľnej funkcie z určitej triedy, avšak pre praktické potreby sú príliš nadhodnotené, a teda nepoužiteľné [16, 25].

V praxi sa obvykle topológia volí heuristicky, napr. v prvej vnútorej vrstve o niečo viac neurónov, ako je vstupov a v druhej vrstve aritmetický priemer medzi počtom výstupov a neurónov v prvej vnútorej vrstve. Po adaptácii sa v prípade veľkej chyby siete prípadne pridá, respektíve pri chudobnej generalizácii odoberie niekol'ko neurónov a adaptívne režim sa celý opakuje pre novú architektúru. Pre test kvality generalizácia neurónovej siete sa počítá chyba siete vzhľadom k tzv. testovacie množine, čo je časť tréningovej množiny, ktorá sa zámerne nevyužila k adaptácii [16, 25].

2.5.4. OpenNN

OpenNN (Open Neural Network Library) je softvérová knižnica pre jazyk C++, ktorá implementuje neurónové siete v oblasti výskumu hlbokého učenia. OpenNN implementuje balík funkcií, ktoré môžu byť vložené do vyvíjaných softvérových nástrojov pomocou API. Hlavnou výhodou OpenNN je veľmi dobrý výkon. Kedže je knižnica je vyvíjaná v jazyku C++,

poskytuje lepšiu výpočtovú rýchlosť a lepšiu správu pamäte. Taktiež implementuje paralelizáciu na CPU pomocou OpenMP a zrýchlenie na GPU pomocou CUDA [20].

2.5.5. Genann

Genann je knižnica pre jazyk C, ktorá implementuje vytváranie a správu neurónových sietí typu MLP. Vďaka tomu, že je implementovaná v ANSI verzii jazyka C, dá sa jednoducho skompilovať pomocou štandardných kompilátorov ako GCC alebo Clang. Je šírená pod voľnou licenciou a teda ju môžeme využiť aj v tomto projekte. Na rozdiel od OpenNN je knižnica Genann veľmi malá a jednoduchá. Vďaka tomu je ale vytváranie malých a stredne veľkých sietí rýchle a efektívne. Knižnica umožňuje okrem vytvárania a testovania sietí aj ukladanie natrénovaného modelu siete do súboru [19].

2.6. Vyhodnotenie analýzy

Metódy neuromonitoringu sa nevyužívajú na všetkých neurologických a chirurgických pracoviskách. V súčasnosti je na trhu niekoľko prístrojov, pomocou ktorých je možné merať, vyhodnocovať a zaznamenávať funkciu nervových štruktúr. Práca s týmito prístrojmi je však zložitá, nie všetky majú dostatočne kvalitné funkcie, práca s nimi je časovo náročná a v nesplodenom rade sú finančne veľmi nákladne. Popísane dôvody môžu odradiť lekárov v praxi od ich zakúpenia a používania, z dôvodu čoho sa v dnešnej dobe výkony, pri ktorých je potrebné hodnotiť funkciu nervových štruktúr centralizujú. Je želané, aby boli metódy vysvetrenia funkcie nervových štruktúr rozšírenejšie a efektívnejšie. Z opisaných dôvodov sme sa rozhodli vytvoriť nové riešenie. Nami vytvorené riešenie sme porovnali s existujúcim riešením. išlo o prístroj IOMED. Po porovnaní funkcií a jednotlivých možností prace s oboma prístrojmi lekárom z praxe, považujeme naše riešenie za prínosne vo viacerých ohľadoch.

Pristroj IOMED, s ktorým sme naše riešenie porovnávali, a ktorý sa pravidelne používa počas operácií neposkytuje o vytvorenom zázname lekárovi bližšie informácie. Takéto prezeranie záznamu trvá aj niekoľko minút a lekár musí vedieť manuálne rozlíšiť jednotlivé vlny od artefaktov.

V našom riešení preto využijeme metódy strojového učenia, ktoré dokážu rýchlo a efektívne napomôcť lekárovi pri analýze záznamu. Vďaka implementácii neurónových sietí do aplikácie dokážeme zaujímavé miesta v zázname extrahovať a rýchlo ich zobraziť lekárovi, prípadne pripraviť na archiváciu v databáze.

3. Opis riešenia

V tejto kapitole sa budeme venovať opisu riešenia, špecifikácií požiadaviek, ktoré musí riešenie spĺňať, opisu implementácie a overeniu riešenia.

Cieľom nášho riešenia má byť vytvorenie funkčnej aplikácie, ktorá na vstupe dostane dátu z neuromonitorovacieho zariadenia a na výstupe ponúkne lekárovi čo najobsiahlejšiu analýzu dát a ich prehľadné zosumarizovanie.

3.1. Špecifikácia požiadaviek

Funkcionálne požiadavky

- Do aplikácie bude vstupovať záznam merania funkčnosti nervov pacienta vo forme súboru alebo prúdu dát.
- Riešenie bude jedna aplikácia, bez grafického používateľského rozhrania, obsahovať bude 2 moduly pre analýzu signálu.
 - Prvý bude staticky analyzovať prúd dát a hľadať pravdepodobné miesta, kde sa nachádzajú impulzy.
 - Druhý modul bude pomocou klasifikátora analyzovať tvar krivky a vyhodnocovať či impulz zodpovedá očakávanému tvaru.
- Výstupom aplikácie budú snímky miest zaujímavých pre lekára – teda snímky jednotlivých vln a artefaktov. Snímky budú okrem samotných dát obsahovať ďalšie metadáta ako názov vlny, maximálna hodnota vo vrchole vlny, percento podobnosti s ideálnym tvarom vlny a čas, kedy bola vlna zachytená.

Nefunkcionálne požiadavky

- Multiplatformovosť – Program bude nutné spustiť na viacerých platformách.
- Predpokladá sa, že po nasadení aplikácie do praxe, bude spustená na stroji s dostatom operačnej pamäte. Naopak ale bude potrebné čo najviac znížiť potrebný výkon na výpočet, aby sa zjednodušilo online analyzovanie dát
- Vzhľadom na to, že táto práca je súčasťou väčšieho projektu, je nutné použiť rovnaké technológie (jazyk a frameworky) ako sa používajú aj v ostatných častiach projektu.

3.2. Návrh riešenia

Táto kapitola sa venuje popisu návrhu riešenia projektu diplomovej práce.

3.2.1. Formát dát

Dáta k aplikácii použijeme tie, ktoré vytvorí neuromonitorovacie zariadenie v nemocnici. Keďže k existujúcemu riešeniu nie je dostupná dokumentácia, dáta sme analyzovali len manuálne. Uložené sú v textovom súbore. Každý riadok obsahuje číslo, ktoré znázorňuje hodnotu na Y osi grafu. Každý riadok, zase znamená posun v čase na osi X, pričom časové intervale medzi jednotlivými riadkami sú rádovo v milisekundách.

3.2.2. Platforma

Aplikáciu vytvoríme tak, aby ju bolo možné spustiť na ľubovoľnom operačnom systéme. Pri vývoji ale budeme preferovať operačný systém LUNIX. Samotná aplikácia bude písaná v jazyku C / C++ a využijeme framework Qt pre vytvorenie jednoduchého a prehľadného používateľského rozhrania.

Pre vytvorenie neurónovej siete, ktorá bude klasifikovať jednotlivé impulzy sme použili opensource framework OpenNN. Tento framework je určený pre jazyk C++ a dá sa ľahko integrovať s frameworkom Qt.

3.2.3. Architektúra aplikácie

Zo špecifikácie požiadaviek vyplýva, že naše riešenie musí poskytovať dve základné funkcie:

1. Lokalizáciu

Tento modul bude mať za úlohu vyhľadať a lokalizovať podľa vopred vybraného vzoru namerané evokované potenciály v dátach a zvýrazniť ich lekárovi aby ich nemusel manuálne hľadať. Na vstupe tohto modulu bude samotný záznam merania, číslo vyjadrujúce počet stimulov, koľko ich lekár urobil a koľko ich je potrebné nájsť. Nepovinný údaj bude aj vzor krivky pre nerv, na ktorom sa meranie vykonávalo. Výstup bude zvýraznená časť dát s hľadanými krivkami grafu.

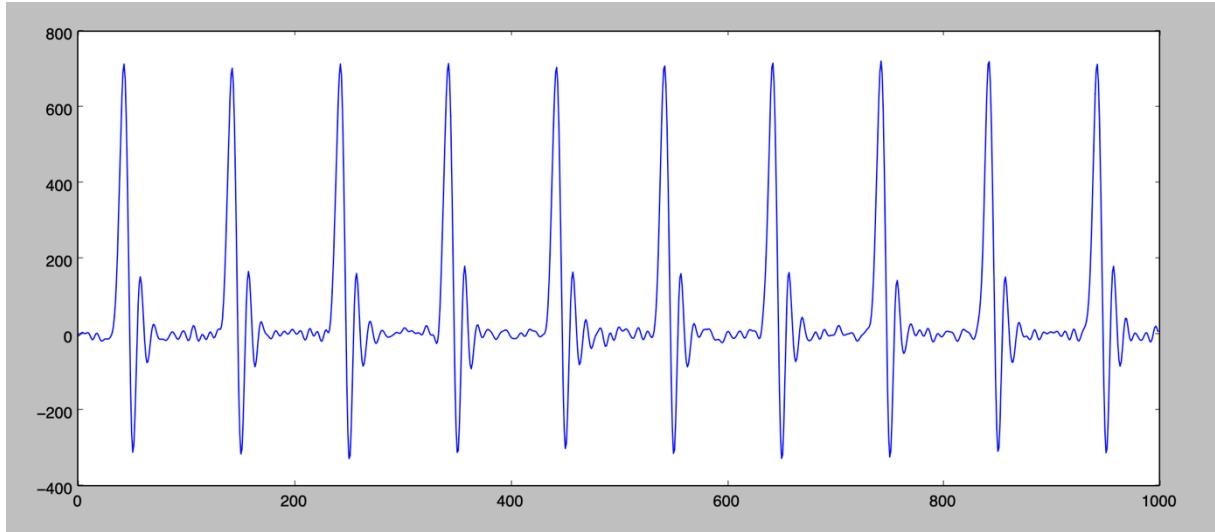
2. Analýzu

Tento modul bude vyhodnocovať základné údaje o jednotlivých nameraných a lokalizovaných stimuloch. Na vstupe modulu bude záznam, pozícia stimuli (z lokalizačného modulu), vzor pre meraný nerv, vek a ďalšie potrebné informácie o pacientovi, ktoré môžu mať vplyv na záznam neuromonitoringu. Na výstupe bude lekárovi poskytnutá informácia o každom stímule – výška, šírka, tvar (percento podobnosti so vzorom) a tiež informácie o celom meraní – priemerné namerané hodnoty stimulov, minimum a maximum.

3.2.4. Generátor signálu

Vzhľadom na ochranu osobných údajov pacientov nie je možné získať všetky potrebné dáta, ktoré by mohli byť použité na analýzu nameraných signálov. Preto sme sa rozhodli vytvoriť vlastný generátor signálov, ktorým dokážeme simulovať meranie evokovaných potenciálov.

Generátor je napísaný v jazyku Python a dokáže generovať časť simulovaného merania, ktoré obsahuje impulzy a aj artefakty. Na obrázku číslo 6 je zobrazený príklad vygenerovaného merania.



Obr. 10: Príklad vygenerovaných trénovacích dát

3.2.5. Vyhľadávanie vzorov v dátach

Aplikácia bude okrem informácií o nameraných dátach tiež poskytovať vyhľadanie a lokalizáciu jednotlivých evokovaných potenciálov. Aby sme toto dosiahli použijeme natrénovaný klasifikátor založený na neurónovej sieti.

3.2.5.1. Trénovanie klasifikátoru

Pred začiatkom analyzovania nameraného signálu, je potrebné natrénovať klasifikátor, ktorý bude vyhodnocovať tvar kriviek zachytených signálov. Máme vytvorený dataset kriviek impulzov, ktoré chceme neskôr vyhodnocovať.

3.2.5.2. Analýza nameraného signálu

Analýza získaného signálu bude prebiehať v dvoch krokoch. V prvom sa vyhľadajú miesta, kde sa predpokladá nameraný impulz. V druhom kroku sa tento predpoklad overí a v prípade potvrdenia, sa nameraný impulz zmeria.

Statické vyhľadanie lokálneho maxima

Záznam sa prehľadáva sekvenčne a zaznamenáva sa aktuálna lokálna priemerná hodnota v danom úseku merania. V prípade, že sa táto hodnota náhle zmení, je pravdepodobné, že v danom úseku merania nastal začiatok impulzu. Ak sa znova priemerná lokálna hodnota

merania stabilizuje, je pravdepodobné, že úsek so zachyteným impulzom sa skončil. Program teda daný úsek, kde predpokladáme impulz zachytí a posunie druhému modulu na ďalšie analyzovanie.

Klasifikácia zachyteného úseku

Modul aplikácie, ktorý zabezpečuje analýzu tvaru krivky jej klasifikáciou, načíta zachytený úsek dát z predchádzajúceho modelu a vyhodnotí s akou pravdepodobnosťou sa na danom úseku nachádza impulz správneho tvaru. Informácia o jeho správnosti je potrebná pri neskoršom zobrazení, keďže impulzy správneho tvaru sa budú odlišovať od artefaktov a impulzov nesprávneho tvaru.

Následne sa zmerajú amplitúdy daného impulzu a jeho celková šírka. Dané informácie sa pripravia na zobrazenie.

3.3. Zmeny v návrhu riešenia

V tejto kapitole sú popísané zmeny v návrhu riešenia a rôzne ďalšie rozšírenia pôvodného návrhu, ktoré sme sa rozhodli počas fázy implementácie urobiť.

3.3.1. Rozšírenie architektúry aplikácie

V pôvodnom návrhu sme predpokladali použitie aplikácie len na „offline“ analýzu jedného záznamu merania. To znamená, že po vykonaní merania, ktoré by bolo uložené v súbore, by lekár spustil analýzu.

Tento návrh sme pozmenili tak, aby bolo možné analyzovanie spustiť okrem takého spôsobu aj „online“ a teda priamo počas výkonu vyšetrenia pacienta. Aplikácia teda nepotrebuje žiadne informácie o počet vln v meraní, stačí len zadat typy vln, ktoré má hľadat. Takisto nie nijak obmedzená dĺžka merania.

Návrh sme tiež doplnili o možnosť paralelnej analýzy viacerých meraní. Ak teda lekár počas vyšetrenia bude merať funkčnosť viacerých svalov pacienta, všetky budú môcť byť analyzované naraz.

3.3.2. Rozšírenie generátoru dát

Takisto sme prispôsobili aj generátor dát tak, aby splňal požiadavky na simuláciu „živého mernaia“. Dáta sú generované v reálnom čase a používateľ môže generátor ovládať niekoľkými jednoduchými príkazmi. V jednom meraní tiež môže byť generovaných viacero typov vln a artefaktov.

3.4. Implementácia

Aplikácia sa skladá z dvoch častí, ktoré navzájom spolu komunikujú pričom obidve sú spustené na rovnakom stroji. Prvá hlavná časť (Signal Analyzer) je zodpovedná za správu niekoľkých paralelných meraní neurosignálov a ich analýzu. Druhá časť generuje dátá a tým simuluje funkčnosť reálnych meracích prístrojov.

3.4.1. Analyzátor signálov

V tejto kapitole je opísaná časť aplikácie, ktorá je zodpovedná za zbieranie údajov z externých meracích prístrojov a tieto namerané dátá analyzuje a vyhodnocuje. Výstupom sú snímky zaujímavých častí jednotlivých meraní.

Pri vytváraní tohto programu sme použili framework Qt, ktorý je založený na programovacom jazyku C++. Jedným z cieľov tohto frameworku je sprístupniť jazyk C++ na čo najviac platforem a tiež zjednodušiť presun aplikácie medzi nimi. Aj keď sme aplikáciu vyvíjali na platforme Macos, je ľahko spustiteľná aj na Windows a Linux.

3.4.1.1. Inicializácia

Aplikácia k svojmu správnemu fungovaniu potrebuje mať v počítači vytvorený špeciálny konfiguračný priečinok s názvom *data*. Tento priečinok je používaný pre uloženie rôznych nastavení programu ale aj pre ukladanie vytvorených snímok meraní. Ďalšia štruktúra tohto špeciálneho priečinka je nasledovná:

- *ann* – priečinok, kde sú uložené súbory obsahujúce vytvorené neurónové siete. Tieto súbory majú názov podľa typu vlny, ktorý daná sieť rozpoznáva a príponu *ann*. Príklad: *wave1.ann*.
- *bluetoothData* – priečinok, ktorý obsahuje textové súbory predstavujúce jednotlivé externé zariadenia snímajúce pacienta. Avšak reálne sú tieto súbory napĺňané generátorom signálov. Dáta z nich sú čítané bluetooth prijímačom a ďalej sa s nimi pracuje akoby pochádzali z reálneho vyšetrenia pacienta. Predpokladá sa, že pri ďalšej práci bude tento priečinok odstránený.
- *config* – priečinok, ktorý obsahuje konfiguračné súbory jednotlivých typov vln, s ktorými sa bude pracovať. Súbory majú názov podľa typu vlny, ktorý predstavujú a príponu *xml*.
- *export* – priečinok, ktorý obsahuje jednotlivé snímky vyexportované aplikáciou. Snímky sú *xml* súbory obsahujúce metadáta každej snímky aj sekvenciu dát, ktorá zachytáva vlnu alebo artefakt.
- *trainData* – priečinok obsahujúci súbory využívané pri trénovaní, testovaní a validácii neurónových sietí jednotlivých typov vln. Jedná sa o CSV súbory obsahujúce snímky vln a artefaktov, ktoré majú názov *train_*, *test_*, *val_* a názov vlny. Príklad: *train_wave1.csv*.

Cesta k hlavnému konfiguračnému priečinku musí byť v programe nastavená do konštanty DATA_PATH. Po správnom nastavení a skompilovaní, je aplikácia pripravená na spustenie.

3.4.1.2. Typy vln a artefakty

Hlavná myšlienka aplikácie je v rozpoznávaní viacerých typov vln v dlhom meraní. Veľkú časť merania tvorí len šum, ktorý nie je ničím pre lekára zaujímavý. Aj napriek tomu sa v pri súčasných riešeniach musí lekár prepracovať cez šum k zaujímavým úsekom merania, ktoré obsahujú vlny. Jednotlivé vlny predstavujú neurofyziologické impulzy nervov. Okrem toho sa v meraní môžu vyskytovať artefakty, čo sú úseky poškodeného merania vyvolané vonkajšími okolnostami. Artefakty môžu mať tvar deformovanej vlny a lekárovi zneprijemňujú prácu. je preto vhodné vedieť ich odlišiť od skutočných vln.

Ked'že rôzne typy vln sa od seba odlišujú počtom vrcholov, dĺžkou a tvarom, vytvorili sme spôsob ako ich zadefinovať pre digitálne spracovanie. Každý typ vlny má vytvorený svoj konfiguračný XML súbor. Obsahuje niekoľko elementov, ktoré popisujú vlnu a neurónovú sieť určenú pre rozpoznávanie danej vlny. Ak sa zachytí úsek merania, ktorého hodnoty naznačujú, že sa môže jednať o vlnu ale jeho tvar sa nezhoduje zo žiadnym typom vlny, ktorý je v meraní nastavený na rozpoznávanie, vyhodnotí sa ako artefakt.

Konfiguračné súbory vln sa nachádzajú v priečinku *data/config*. Pri použití v aplikácii je potrebné len zadať názov súbor a ten sa načíta z tejto adresy. Súbor obsahuje jeden koreňový element root. Ten obsahuje už konkrétné elementy pre vlnu:

- name – element obsahuje textovú informáciu o názve typu vlny.
- threshold – číselná informácia, ktorá môže byť kladná aj záporná. Ak hodnota nejakého bodu merania prekročí túto hranicu môže sa jednať o vlnu a preto okolie tohto bodu bude vyseknuté a skontrolované neurónovou sieťou. Ak nie je rozpoznaná žiadna vlna, úsek sa posúdi ako artefakt.
- length – kladný celo-číselný údaj hovoriaci o dĺžke priebehu danej vlny. Zároveň sa jedná o počet neurónov vo vstupnej vrstve neurónovej siete.
- th_cross – Bod v priebehu vlny, v ktorom sa očakáva prechod hodnôt údajov cez hranicu threshold.
- ann_file – textová informácia predstavujúca názov súboru s príponou *ann*, v ktorom bude uložená neurónová sieť. Súbor s týmto názvom sa bude hľadať na adrese *data/ann*.
- ann_topology – pole štyroch celých čísel, oddelených bodkočiarkou. Tieto čísla predstavujú topológiu neurónovej siete, ktorá sa použije pri rozpoznávaní danej vlny. Jednotlivé čísla predstavujú:
 1. Prvé číslo – počet vstupnej vrstvy neurónovej siete
 2. Druhé číslo – počet skrytých vrstiev siete
 3. Tretie číslo – počet neurónov v skrytých vrstvách siete
 4. Štvrté číslo – počet neurónov výstupnej vrstvy

3.4.1.3. Bluetooth prijímač dát

Dáta, ktoré aplikácia analyzuje sú získavané z externých zariadení pripojených cez bluetooth. Tieto externé zariadenia vykonávajú meranie na pacientovi a posielajú namerané dáta cez

bluetooth komunikáciu do stroja, na ktorom je spustený analyzátor. Keďže táto časť bola predmetom inej diplomovej práce, v tejto práci sme vytvorili spôsob ako tieto zariadenia simulovať.

Meranie simulujeme generátorom signálu, ktorý dátu zapisuje do textového súboru. Bluetooth prijímač tento súbor číta a získava z neho dátu. Ďalej sa už ale s každým údajom pracuje rovnako, ako keby bol prijatý z reálneho merača. Trieda, ktorá obsahuje implementáciu bluetooth prijímača zahŕňa aj metódy, ktoré zodpovedajú za správne pridávanie dát do analyzátoru. Bluetooth prijímač pri inicializácii získa objekty jednotlivých meraní, ktoré v sebe držia všetky potrebné premenné a polia k analýze merania. Pri ďalšej práci bude nutné bluetooth prijímač doplniť o správu reálnych pripojených zariadení, avšak použijú sa už implementované metódy určené na pridávanie dát do jednotlivých meraní.

Prijímanie dát

Aktuálne bluetooth prijímač otvorí všetky dátové súbory, ktoré pri inicializácii boli do neho pridané a naparované na všetky merania. V cykle tieto súbory prechádza a z každého načíta jeden riadok. Ak sa riadok nie je prázdný, znamená to, že do súboru pribudol nový údaj a je spracovaný a pridaný do bufferu príslušného merania.

Samotné pridanie údaju do bufferu si vyžaduje smerník na buffer, veľkosť bufferu, adresu poslednej zapísanej pozície v bufferi (posledná aktualizácia bufferu) a samotnú hodnotu, ktorá má byť zapísaná. K smerníku na začiatok bufferu sa pripočíta číslo indexu poslednej aktualizácie a tým dostaneme adresu, kam sa má nová hodnota zapísat. Tiež sa aktualizuje index poslednej aktualizácie bufferu.

Ukončenie merania

Ak hodnota údaju načitaná z dátového súboru je vyššia ako 10 000, je to informácia o ukončení merania. Táto hodnota sa už nepridá do bufferu merania ale nastaví sa pre dané meranie príznak *end* na hodnotu 1. Týmto je meranie ukončené. Rovnaký spôsob ukončenia sa použije aj pri zmene implementácie bluetooth prijímača.

3.4.1.4. Dátové štruktúry v meraní

V aplikácii je analýza údajov merania vykonávaná v objekte merania triedy *Measurement*. Pre každé meranie je vytvorený takýto objekt, ktorý sa stará o správne fungovanie analýzy a nijak nekomunikuje ani nezasahuje do iných paralelných meraní.

Kruhový buffer

Najdôležitejšou dátovou štruktúrou, ktorú objekt merania obsahuje je kruhový buffer. Tento buffer obsahuje dátu merania určené na analýzu. Údaje do neho pridáva bluetooth prijímač, ktorý tiež nastavuje hodnotu *gpt* – čo je pozícia posledného pridaného údaju. Samotný analyzátor merania z kruhového bufferu dátu len číta. Udržuje si adresu *actPointer* čo je adresa

posledného prečítaného (analyzovaného) bodu v bufferi. Táto hodnota nesmie predbehnúť *gpt* ale môže byť menšia.

Zoznam pravdepodobných vrcholov

Ak pri analýze nejakého bodu je splnená podmienka bodu *th_cross*, je adresa tohto bodu pridaná do zoznamu pravdepodobných vrcholov. Zoznam sa udržuje spôsobom FIFO. To, či bod je skutočne bodom vlny alebo iba bodom v artefakte, určí až ďalšia analýza pomocou neurónovej siete. Táto analýza sa spúšťa vždy pre prvý bod zoznamu, ktorého okolie v hlavnom bufferi sa vysekne. Po analýze sa bod zo zoznamu odstráni.

Zoznam rozpoznávaných vĺn

Každé meranie obsahuje zoznam vĺn, ktoré pri analýze rozpoznáva. Vlna je definovaná svojím objektom, ktorý načíta pri inicializácii dát z konfiguračného súboru. Neskôr pri nastavovaní merania sú tieto objekty do neho priradené. Jeden typ vlny môže byť prítomný vo viacerých meraniach. Jedno meranie, nemôže obsahovať dva objekty s rovnakým typom vlny.

3.4.1.5. Analýza signálov

Analýza signálu prebieha samostatne pre každé meranie, nezávisle na ostatných. Každé meranie sa analyzuje postupne pre každý jeho bod. Pre tento údaj, ktorý sa prečíta z bufferu merania na adresе *actPointer*, sa vykonáva analýza. Tá sa skladá z dvoch základných krokov:

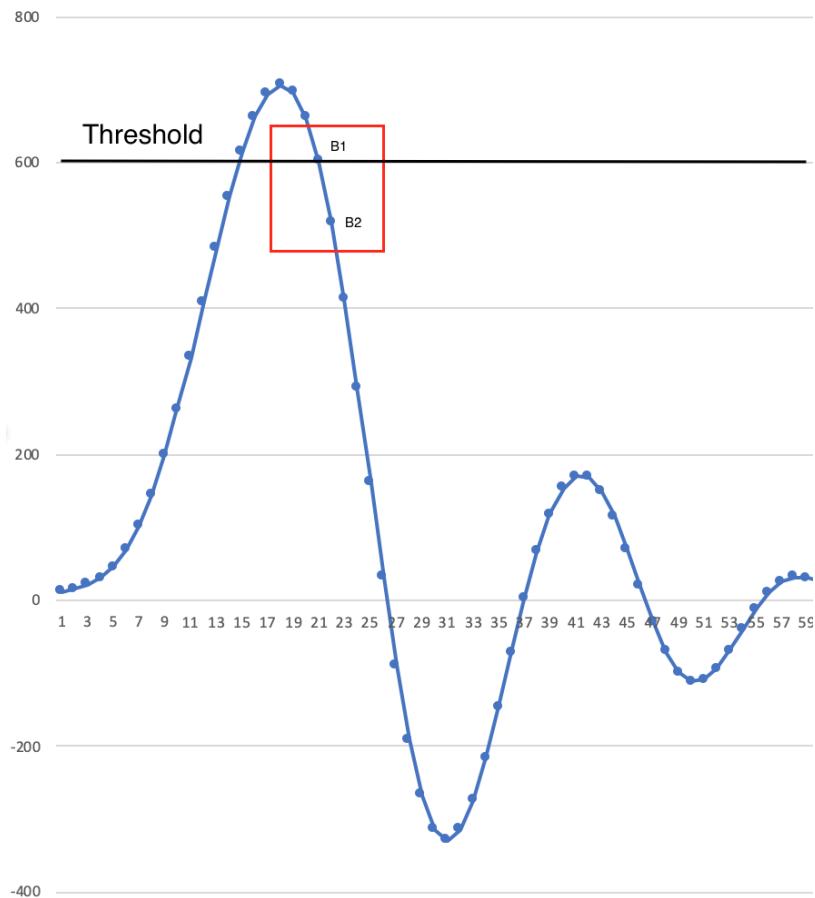
1. Zistenie, či daný bod spĺňa podmienku bodu *th_cross* – bol zachytený vrchol vlny. Ak áno, bod je pridaný do zoznamu pravdepodobných vrcholov vlny.
2. Kontrola či vzdialenosť aktuálneho bodu od najstaršieho bodu v zozname pravdepodobných vrcholov je presne *gMaxWait*. Ak áno, nastáva analýza neurónovou sieťou a vytvorenie snímky.

Po vyhodnotení podmienok a prebehnutí prípadnej analýzy neurónovou sieťou, je zvýšená hodnota *actPointer*, tak aby adresa stále ukazovala na hodnoty v kruhovom buffer. Tiež je zvýšená hodnota *actTime*, na základe vzorkovania.

Kontrola podmienky *th_cross*

Skratka *th_cross* je vytvorená zo slov „threshold cross“ – teda je to miesto v priebehu vlny, kedy hodnota prechádza z hranice nad thresholdom pod jeho hranicu (v prípade zápornej vlny naopak).

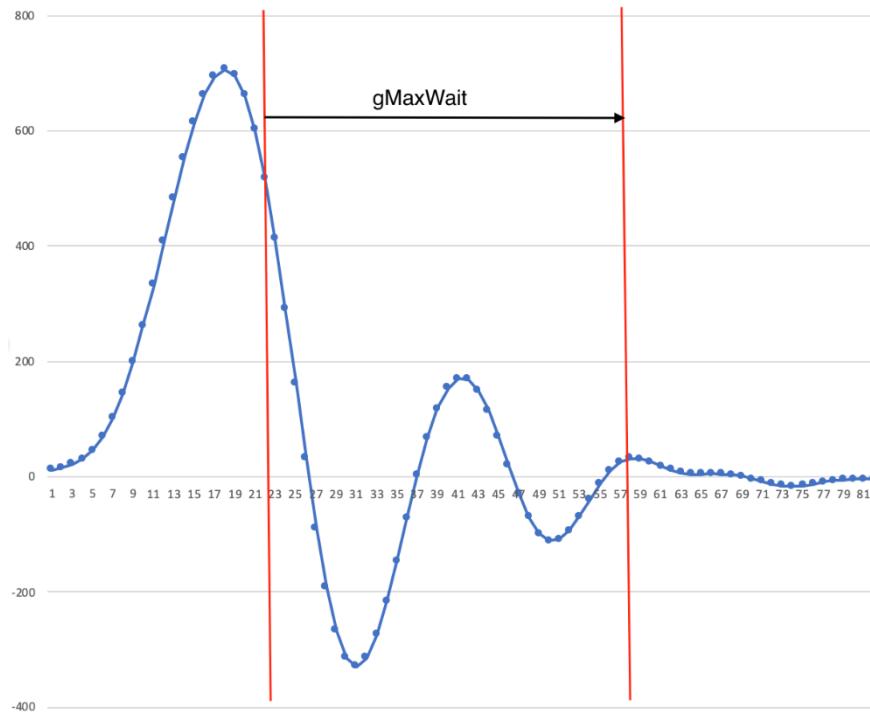
Pre každý typ vlny, je v jej priebehu určené miesto, kde sa predpokladá takéto správanie hodnôt. Od tohto bodu sa ďalej určuje rozsah dát, ktoré sa vyseknú z dátového buffera. V rámci tohto rozsahu sa predpokladá, že bude zachytený celý priebeh vlny. Ak sa nájdú dva za sebou idúce body, ktoré vykazujú dané hodnoty, druhý z týchto bodov sa použije ako pravdepodobný vrchol vlny – avšak nemusí to byť aj reálne najvyššia hodnota v danej vlne.



Obr. 11: Bod B2 - splňujúci podmienku th_cross

Kontrola vzdialenosťi $gMaxWait$

Pri pridávaní typu vlny do merania sa vypočíta rozdiel bodov th_cross a $dlzka$ vlny. Výsledok je vzdialosť, koľko bodov je nutné počkať od nájdenia pravdepodobného vrcholu vlny po skončenie celého jej priebehu. V rámci merania sa potom nájde maximálna hodnota toho rozdielu medzi všetkými pridanými typmi vln. Ak pre aktuálne analyzovaný bod je jeho vzdialenosť od najstaršieho nájdeného vrcholu vlny rovná $gMaxWait$, vieme, že môžeme vyseknúť úsek dát zodpovedajúci pre najdlhšiu vlnu a tento úsek zmerať neurónovou sieťou a vytvoriť snímku.



Obr. 12: Vzdialenosť od bodu th_cross po koniec vlny

Po splnení predchádzajúcich podmienok, nastáva analýza okolia nájdeného vrcholu. Pre každý typ vlny, ktorý sa v meraní nachádza sa vysekne okolie daného bodu a skontroluje sa príslušnou neurónovou sietou. Tá pre úsek dát vypočíta pravdepodobnosť, že sa ľahom nachádza vlna. Nájde sa maximálna pravdepodobnosť naprieč všetkými typmi vln. Ak je vyššia ako 90% predpokladáme, že sa naozaj jedná o vlnu a vytvorí sa snímka s daným typom vlny, ktorý získal najvyššiu pravdepodobnosť. Ak je nižšia ako 90%, jedná sa o artefakt.

3.4.1.6. Neurónová siet

Pre zistenie, či nejaký úsek dát obsahuje hľadanú vlnu alebo iba artefakt, používame neurónovú sieť. Pre samotný základ neurónovej siete sme použili knižnicu Genann, ktorá je dostupná pod voľnou licenciou. Okolo tejto knižnice sme vytvorili „obal“, ktorý zabezpečuje správny chod a komunikáciu v rámci analyzátoru signálu.

Knižnica Genann poskytuje vytvorenie neurónovej siete typu MLP – multi layer perceptron. Tento typ neurónovej siete je postačujúci pre spracovanie tabuľkových dát pevnej dĺžky, čomu zodpovedá reprezentácia vlny v tomto projekte. Keďže ale každý typ vlny môže mať dĺžku rozdielnú, vytvorili sme pre každý typ vlny samostatnú siet s vlastnou topológiou.

Trieda *Ann* obsahuje objekt typu vlny, ktorý v sebe nesie informáciu o topológii siete. Na základe tohto sa siet vytvorí. Pre každý typ vlny je vytvorený tréningový dataset, ktorý sa využíva pre trénovanie siete. Dataset je rozdelený na 3 časti:

- Tréningová časť
- Testovacia časť

- Validačná časť

Po vytrénovaní je sieť pripravená na ďalšiu prácu. Môže byť uložená do súboru pre neskôršie použitie, dotrénovaná z novo-získaných dát alebo jednoducho spustená pre jedno pole dát, ktoré chceme analyzovať. Toto pole (úsek s vlnou) musí mať rovnakú dĺžku ako je počet neurónov vo vstupnej vrstve siete.

Výstupom jedného dopredného spustenia siete so zvoleným poľom dát je číslo z intervalu $<0,1>$, ktoré hovorí o pravdepodobnosti, že sa v danom poli nachádza rovnaký typ vlny, na aký bola daná siet natrénovaná.

3.4.1.7. Výstup programu

Výstupom programu sú vytvorené snímky zaujímavých miest v meraní. Jedná sa o vlny a artefakty. Tieto snímky sa ukladajú do XML súborov na adrese *data/export*. Názvy súborov so snímkami sú formátované v tvare „*snap_typVlny_poradieSnímky_datumČas.xml*“. Súbor so snímkou obsahuje okrem koreňového elementu *wave* niekoľko ďalších elementov:

- name – textová informácia predstavujúca názov typu vlny.
- max – číselná hodnota, maximálna hodnota jedného bodu dát v snímke.
- similarity – číselná hodnota v intervale $<0,1>$, ktorá predstavuje percento pravdepodobnosti, že sa jedná o daný typ vlny. Ak je táto hodnota väčšia ako 0.9, jedná sa o vlnu. Ak je menšia, jedná sa o artefakt.
- time – čas v meraní, kedy bola vlna alebo artefakt zachytené.
- data – pole čísel. Samotné dáta priebehu vlny alebo artefaktu.

3.4.2. Generátor signálov

V tejto kapitole je opísaná časť aplikácie, ktorá simuluje činnosť externých meracích zariadení používaných pri neurofyziologických vyšetreniach. Tieto zariadenia merajú činnosť nervov a svalov pacientov – zachytávajú nervové impulzy, ktoré sú reprezentované vlnami v grafe. Dáta odosiela prvej časti aplikácie, ktorá ich analyzuje. Postupnosť jednotlivých vln a artefaktov môže používateľ ručne ovládať príkazmi. Na základe týchto príkazov sa v reálnom čase vygenerujú rôzne typy vln. Predpokladá sa, že v budúcnosti bude táto časť úplne nahradená reálnymi zariadeniami, ktoré snímajú pacientove svaly alebo nervy.

Táto časť aplikácie je písaná v jazyku Python, pretože to bol najrýchlejší a najjednoduchší spôsob ako vytvoriť simulované meranie pacienta. Pri vývoji neboli použití žiadne špeciálne framework, iba niekoľko knižníč, ktoré pomáhajú pri generovaní signálu a jeho vizualizácii.

3.4.2.1. Typy vln a artefakty

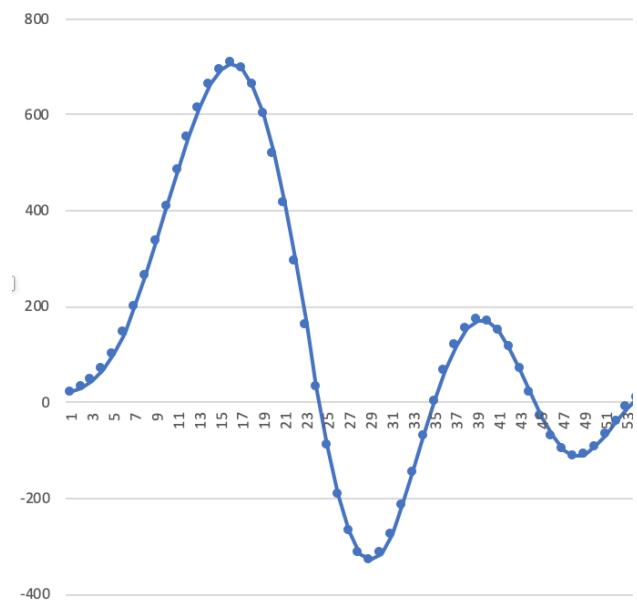
Simulátor merania je schopný generovať niekoľko typov signálu. Všetky typy signálu sa striedajú a tak vytvárajú plnohodnotnú simuláciu merania.

Šum

Prvým základným typom signálu je šum. Ide o body v meraní, ktorých hodnota sa pohybuje vo veľmi malom intervale $<-20,20>$ okolo hodnoty 0. Tento druh signálu nie je pre lekára ničím zaujímavý a snažíme sa jeho prehľadávanie čo najviac odbúrať. Generuje sa automaticky stále vtedy, keď sa práve negeneruje iný typ signálu. Môže vyplňať medzery medzi jednotlivými vlnami alebo artefaktami.

Vlna typu 1

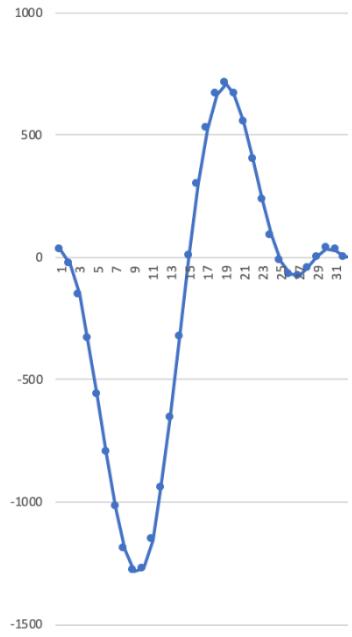
Prvý typ vlny sa vytvára z pola čísel, ktoré sa vygeneruje pomocou funkcie `wavelets.daub(10)`, ktorá je z balíku `scipy.signal`. Toto pole sa ďalej pre-vzorkuje pomocou funkcie `resample()` z balíka `scipy.signal` a zvýši sa rozlíšenie tak, že sa všetky body vynásobia číslom 1024. Dĺžka priebehu vlny je 50 bodov merania.



Obr. 13: Priebeh vlny typu 1

Vlna typu 2

Druhý typ vlny sa vytvára z pola piatich celých čísel [1000, 2500, 1700, 300, 1000]. Pri generovaní, môžu byť tieto čísla mierne odlišné aby sme získali mierne odlišnú vlnu. Tvar bude rovnaký ale extrémy vo vrcholoch môžu byť odlišné. Ďalej sa využije funkcia `resample()` knižnice `scipy.signal` a toto pole sa pre-vzorkuje na pole tridsiatich čísel. Všetky čísla sa vynásobia -1. A pripočítá sa ku každému hodnote 1000. Týmto spôsobom dostaneme vlnu, ktorá začína záporným vrcholom a pokračuje menším kladným vrcholom. Dĺžka priebehu vlny je 30 bodov.

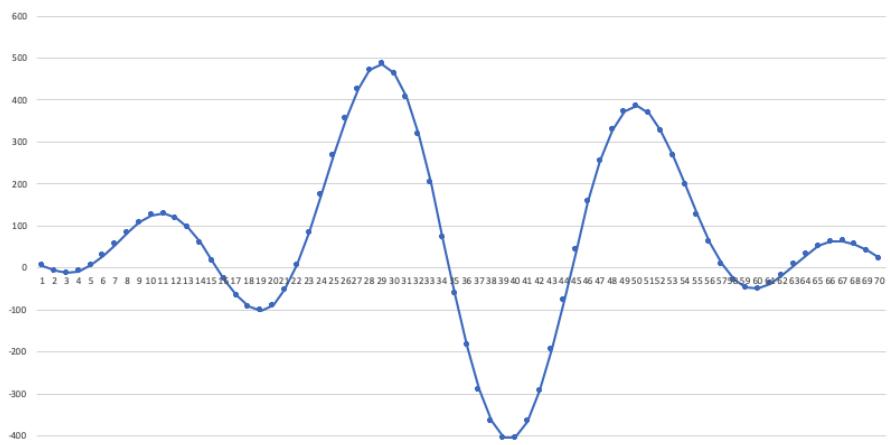


Obr. 14: Priebeh vlny typu 2

Vlna typu 3

Tretí typ vlny sa vytvára z poľa čísel [0, 100, -50, 200, 300, -400, 300, 100, 10]. Ako pri predchádzajúcim type vlny, tak aj pri tomto sa jednotlivé čísla pri generovaní môžu trochu meniť aby sme získali rozdielne hodnoty vo vrcholoch.

Použije sa funkcia `resample()` z balíku `scipy.signal` a toto pole sa pre-vzorkuje na pole sedemdesiatich čísel.



Obr. 15: Priebeh vlny typu 3

Artefakt

Artefakt je úsek merania, kde hodnoty nadobúdajú nepredvídateľné, náhodné hodnoty. Pri reálnom meraní je vznik artefaktov spôsobený vonkajším prostredím. V simulovanom meraní vytvárame artefakty generovaním náhodných hodnôt v určitom intervale. Dĺžka každého artefaktu je tiež náhodná hodnota z určitého intervalu. Celkovo je program schopný generovať 5 typov artefaktov:

1. Hodnoty v intervale $< -100, 100 >$
2. Hodnoty v intervale $< -500, 500 >$
3. Hodnoty v intervale $< -700, 700 >$
4. Hodnoty v intervale $< -1000, 1000 >$
5. Hodnoty v intervale $< -1300, 1300 >$

Pri vytváraní artefaktu sa náhodne určí typ, ktorý sa vygeneruje. Tiež sa vygeneruje náhodné číslo z intervalu $< 15, 50 >$, ktoré predstavuje dĺžku artefaktu.

3.4.2.2. Zobrazenie generovaného signálu

Po spustení generovania signálu naživo sa otvorí okno, ktoré v reálnom čase zobrazuje priebeh merania. Pre zobrazenie grafu používame balík *matplotlib.pyplot*, ktorý je určený pre grafické vytváranie dvojrozmerných grafov.

V programe sa udržuje pole, ktoré sa pri každej zmene zobrazí na grafe. Zmena poľa nastáva vtedy, keď sa v rámci merania vytvorí nový bod. Vtedy sa najstarší bod zobrazovaného poľa odstráni a pridá sa nový. Následne sa toto pole zobrazí na grafe.

Na začiatku je zobrazované pole vynulované a táto časť sa ešte nezapisuje do výstupného súboru aj keď je zobrazená na grafe. Následne keď sa začne generovať šum, vlny a artefakty zmení sa zobrazovaná mierka ypsilonovej osi podľa maximálnej, resp. minimálnej hodnoty. Mierka sa nezmení, ani keď sa už daný extrém posunie v čase a nebude zobrazený.

3.4.2.3. Ovládanie

Živé generovanie signálu je ovládané cez písanie príkazov do príkazového súboru. Príkazové súbory sú pripravené pri inicializácii. Keďže pri väčšom počte súčasne generovaných meraní, by bolo ovládanie z klávesnice, rozhodli sme sa ovládať generovanie týmto spôsobom. Používateľ si pre každé meranie otvorí jeho príkazový súbor a môže do neho písat príkazy, ktoré vygenerujú typ vlny 1 a 2, artefakt alebo ukončenie generovania. Každý príkaz sa napíše na samostatný riadok a potvrdí sa opäťovným uložením súboru.

3.4.2.4. Výstup programu

Výstupom tohto programu je vygenerovanie simulovaného merania neurofiziologických signálov pacienta. Skript sa môže na jednom stroji spustiť v rovnakom čase aj viac-krát, vtedy je možné simulovanie viacerých meraní naraz. Dôležité ale je, pre každú spustenú inštanciu

skriptu nastaviť rozdielny cieľový súbor. Na každom riadku je jedno číslo, ktoré predstavuje jeden bod merania. Tento súbor je potom čítaný analyzátorom signálu.

3.4.2.5. Doplňujúce funkcie

Skript generator obsahuje okrem funkcií zodpovedných za simuláciu neurofyziologických meraní aj funkcie, ktoré využívame pri vytváraní tréningových datasetov pre neurónovú sieť. Napríklad funkciu, ktorá do CSV súboru, v ktorom sú snímky vĺn doplní snímky šumu. Takisto tento skript obsahuje funkcie pre samostatné vygenerovanie jednotlivých vĺn do súborov.

3.5. Overenie riešenia

Hlavnou úlohou aplikácie je správne a rýchlo analyzovať priebeh dát merania a rozpoznať v nich jednotlivé typy vĺn od artefaktov. Výsledkom je súbor snímok zaujímavých miest merania. Na týchto miestach sa nachádzajú spomenuté vlny alebo artefakty. Nasledujúce testovanie bolo vykonané na vlny typu 3, keďže táto vlna je najdlhšia a najzložitejšia. Podobne sme ale postupovali aj pri ostatných typoch vĺn.

3.5.1. Vytvorenie tréningového datasetu

Pomocou generátoru sme vytvorili CSV súbor, ktorý obsahoval niekoľko tisíc správnych snímok požadovanej vlny. Každá snímka bola uložená na jednom riadku, hodnoty boli oddelené bodkočiarkou. Na posledné miesto v riadku bolo doplnané pri generovaní číslo 1, ktoré označuje, že na riadku sa nachádza snímka vlny. Súbor bol ďalej doplnený ďalšie snímky šumu. Počet snímok šumu bol rovnaký ako počet snímok vĺn – veľkosť súboru sa teda zdvojnásobila. Koniec riadku bol označený číslom 0.

Tento postup sme zopakovali 3-krát. Vytvorili sme tak rovnakým spôsobom 3 datasety pre trénovanie. Rozdiel bol vo veľkosti súborov:

1. Dataset - veľkosť 100 snímok vĺn + 100 snímok šumu
2. Dataset - veľkosť 1 000 snímok vĺn + 1 000 snímok šumu
3. Dataset - veľkosť 10 000 snímok vĺn + 10 000 snímok šumu

Každý súbor sme rozdelili na 2 menšie súbory – tréningový a testovací pomerom: $\frac{3}{4}$ pre tréningovú a $\frac{1}{4}$ pre testovaciu časť.

Potom sme vytvorili veľký súbor s 50 000 snímkami vĺn a 50 000 snímkami šumu, na ktorom budeme validovať natrénovanú sieť. Takto vytvorené súbory sme ďalej využili pre testovanie aplikácie.

3.5.2. Scenár testovania

Pre otestovanie rýchlosťi a správnosti natrénovanej neurónovej siete, sme vytvorili scenár, ktorým sme tieto údaje merali:

1. Vytvorenie nového objektu neurónovej siete
2. Načítanie jednotlivých datasetov
3. Spustenie tréningu siete
4. Vyhodnotenie výsledku po validácii siete

Tieto kroky sme zopakovali dvadsať-krát, pričom sme zaznamenávali:

- Priemerný, najlepší a najhorší počet epoch jedného z tréningov
- Priemerný, najlepší a najhorší čas trvania jedného z tréningov
- Najhoršiu a najlepšiu sieť z pohľadu percenta úspešnosti pri validácii

Samotný bod 3 – *trénovanie siete* pozostáva z postupného spúšťania epoch trénovania. Jedna epocha v trénovaní siete pozostáva z prejdenia všetkých snímok v súbore na ktorých sa sieť trénuje. Epochu pokračuje v následnom spustení testovacej funkcie, ktorá vráti úspešnosť aktuálneho nastavenia. Ak je táto úspešnosť menšia ako 100%, pokračuje sa ďalšou epochou. Ak úspešnosť nedosahuje hranicu 100% ale počet spustených epoch presiahol maximálny nastavený počet, trénovanie sa ukončí.

Pri testovaní sa prechádza každá snímka v testovacom súbore a vyhodnocuje sa pravdepodobnosť, že sa na snímke nachádza vlna alebo šum. Táto pravdepodobnosť je získaná zo spustenia siete pre danú snímku. V tomto kroku sa už sieť nijak netrénuje ani nezlepšuje. Ak je pravdepodobnosť vyššia ako 0.9, zaokrúhli sa na číslo 1. Ak je menšia ako 0.1, zaokrúhli sa na 0. Ak je hodnota medzi týmito hranicami, zmení sa na číslo 2. Pre každú snímku v testovacom súbore je určené číslo 1 alebo 0, podľa toho či je na snímke vlna alebo šum. Ak výsledok výpočtu siete zhoduje s týmto číslom, započíta sa to ako správne uhádnutie. Ak je výsledok 2 nikdy sa nemôže započítať ako správny. Výsledkom celého testovania je počet správne a nesprávne uhádnutých snímok. Z týchto čísel sa vypočíta percento úspešnosti siete. Snímky posudzujeme s toleranciou 10% keďže aj správne vlny sa môžu mierne lísiť.

Bod 4 – *validácia siete* je vlastne spustenie testovania siete na novom datasete, ktorý neboli použitý pri trénovaní. Navyše vyhodnocuje aj ostatné štatistiky ako počty epoch alebo časy namerané pre jednotlivé trénovania.

3.5.3. Rozdielne topológie

Následne sme pri použití datasetu s 2000 snímkami spúšťali testovací scenár pre rôzne topológie siete. Porovnali sme výsledky a najlepšiu topológiu sme vybrali. Rozdiel medzi nimi bol len v počte skrytých vrstiev a v počte ich neurónov. Vstupná a výstupná vrstva boli stále rovnaké. Tvar topológie závisí od knižnice Genann, ktorú sme použili. Skladá sa zo 4 čísel pričom prvé číslo hovorí o veľkosti vstupnej vrstvy, druhé o počte skrytých vrstiev, tretie o počte neurónov v skrytých vrstvách a štvrté číslo po veľkosti výstupnej vrstvy.

Topológia 1 – málo skrytých vrstiev s málo neurónmi

Prvá topológia, ktorú sme testovali mala tvar: (70, 1, 10, 1) – 70 vstupných neurónov, jedna skrytá vrstva s desiatimi neurónmi a jeden výstupný neurón. Zaznamenali sme nasledujúce výsledky:

```
Topologia: 70,1,10,1
Maximum epoch: 1000

Pocet epoch treningov:
    Priemer: 1000
    Najmenej: 1000
    Najviac: 1000

Cas trenovania:
    Priemer: 13107 msec
    Najkratsi: 11487 msec
    Najdlhsi: 14532 msec

Cas validacie:
    Priemerny: 523 msec
    Najkratsi: 406 msec
    Najdlhsi: 693 msec

Najhorsia siet:
    SPRAVNE: 96973
    NESPRAVNE: 3027
    Uspesnost: 96.9730%

Najlepsia siet:
    SPRAVNE: 99035
    NESPRAVNE: 965
    Uspesnost: 99.0350%
```

Z výsledku vidíme, že neurónovej sieti sa nepodarilo ani raz dosiahnuť úspešnosť 100% pri tréningu. Na validačných dátach bola najlepšia úspešnosť 99,035%, pričom 965 snímok zo 100 000 bolo zle posúdených.

Topológia 2 – veľa skrytých vrstiev s málo neurónmi

Druhá topológia, ktorú sme testovali mala tvar: (70, 5, 10, 1) – 70 vstupných neurónov, päť skrytých vrstiev, každá s desiatimi neurónmi a jedným výstupným neurónom. Zaznamenali sme nasledujúce výsledky:

```
Topologia: 70,5,10,1
Maximum epoch: 1000

Pocet epoch treningov:
    Priemer: 928
    Najmenej: 281
    Najviac: 1000

Cas trenovania:
    Priemer: 19443 msec
    Najkratsi: 5925 msec
    Najdlhsi: 22066 msec

Cas validacie:
    Priemerny: 603 msec
    Najkratsi: 549 msec
    Najdlhsi: 699 msec

Najhorsia siet:
    SPRAVNE: 99069
    NESPRAVNE: 931
    Uspesnost: 99.0690%

Najlepsia siet:
    SPRAVNE: 99686
    NESPRAVNE: 314
    Uspesnost: 99.6860%
```

Pri topológií s väčším počtom skrytých vrstiev pozorujeme zlepšenie, avšak výsledok pri validácii stále nie je perfektný.

Topológia 3 – málo skrytých vrstiev s veľa neurónmi

Tretia topológia, ktorú sme testovali mala tvar: $(70, 1, 200, 1) - 70$ vstupných neurónov, jedna skrytá vrstva s dvesto neurónmi a jedným výstupným neurónom. Zaznamenali sme nasledujúce výsledky:

```
Topologia: 70,1,200,1
Maximum epoch: 1000

Pocet epoch treningov:
    Priemer: 314
    Najmenej: 0
    Najviac: 1000

Cas trenovania:
    Priemer: 54999 msec
    Najkratsi: 178 msec
    Najdlhsi: 181722 msec

Cas validacie:
    Priemerny: 4901 msec
    Najkratsi: 4662 msec
    Najdlhsi: 5523 msec

Najhorsia siet:
    SPRAVNE: 99469
    NESPRAVNE: 531
    Uspesnost: 99.4690%

Najlepsia siet:
    SPRAVNE: 99945
    NESPRAVNE: 55
    Uspesnost: 99.9450%
```

Pri tejto topológií sme namerali opäť vyššiu úspešnosť avšak rádovo sa zhoršil čas potrebný pri validácii. To znamená, že aj pri ostrom nasadení bude trvať posúdenie jednej snímky dlhšie. Avšak udávané časy sú pre celý validačný set, ktorý je veľký 100 000 snímok. Posúdenie jednej snímky je stále dostatočne rýchle.

Topológia 4 – viacero skrytých vrstiev s veľa neurónmi

Štvrtá topológia, ktorú sme testovali mala tvar: (70, 5, 200, 1) – 70 vstupných neurónov, päť skrytých vrstiev s dvesto neurónmi a jedným výstupným neurónom. Zaznamenali sme nasledujúce výsledky:

```
Topologia: 70,5,200,1
Maximum epoch: 1000

Pocet epoch treningov:
    Priemer: 4
    Najmenej: 1
    Najviac: 22

Cas trenovania:
    Priemer: 14456 msec
    Najkratsi: 3368 msec
    Najdlhsi: 73991 msec

Cas validacie:
    Priemerny: 53012 msec
    Najkratsi: 52749 msec
    Najdlhsi: 53300 msec

Najhorsia siet:
    SPRAVNE: 99682
    NESPRAVNE: 318
    Uspesnosť: 99.6820%

Najlepsia siet:
    SPRAVNE: 99969
    NESPRAVNE: 31
    Uspesnosť: 99.9690%
```

Najlepší prípad, ktorý dosiahla natrénovaná siet' už je relatívne uspokojivý avšak pri tejto topológií sa opäť dramaticky zvýšil čas potrebný na validáciu.

Topológia 5 – kombinácia predchádzajúcich

Na základe predchádzajúcich výsledkov sme ich kombináciou vytvorili topológiu: (70,2,150,1) – 70 vstupných neurónov, dve skryté vrstvy so sto neurónmi a jeden výstupný. Zaznamenali sme nasledujúce výsledky:

```
Topologia: 70,2,150,1
Maximum epoch: 1000

Pocet epoch treningov:
    Priemer: 35
    Najmenej: 3
    Najviac: 133

Cas trenovania:
    Priemer: 21011 msec
    Najkratsi: 2078 msec
    Najdlhsi: 77698 msec

Cas validacie:
    Priemerny: 10849 msec
    Najkratsi: 10333 msec
    Najdlhsi: 11446 msec

Najhorsia siet:
    SPRAVNE: 99492
    NESPRAVNE: 508
    Uspesnosť: 99.4920%

Najlepsia siet:
    SPRAVNE: 99989
    NESPRAVNE: 11
    Uspesnosť: 99.9890%
```

Pri použití tejto topológií pozorujeme najlepší výsledok presnosti. Takisto aj čas validácie je relatívne malý.

Pri výhodnocovaní všetkých typov topológií sme pozorovali, že pri chybných prípadoch spravidla nedochádzalo k zlému posúdeniu vlny. Chyby nastávali len pri určovaní snímok šumu, ktorým bola priradená vyššia podobnosť s vlnou ako 10%.

3.5.4. Rozdielne datasety

Ďalej sme sa pokúsili natrénovať neurónovú sieť pomocou vytvorených datasetov s rozdielnou veľkosťou. Pre každý z týchto datasetov sme vykonali rovnaký scenár testovania. Použili sme topológiu siete číslo 5 z predchádzajúceho testovania – 70, 2, 150, 1.

Dataset 1 – 200 snímok

Pri použití tréningového súboru o veľkosti 150 snímok a testovacieho o veľkosti 50 snímok bola neurónová sieť schopná sa natrénovať po v priemere 565 epochách. Čas trénovania bol 32830 milisekúnd, pričom priemerný čas jednej epochy bol 26 milisekúnd

Úspešnosť takto natrénovanej siete na validačnom súbore bola v najlepšom prípade 99,63%. Zo 100 000 snímok bolo 99 634 posúdených správne ale 366 snímok bolo posúdený nesprávne.

Dataset 2 – 2 000 snímok

Pri použití tréningového súboru o veľkosti 1 500 snímok a testovacieho o veľkosti 500 snímok bola neurónová sieť schopná sa natrénovať v priemere po 35 epochách. Priemerný čas trénovania bol 21 011 milisekúnd.

Úspešnosť takto natrénovanej siete na validačnom súbore bola v najlepšom prípade 99,989%. Zo 100 000 snímok bolo 99 989 posúdených správne a len 11 nesprávne

Dataset 3 – 20 000 snímok

Pri použití tréningového súboru o veľkosti 15 000 snímok a testovacieho s 5 000 snímkami bola sa dokázala sieť natrénovať v priemere po 5 epochách. Priemerný čas trvania tréningu bol 30 507 milisekúnd.

Úspešnosť takto natrénovanej siete na validačnom súbore bola v najlepšom prípade 99,999%. Zo 100 000 snímok bolo zle posúdená len 1 snímka.

Z týchto výsledkov vieme povedať, že ak budeme v budúcnosti pridávať nový typ vlny, bude vhodné pre tréning vytvoriť dataset s aspoň 10 000 snímkami danej vlny a 10 000 snímkami šumu.

3.5.5. Analýza reálneho merania

Posledným testom riešenia bola analýza vygenerovaného záznamu merania. Rovnakým spôsobom ako v predchádzajúcich testoch, sme vytrénovali model neurónovej siete aj pre ďalšie dva typy vln. Pomocou generátora sme vytvorili záznam s 500 snímkami každého typu vlny a 1 000 artefaktmi. Spustili sme analýzu a hodnotili sme, koľko snímok jednotlivých vln bolo vytvorených. Podľa očakávania, bolo vytvorených 500 snímok každého typu vlny. Snímok artefaktov bolo vytvorených viac ako 1 000, pretože v rámci jednej sekvencie artefaktu sa nachádza viacero bodov, ktoré spĺňajú podmienku *th_cross* aspoň jedného typu vlny v meraní.

4. Zhodnotenie

Cieľom práce bolo vytvorenie aplikácie pre analýzu a automatické vyhodnocovanie neurofyziologických signálov.

Ako prvé sme analyzovali medicínsku oblasť, pri ktorej sa využíva neuromonitoring. Rozobrali sme príklady niekoľkých ochorení, ktorými trpia pacienti. Pri liečbe lekári využívajú aj monitorovanie ich nervov či svalov. Táto problematika je rozobraná v kapitole 2.1. *Diagnózy* a 2.2. *Liečebná metóda – selektívna dorzálna rizotómia*. Ďalej sme sa podrobnejšie venovali práve technikám a spôsobom využitia neuromonitoringu či už počas operácie pacienta alebo aj mimo operácie. Rozobrali sme aj typy kriviek, ktoré s ktorými lekár pracuje počas mimooperačného vyšetrenia EMG. V kapitole 2.3. *Neuromonitoring* sme uviedli typy hľadaných kriviek ale možné deformácie a vznik nechcených chýb pri meraní, ktoré sú spôsobné vonkajším prostredím a stážajú prácu lekárovi – takzvané artefakty. Pre lepšiu predstavu ako sa počas monitorovania pacienta zbierajú dátá, sme tento postup opísali v kapitole 2.4. *Získavanie monitorovacích dát*.

V ďalších kapitolách sme sa už venovali spôsobom ako sa tieto namerané signály vyhodnocovať. V kapitole 2.5.1. *Atribúty signálu* sme rozobrali ako definovať krivku signálu a aké informácie budeme a o krivke zbierať a vyhodnocovať. V nasledovnej kapitole sme analyzovali spôsoby ako typy kriviek rozoznávať – ako klasifikovať jednotlivé krivky. V kapitole 2.5.3. *Neurónové siete* sme detailnejšie rozobrali ako fungujú neurónové siete. Aké sú výhody použitia práve tejto metódy, vďaka ktorým dokážeme jednotlivé krivky rozoznávať s vysokou presnosťou a ktorá sa časom dokáže ešte zlepšovať. Touto časťou práce sa skončila analýza problému.

Následne bolo potrebné špecifikovať požiadavky aplikácie. Tie sú spísané v kapitole 3.1. *Špecifikácia požiadaviek*. Odvodili sme ich na základe konzultácií s lekárkou, kedy nám vysvetlila v akom prostredí môže byť aplikácia nasadená. Ďalej sme opísali návrh riešenia ako implementovať aplikáciu. Keďže nastal problém so získaním reálnych dát, nameraných pri vyšetreniach v nemocnici, rozhodli sme sa pre vytvorenie generátoru dát. Návrh riešenia je opísaný v kapitole 3.2. *Návrh riešenia*.

Nasledovala fáza implementácie, opísaná v kapitole 3.4. *Implementácia*, v ktorej sme najprv vytvorili prototyp generátoru dát. Ten spočiatku generoval len jeden typ vlny. Postupným vylepšovaním sme do neho pridali ďalšie dva typy vĺn, artefaktov a šum. V rámci generátoru sú tiež implementované funkcie, ktoré rýchlo a efektívne dokážu vygenerovať dataset pre každý typ vlny. Tieto datasety sme využili pri trénovaní neurónových sietí. Generátor tiež obsahuje funkciu, ktorou dokáže generovať signál v reálnom čase. Dĺžka generovania nie je obmedzená, preto je implementovaný generátor solídnym simulátorom reálneho merania pacienta. Táto časť práce bola implementovaná v jazyku Python. Ďalšou časťou, ktorú sme implementovali bol analyzátor signálov. Táto aplikácia bola vytváraná v jazyku C++ a využili sme framework Qt. Vybraný framework najlepšie splňal špecifikované požiadavky na multiplatformovosť a tiež pomohol pri spojení tejto práce s ďalšou, v ktorej sa študent venoval vytváaniu grafického rozhrania pre analyzátor. Navrhli sme formát XML súboru, ktorý zahŕňa pre typ vlny všetky informácie, ktoré sme analyzovali v predchádzajúcich kapitolách. To zabezpečuje jednoduché pridanie nového typu vlny do systému. V aplikácii analyzátoru sme ďalej vytvorili spôsob statického vyhľadania oblastí dát, v ktorých

predpokladáme, že sa nachádzajú vlny. Tieto oblasti sa určujú práve na základe konfiguračného XML súboru pre jednotlivé typy vln. Následne sme vytvorili modul, ktorý obsahuje neurónovú sieť. Ten je zodpovedný za posúdenie, či sa danom úseku dát nachádza vlna alebo artefakt. Po vyhodnotení sa vytvára snímka danej oblasti dát aj s nameranými dátami, pre ďalšie použitie lekárom.

Následné testovanie, ktoré je popísané v kapitole 3.5. *Overenie riešenia*, malo potvrdiť správnosť riešenia a najst' najlepšiu konfiguráciu pre vyhľadávanie tvarov vln v dátach. Vytvorili sme veľké množstvo snímok vln a pomocou nich sme natrénovali neurónovú sieť. Následne sme spustili automatickú simuláciu merania, ktorá vytvorila niekoľko stoviek vln. Týmto spôsobom bolo simulované reálne meranie pacienta pri vyšetrení. Výsledky ukázali správne fungovanie aplikácie analyzátoru.

Výsledná aplikácia splňa všetky požiadavky, ktoré vyplývali zo zadania a je plne pripravená na ďalšie používanie, vývoj a testovanie. Môže mať reálne uplatnenie pri vyšetreniach pacientov v nemocnici a dokáže lekárovi ušetriť čas pri analýze celého merania. Lekári, ktorí používajú aktuálny softvér musia celé meranie sledovať ako video, naše riešenie ponúka automatickú extrakciu zaujímavých častí a ich analýzu. Výhodou aplikácie je jej modulárna architektúra a teda pri ďalšom vývoji je jednoduché vymeniť alebo doplniť ďalšiu funkcionality. Takisto je relatívne jednoduché doplniť nový tvar vlny – stačí ju len popísať v konfiguračnom súbore a vytvoriť dataset, ktorým sa natrénuje neurónová sieť. Tiež sa aktuálne vytvorené siete dokážu časom zlepšovať, keďže aplikácia dokáže z novo-vytvorených snímok tieto siete dotrénovať. Ďalšia práca si bude vyžadovať doplniť funkčné napojenie externých meracích zariadení cez bluetooth.

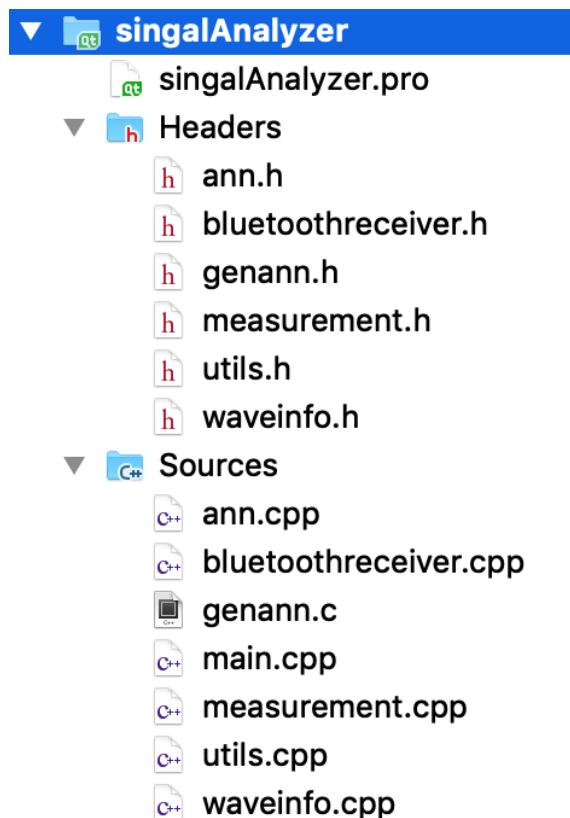
5. Technická dokumentácia

Táto kapitola sa venuje technickej dokumentácii projektu. Je rozdelená na dve časti, pričom prvá časť obsahuje zdrojové kódy aplikácie a ich vysvetlenia. Druhá časť obsahuje používateľskú príručku.

5.1. Dokumentácia implementácie

Táto kapitola je venovaná detailnému popisu jednotlivých tried a metód, ktoré boli v rámci projektu diplomovej práce implementované.

Hlavná aplikácia, ktorá analyzuje namerané neurosignály je písaná v jazyku C++, vo frameworku Qt. Pre neurónovú sieť sme využili knižnicu Gennan, ktorá je napísaná v ANSI C. Každá trieda zdrojového kódu je rozdelená do hlavičkového s príponou .h a zdrojového súboru s príponou .cpp. Knižnica Genna má tiež hlavičkový súbor ale implementácia je uložená súbore s príponou .c. Súbor s príponou .pro je konfiguračný súbor Qt frameworku. Všetky súbory aplikácie sú organizované nasledovne:



Obr. 16: Zoznam súborov aplikácie

Druhá aplikácia, ktorá simuluje reálne meranie neurosignálov a pre potreby vývoja hlavnej aplikácie ich generuje, je písaná v jazyku Python 3.6. bez použitia špeciálneho frameworku. Celá je uložená len v jedno súbore: generator.py.

5.1.1. Main.cpp

Je to spúšťací súbor konzolovej aplikácie, ktorá je určená pre prezentačné účely.

```
int main(){

    QTextStream qtin(stdin);
    QString line;
    QList<Measurement*> merania;

    // vytvorenie bluetooth receiveru
    BluetoothReceiver *br = new BluetoothReceiver();

    printf("\n\n");
    QFile file(QString(DATA_PATH) + QString("nadpis.txt"));
    file.open(QIODevice::ReadOnly | QIODevice::Text);
    QByteArray nadpis = file.readAll();
    printf("%s\n\n", nadpis.toStdString().c_str());
    printf("Prikazy:\n");
    printf("h - vypis helpu\n");
    printf("addm - vytvorenie noveho merania\n");
    printf("w2m - pridanie typu vlny do merania\n");
    printf("status - vypis aktualneho nastavenia programu\n");
    printf("start - spustenie analyzy vsetkych vytvorených merani\n");
    printf("exit - ukoncenie apliakcie\n\n");

    while(1){
        printf("Main>: ");
        qtin >> line;
        line = line.toLower();

        if(line.contains("exit")){
            printf("Koniec programu\n");
            break;
        }
        if(line.contains("h")){
            printf("h - vypis helpu\n");
            printf("addm - vytvorenie noveho merania\n");
            printf("w2m - pridanie typu vlny do merania\n");
            printf("status - vypis aktualneho nastavenia programu\n");
            printf("start - spustenie analyzy vsetkych vytvorených merani\n");
            printf("exit - ukoncenie apliakcie\n\n");
        }

        if(line.contains("status")){
            printf("\nPocet merani: %d\n", merania.size());
            for(int i=0; i<merania.size(); i++){
                merania.at(i)->printStatus();
            }
            br->printStatus();
        }
    }
}
```

```

        if(line.contains("addm")){
            printf("Zadaj nazov merania: ");
            qtin >> line;
            merania.append(new Measurement(line, 1000));
            Measurement *m = merania.last();
            br->addMeasurement(m);
            do{
                printf("Zadaj nazov suboru pre bluetooth receiver: ");
                qtin >> line;
                line = line.toLower();
            }while(br->addFile(QString(DATA_PATH) + QString("bluetoothData/") + QString(line)) == 1);

        }

        if(line.contains("w2m")){
            printf("Zadaj nazov merania: ");
            qtin >> line;
            for(int i=0; i<merania.size(); i++){
                if(merania.at(i)->mName == line){
                    printf("Zadaj nazov vlny: ");
                    qtin >> line;
                    line = line.toLower();
                    merania.at(i)->addWave(new Ann(new Waveinfo(QString(DATA_PATH), QString(line))));
                    break;
                }
            }
        }

        if(line.contains("start")){
            QThreadPool::globalInstance()->start(br);

            while(1){
                int ended = 0;
                for(int i=0; i<merania.size(); i++){
                    if(merania.at(i)->step() == 1) ended++;
                }
                if(ended == merania.size()){
                    printf("Main: koniec merania\n");
                    break;
                }
            }
            printf("Main: Ukoncene vsetky merania\n");
        }

    }

    return 0;
}

```

Aplikácia sa môže nachádzať v dvoch stavoch: *konfigurácia* a *analýza*. V konfiguračnom stave sa v aplikácii môžu pridať alebo odoberať merania a tiež sa dajú jednotlivé merania nastavovať – môžu sa do nich pridať alebo odoberať vlny. Môže sa vytvoriť nový typ vlny a k nej prislúchajúca neurónová sieť. Tiež je možné novo-vytvorenú neurónovú sieť trénovať alebo dotrénovať už existujúcu sieť pomocou snímok, ktoré boli predtým vytvorené. V stave analýzy nie je možné nijako meniť nastavenie programu, pretože prebieha analýza jednotlivých meraní. Dáta sa preberajú z bluetooth zariadenia a spracovávajú sa.

Po spustení sa vytvorí objekt, ktorý spravuje zoznam meraní a tiež sa vytvorí objekt určený na prijímanie dát cez bluetooth.

Ďalej sa spustí načítavanie príkazov, ktoré reprezentujú rovnaké use-casy aké budú vo finálnej grafickej aplikácii. Zoznam príkazov:

- h – vypíše help
- exit – ukončenie aplikácie
- addm – vytvorenie nového merania
- w2m – pridanie typu vlny do merania
- status – vypísanie aktuálneho nastavenia programu
- start – spustenie analýzy

Príkaz *h* – po zadaní sa vypíše zoznam jednotlivých príkazov, ktoré je možné zadávať do aplikácie a tým ju ovládať.

Príkaz *exit* – ukončí aplikáciu ak sa nachádza v konfiguračnom stave

Príkaz *addm* – pridanie nového merania do programu. Po zadaní tohto programu aplikácia ďalej požaduje zadanie názvu merania. Potom sa meranie pridá do bluetooth prijímača. Pre prezentáčné účely sa ešte musí zadať názov súboru, z ktorého bude bluetooth prijímač získavať k danému meraniu dátu.

Príkaz *w2m* – pridanie nového typu vlny do existujúceho merania. Po zadaní tohto príkazu je nutné zadať názov merania, do ktorého sa nový typ vlny pridá. Znamená to, že v tomto meraní sa bude rozpoznávať ďalší typ vlny. Ak je zadaný názov existujúceho merania, je ďalej nutné zadať názov konfiguračného súboru požadovanej vlny. Ak ešte takáto vlna v meraní priradená nie je, učiní sa tak.

Príkaz *status* – vypísanie aktuálneho stavu nastavenia programu. Vypíše sa všetky merania a pre každé meranie sa vypíše zoznam vln, s ktorými dané merania pracujú. Takisto sa vypíše nastavenie bluetooth prijímača – čiže tiež zoznam meraní a súbory k nim priradené.

Príkaz *start* – spustenie analýzy meraní. Aplikácia sa prepne z konfiguračného stavu do stavu analýzy. Bluetooth prijímač začne získavať dátu a jednotlivé merania začnú analyzovať ich začnú analyzovať a vyhľadávať v nich prednastavené vlny. Tento stav trvá až kým nie je ukončené posledné meranie.

5.1.2. Waveinfo

Trieda *Waveinfo* je určená pre správu informácií o jednom type vlny. Do konštruktora triedy vstupuje názov XML súboru, v ktorom sú údaje o type vlny. Tento súbor sa hned' v konštruktore otvorí a získané údaje z neho sa uložia do objektu typu *Waveinfo*.

Waveinfo.h

Hlavičkový súbor triedy Waveinfo.

```
#include <QFile>
#include <QString>
#include <QtXml>

class Waveinfo{

public:
    /* Vytvorenie konfiguracneho objektu vlny */
    Waveinfo(QString dataPath, QString fileName);

    /* Threshold vlny */
    int waveThreshold;

    /* Bod vlny, v ktorom vlna prechadza z urovne nad thresholdom pod tuto uroven
       (a naopak ak je threshold zaporny) */
    int waveTh_cross;

    /* Dlzka priebehu vlny */
    int waveLength;

    /* Nazov vlny */
    QString waveName;

    /* Nazov suboru neuronovej siete pre vlnu */
    QString waveAnnFile;

    /* Topologia neuronovej siete pre vlnu */
    int waveAnnTopology[4];

private:

};
```

Údaje, ktoré o vlnie drží objekt typu Waveinfo sú:

- waveThreshold – úroveň, od ktorej sa rozpoznáva vlna
- waveTh_cross – bod v priebehu vlny, kedy hodnota prechádza z úrovne nad threshodlom pod jeho úroveň
- waveLength – dĺžka priebehu vlny
- waveName – názov vlny
- waveAnnFile – názov súboru, v ktorom je uložená neurónová sieť danej vlny
- waveAnnTopology – pole údajov, ktoré predstavuje topológiu neurónovej siete

Waveinfo.cpp

Tento zdrojový súbor obsahuje len konštruktor objektov typu tejto triedy.

Waveinfo() – konštruktor objektu typ Waveinfo.

Parametre:

- `QString dataPath` – cesta do dátového priečinku aplikácie
- `QString fileName` – názov konfiguračného súboru

```

Waveinfo::Waveinfo(QString dataPath, QString fileName){
    QDomDocument xmlWaveInfo;

    // Otvorenie suboru a nacitanie do XML objektu
    QFile file(dataPath + QString("config/") + fileName);
    if (!file.open(QIODevice::ReadOnly))
        printf("Waveinfo FILE ERROR 1: Subor s informaciami o vlnе neboli najdeny\n");
    xmlWaveInfo.setContent(&file);
    file.close();

    // Pripojenie k datam v XML objekte a ich ulozenie do vlastneho objektu
    QDomElement waveRoot = xmlWaveInfo.documentElement();
    QDomElement wave = waveRoot.firstChild().toElement();

    do{
        if(wave.tagName() == "name"){
            Waveinfo::waveName = QString(wave.text());
        }

        if(wave.tagName() == "threshold"){
            Waveinfo::waveThreshold = wave.text().toInt();
        }

        if(wave.tagName() == "dlzka_priebehu"){
            Waveinfo::waveLength = wave.text().toInt();
        }

        if(wave.tagName() == "th_cross"){
            Waveinfo::waveTh_cross = wave.text().toInt();
        }

        if(wave.tagName() == "ann_file"){
            Waveinfo::waveAnnFile = dataPath + QString("ann/") + QString(wave.text());
        }

        if(wave.tagName() == "ann_topology"){
            QString line = wave.text();
            line.replace(",",".");
            QStringList data = line.split(':');

            for(int t=0; t<4; t++){
                Waveinfo::waveAnnTopology[t] = data[t].toInt();
            }
        }
        wave = wave.nextSibling().toElement();
    }while(!wave.isNull());
    printf("Konfiguracny subor vlny nacitany\n");
}

```

Do konštruktu ako parameter vstupuje názov súboru s konfiguračnými údajmi o vlne a tiež cesta k hlavnému dátovému priečinku aplikácie. Súbor s údajmi o vlne sa bude hľadať adresu, ktorá vznikne spojením adresy hlavného dátového priečinku s priečinkom *config*. Na tejto adrese sa nachádzajú konfiguračné súbory jednotlivých vln. Načíta sa teda požadovaný súbor. Ak sa ho naozaj podarilo otvoriť, spustí sa cyklus, ktorý hľadá jednotlivé XML elementy a získava z nich údaje o vlne. Na poradí elementov v súbore nezáleží.

5.1.3. Ann

Trieda *Ann* je určená pre správu jedného typu vlny a neurónovej siete pre danú vlnu. Táto trieda priamo využíva funkcie knižnice Genann, ktorá je zodpovedná sa za výpočty neurónovej siete. Táto trieda zodpovedá za načítavanie a ukladanie siete, jej trénovanie, evaluáciu a tiež spúšťanie. Takisto táto trieda je zodpovedná za načítavanie a správu datasetov potrebných pri trénovaní neurónovej siete.

Ann.h

Hlavičkový súbor triedy *Ann*.

```
#include <QString>
#include <QFile>
#include <QList>
#include <genann.h>
#include <waveinfo.h>
#include <stdlib.h>

class Ann{
public:
    /* Konstruktor - vytvorenie noveho objektu neuronovej siete*/
    Ann(Waveinfo *wi);

    /* Konfiguracny subor danej vlny*/
    Waveinfo *wi;

    /* Vytvorenie novej neuronovej siete */
    void createNN();

    /* Nacitanie neuronovej siete zo suboru */
    int loadNN();

    /* Ulozenie neuronovej siete do suboru */
    int storeNN();

    /* Nacitanie datasetov pri trenovani neuronovej siete*/
    void loadDataset(QString fileName, int type);

    /* Spustenie trenovania neuronovej siete */
    void trainNN(int maxEpochs);

    /* Spustenie testu neuronovej siete */
    double testNN();

    /* Spustenie evaluacie neuronovej siete */
    void evaluateNN();

    /* Vyhodnotenie dat neuronovou sietou */
    double runNN(double *input);

    /* Uvolnenie neuronovej siete */
    void freeNN();
```

```

private:
    /* Neuronova siet */
    genann *ann;

    /* Trenovaci dataset */
    QList<QList<double>> trainSet;

    /* Testovaci dataset */
    QList<QList<double>> testSet;

    /* Validacny dataset */
    QList<QList<double>> valSet;

    /* Nazov suboru, v ktorom je ulozena neuronova siet */
    QString annFileName;

};


```

Údaje, ktoré o neurónovej sieti obsahuje objekt typu Ann:

- ann – smerník na štruktúru siete. Štruktúra je definovaná v knižnici Genann
- trainSet – tréningový dataset siete
- testSet – testovací dataset siete
- valSet – validačný dataset siete
- annFileName – názov súboru pre uloženie a načítanie neurónovej siete
- wi – objekt typu Waveinfo, ktorý obsahuje konfiguračné údaje pre prácu s neurónovou sieťou

Do konštruktoru triedy Ann vstupuje objekt typu Waveinfo – konfiguračný objekt vlny. Tento objekt sa pre inštanciu triedy Ann uloží. Ako verejné sú povolené metódy, ktoré sú využívané inými objektami na manipuláciu s neurónovou sieťou. Tiež je verejne dostupný aj objekt s konfiguračnými údajmi vlny. Samotná štruktúra, v ktorej je uložená neurónová sieť je privátna a tiež aj načítané datasety, ktoré sa využívajú pri trénovaní.

Ann.cpp

Súbor obsahuje konkrétnie implementácie metód definovaných v hlavičkovom súbore triedy.

Ann() – konštruktor objektu typu Ann

Parametre:

- Waveinfo *wi – objekt typu Waveinfo

```
Ann::Ann(Waveinfo *wi){  
    srand(time(0));  
    Ann::wi = wi;  
  
    Ann::annFileName = wi->waveAnnFile;  
}
```

trainNN() – táto metóda je zodpovedná za natrénovanie neurónovej siete na požadovanú úroveň.

Parametre:

- int epochs – maximálny počet epoch

Návratová hodnota: žiadna

```
void Ann::trainNN(int epochs){  
    printf("Start training...\n");  
    printf("Train set size: %d\n", trainSet.size());  
    int e;  
    for(e=1; e<=epochs; e++){  
  
        for(int index=0; index<trainSet.size(); index++){  
            QList<double> lineData = trainSet.at(index);  
  
            double* inputs = (double*)malloc(wi->waveLength*sizeof(double));  
            for(int x=0; x<wi->waveLength; x++){  
                inputs[x] = lineData.at(x);  
            }  
            double outputs = lineData.at(wi->waveLength);  
  
            genann_train(ann, inputs, &outputs , 0.1);  
        }  
        double res = testNN();  
  
        printf("Epoch: %d / %d : %lf%\n", e, epochs, res);  
        if(res >= 99.90 || e > 1000){  
            break;  
        }  
    }  
}
```

Ako parameter vstupuje do tejto metódy číslo, ktoré hovorí o maximálnom počte epoch, ktoré počas trénovania môžu prebehnúť. Spustí sa cyklus, ktorý tieto epochy počítá. Pre každú epochu sa spustí cyklus, ktorý prejde celý trénovací dataset a pre každú položku z tohto datasetu spustí funkciu trénovania siete. Veľkosť tréningového balíku je vždy 1, kedže knižnica Genann neponúka možnosť toto nastaviť. Potom sa vyhodnotí úspešnosť siete na testovacích dátach. Ak je miera úspešnosti siete dostatočne veľká alebo je dosiahnutý maximálny počet epoch, trénovanie sa končí, inak sa pokračuje ďalšou epochou.

testNN() – Táto metóda je zodpovedná na testovanie úspešnosti neurónovej siete na testovacích dátach.

Parametre: žiadne

Návratová hodnota:

- double result - percento úspešnosti neurónovej siete

```
double Ann::testNN(){
    double result;
    int good = 0;
    int bad = 0;
    for(int t=0; t<testSet.size(); t++){
        QList<double> lineData = testSet.at(t);

        double* inputs = (double*)malloc(wi->waveLength*sizeof(double));
        for(int x=0; x<wi->waveLength; x++){
            inputs[x] = lineData.at(x);
        }
        double outputs = lineData.at(wi->waveLength);

        double res = *genann_run(ann, inputs);
        if(res > 0.9) res = 1;
        else if(res < 0.1) res = 0;
        else res = 2;

        if(res == outputs){
            good++;
        }else{
            bad++;
        }
    }
    result = ((float)good/testSet.size())*100.0;
    return result;
}
```

Premenné *good* a *bad* počítajú počet správnych výsledkov siete. Z nich sa na konci testovania vypočíta pomer úspešnosti k celkovému počtu testovaných položiek. Pri testovaní sa prechádzajú všetky položky testovacieho datasetu a pre každú sa vypočíta hodnota – pravdepodobnosť či ide o vlnu alebo o šum. Ak výsledok obsahuje odchýlku maximálne 10% považuje sa za správny. Ak je odchýlka väčšia, výsledok sa považuje za nesprávny.

evaluateNN() – táto metóda je zodpovedná za evaluáciu neurónovej siete.

Parametre: žiadne

Návratová hodnota: žiadna

```
void Ann::evaluateNN(){
    int good = 0;
    int bad = 0;
    for(int t=0; t<valSet.size(); t++){
        QList<double> lineData = valSet.at(t);

        double* inputs = (double*)malloc(wi->waveLength*sizeof(double));
        for(int x=0; x<wi->waveLength; x++){
            inputs[x] = lineData.at(x);
        }
        double outputs = lineData.at(wi->waveLength);

        double res = *genann_run(ann, inputs);
        double real = res;
        if(res > 0.9) res = 1;
        else if(res < 0.1) res = 0;
        else res = 2;

        printf("VAL input[%lf, %lf, ... : %lf] get %lf (%lf) ->", inputs[0], inputs[1], outputs,
               res, real);
        if(res == outputs){
            good++;
            printf("good\n");
        }else{
            bad++;
            printf("BAD\n");
        }
    }
    printf("\nTopologia:");
    printf("\t%d,%d,%d,%d\n\n", wi->waveAnnTopology[0], wi->waveAnnTopology[1], wi-
          >waveAnnTopology[2], wi->waveAnnTopology[3]);
    printf("Z poctu %d je:\n", valSet.size());
    printf("\tGOOD: %d\n", good);
    printf("\tBAD: %d\n", bad);
    printf("\tUspesnost: %.2f%%\n", ((float)good/valSet.size())*100.0);
}
```

Vyhodnotí sa jej úspešnosť na dátach, ktoré sú rozdielne od tréningových a testovacích. Priebeh je takmer totožný ako pri metóde *testNN* s tým rozdielom, že na konci sa vypíše obsiahlejšie zhodnotenie úspešnosti siete.

createNN() – táto metóda je zodpovedná za vytvorenie novej neurónovej siete.

Parametre: žiadne

Návratová hodnota: žiadna

```
void Ann::createNN(){
    ann = genann_init(
        wi->waveAnnTopology[0],
        wi->waveAnnTopology[1],
        wi->waveAnnTopology[2],
        wi->waveAnnTopology[3]
    );
}
```

Použije sa funkcia knižnice Genann, ktorej parametre sa nastavia podľa vopred načítanej topológie siete. Prvý parameter hovorí o veľkosti vstupnej vrstvy, druhý o počte skrytých vrstiev, tretí o veľkosti skrytých vrstiev a štvrtý o veľkosti výstupnej vrstvy neurónovej siete.

runNN() – táto metóda je zodpovedná za jedno spustenie výpočtu neurónovej siete na základe jedného vstuпу.

Parametre:

- double *input – smerník na pole so vstupom do siete

Návratová hodnota:

- double res – výsledok po jednom spustení neurónovej siete pre zadaný vstup

```
double Ann::runNN(double *input){
    double res = *genann_run(ann, input);
    return res;
}
```

Ako parameter vstupuje pole, ktoré je veľké ako vstupná vrstva neurónovej siete. Ako výstup je číslo z intervalu <0,1> ktoré tvorí výsledok jednej iterácie neurónovej siete. Pri spustení tejto metódy sa neurónová sieť netrénuje, len počíta výsledok.

storeNN() – táto metóda je zodpovedná za uloženie vytvorenej neurónovej siete do súboru.

Parametre: žiadne

Návratová hodnota:

- int – úspešnosť zápisu siete do súboru

```
int Ann::storeNN(){
    char *fileName = annFileName.toLocal8Bit().data(); // konverzia stringu kvôli nasledovnej
                                                       funkcie fopen
    FILE *file = fopen(fileName, "w");

    if(!file){
        printf("Ann FILE ERROR 2: Subor pre ulozenie neuronovej siete neboli otvorený\n");
        return 1;
    }

    genann_write(ann, file);
    printf("Neuronova siet %s ulozena\n", annFileName.toStdString().c_str());
    return 0;
}
```

Pri ukladaní sa kontroluje správnosť otvorenia súboru a pre samotné uloženie sa využíva procedúra knižnice Genann. Názov súboru, v ktorom bude siet uložená bol definovaný v konfiguračnom súbore vlny.

loadNN() – táto metóda je zodpovedná za načítanie vytvorenej neurónovej siete zo súboru.

Parametre: žiadne

Návratová hodnota:

- int – úspešnosť načítania siete zo súboru

```

int Ann::loadNN(){
    char *fileName = annFileName.toLocal8Bit().data();
    FILE *file = fopen(fileName, "r");

    if(!file){
        printf("Ann FILE ERROR 3: Subor s neuronovou sietou neboli najdeny\n");
        return 1;
    }

    ann = genann_read(file);

    if (!ann) {
        printf("Ann LOAD ERROR 1: Nebolo mozne nacitat neuronovu siet zo suboru: %s.\n", fileName);
        return 1;
    }

    printf("Neuronova siet %s nacitana\n", annFileName.toString().c_str());
    return 0;
}

```

Pri načítaní sa kontroluje správnosť otvorenia súboru a pre samotné načítanie sa využíva funkcia knižnice Genann. Tiež sa kontroluje správne načítanie siete. Názov súboru, v ktorom je sieť uložená bol definovaný v konfiguračnom súbore vlny.

freeNN() – táto metóda je zodpovedná na uvoľnenie vytvorennej neurónovej siete. Pre samotné uvoľnenie sa využíva procedúra knižnice Genann.

Parametre: žiadne

Návratová hodnota: žiadna

```

void Ann::freeNN(){
    genann_free(Ann::ann);
}

```

5.1.4. Bluetoothreceiver

Trieda bluetooth prijímaču je určená pre správu externých zariadení, ktoré získavajú dátu meraním. Trieda dedí z triedy QRunnable, ktorá je súčasťou frameworku Qt a umožňuje jednoduché spustenie vlastných procedúr na novom procesorovom vlákne. Trieda Bluetoothreceiver obsahuje jednotlivé merania a k nim párované bluetooth zariadenia. Pre prezentáčné účely tejto práce sú bluetooth zariadenia simulované súbormi, ktoré obsahujú dátu meraní. Po spustení bluetooth prijímač začne získavať dátu zo zariadení (súborov) a priradovať

ich jednotlivým meraniam do ich bufferov a zároveň označuje adresu bufferu s najaktuálnejším zápisom. Bluetooth prijímač beží na inom procesorovom vlákne ako časť aplikácie, ktorá spravuje merania a analyzuje dátu.

Bluetoothreceiver.h

Okrem hlavičiek metód a premenných súbor obsahuje aj implementáciu metódy run(), ktorej kód sa spúšťa v novom procesorom vlákne. Vytvorenie a správu tohto vlákna zabezpečuje Qt framework v triede QRunnable. Pri nastavovaní bluetooth prijímača je nutné zavolať metódy addMeasurement() a addFile() za sebou, pretože súbor, ktorý bude pridaný v metóde addFile() sa automaticky páruje na meranie na rovnakom indexe v zozname meraní.

```

#include <QThreadPool>
#include <QDebug>
#include <QString>
#include <QFile>
#include <QList>

#include <measurement.h>
#include <utils.h>

class BluetoothReceiver : public QRunnable{
    void run() override
    {
        qDebug() << "\nSpustenie bluetooth receivera\n" << QThread::currentThread();
        recieveData();
    }
public:
    /* Konstruktor - vytvorenie objektu bluetooth prijimaca */
    BluetoothReceiver();

    /* Pridanie suboru, z ktoreho bluetooth receiver nacitava data - potrebne len pre prezentaciu */
    int addFile(QString fileNameArg);

    /* Pridanie vytvoreneho merania */
    int addMeasurement(Measurement *m);

    /* Spustenie prijimania dat */
    void recieveData();

    /* Vypis aktualneho nastavenia bluetooth receivera */
    void printStatus();

private:
    /* Zoznam merani */
    QList<Measurement *> merania;

    /* Zoznam suborov, z ktorych bluetooth receiver nacitava data - potrebne len pre prezentaciu */
    QList<QString> filesNames;

    /* Pridanie dat do buffera merania */
    void appendBuffer(double *buffer, int bufferSize, int *gpt, double value);
};


```

Údaje, ktoré obsahuje Bluetoothreceiver:

- merania – zoznam všetkých meraní, do ktorých sa budú zapisovať namerané údaje
- filesNames – zoznam súborov, z ktorých sa budú dátá získavať a ktoré simullujú extrené zariadenia

Bluetooth.cpp

Tento súbor obsahuje implementácie metód, ktoré sú definované v hlavičkovom súbore bluetooth prijímača. Konštruktor tejto triedy je prázdny.

receiveData() – táto metóda je zodpovedná za prijímanie dát z externých bluetooth zariadení.

Parametre: žiadne

Návratová hodnota: žiadna

```
void BluetoothReceiver::recieveData(){

    if(merania.size() != filesNames.size()){
        printf("Bluetooth Receiver FILE ERROR 1: Pocet merani nie je rovnaky ako pocet suborov na citanie\n");
        return;
    }

    int NUM_MES = merania.size();
    const int LINE_CHARS_LEN = 25;
    double lineVal = 0.0;
    char lineChars[LINE_CHARS_LEN];

    QFile *files[NUM_MES];
    for(int i=0; i<NUM_MES; i++){
        files[i] = new QFile(filesNames.at(i));
        if (!files[i]->open(QIODevice::ReadOnly | QIODevice::Text))
            printf("Bluetooth Receiver: FILE ERROR 2: Subor %s neboli otvorený\n", filesNames.at(i).toString().c_str());
    }

    int ended = 0;
    while(1){
        for(int i=0; i<NUM_MES; i++){ // pre vsetky subory a merania
            Measurement *m = merania.at(i); // otvorí sa dane meranie
            if(m->end == '1'){ // ak je uz dane meranie ukoncene, pokracuje sa dalsim meranim
                continue;
            }

            QByteArray line = files[i]->readLine();
            int len =line.size();

            if(len > 0){ // udaje sa spracovavaju len ak sa nejake nacitali
                line.resize(line.size()-1);
                line.replace(", ", ".");
                lineVal = line.toDouble(); // prevod nacitaneho riadku na double
                if(lineVal > 10000){ // ukoncovaci znak merania
                    printf("\nBluetooth: NAJDENY KONIEC\n");
                    ended++; //sa zvysi pocet ukoncenych mernani
                    m->end = '1';
                    continue;
                }

                appendBuffer(m->gBuffer, m->gBufferSize, &m->gpt, lineVal); // pridanie do bufferu nacitaneho merania
            }
            memset(lineChars, '\0', LINE_CHARS_LEN);

        }
        if(ended == merania.size()){
            printf("Bluetooth: Koniec citania suborov\n");
            break;
        }
    }

    for(int i=0; i<NUM_MES; i++){
        files[i]->close();
    }
}
```

Pre potreby prezentácie tohto projektu sú zariadenia simulované súbormi a pri neskoršej práci bude potrebné túto metódu prerobiť. Prezentuje ale funkčné priradovanie dát do bufferov jednotlivých meraní, ktoré sú v programe nastavené. Ako prvé sa skontroluje, či je v objekte načítaný rovnaký počet meraní a dátových súborov. Následne sa všetky dátové súbory otvoria

a spustí sa cyklus, ktorý tieto súbory číta. Ak sa nejaké meranie ukončí – d'alej už nepribúdajú do dátového súboru údaje, tak sa cyklus nekončí ale číta zo všetkých ďalších súborov, ktoré sú priradené neukončeným meraniam. Pre každý súbor sa opakuje postup kedy sa otvorí mernie priradené k danému súboru, zo súboru sa načíta jeden riadok a ak nie je prázdný, tak sa správne naformátuje a priradí sa do bufferu daného merania. Pre prezentačné účely je ukončenie merania značené hodnotou väčšou ako 10 000. Pri reálnom meraní dá externé zariadenie informáciu o ukončení merania iným spôsobom. Ak teda nastane ukončenie merania zo strany extreného zariadenia, nastaví sa pre dané meranie v aplikácii príznak *end* na hodnotu 1.

appendBuffer() – táto metóda zodpovedá za pridávanie nových údajov do bufferu merania.

Parametre:

- double *buffer – smerník na buffer merania
- int bufferSize – veľkosť bufferu merania
- int *gpt – smerník na adresu, ktorá označuje najaktuálnejší údaj v bufferi
- double value – hodnota, ktorá ma byť do buffera vložená

Návratová hodnota: žiadna

```
void BluetoothReceiver::appendBuffer(double *buffer, int bufferSize, int *gpt, double value){  
    double *helpPtr = buffer;  
  
    helpPtr = (buffer + *gpt);  
    *helpPtr = value;  
  
    incGPT(gpt, bufferSize);  
}
```

Ako parametre vstupujú do tejto metódy smerník na buffer, veľkosť bufferu, smerník na údaj hovoriaci o adrese najaktuálnejšieho údaju v bufferi a hodnota, ktorá má byť zapísaná. Hodnota sa zapíše do bufferu a nastaví sa hodnota *gpt*.

addMeasurement() – táto metóda je zodpovedná za pridanie nového merania do bluetooth prijímača.

Parametre:

- Measurement *m – objekt typu Measurement

Návratová hodnota:

- int – úspešnosť pridania merania do bluetooth prijímača

```

int BluetoothReceiver::addMeasurement(Measurement *m){
    for(int i=0; i<merania.size(); i++){
        if(merania.at(i)->mName == m->mName){
            printf("Bluetooth receiver MEASURE ERROR 1: Dane meranie uz je priradene v Bluetooth
                   receiveri\n");
            return 1;
        }
    }
    merania.append(m);
    return 0;
}

```

Ako parameter vstupuje objekt typu Measurement. Podľa názvu merania sa skontroluje, či už dané meranie nie je priradené v bluetooth prijímači. Ak nie, tak sa priradí.

addFile() – táto metóda je zodpovedná za pridanie dátového súboru do bluetooth prijímača.

Parametre:

- QString fileNameArg – názov súboru

Návratová hodnota:

- int – úspešnosť pridania súboru do bluetooth prijímača

```

int BluetoothReceiver::addFile(QString fileNameArg){
    if(filesNames.contains(fileNameArg)){
        printf("Bluetooth receiver FILE ERROR 3: Zadany subor %s uz je pouzivany\n",
               fileNameArg.toStdString().c_str());
        return 1;
    }
    filesNames.append(fileNameArg);
    printf("Pridany subor do merania: %s\n", fileNameArg.toStdString().c_str());
    return 0;
}

```

Využíva sa pre prezentačné účely – dátový súbor simuluje reálne externé zariadenie. Súbor, ktorý sa v tejto metóde pridá do zoznamu, je automaticky párovaný na meranie, ktoré je v zozname meraní na rovnakom indexe. Ako parameter vstupuje do metódy názov súboru, ktorý by už mal obsahovať celú adresu súboru. Skontroluje sa, či už daný súbor nie je pridaný i inému meraniu. Ak nie, priradí sa.

printStatus() – táto metóda je zodpovedná vypísanie aktálneho nastavenia bluetooth prijímača. Vypíše sa zozname meraní a k nim priradených dátových súborov.

Parametre: žiadne

Návratová hodnota: žiadna

```
void BluetoothReceiver::printStatus(){
    printf("\nBluetooth status:\n");
    printf("\tPocet merani: %d\n", merania.size());
    for(int i=0; i<merania.size(); i++){
        printf("\t\t%s : %s\n", merania.at(i)->mName.toStdString().c_str(),
               filesNames.at(i).toStdString().c_str());
    }
}
```

5.1.5. Measurement

Inštancia tejto triedy zodpovedá za správu jedného merania v aplikácii – spravuje typy vín, ktoré sú v meraní rozpoznávané, vytvára snímky časti dát (snímky vín) a drží buffer, v ktorom sa nachádzajú namerané údaje.

Measurement.h

```

#include <QString>
#include <QList>
#include <ann.h>
#include <utils.h>

#ifndef _WIN32
#define DATA_PATH "D:\\Codes\\DP_peakFinder\\data\\"
#endif

#if __APPLE__
#define DATA_PATH "/Users/dejvid/Codes/DP/DP_peakAnalyzer/data/"
#endif

class Measurement{

public:
    /* Vytvorenie noveho merania */
    Measurement(QString mName, int bufferSize);

    /* Vypis aktualneho nastavenia merania */
    void printStatus();

    /* Pridanie typu vlny do merania */
    int addWave(Ann *wave);

    /* Jeden krok pri analyze dat merania */
    int step();

    /* Vytvorenie snapshotu */
    void createSnapShot(double *arr, int dlzka, int actTime, QString mName, double similarity);

    /* Zistenie percenta podobnosti useku dat s vlnami, ktore meranie rozpoznava */
    double snapshotSimilarity(double *arr, Ann *ann);

    /* Vysek z dat */
    double *cutSnapShot(int index, int dlzka);

    /* Nazov merania */
    QString mName;

    /* Zoznam vln v merani */
    QList<Ann*> mWaves;

    /* Velkost bufferu merania */
    int gBufferSize;

    /* Buffer merania */
    double *gBuffer;

    /* Adresa buffera s poziciou najaktualnejsim udajom, ktory pridal bluetooth receiver */
    int gpt = 0;

    /* Počet udajov, ktore boli do buffera pridane od zaciatku merania */
    int gTime = 0;
}

```

```

/* Vzorkovanie merania */
int gSamplingRate = 1;

/* Najvacsia dlzka od th_cross po koniec vlny */
int gMaxWait = 0;

/* Ukoncovaci znak merania */
char end = '0';

const int WAVES = 4;

private:
    /* Vytvorenie snapshotu, ktorý neobsahuje žiadne metadata */
    void createSnapShotRAW(int index, int dlzka);

    /* Počet exportovanych snapshotov */
    int exported = 0;

    /* Adresa s aktualou poziciou, ktorá je prehľadavana v merani */
    int actPointer = gpt;

    /* AKtualna hodnota nacitana z adresy actPointer */
    double val = 0.0;

    /* Zoznam bodov, ktore splnili podmienku najdenia vlny a cakaju na analyzu */
    QList<int> peaks;

    /* Najdena vlna */
    Ann *foundWave;

    /* Data, ktore obsahuju vysek dat */
    double *arr;

    /* Maximalna podobnosť s vlnami daneho useku dat */
    double maxSim = 0.0;

};


```

Údaje, ktoré obsahuje objekt typu Measurement:

- mName – názov merania
- wWaves – zoznam typov vĺn, ktoré sa v meraní rozpoznávajú
- peaks – zoznam bodov, ktoré sú pravdepodobnými vrcholmi vlny
- foundWave – aktuálne nájdená vlna počas analýzy

Measurement.cpp

Measurement() – konštruktor objektu. Ako parametre vstupujú názov merania a veľkosť bufferu, ktorý sa bude pri analýze merania používať. V konštruktore sa alokuje miesto pre tento buffer.

Parametre:

- QString name – názov merania
- int bufferSize – veľkosť bufferu, ktorý sa pri analýze daného merania bude používať

```

Measurement::Measurement(QString name, int bufferSize){
    Measurement::mName = name;
    Measurement::gBufferSize = bufferSize;

    Measurement::gBuffer = (double*)malloc(gBufferSize * sizeof(double));
}

```

step() – táto metóda je zodpovedná na analýzu jedného bodu (údaju) v rámci merania.

Parametre: žiadne

Návratová hodnota:

- int – ukončenie merania alebo čakanie na ďalšie volanie

```

int Measurement::step(){
    if(end == '1' && actPointer == gpt) return 1;
    if(actPointer == gpt) return 0;

    val = gBuffer[actPointer];

    // Pre kazdy bod merania je nutne zistit ci neprekracuje threshold lubovolnej vlny
    // Ak ano, je vytvorený nový objekt peak, ktorý sa bude o dane data starat
    int h = checkIndex(actPointer-1, gBufferSize);
    for(int index=0; index<mWaves.size(); index++){
        Waveinfo *wi = mWaves.at(index)->wi;

        if(wi->waveThreshold >= 0){
            if((gBuffer[h] >= wi->waveThreshold) && val < wi->waveThreshold){
                if(peaks.empty() || peaks.last() != actPointer){
                    peaks.append(actPointer);
                }
            }
        }else{
            if((gBuffer[h] < wi->waveThreshold) && val >= wi->waveThreshold){
                if(peaks.empty() || peaks.last() != actPointer){
                    peaks.append(actPointer);
                }
            }
        }
    }
}

```

```

// ak od najstarsieho najdeneho peaku prebehol cas najdlhsej vlny

if(!peaks.empty() && gMaxWait == checkIndex(actPointer - peaks.at(0), gBufferSize) ){
    for(int index=0; index<mWaves.size(); index++){
        Waveinfo *wi = mWaves.at(index)->wi;
        arr = cutSnapShot(checkIndex(peaks.at(0) - wi->waveTh_cross, gBufferSize), wi->waveLength);
        double sim = snapshotSimilarity(arr, mWaves.at(index));
        if(sim > maxSim){
            maxSim = sim;
            foundWave = mWaves.at(index);
        }
    }
    Waveinfo *wi = foundWave->wi;
    if(maxSim > 0.9){
        createSnapShot(arr, wi->waveLength, gTime - (gSamplingRate * wi->waveLength), QString(wi->waveName), maxSim);
        printf("%s : Exportovana VLNA %s (%lf)\n", mName.toStdString().c_str(), wi->waveName.toStdString().c_str(), maxSim);
    }else{
        createSnapShot(arr, wi->waveLength, gTime - (gSamplingRate * wi->waveLength), QString("artefakt"), maxSim);
        printf("%s : Exportovany artefakt (%lf)\n", mName.toStdString().c_str(), maxSim);
    }
    foundWave = NULL;
    maxSim = 0.0;
    peaks.removeAt(0);
}
incGPT(&actPointer, gBufferSize);
gTime += gSamplingRate;
return 0;
}

```

Údaj, ktorý sa bude analyzovať je určený adresou v bufferi *actPointer*. Ak je táto adresa rovnaká ako *gpt*, znamená to, že od poslednej analýzy bodu merania neboli do bufferu zapísaný žiadnen nový údaj. Preto sa v analýze nepokračuje a čaká sa na ďalšie zavolanie tejto metódy, kedy sa predpokladá, že nejaký nový údaj pribudne. Ak bluetooth prijímač nový údaj zapíše, hodnota *gpt* sa zmení. Ak sú hodnoty *gpt* a *actPointer* rovnaké a zároveň je príznak *end* nastavený na 1, znamená to, že dané meranie bolo ukončené a už sa nebude ďalej spúšťať. Ak predošlé podmienky neboli splnené pokračuje sa analýzou nového údaju.

Pre každý typ vlny, ktorý je v meraní prítomný, sa vyhodnotí, či aktuálny bod nespĺňa podmienku *th_cross*. Ak áno pridá sa do zoznamu bodov, o ktorých sa predpokladá, že môžu byť vrcholom nejakej vlny. Ďalej ak zoznam nájdených predpokladaných vrcholov vln nie je prázdny a zároveň aktuálne analyzovaný bod je od najstaršieho nájdeného vrcholu vlny vzdialenosť *gMaxWait*, vyhodnotí sa či sa v okolí týchto bodov nenachádza vlna.

Pre všetky typy vln prítomných v meraní, sa na základe najstaršieho nájdeného vrcholu vlny vysekne úsek dát a pomocou neurónovej siete sa zistí podobnosť tohto úseku s vlnou. Takto sa nájde maximálna podobnosť s nejakým typom vlny. Ak je podobnosť väčšia ako 90%, vytvorí sa snímka s názvom vlny. Ak je podobnosť nižšia, vytvorí sa snímka s artefaktom.

Na konci sa zvýší hodnota *actPointer* čím sa značí posun v analýze merania. Tiež sa zvýší čas merania na základe vzorkovania.

createSnapShot() – táto metóda je zodpovedná za vytvorenie snímky úseku dát – vytvorenie XML súboru.

Parametre:

- double *arr – úsek dát snímky
- int dlzka – dĺžka pola dát
- int actTime – čas, kedy bola vlna nameraná
- QString name – názov vlny alebo artefaktu
- double similarity – podobnosť úseku dát s daným typom vlny

Návratová hodnota: žiadna

```
void Measurement::createSnapShot(double *arr, int dlzka, int actTime, QString name, double similarity){  
    int max = 0;  
  
    for(int i=0; i< dlzka; i++){  
        if(arr[i] > max) max = arr[i];  
    }  
  
    QDateTime dateTime = QDateTime::currentDateTime().toString().replace(QString(":"), QString("")).replace(QString(" "), QString(" "));  
    QString snapName = QString("export/snap_") + QString(name)+ QString("_") +QString::number(exported++) + QString("_") + dateTime +  
        QString(".xml");  
    QFile file(QString(DATA_PATH) + snapName);  
  
    if(file.open(QIODevice::Append)){  
        QXmlStreamWriter xmlw(&file);  
        xmlw.setAutoFormatting(true);  
        xmlw.writeStartDocument();  
  
        xmlw.writeStartElement("wave");  
        xmlw.writeTextElement("name", name);  
        xmlw.writeTextElement("max", QString::number(max));  
        xmlw.writeTextElement("similarity", QString::number(similarity));  
        xmlw.writeTextElement("time", QString::number(actTime));  
        xmlw.writeStartElement("data");  
        xmlw.writeCharacters("\n");  
        for(int x=0; x<dlzka; x++){  
            xmlw.writeCharacters(QString::number(arr[x]) + "\n");  
        }  
        xmlw.writeEndElement();  
        xmlw.writeEndElement();  
    }  
  
    file.close();  
}
```

Okrem samotných dát sa zapíšu aj metadáta o názve vlny (artefaktu), maximálnej hodnote, percente pravdepodobnosti a čase. Ako parametre vstupujú do tejto metódy pole (úsek) dát, dĺžka tohto pola, čas v meraní, kedy bol úsek nameraný, názov vlny a podobnosť. Na začiatku sa zistí maximálna hodnota úseku dát. Potom sa vytvorí súbor, ktorého názov je naformátovaný: „snap_typVlny_poradie_dátum.xml“. Ďalej sa do súboru zapíšu všetky údaje.

createSnapShotRAW() – táto metóda vytvorí snímu rovnakým spôsobom ako predchádzajúca ale s rozdielom, že nezapisuje do súboru metadáta ale iba samotný úsek dát.

Parametre:

- int nindex – adresa v bufferi, kde sa začína úsek požadovanej snímky
- int dlzka – dĺžka úseku dát snímky

Návratová hodnota: žiadna

```
void Measurement::createSnapShotRAW(int index, int dlzka){
    index = checkIndex(index, gBufferSize);
    int i=0;

    QString dateTime = QDateTime::currentDateTime().toString().replace(QString(":"), QString(" ")).replace(QString(" "), QString(""));
    QString snapName = QString("export/snap_") + dateTime + QString(".xml");
    QFile file(QString(DATA_PATH) + snapName);

    if(file.open(QIODevice::Append)){
        QTextStream out(&file);
        for(i=0; i< dlzka; i++){
            QString val = QString::number(gBuffer[index]);
            out << val << endl;

            index = checkIndex(++index, gBufferSize);
        }
    }

    printf("Exportovany subor:%d\n", exported-1);
    file.close();
}
```

snapshotSimilarity() – táto metóda je zodpovedná za zistenie pravdepodobnosti, že sa v úseku dát nachádza vlna.

Parametre:

- double *arr – smerník na pole s dátami snímky
- Ann *ann – objekt s neurónovou sieťou

Návratová hodnota:

- double similarity – podobnosť úseku dát s daným typom vlny

```
double Measurement::snapshotSimilarity(double *arr, Ann *ann){
    double similarity = 0.0;

    similarity = ann->runNN(arr);

    return similarity;
}
```

Ako parametre vstupujú pole dát a objekt typu Ann, ktorý drží neurónovú sieť daného typu vlny. Návratová hodnota je číslo v intervale <0,1>, ktoré je získané zavolaním funkcie knižnice Genann, ktorá jeden krát spustí výpočet neurónovou sieťou pre daný úsek dát.

cutSnapShot() – táto metóda je zodpovedná za „vyseknutie“ poľa dát z merania.

Parametre:

- int index – adresa začiatku úseku dát, ktoré sa majú skopírovať do nového poľa
- int dlzka – dĺžka úseku dát

Návratová hodnota:

- double *arr – pole s vybraným úsekom dát

```
double *Measurement::cutSnapShot(int index, int dlzka){  
    int helpIndex = index;  
    double *arr;  
  
    arr = (double*)malloc(dlzka * sizeof (double));  
    for(int i=0; i< dlzka; i++){  
        arr[i] = gBuffer[helpIndex];  
  
        helpIndex = checkIndex(++helpIndex, gBufferSize);  
    }  
    return arr;  
}
```

Využíva pri nájdení miesta, kde sa pravdepodobne nachádza nejaká vlna. Ako parametre vstupujú do metódy adresa, kde sa predpokladaná vlna začína a dĺžka vlny pre ktorú sa má vytvoriť snímka. Návratová hodnota je pole dát, ktoré sa nakopírovali z bufferu merania.

addWave() – táto metóda je zodpovedná za pridanie ďalšieho typu vlny na rozpoznávanie do merania.

Parametre:

- Ann *wave – objekt typu vlny

Návratová hodnota:

- int – úspešnosť pridania ďalšieho typu vlny do merania

```

int Measurement::addWave(Ann *wave){
    for(int i=0; i<mWaves.size(); i++){
        if(wave->wi->waveName == mWaves.at(i)->wi->waveName){
            printf("Measurement WAVE ERROR 1: Vlna uz v danom merani je...\n");
            return 1;
        }
    }
    mWaves.append(wave);
    if(wave->loadNN() == 1){
        printf("Measurement WAVE ERROR 2: Neuronovu siet sa nepodarilo nacitat\n");
        return 1;
    }
    if((mWaves.last()->wi->waveLength - mWaves.last()->wi->waveTh_cross) > gMaxWait)
        gMaxWait = mWaves.last()->wi->waveLength - mWaves.last()->wi->waveTh_cross;
    return 0;
}

```

Ako parameter vstupuje objekt typu Ann, ktorý drží o type vlny všetky informácie vrátanie neurónovej siete. Najprv sa skontroluje, či už daný typ vlny nie je v meraní pridaní. Ak nie, pridá sa a zo súboru sa načíta neurónová sieť. Tiež sa zistí či úsek th_cross-koniec vlny nie je pri novej vlne dlhší ako bol doteraz zaznamenaný. Ak áno, táto maximálna hodnota sa zmení.

printStatus() – táto metóda je zodpovedná za výpis aktuálneho nastavenia merania. Vypíše sa zoznam meraní aj s ich názvami. Pre každé meranie sa vypíše zoznam typov vín, ktoré sú meraní rozpoznávané.

Parametre: žiadne

Návratová hodnota: žiadna

```

void Measurement::printStatus(){
    printf("=> %s:\n", mName.toStdString().c_str());
    printf("\tVlny(%d):\n", mWaves.size());
    for(int i=0; i<mWaves.size(); i++){
        printf("\t\t-> %s\n", mWaves.at(i)->wi->waveName.toStdString().c_str());
    }
}

```

5.1.6. Utils

Z tejto triedy sa inštancie nevytvárajú, ponúka len globálne funkcie pre posun indexu v kruhovom bufferi.

Utils.h

Obsahuje len definície funkcií, ktoré zabezpečujú správne správanie sa kruhového bufferu.

```

/* Zvýšenie indexu v kruhovom bufferi */
void incGPT(int *ptr, const int bufferSize);

/* Zniženie indexu v kruhovom bufferi */
void decGPT(int *ptr, const int bufferSize);

/* Kontrola indexu v kruhovom bufferi */
int checkIndex(int index, const int bufferSize);

```

Utils.cpp

Obsahuje implementácie funkcií pre posun indexu v kruhom bufferi.

incGPT() – táto procedúra sa používa pri zvýšení indexu v kruhovom bufferi o 1.

Parametre:

- **int** *ptr – smerník na adresu v bufferi
- **const int** bufferSize – veľkosť bufferu

Návratová hodnota: žiadna

```

void incGPT(int *ptr, const int bufferSize){
    *ptr += 1;
    *ptr >= bufferSize ? *ptr = 0 : *ptr;
}

```

Ako parametre vstupujú do procedúry smerník na integer – adresa v bufferi a veľkosť bufferu. Hodnota na adrese sa zvýší. Ak presiahne veľkosť bufferu, tak sa na adresu, kam ukazuje smerník zapíše hodnota 0 a tak vznikne adresa na začiatku bufferu.

decGPT() – táto procedúra sa používa pri znížení indexu v kruhovom bufferi o 1.

Parametre:

- **int** *ptr – smerník na adresu v bufferi
- **const int** bufferSize – veľkosť bufferu

Návratová hodnota: žiadna

```

void decGPT(int *ptr, const int bufferSize){
    *ptr -= 1;
    *ptr < 0 ? *ptr = bufferSize : *ptr;
}

```

Ako parametre vstupujú do procedúry smerník na integer – adresa v bufferi a veľkosť bufferu. Hodnota na adrese za zníži. Ak hodnota bude záporná, tak sa do tejto hodnoty zapíše veľkosť bufferu zmenšená o 1 a tak vznikne adresa na konci kruhového bufferu.

checkIndex() – táto funkcia sa používa na kontrolu už vypočítaného indexu kruhového bufferu, či sa naložaj nachádza v predpokladanom rozmedzí.

Parametre:

- int index – hodnota, ktorá má byť skontrolovaná
- const int bufferSize – veľkosť bufferu

Návratová hodnota:

- int index – skontrolovaná adresa v bufferi

```
int checkIndex(int index,const int bufferSize){  
    if(index < 0 ) return bufferSize + index;  
    if(index >= bufferSize) return index - bufferSize;  
    return index;  
}
```

Ako parametre vstupuje číslo – index a veľkosť bufferu. Ak je hodnota menšia ako 0, tak sa k nej pripočíta veľkosť bufferu a naopak, ak je hodnota vyššia alebo rovná ako veľkosť bufferu, tak sa od nej veľkosť bufferu odpočíta. Výsledná hodnota sa vracia z tejto funkcie.

5.1.7. Generator.py

Táto časť aplikácie je na rozdiel od prechádzajúcich tried písaná v jazyku Python. Nebol použitý žiadny špeciálny framework ale bolo použitých viacero knižníc, ktoré pomahajú pri generovaní a vizualizácii údajov meraní. V súbore generator.py je implementovaných niekoľko funkcií, ktoré je možné spúšťať samostatne.

Zoznam balíkov potrebných k spusteniu:

- pylab
- numpy
- scipy
- matplotlib

liveGenerating() – táto funkcia vykonáva generovanie signálu naživo.

Parametre:

- subor – Názov súboru, do ktorého sa bude vygenerovaný signál zapisovať.

Návratová hodnota: žiadna

```

def liveGenerating(subor):
    size = 100
    x_vec = numpy.linspace(0, 1, size + 1)[0:-1]
    y_vec = numpy.random.randn(len(x_vec))
    line1 = []
    array = []
    file = open("prikazy/prikazy_" + subor, "r")

    while 1:
        cPrikaz = file.readline()
        if cPrikaz.__contains__("w1"):
            wave = generate_type_1_wave()
            for w in wave:
                array.append(w)

        if cPrikaz.__contains__("w2"):
            cPrikaz = ""
            #arr = [1000, 2500, 1700, 300, 1000]
            arr = [random.randint(950, 1050), random.randint(1950, 2050),
                    random.randint(1650, 1750), random.randint(250, 350), random.randint(950, 1050)]
            arr_sampled = signal.resample(arr, 30)
            arr_sampled *= -1
            arr_sampled += 1000
            for k in arr_sampled:
                array.append(k)

        if cPrikaz.__contains__("a"):
            cPrikaz = ""
            aType = random.randint(0, 4)
            if aType == 0:
                arr = numpy.random.normal(0, 100, random.randint(15, 50))
            elif aType == 1:
                arr = numpy.random.normal(0, 500, random.randint(15, 50))
            elif aType == 2:
                arr = numpy.random.normal(0, 1000, random.randint(15, 50))
            elif aType == 3:
                arr = numpy.random.normal(0, 1300, random.randint(15, 50))
            else:
                arr = numpy.random.normal(0, 700, random.randint(15, 50))
            for k in arr:
                array.append(k)

        if cPrikaz.__contains__("o"):
            foutMeranie = open("/Users/dejvid/Codes/DP/DP_peakAnalyzer/data/bluetoothData/" + subor, "a")
            foutMeranie.write(str(110000) + "\n")
            foutMeranie.close()
            cPrikaz = ""
            break

        if len(array) > 0:
            y_vec[-1] = array[0]
            foutMeranie = open("/Users/dejvid/Codes/DP/DP_peakAnalyzer/data/bluetoothData/" + subor, "a")
            foutMeranie.write(str(array[0]) + "\n")
            foutMeranie.close()
            array.pop(0)
        else:
            rand_val = numpy.random.normal(0, 20)
            y_vec[-1] = rand_val
            foutMeranie = open("/Users/dejvid/Codes/DP/DP_peakAnalyzer/data/bluetoothData/" + subor, "a")
            foutMeranie.write(str(rand_val) + "\n")
            foutMeranie.close()

    line1 = live_plotter(x_vec, y_vec, line1)
    y_vec = numpy.append(y_vec[1:], 0.0)

```

Ako parameter do funkcie vstupuje názov súboru, do ktorého sa budú vygenerované údaje zapisovať. Tento istý názov sa použije aj pri otvorení súboru, ktorý sa využíva pre zadávanie príkazov pri generovaní signálu. Generátor sa ovláda spôsobom, že používateľ zapisuje do súboru príkazy a na základe nich sa vygeneruje nejaká vlna alebo artefakt. Ak v súbore s príkazmi nie je žiadnen nový príkaz, generuje sa len šum.

Po úvodnej inicializácii sa spustí hlavný cyklus ktorý načíta príkazy zo súboru a podľa nich generuje signál. Zoznam príkazov:

- w1 – vygenerovanie vlny typu 1
- w2 – vygenerovanie vlny typu 2
- a – vygenerovanie artefaktu
- q – ukončenie generovania

Po vygenerovaní požadovanej sekvencie dát, sa zapíšu do súboru a zobrazia na grafe. Zobrazovanie sa vykonáva zavolaním funkcie *live_plotter()*.

generate_type1_wave() – táto funkcia je zodpovedná za vygenerovanie jednej vlny typu 1.

Parametre: žiadne

Návratová hodnota:

- ecg – pole s vygenerovanou vlnou

```

def generate_type_1_wave():
    print('Simulating heart ecg')

    # The "Daubechies" wavelet is a rough approximation to a real,
    # single, heart beat ("pqrst") signal
    pqrst = signal.wavelets.daub(10)

    # Add the gap after the pqrst when the heart is resting.
    samples_rest = 10
    zero_array = numpy.zeros(samples_rest, dtype=float)
    pqrst_full = numpy.concatenate([pqrst, zero_array])
    pqrst_full = pqrst_full[:12]

    # Simulated Beats per minute rate
    # For a health, athletic, person, 60 is resting, 180 is intensive exercising
    bpm = 60
    bps = bpm / 60

    # Simulated period of time in seconds that the ecg is captured in
    capture_length = 1

    # Calculate the number of beats in capture time period
    # Round the number to simplify things
    num_heart_beats = int(capture_length * bps)

    # Concatenate together the number of heart beats needed
    ecg_template = numpy.tile(pqrst_full, num_heart_beats)

    # Add random (gaussian distributed) noise
    noise = numpy.random.normal(0, 0.01, len(ecg_template))
    ecg_template_noisy = noise + ecg_template

    # Simulate an ADC by sampling the noisy ecg template to produce the values
    # Might be worth checking nyquist here
    # e.g. sampling_rate >= (2 * template sampling rate)
    sampling_rate = 50.0
    num_samples = sampling_rate * capture_length
    ecg_sampled = signal.resample(ecg_template_noisy, int(num_samples))

    # Scale the normalised amplitude of the sampled ecg to whatever the ADC
    # bit resolution is
    # note: check if this is correct: not sure if there should be negative bit values.
    adc_bit_resolution = 1024
    ecg = adc_bit_resolution * ecg_sampled
    return ecg

```

Pri generovaní vlny typu 1 sa využije funkcia knižnice signal a vygeneruje sa kostra vlny. Táto kostra sa ďalej upravuje pridaním šumu, pridaním ďalších bodov na základe vzorkovania a tiež zväčšením rozlíšenia.

appendNoise() – táto funkcia vytvára snímky šumu a pridáva ich do tréningového datasetu neurónovej siete.

Parametre:

- length – veľkosť snímky

Návratová hodnota: žiadna

```
def append_noise(length):
    lines = []
    with open("vlny.csv", 'r') as file:
        lines = file.readlines()
    file.close()

    noised_range = [0, len(lines) - 1] # rozsah riadkov kde sa bude generovať sum

    for i in range(0, len(lines)):
        i = 0
        noise = numpy.random.normal(0, 20, length)
        lineNum = random.randint(noised_range[0], noised_range[1])
        print(lineNum)

        line = ""
        for c in noise:
            c = "%-.4f" % c
            l = str(c).replace('.', ',')
            line += l + ";"

        line += "\n"
        print(line)

        lines.insert(lineNum, line)
        noised_range[1] += 1

    with open("noised_vlny.csv", 'w') as file:
        for l in lines:
            file.write(l)
    file.close()
```

Funkcia otvorí súbor a načíta všetky snímky do poľa. Následne vygeneruje počet snímok šumu rovnaký ako bol pôvodný počet snímok v súbore. Novo-vygenerované snímky ukladá na náhodné miesta medzi pôvodné snímky. Následne celý dataset uloží do nového súboru.

5.2. Používateľská príručka

Táto kapitola je venovaná použitiu aplikácie z pohľadu používateľa. Jedná sa len o spustenie prezentačného módu aplikácie keďže finálne použitie sa predpokladá s grafickým rozhraním, ktoré je predmetom inej diplomovej práce. Sú tu teda zhrnuté informácie ako správne spustiť

konzolovú časť aplikácie. Tiež je v tejto kapitole vysvetlené ako správne spojiť generátor signálov s analyzátorom cez dátové súbory.

5.2.1. Použitie analyzátoru

Aplikácia analyzátoru nemá grafické rozhranie ale spúšťa sa len z konzoly počítača. Pre správne fungovanie je potrebné mať skompilovaný binárny súbor. Pre kompliaciu na vlastnom stroji je potrebné mať nainštalovaný framework Qt vo verzii 5.11.2. Tiež sme použili IDE Qt Creator vo verzii 4.7.1.

Pre spustenie generátoru zadajte v konzole príkaz: **./signalAnalyzer**

5.2.1.1. Zobrazenie analyzátoru dát

Po spustení sa otvorí okno prezentačnej aplikácie. Ako prvé sa zobrazí názov aplikácie a zoznam príkazov, ktoré môže používateľ zadávať.



Main>: █

Okno prezentačnej aplikácie

Riadok s označením **Main>**: označuje miesto, kam môže používateľ písat príkazy.

5.2.1.2. Ovládanie analyzátoru dát

Príkaz **h**

Pre zobrazenie legendy s príkazmi zadajte **h**. Jednotlivé príkazy sa tiež zobrazia hned po štarte aplikácie.

Príkaz **addm**

Pre vytvorenie nového merania zadajte príkaz **addm**. Aplikácia dokáže spravovať ľubovoľný počet paralelných meraní. Každé z meraní môže pracovať s jedným alebo viacerými typmi vln. Každé meranie ale môže byť napojené len na jedno bluetooth zariadenie (v prezentačnom móde to je dátový súbor s vygenerovaným meraním).

Po zadaní príkazu je používateľ vyzvaný k zadaniu názvu merania a dátového súboru, z ktorého bluetooth prijímač bude získavať dátu.

```
Main>: addm
Zadaj nazov merania: Meranie_Vlny
Zadaj nazov suboru pre bluetooth receiver: meranie1.txt
Pridany subor do merania: /Users/dejvid/Codes/DP/DP_peakAnalyzer/data/bluetoothData/meranie1.txt
Main>: █
```

Zadanie príkazu *addm*

Príkaz *w2m*

Pre pridanie typu vlny do merania zadajte príkaz *w2m*. Po vytvorení merania je ešte nutné určiť jeden alebo viac typov vln, ktoré budú v meraní rozoznávané. Pre pridanie viacerých typov vln zadajte tento príkaz opakovane.

Po zadaní príkazu je používateľ vyzvaný aby zadal názov merania, do ktorého chce vlnu pridať a tiež názov vlny (resp. meno konfiguračného súboru s vlnou)

```
Main>: w2m
Zadaj nazov merania: Meranie_Vlny
Zadaj nazov vlny: wave1.xml
Konfiguracny subor vlny nacitany
Neuronova siet /Users/dejvid/Codes/DP/DP_peakAnalyzer/data/ann/wave1.ann nacitana
Zadanie príkazu w2m
```

Príkaz *status*

Pre zobrazenie aktuálneho nastavenia aplikácie, zadajte príkaz *status*. Zobrazí sa výpis meraní a k nim priradených vln. Tiež sa zobrazí aktálne nastavenie bluetooth prijímača.

```
Main>: status
-
Pocet merani: 1
=> Meranie_Vlny:
    Vlny(1):
        -> type1

Blueatooth status:
    Pocet merani: 1
        Meranie_Vlny : /Users/dejvid/Codes/DP/DP_peakAnalyzer/data/bluetoothData/meranie1.txt
Zadanie príkazu status
```

Príkaz *start*

Pre spustenie analýzy meraní, zadajte príkaz *start*. Týmto príkazom sa aplikácia prepne z konfiguračného stavu do analyzujúceho. V tomto bode sa nedajú pridávať nové vlny ani nijako meniť nastavenie programu. Prebieha analýza dát a vytvárajú sa snímky vln. Po ukončení analýzy sa aplikácie prepne opäť do konfiguračného módu.

```

Main>: start

Spustenie bluetooth receivera
    QThread(0x7fed0de00f80, name = "Thread (pooled)")
Meranie_Vlny : Exportovana VLNA type1 (0.983247)
Meranie_Vlny : Exportovany artefakt (0.000654)
Meranie_Vlny : Exportovana VLNA type1 (0.985391)
Meranie_Vlny : Exportovana VLNA type1 (0.972727)
Meranie_Vlny : Exportovany artefakt (0.000308)
Meranie_Vlny : Exportovany artefakt (0.013889)
Meranie_Vlny : Exportovany artefakt (0.000903)
Meranie_Vlny : Exportovany artefakt (0.000122)
Meranie_Vlny : Exportovany artefakt (0.001218)
Meranie_Vlny : Exportovany artefakt (0.000617)
Meranie_Vlny : Exportovana VLNA type1 (0.982881)

Bluetooth: KONIEC
Main: koniec merania
Main: Ukoncene vsetky merania

```

Príklad spustenia analyzátoru

Príkaz *exit*

Pre ukončenie aplikácie zadajte príkaz *exit*.

5.2.2. Použitie generátoru

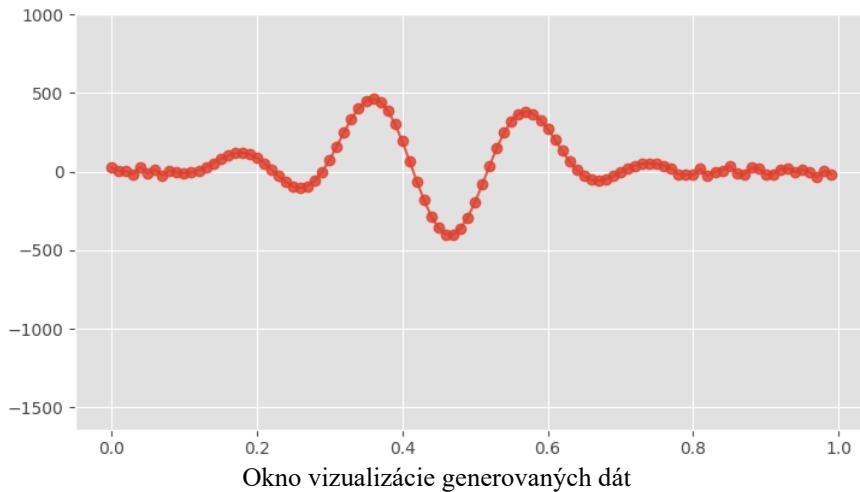
Samotná aplikácia generátoru nemá grafické rozhranie a je spúšťaná len z konzoly počítača. Pre správne fungovanie odporúčame vytvoriť virtuálne prostredie a nainštalovať nasledovné balíky:

- pylab
- numpy
- scipy
- matplotlib

Pre spustenie spusťte program príkazom: **python generator.py <súbor.txt>** pričom ako parameter zadajte súbor, do ktorého sa budú generovať dátá. Zároveň

5.2.2.1. Zobrazenie generovaného signálu

Po spustení generátoru sa zobrazí okno, ktoré vizualizuje simulované meranie pacienta. Ak sa skript spustí paralelne viac krát, otvorí sa pre každú inštanciu nové, samostatné okno. Toto okno sa používa len na zobrazovanie dát a nemá žiadne aktívne elementy, s ktorými by mohol používateľ interagovať.



5.2.2.2. Ovládanie generátora

Pri spustení skriptu s generátorom bol zadaný ako argument názov súboru. V priečinku s generátorom sa vytvorí nový súbor v tvare: **prikazy_<zadaný súbor>.txt**. Toto je ovládací súbor generátoru. Používateľ do neho píše príkazy, ktorými generuje buď vlny alebo artefakty.

w1 – Príkaz pre vygenerovanie vlny typu 1

w2 – Príkaz pre vygenerovanie vlny typu 2

w3 – Príkaz pre vygenerovanie vlny typu 3

a – Príkaz pre vygenerovanie artefaktu

q – Príkaz pre ukončenie simulácie

Každý príkaz sa zadáva na samostatný riadok. Pre potvrdenie zadania príkazu, súbor opakovane uložte.

```
prikazy_mernaie1.txt
1 w1
2 w2
3 a
4 w3
5 w2
6 a
7 w1
8 w3
9 w2
10 q
11
```

Príklad súboru so zadanými príkazmi pre generátor dát

6. Zoznam použitej literatúry

- [1] SÁRKÁNYOVÁ, M.: Neuromonitoring, 2017.
- [2] MOLLER, A. R.: Intraoperative Neurophysiological Monitoring, 2006, ISBN 1588297039.
- [3] Jameson LC, Sloan TB. Monitoring of the brain and spinal cord. Anesthesiol Clin. 2006;24(4):777–91.
- [4] Jameson LC, Sloan TB. Neurophysiologic monitoring in neurosurgery. Anesthesiol Clin. 2012;30(2):311–31.
- [5] Sala F, Lanteri P, Bricolo A. Motor evoked potential monitoring for spinal cord and brain stem surgery. Adv Tech Stand Neurosurg. 2004;29:133–69
- [6] CRAM, J.R., KASMAN, G.S., Holtz J.: Introduction to Surface, Electromyography. Aspen Publishers Inc.; Gaithersburg, Maryland, 1998.
- [7] HUSAIN, A. M.: A Practical Approach to Neurophysiologic Intraoperative Monitoring, 2008, ISBN 9781933864, 1933864095
- [8] AANEM: Recommended Policy for Electrodiagnostic Medicine American Association of Neuromuscular & Electrodiagnostic Medicine, 2011, https://www.aanem.org/getmedia/3275d71c-81dc-4b23-96a7-03173ecf8446/Recommended_Policy_EDX_Medicine_062810.pdf.
- [9] KLEISSEN, R. F. M., BUURKE, J.H., HARLAAR, J., ZILVOLD, G.: Electromyography in the biomechanical analysis of human movement and its clinical application. Gait Posture. 1998;8(2):143–158. doi: 10.1016/S0966-6362(98)00025-3.
- [10] Jee Honq Quach: Surface Electromyography: Use, Design & Technological Overview, Concordia University, Project report in partial fulfillment of: ENGR 6191, Introduction to Biomedical Engineering, 2007
- [11] Storn, R., Price, K. Differential Evolution a Simple and Efficient Heuristic for Global Optimization. J. Global Optimization, 1997
- [12] Kvasnička, V., Pelikán, M., Pospíchal, J.: Hill climbing with learning (an abstraction of genetic algorithm). Neural network world, 1996
- [13] Kvasnička, V., Beňušková, L., Pospíchal, J., Farkaš, I., Tiňo, P., Kráľ, A.: Úvod do teórie neurónových sietí. IRIS, Bratislava, 1997
- [14] Fausett, L. V.: Fundamentals of Neural Networks. PrenticeHall, Inc., Englewood Cliffs, New Jersey, 1994
- [15] Hussain, T.: Modularity within neural networks. Queen's University Kingston, Ontario, Canada, 1995
- [16] Beale, R. - Jackson, T.: Neural Computing: An Introduction. J W Arrowsmith Ltd, Bristol, Great Britain, 1992
- [17] MOHYLOVÁ, J. – KRAJČA, V.: Zpracování biologických signálů. Ostrava, 2007. ISBN 978-80-248-1491-9.
- [18] KISHNER, S., ELLIOTT, L.: Electromyography and Nerve Conduction Studies, 2015, <http://emedicine.medscape.com/article/2094544-overview>.
- [19] Genann neural networks, <https://github.com/codeplea/genann>
- [20] OpenNN, www.opennn.net

- [21] MUDr. Martin Kucharík, MUDr. Ján Tomka, CSc., MHA, MUDr. Ivan Vulev, PhD.: Periprocedurálny neurologický manažment operácií a endovaskulárnych intervencií na prívodných mozgových tepnách, *Vask. med.*, 2016, 8(1): 14–16
- [22] Ryad Zemouri, Noureddine Zerhouni, Daniel Racoceanu,: "Deep Learning in the Biomedical Applications: Recent and Future Status", *Applied Sciences*, vol. 9, pp. 1526, 2019.
- [23] Phinyomark, A., Scheme, E.: "EMG Pattern Recognition in the Era of Big Data and Deep Learning", *Big Data and Cognitive Computing*, vol. 2, pp. 21, 2018.
- [24] M. Heywood, P. Noakes,: "A framework for improved training of Sigma-Pi networks", *Neural Networks IEEE Transactions on*, vol. 6, no. 4, pp. 893-903, 1995.
- [25] N.Y. Nikolaev, H. Iba, "Learning polynomial feedforward neural networks by genetic programming and backpropagation", *Neural Networks IEEE Transactions on*, vol. 14, no. 2, pp. 337-350, 2003.

Príloha A Plán práce na diplomovej práci 1

1. Analýza chorôb a diagnóz pacientov neurodegeneratívnymi ochoreniami
2. Analýza spôsobov liečby neurodegeneratívnych ochorení
3. Analýza využitia neuromonitoringu pri vyšetreniach
4. Analýza druhov a techník neuromonitoringu
5. Analýza dát a signálov získaných pri neuromonitoringu pacienta
6. Analýza EMG záznamu a faktorov ovplyvňujúcich záznam signálu
7. Analýza získavania dát a ich prevod do digitálnej formy
8. Analýza metód a spôsobov vyhodnocovania signálov
9. Analýza spôsobov klasifikácie tvaru impulzu
10. Analýza neurónových sietí (1/2)
11. Analýza neurónových sietí (2/2)
12. Spísanie návrhu riešenia

Pri vytváraní analýzy sa odporúča dodržať stanovené poradie krokov ale nie je to nevyhnutné.

Príloha B Plán práce na diplomovej práci 2

1. Oprava chýb, ktoré vedúci spomenul v posudku za predchádzajúci semester
2. Vytvorenie návrhu dátového súboru, ktorý opisuje typ vlny
3. Návrh a implementácia štruktúry dátového priečinku pre analyzátor meraní
4. Implementácia generátoru dát (1/3)
5. Implementácia generátoru dát (2/3)
6. Implementácia generátoru dát (3/3)
7. Vytvorenie prvých datasetov so simulovanými meraniami
8. Implementácia prototypu analyzátoru merania (1/3)
9. Implementácia prototypu analyzátoru merania (2/3)
10. Implementácia prototypu analyzátoru merania (3/3)
11. Testovanie prototypu
12. Dokončenie písania výsledkov práce po semestri

Pri implementovaní jednotlivých komponentov práce nie je nutné striktne dodržiavať postupnosť všetkých krokov. Vykonanie niektorých krokov sa môže časovo lísiť od predpokladaného stavu.

Príloha C Plán práce na diplomovej práci 3

1. Rozšírenie návrhu zadania
2. Implementácia ďalších typov vĺn v generátore
3. Implementácia generovania dát v reálnom čase (1/2)
4. Implementácia generovania dát v reálnom čase (2/2)
5. Implementácia vyhľadávania viacerých typov vĺn súčasne (1/2)
6. Implementácia vyhľadávania viacerých typov vĺn súčasne (2/2)
7. Implementácia paralelnej analýzy viacerých meraní
8. Vytvorenie testovacích scenárov
9. Testovanie simulovaného merania v reálnom čase
10. Testovanie analyzátoru pri analýze signálov v reálnom čase
11. Dokončovanie písania výsledkov práce dokumentu (1/2)
12. Dokončovanie písania výsledkov práce (2/2)

Pri implementovaní jednotlivých komponentov práce nie je nutné striktne dodržiavať postupnosť všetkých krokov. Vykonanie niektorých krokov sa môže časovo lísiť od predpokladaného stavu.

Príloha D Zhodnotenie plnenia práce v jednotlivých etapách

Na konci prvej fázy sme mali vytvorenú dostatočne obsiahlu analýzu, na základe ktorej sme vedeli vyhodnotiť ďalší postup v práci. V tejto fáze sme sa tiež poradili lekárkou, ktorá nás usmernila v oblasti medicínskych tém a vysvetlila nám aktuálny stav.

V ďalšej fáze sme pracovali na implementácii prototypu aplikácie. Vytvorili sme generátor dát a jednoduchý analyzátor. V tomto bode sme však pracovali len s jedným typom vlny a generovanie s analýzou dát neprebiehalo v reálnom čase.

V poslednej fáze sme doplnili generátor o ďalšie typy vln. Takisto sme rozšírili analyzátor o paralelné spracovanie viacerých meraní a tiež sme doplnili vyhľadávanie viacerých typov vln v jednom meraní. Generovanie a analýza dát už prebiehalo v reálnom čase

Možno skonštatovať, že všetky časti plánu práce boli splnené včas. Väčšina krokov bolo dodržaných v stanovenom poradí, avšak v niektorých prípadoch nastala zmena poradia ich naplnenia. Tiež sa vyskytli prípady kedy došlo k zmene funkcionálit. Tieto zmeny sú popísané v kapitole 3.3, *Zmeny v návrhu riešenia*.

Príloha E Opis digitálnej časti práce

Evidenčné číslo práce v informačnom systéme: FIIT-136227-71997

Obsah digitálnej časti práce (archív ZIP):

<i>Priečinok</i>	<i>Opis</i>
/generator	Obsahuje súbory potrebné ku generátoru dát
/prikazy	Obsahuje súbory pre zadávanie príkazov
generator.py	Python skript s programom
requirements.txt	Zoznam potrebných balíkov
/signal_analyzer	Zdrojové súbory a dáta analyzátoru signálu
/data	Dátový priečinok opísaný v dokumentácii
/src	Priečinok so zdrojovými kódmi aplikácie
/praca-pdf	pdf verzia záverečnej správy
/praca.pdf	

Názov odovzdaného archívu: DP_prilohy_digital_beno.zip