Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Katedra informatiky a výpočetní techniky

BAKALÁŘSKÁ PRÁCE

Plzeň, 2012

Jan Strejc

Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Katedra informatiky a výpočetní techniky

Bakalářská práce

Klasifikace ERP komponent Kohonenovou neuronovou sítí

Plzeň 2012 Jan Strejc

Zadání

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

.....

V Plzni dne 11.5.2012, Jan Strejc

Abstract

In this thesis I will deal with the processing of brain activity. This problem falls within the area, which is called neuroinformatics. Brain activity is usually measured by the method known as electroencephalography (EEG) and I'll be interested in the effects concerning event-related potentials (ERP). ERP is electrically detectable brain response to specific stimulation (visual, auditory). There are many ways of searching event-related potentials in the EEG, but none of them does universal and reliable detection.

The purpose of this work is to try using Kohonen neural network to look for the ERP. Neural networks are used to solve problems, which are unsolvable by algorithmical means, but are relatively easy for humans, by trying to simulate human brain. For example pattern classification and recognition, signal processing and function approximation.

For classification purposes signal will be decomposed to parts by matching pursuit algorithm, which creates number of atoms representing the signal. Neural network will be used to group similar atoms together.

The goal is to test if it is possible to use neural network to find the ERP components in EEG signal.

Obsah

1 Úvod

V této práci se budu zabývat zpracováváním mozkové aktivity. Tato oblast spadá do odvětví, které se nazývá neuroinformatika. Činnost mozku se zpravidla měří metodou zvanou elektroencefalografie (EEG) a já se budu zajímat o jevy týkající se evokovaných potenciálů, anglicky event-related potentials (ERP). ERP jsou elektricky zaznamenatelné reakce mozku na určité stimulace (vizuální, zvukové). Způsobů jak hledat evokované potenciály v EEG záznamu existuje mnoho, ale žádný zatím neposkytuje univerzální a spolehlivou schopnost jejich detekce.

Mým cílem je otestovat možnost jejich hledání s využitím neuronových sítí, konkrétně Kohonenovou neuronovou sítí. Neuronovými sítěmi lze řešit úlohy, které se tradičními algoritmickými metodami řeší velmi špatně, hledání evokovaných potenciálů je jednou takovou úlohou. Neuronové sítě se snaží napodobit chování mozku a tak řešit problémy, které člověk zvládá řešit relativně snadno, ale pro počítač jsou normálně prakticky neřešitelné. Mezi tyto úlohy patří například klasifikace a rozpoznávání vzorů, zpracování signálů, aproximace funkcí a předpovídání časových řad.

Signál naměřený pomocí EEG bývá často zašuměný a obsahuje artefakty, proto bude nutné použít různé filtry pro jejich odstranění.

Aby mohla být provedena klasifikace ERP komponent musí se napřed vhodným způsobem naměřený signál předzpracovat a extrahovat z něho vhodné příznaky. Zde se nabízí algoritmus matching pursuit, který rozkládá vstupní signál na části reprezentující trend signálu (atomy), a ty již bude možné klasifikovat neuronovou sítí.

Neuronová síť dokáže roztřídit atomy podle jejich podobnosti a bude nutné vytvořit algoritmus, který v těchto skupinách (shlucích) podobných atomů bude hledat případné ERP komponenty.

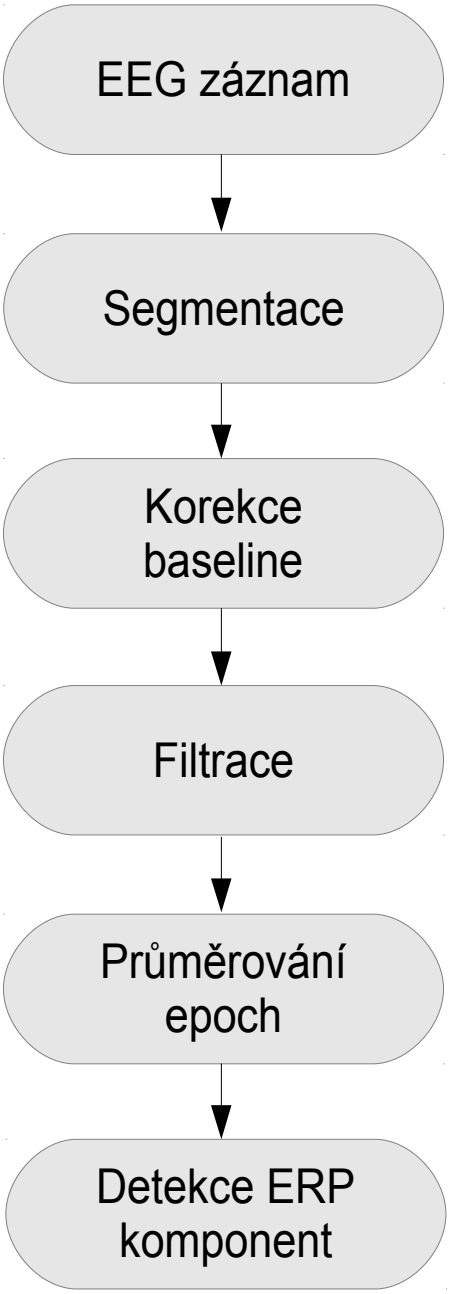
Další důležitou částí této práce bude vizualizace výsledků, pro jejich přehledné zobrazení bude vytvořeno grafické uživatelské rozhraní.

V ideálním případě tato metoda umožní nalezení komponent evokovaných potenciálů v EEG signálu, jako další možné využití se jeví její použití jako filtru pro odstranění nežádoucích částí signálu (artefakty, šum). ERP komponenty v rámci jednotlivých epoch by si měli být podobné, proto by je neuronová síť měla klasifikovat do stejného shluku. Ten pak lze považovat za podstatnou část signálu a zbytek tak můžeme odstranit.

2 Teoretická část

2.1 Dekompozice problému

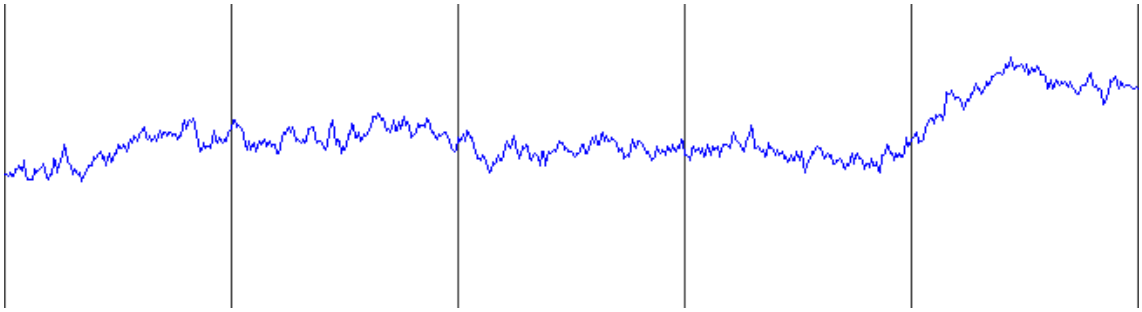
Následující diagram (Obrázek 2.1) zobrazuje dekompozici problému. Jednotlivé části budou dále podrobně popsány. Pořadí není nutné dodržet, například filtraci je možno provádět ještě před segmentací.



Obrázek 2.1: Dekompozice problému

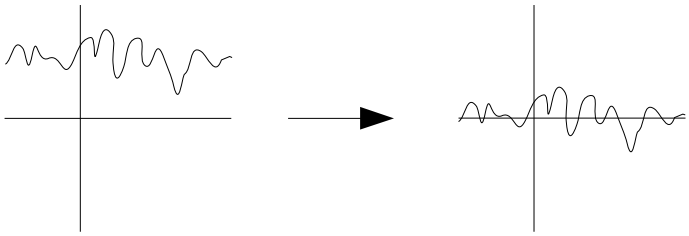
2.2 EEG signál

Elektroencefalografie (EEG) je metoda pro měření aktivity mozku. EEG signál (Obrázek 2.2) je měřen pomocí elektrod připojených na hlavu testovaného člověka pomocí speciální čepice. Zaznamenává se amplituda signálu, která je řádově v desítkách mikrovoltů. Testovaný člověk je stimulován (například se dívá na několik různobarevně blikajících diod) a chvíle kdy došlo k bliknutí se také zaznamenává a později poslouží k rozdělení signálu na epochy.

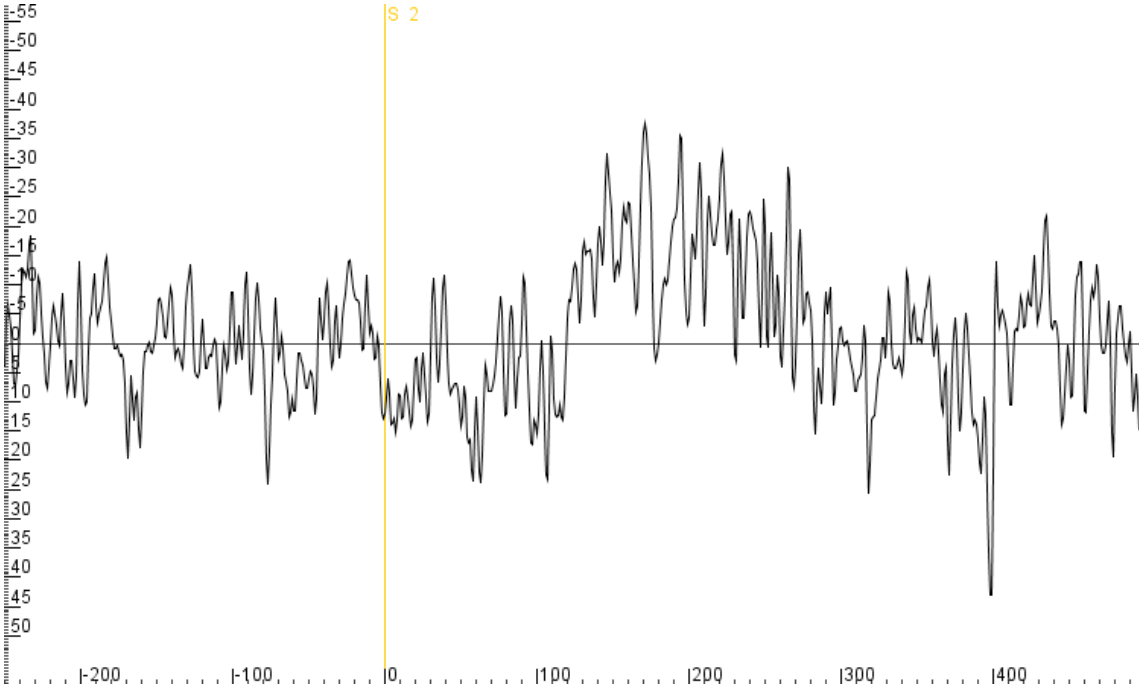


Obrázek 2.2: EEG Signál

- **Kanál** - Představuje signál nahraný z jedné elektrody, každá elektroda nahrává jiný signál.
- **Marker** – Představuje okamžik kdy došlo ke stimulaci. Může být spojen jen s některými kanály, ale často je spojen se všemi (některé stimulace vyvolávají změny v signálu, kterou je možno měřit jen na některých místech hlavy a tak hledat tyto změny na všech kanálech by nemělo smysl).
- **Epocha** – Pro účely klasifikace je nutné provést segmentaci, která rozdělí signál jednoho kanálu na krátké části kolem naměřených markerů – epochy (Obrázek 2.4). Zpravidla to bývá například 500 ms před markerem a 1000 ms po markeru. Na epochách se provádí ještě korekce baseline, která má za úkol srovnat signál v epochách na stejnou úroveň kolem 0 (Obrázek 2.3).



Obrázek 2.3: Korekce baseline



Obrázek 2.4: Epocha

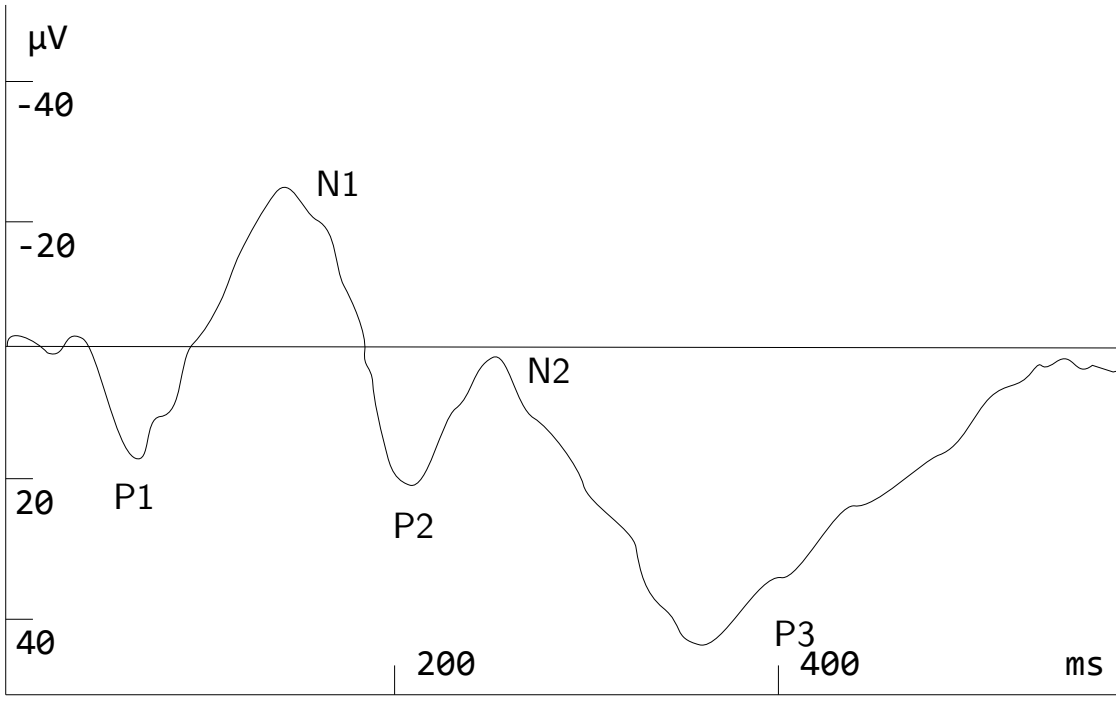
2.3 ERP Komponenty

Cílem je najít v signálu ERP komponenty (event-related potentials) neboli evokované potenciály, jsou diskrétní změny elektrické aktivity nervového systému (zejména mozku).

Značí se běžně podle polarity a pořadí v epoše – P pro pozitivní, N pro negativní vlnu a C pro komponentu, která může mít různou polaritu. Číslo za názvem komponenty představuje buď čas v milisekundách, kde se komponenta nachází, nebo pořadí komponenty v epoše. (Obrázek 2.5). Konvencí je zobrazovat zápornou osu nahoru a kladnou dolu. [1]

Mým cílem je primárně hledat *N1* a *P3*, protože jsou nejvýraznější.

- **P3 (P300)** – Pozitivní vlna evokovaného potenciálu, vyskytující se zhruba 300 ms od stimulace.
- **N1** – Vizuální negativní evokovaný potenciál. Vrchol komponenty je okolo 100 až 200 ms po stimulu.



Obrázek 2.5: ERP komponenty

Pro hledání ERP komponent se běžně používá následujících metod:

- **Průměrování epoch** – po zprůměrování signálu všech epoch vyniknou ty části které se nacházejí ve většině epoch a jsou si podobné.
- **Metoda korelace** – průměrováním získáme napřed vzorovou komponentu, kterou hledáme, tu pak posouváme po signále zkoumané epochy a měříme korelaci mezi oběma signály. V případě vysoké míry shody si zapamatujeme pozici posouvaného signálu jako nalezenou komponentu. [2]
- **Waveletová transformace** – viz [2].
- **PCA a ICA** – Obě metody hledají ERP komponenty přes všechny elektrody, pokud tedy nemáme k dispozici signál více elektrod, nebo se komponenta vyskytuje pouze na některých elektrodách není tato metoda použitelná. Více informací v [1].
- **Matching Pursuit** – viz 2.5 a [2]. Na rozdíl od PCA a ICA je možné jej použít na signál pouze jedné elektrody. Já budu pro detekci komponent vzniklé atomy klasifikovat Kohonenovou neuronovou sítí.

2.4 Filtry

Naměřený signál je plný šumu a obsahuje artefakty (například mrknutí), proto je nutné jej filtrovat. Epochy které obsahují artefakty je potřeba odstranit. [3]

2.4.1 Jednoduchý filtr prahu napětí

Projede celou epochu a testuje jestli signál někde nepřesáhl zvolené maximum, nebo minimum. Pokud ano je taková epoch odstraněna.

2.4.2 Pohybující se okénko

Složitější varianta předchozího filtru, Na začátku se zvolí délka okénka v milisekundách, o kolik se posune v každém kroku a maximální rozdíl napětí signálu v rámci testovaného okénka. V každém kroku se testuje jestli rozdíl mezi nejnižší a nejvyšší hodnotou signálu uvnitř okénka nepřesáhne zvolené maximum, pak se okénko posune a testuje se znovu dokud nedojede nakonec. Pokud testovaná hodnota přesáhne maximum je epocha odstraněna.

2.4.3 Průměrování signálu

Pro odstranění šumu je možné použít průměrování signálu, kdy se spočítá nová hodnota signálu pro každý bod jako průměr jeho okolí. Pokud se zvolí okolí příliš velké může dojít ke ztrátě detailů v signálu (bude příliš vyhlazen). Filtr je možné používat opakovaně, ale také se musí dát pozor aby se ze signálu neodstranily důležité detaily.

2.4.4 FIR filtr

Diskrétní lineární filtr s konečnou impulzní odezvou (finite impulse response). Složí k výběru určitých složek (frekvenčních oblastí) signálu a k potlačení jiných. Každý signál lze rozložit na množinu sinusových funkcí s různou frekvencí a fází, které když se sečtou tak dají dohromady původní signál. Různé typy filtrů mají za úkol potlačit nebo propustit určité frekvence v signálu. [1]

Hlavní typy jsou:

- **Dolní pásmová propusť** – má za úkol odstranit vysoké frekvence a propustit nízké.
- **Horní pásmová propust** – má za úkol odstranit nízké frekvence a propustit ty vysoké.
- **Band-pass filtr** – má za úkol propustit frekvence v určitém intervalu a odstranit ty mimo něj.

2.5 Matching pursuit

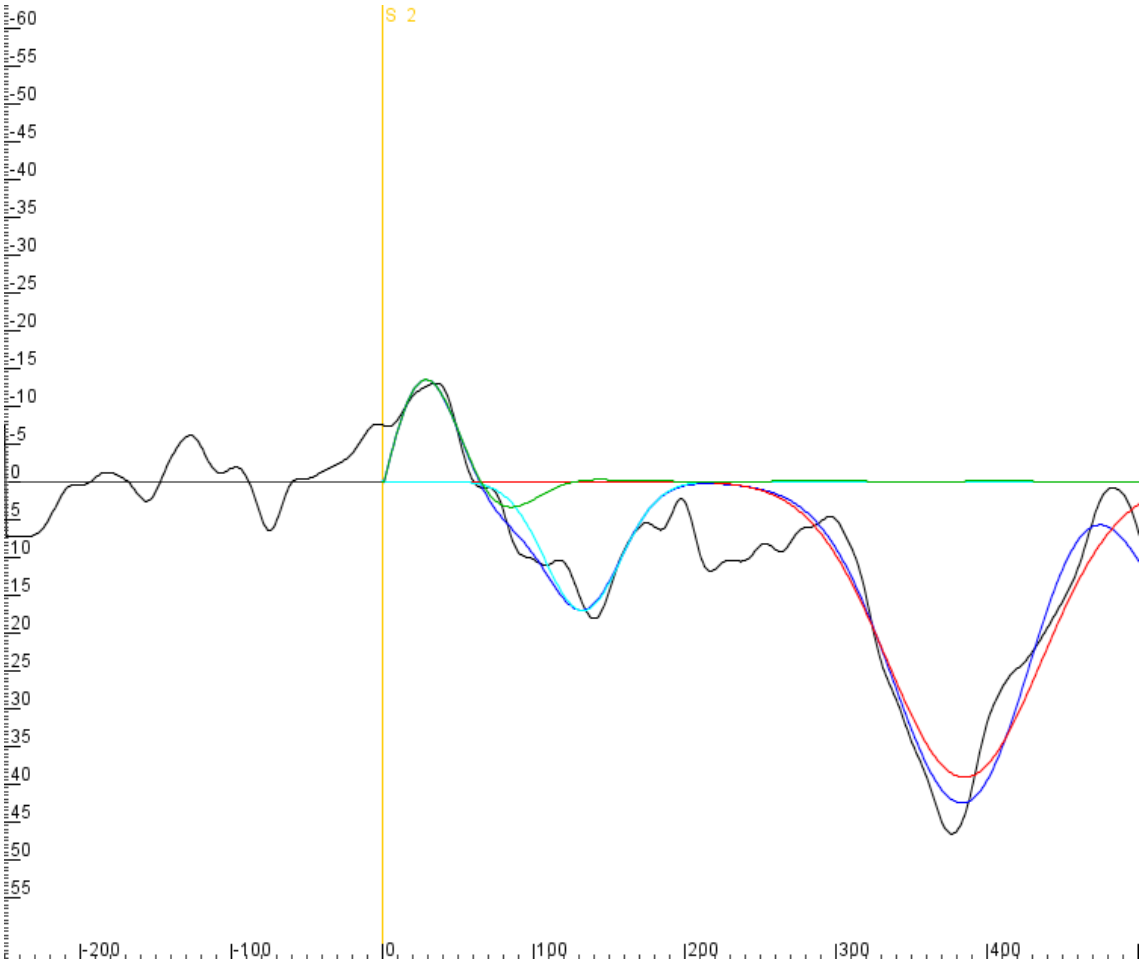
Algoritmus matching pursuit aproximuje signál tak, že ho rozkládá pomocí lineární kombinace funkcí ze svého slovníku. Ve slovníku je možné použít libovolnou množinu funkcí, ale nejčastěji se používají Gáborovy atomy (2.1). V každé iteraci je od signálu odečtena funkce, která jej nejlépe aproximuje,

zbytek (residuum) je vstupním signálem další iterace. Tak postupně získáme několik atomů a zbytek signálu blížící se nulovému signálu. Zvýšením počtu atomů se aproximace zpřesňuje, ale velké množství atomů je poté velmi blízko nule a tak by pro budoucí zpracování neměli prakticky žádný význam (aproximovaly by vlastně jen šum).

$$G_{(s,u,v,w)}(t) = g\left(\frac{t-u}{s}\right) \cdot \cos(vt+w) \tag{2.1}$$

Kde $g(t) = e^{-\pi \cdot t}$ je Gaussovské okénko. Parametry funkce jsou měřítko s, posunutí u, frekvence v a fázový posun w. [2] [4]

Na obrázku 2.6 je vidět epochu délky 750 ms (Začíná 250 ms před a končí 500 ms po markeru). Marker představuje svislá oranžová čára s popiskem *S 2*. Signál epochy je zobrazen černě, aproximace matching pursuitem je zobrazena modře a jedná se o součet pěti atomů. Některé z těchto atomů jsou zobrazeny zelenou, azurovou, a červenou barvou. Některé z těchto atomů by mohli odpovídat ERP komponentám.

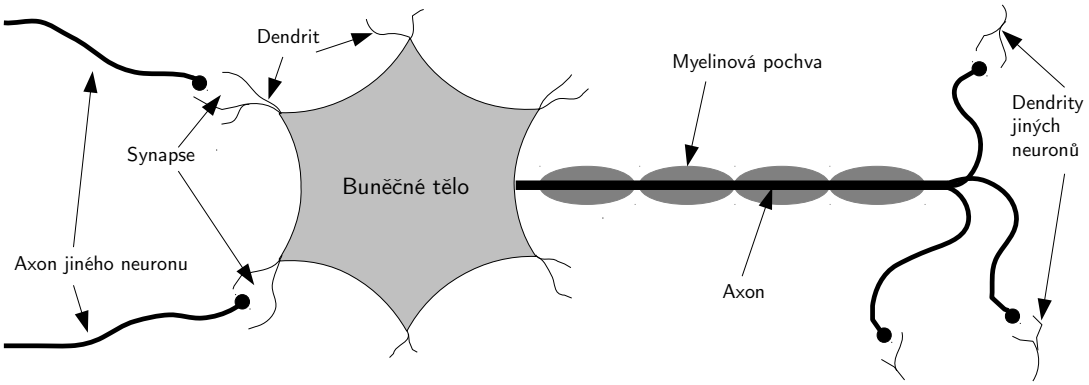


Obrázek 2.6: Epocha s vytvořenými atomy

2.6 Biologická neuronová síť

Stavební jednotky biologické neuronové sítě jsou neurony. Neuron se skládá z několika částí (Obrázek 2.7): [5]

- **Dendrity** – přijímají signál z jiných neuronů, bývá jich větší množství. Signály tvoří elektrické impulzy, které jsou přeneseny přes synapsi (z jednoho neuronu na druhý) chemickým procesem neurotransmitery. Chemický neurotransmiter modifikuje vstupující signál podobným způsobem jako váhy v umělé neuronové síti.
- **Buněčné tělo (soma)** – sčítá příchozí signály z dendritů a ve chvíli kdy má dostatečný vstup jej odvysílá axonem do dalších neuronů.
- **Axon** – je pouze jeden, slouží k přenosu signálu na dendrity jiných neuronů.



Obrázek 2.7: Biologický neuron

Biologické neurony tvoří nervový systém, který se umělé nervové sítě snaží simulovat. Některé důležité vlastnosti biologických neuronových sítí se tak projevují i v těch umělých:

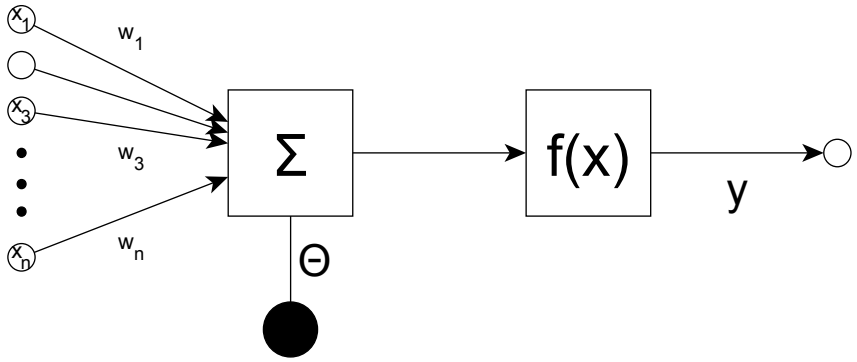
1. Neuron přijímá více signálů.
2. Signál může být modifikován váhami.
3. Neuron sčítá příchozí signály.
4. Pokud má neuron dostatečný vstup tak vyšle jeden výstup.
5. Výstup jednoho neuronu se může dostat na více dalších neuronů.

Další důležitou vlastností je tolerance k chybám. Projevuje se dvěma způsoby. Za prvé jsme schopni rozpoznat signály, které jsou trochu jiné než cokoli co jsme kdy viděli. Za druhé jsme schopni tolerovat poškození nervového systému. Jiné neurony se dokáží naučit vykonávat činnost poškozených neuronů.

I když se nesnažíme vytvořit simulaci biologické neuronové sítě, může pro nás být užitečné se jim přiblížit, protože to může vylepšit výpočetní schopnosti umělé neuronové sítě. [5]

2.7 Umělá neuronová síť

Umělá neuronová síť je systém pro zpracování informací, dá se považovat za zjednodušený model mozku. Základním stavebním prvkem je umělý neuron (Obrázek 2.8) stejně jako neuron u biologických neuronových sítích. Signály $(x_1 \ldots x_n)$ se mezi neurony přenáší spojeními, která jsou ohodnocena váhami $(w_1 \ldots w_n)$. Každý neuron na sečtený signál aplikuje aktivační funkci f a vytvoří tak svůj výstup y (výstup je jen jeden, ale může směřovat na více neuronů). Θ představuje práh, který určuje kdy je vstup dostatečný a může se vytvořit výstup. [5]



Obrázek 2.8: Model umělého neuronu

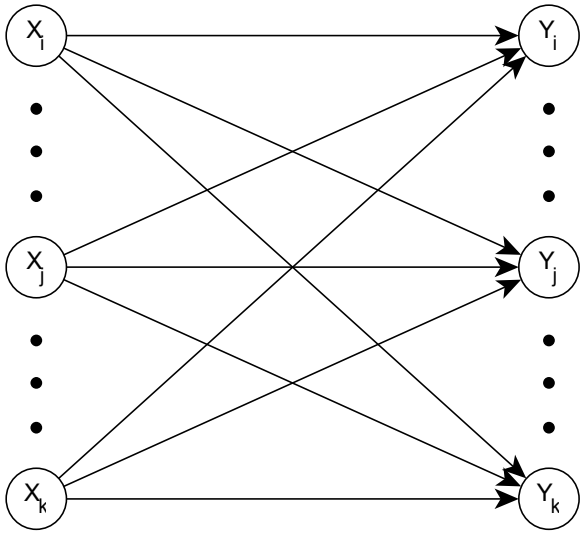
Umělé neuronové sítě mají uplatnění při zpracování signálů, rozpoznávání a klasifikaci vzorků, v medicíně, při tvorbě mluveného hlasu, při rozpoznávání řeči.

2.7.1 Architektura neuronových sítí

Neurony se často zobrazují do vrstev. Tomuto typu architektury se říká vrstvená architektura. Neuronové sítě mají většinou vstupní vrstvu do které přichází externí signál. Dělí se na jednovrstvé a vícevrstvé. Když zjišťujeme počet vrstev tak se vstupní jednotky většinou nepočítají jako vrstva, protože neprovádí žádné výpočty.

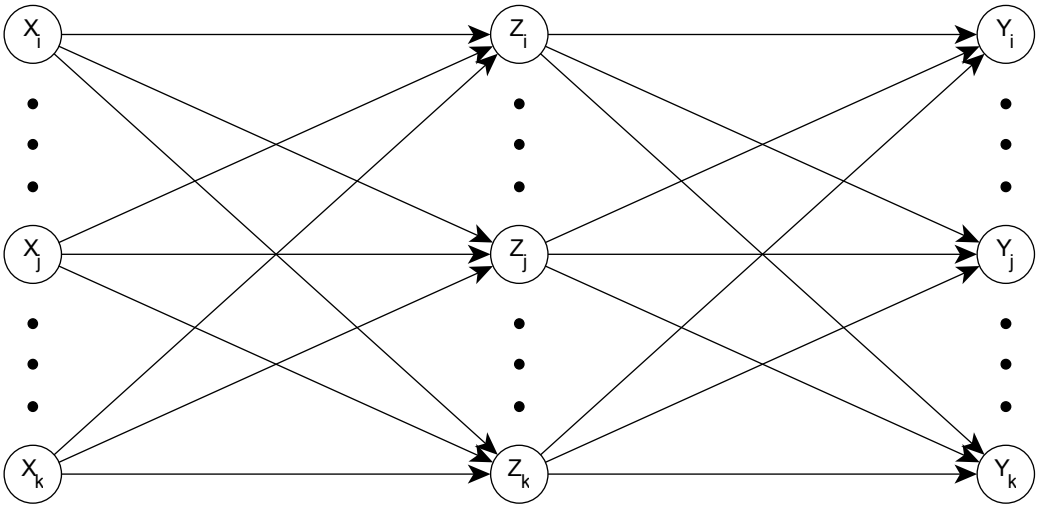
Jednovrstvá i vícevrstvá architektura patří do skupiny takzvaných sítí s dopředným šířením, to znamená, že vstup cestuje pouze jedním směrem od vstupu k výstupu a netvoří žádné smyčky. Naopak sítě rekurentní obsahují uzavřené smyčky, se zpětnou vazbou (například Hopfieldova síť). [5]

Jednovrstvá architektura
Jednovrstvá architektura obsahuje jednu vrstvu vážených spojení (Obrázek 2.9). Spojeny jsou vstupní jednotky (x_i - x_k), do kterých vstupuje signál z vnějšku s výstupními jednotkami (y_i - y_k), ze kterých je možno číst výstup neuronové sítě. Vstupní jednotky nejsou propojeny sami se sebou a to stejné platí pro výstupní jednotky. U rozpoznávání vzorků představuje každá výstupní jednotka kategorii, do které může vstupní vektor patřit. [5]



Obrázek 2.9: Jednovrstvá architektura

Vícevrstvá architektura
Vícevrstvá architektura obsahuje jednu nebo více vrstev (úrovní) skrytých jednotek *(Zi - Zk)* (Obrázek 2.10). Dokáže vyřešit komplikovanější úlohy než architektura jednovrstvá. [5]



Obrázek 2.10: Vícevrstvá architektura

Architektura založená na soutěžení
Tyto sítě používají soutěžení, kdy se nemůže stát aby vstupní vzorek vyvolal výstup na více neuronech. Jen vítězný neuron, nebo jeho malé okolí má dovoleno se učit. Jsou vhodné například při rozpoznávání znaků, kdy znak nemůže být zároveň E a F, ale neuronová síť si musí vybrat který to je. Tento mechanizmus se nazývá soutěžení.

Variantou je typ vítěz bere vše (winner take all), kde pouze jeden neuron bude mít po ukončení soutěžení nenulový výstup, typickým příkladem je síť Maxnet. [5]

2.7.2 Učení

Cílem je nastavit neuronovou síť tak aby dávala přesné výsledky. Zkušenosti sítě jsou uloženy v jejích vahách. [5]

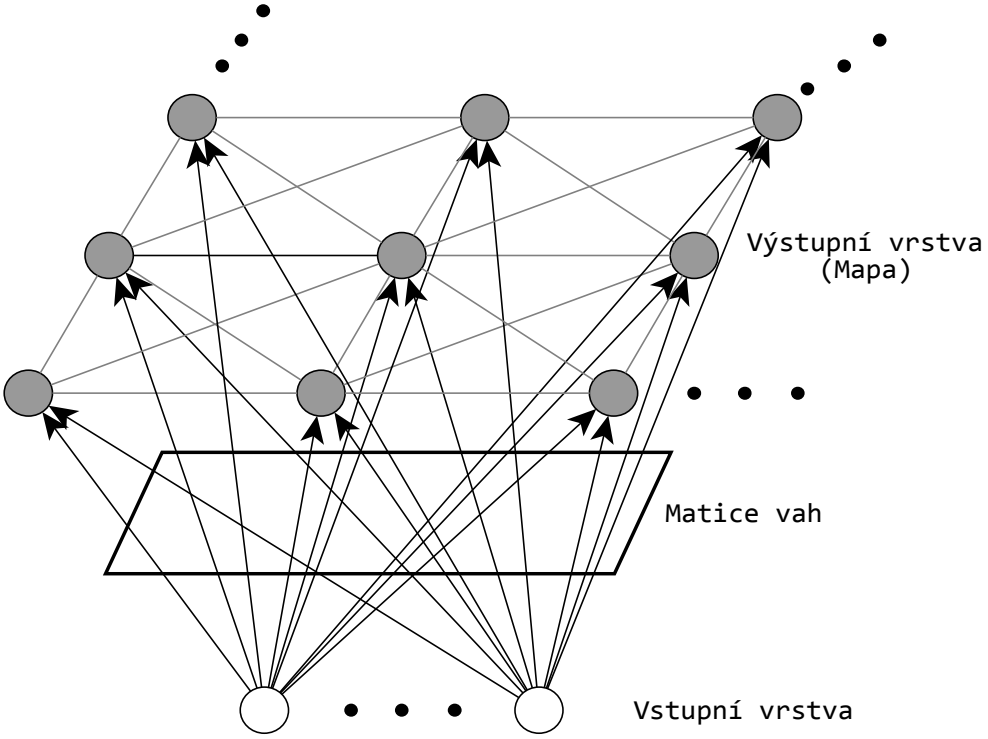
- **Učení s učitelem** – kromě vstupních dat se do neuronové sítě vloží i množina očekávaných výstupů, při trénování se pak váhy v neuronové síti upravují tak aby se co nejvíce přiblížily požadovanému výstupu.
- **Učení bez učitele** – při tomto typu učení si síť třídí vzorky sama a upravuje si váhové vektory.
- **Fixní váhy** – některé neuronové sítě mají nastaveny váhové vektory od začátku a ty se v průběhu již nemění.

2.8 Kohonenova neuronová síť

Tato neuronová síť předpokládá topologickou strukturu mezi jednotlivými neurony. Data se v průběhu klasifikace organizují do mapy, proto se tato síť nazývá také jako samoorganizující mapa (SOM). Často se používá pro zobrazení vícerozměrných dat do dvourozměrné mapy. V průběhu klasifikace se pro každý vstupní vektor zjistí, kterému neuronu je nejblíže (vzdálenost od váhového vektoru neuronu). Neuron který zvítězí a jeho okolí (podle topologie neuronů) upraví své váhy podle klasifikovaného vektoru. [5]

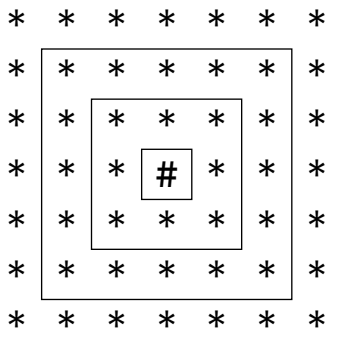
2.8.1 Architektura

Architektura Kohonenovi neuronové sítě se nachází na obrázku 2.11. Jedná se o jednovrstvou architekturu s topologickou strukturou.

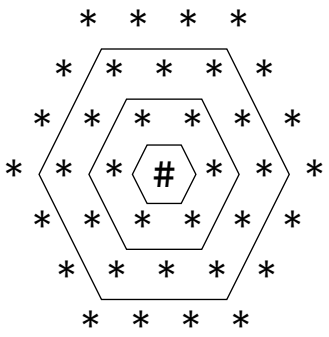


Obrázek 2.11: Architektura Kohonenovi neuronové sítě

Nejčastěji použitá topologická struktura je dvourozměrná mapa. Kromě vítězného neuronu se učí i jeho okolí – radius. Radius se používá buď čtverec (Obrázek 2.12), hexagon (Obrázek 2.13), nebo diamant. # představuje vítězný neuron a * okolní neurony. Pokud se učí pouze vítězný neuron jedná se o radius 0, na čtvercovém se pro radius 1 učí 9 neuronů a pro radius 2 se učí 25 neuronů. V případě že se vítězný neuron nachází na kraji tak se učí menší počet neuronů, ty co by byli mimo mapu se neberou z druhé strany.



Obrázek 2.12: Čtvercová mapa



Obrázek 2.13: Hexagonální mapa

2.8.2 Obecný algoritmus

```
Inicializace vah: váhy[délka_vektoru][počet_neuronů]
Nastavení topologie a radius
Nastavení parametru učení alpha

FOR počet_iterací DO
BEGIN
      FOR vektor x FROM vstupní_vektory DO
      BEGIN
            FOR j = 0 TO počet_neuronů DO
            BEGIN
                  D[j]=∑_i (váhy[i][j]−x[i])^2
            END
            FIND index J SUCH D[J] IS MINIMUM
            FOR all j IN radius OF J
            BEGIN
                  FOR i = 0 TO délka_vektoru DO
                  BEGIN
                        váhy[i][j] += alpha * (x[i] - váhy[i][j])
                  END
            END
      END
      UPDATE alpha
      REDUCE radius
END
```

K formování mapy dochází ve dvou fázích. V první se formuje správné pořadí a ve druhé dochází k finální konvergenci. Druhá fáze vyžaduje nízký parametr učení a velké množství iterací. Podle toho která z těchto fází je dominantní bude vypadat výsledná mapa. Pokud je dominantní první fáze, budou si váhy sousedních neuronů velmi podobné, ale jednotlivé vektory budou svým neuronům podobné méně a naopak v případě dominantní druhé fáze.

Váhy se nastavují většinou na náhodné hodnoty v rozsahu vstupních dat. Pokud je známa nějaká informace o rozmístění neuronů a jejich vahách, tak je možné podle ní upravit počáteční váhy. Například vytvořit váhové vektory podobné těm co hledáme a tyto neurony je pak k sobě budou přitahovat již od začátku To může urychlit proces trénování. [5]

2.8.3 Vstup

Vstupem do sítě je vektor určený ke klasifikaci (v tomto případě signál jednotlivých atomů zrekonstruovaný z jejich parametrů). Všechny složky vstupního vektoru jsou přivedeny na všechny neurony. Neurony jsou rozmístěny ve dvourozměrné mapě, kde sousední neurony mají mezi sebou vztah viz obrázek 2.11 na straně 12.

Pro každý vstupní vektor vygenerují všechny neurony výstupní hodnotu D, která se rovná druhé mocnině euklidovské vzdálenosti klasifikovaného vektoru x od vektoru *vah* daného neuronu (2.2). [5]

$$D = \sum_{i}^{n} (váhy_i - x_i)^2 \tag{2.2}$$

2.8.4 Trénování

Neuron s nejnižší hodnotou D se nazývá vítězný neuron. Pouze vítězný neuron a jeho zvolené okolí se bude učit od klasifikovaného vektoru. Učení probíhá iterativně a okolí, které se učí se postupně zmenšuje. Samotné učení je modifikace vah neuronu a jeho okolí podle (2.3). [5]

$$váhy_{K+1} = váhy_K + a(váhy_K - x) \tag{2.3}$$

Parametr a představuje míru učení, ta je na počátku vysoká, aby se váhy rychle přizpůsobily množině vstupních vektorů. Pak se snižuje současně s velikostí okolí, které se učí a tak začnou váhy neuronu získávat podobu jen těch vektorů kterým jsou nejpodobnější. Tento proces vyžaduje větší množství iterací, nízkou hodnotu a a malé okolí.

2.8.5 Klasifikace

Když máme neuronovou síť natrénovanou, tak vstupní vektor bude patřit neuronu, jehož hodnota D podle (2.2) bude nejmenší. Tak vznikne nad každým neuronem shluk, který obsahuje jemu podobné vektory. Váhový vektor neuronu je možné zobrazit jako signál.

2.9 Hlednání komponent

Skupina signálů které jsou navzájem podobné se nazývá shluk (cluster). Shluk se většinou vytvoří nad jedním neuronem ze signálů, které jsou mu podobné, ale u Kohonenovi sítě vznikají shluky často v rámci několika sousedních neuronů. Je tedy nutné porovnat také podobnost vah dvou sousedních neuronů a zjistit zda netvoří komponentu společně.

Za komponentu je možné považovat shluk atomů z jednoho a více neuronů, který splňuje následující podmínky.

1. Sousední neurony jsou si dostatečně podobné podle zvoleného kritéria.
2. Signál jednotlivých atomů je dostatečně podobný s vahami neuronů do kterých patří.
3. Výsledný shluk obsahuje atomy dostatečně rozprostřené po epochách.

2.10 Porovnání signálů

Je nutné použít vhodné metody pro porovnání dvou signálů, které neuronová síť klasifikovala a rozhodnout zda jsou si podobné či nikoliv. Při porovnání je nutné brát v potaz, že amplituda komponenty může být v signálu posunuta nahoru nebo dolů, podle toho jak prochází epochou baseline a také se může lišit i její amplituda.

Mezi nejběžnější typy porovnání dvou diskrétních signálů x a y patří: [6]

Normalizovaná euklidovská vzdálenost

Nejjednodušší metoda porovnání, pro všechny signály se spočítá jejich vzájemná euklidovská vzdálenost (2.4) a následně se vydělí nejvyšší z nich. Tím se dosáhne toho že výsledná podobnost je z intervalu *<0, 1>*. Pro stejné vektory *0* a *1* pro ty nejodlišnější.

$$Euklid(x,y)=\sqrt{\sum_i (x_i-y_i)^2} \tag{2.4}$$

Kosinová metoda

Tato metoda je založena na úhlu mezi normalizovanými verzemi dvou signálů, při porovnání se bere pouze směr (2.5). Toto porovnání akceptuje různou velikost amplitudy. Výsledky jsou z intervalu *<-1, 1>*. *1* pro signály směřující stejným směrem, *-1* pro opačné a *0* pro odlišné.

$$Cosine(x,y)=\frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2 \sum_i y_i^2}} \tag{2.5}$$

Korelační metoda

Porovnání korelace (2.6) dvou signálů poskytuje podobné výsledky jako kosinová metoda, ale navíc akceptuje i posunutí amplitudy. Výsledky jsou také z intervalu *<-1, 1>*. *1* pro signály závislé, *-1* pro opačnou závislost a *0* pro signály bez závislosti.

$$Korelace(x,y)=\frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i-\bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i-\bar{y})^2}} \tag{2.6}$$

Kde $\bar{x}=\frac{1}{n}\sum_{i=1}^{n} x_i$ a $\bar{y}=\frac{1}{n}\sum_{i=1}^{n} y_i$ je průměrná hodnota signálů x a y.

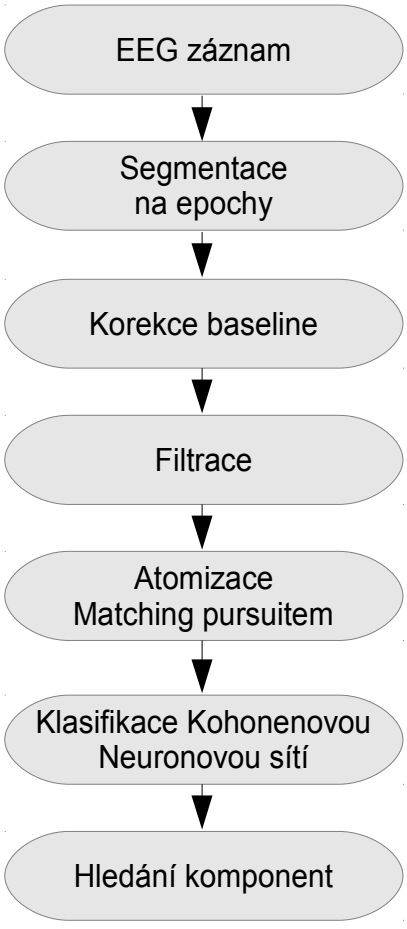
2.11 Shrnutí

Zpracovávat budu signál z jedné elektrody, tento signál segmentací rozdělím na epochy a ty pak budu filtrovat pomocí filtrů popsaných v podkapitole 2.4. Takto upravený signál pak algoritmem matching pursuit rozdělím na atomy (zhruba 10 atomů na jednu epochu). Za předpokladu, že se ERP komponenta nachází ve velkém množství epoch a její tvar se v rámci prohledávaných epoch příliš neliší, měly by se atomy, které jí aproximují sobě podobat. Atomy poté budu třídit Kohonenovou neuronovou sítí na shluky podobných atomů, v těchto shlucích pak budu hledat případnou ERP komponentu.

3 Realizační část

3.1 Cíl

Cílem projektu je ověřit možnost hledání ERP komponent v naměřeném EEG signálu Kohonenovou neuronovou sítí. Načtený signál je po provedení segmentace filtrován pomocí filtrů, aby se odstranil nežádoucí šum. Na dekompozici je použit obecný algoritmus matching pursuit z knihovny EEGDSP vyvíjené na Katedře informatiky a výpočetní techniky (KIV) a matching pursuit implementovaný s využitím genetických algoritmů [7]. Samotná klasifikace se bude provádět Kohonenovou neuronovou sítí, ta je schopna se sama natrénovat ze vstupního vzorku atomů a nepotřebuje mít k dispozici vzorové výsledky. Výsledné rozložení shluků v podobě dvourozměrné mapy by mělo velice usnadnit hledání komponent. Diagram postupu je na obrázku 3.1.



Obrázek 3.1: Postup zpracování

3.2 Vstupní data

Program čte data ve formátu Brain Vision. Data obsahují 3 soubory:

- Hlavičkový textový soubor soubor `.vhdr` (Obrázek 3.2) obsahující informace o naměřených datech.
- Binární soubor `.eeg` s EEG signálem kanálů.
- Textový soubor s pozicemi markerů `.vmrk` (Obrázek 3.3).

```
Brain Vision Data Exchange Header File Version 1.0
; Data created by the Vision Recorder

[Common Infos]
Codepage=UTF-8
DataFile=dominik.eeg
MarkerFile=dominik.vmrk
DataFormat=BINARY
; Data orientation: MULTIPLEXED=ch1,pt1, ch2,pt1 ...
DataOrientation=MULTIPLEXED
NumberOfChannels=19
; Sampling interval in microseconds
SamplingInterval=1000

[Binary Infos]
BinaryFormat=INT_16

[Channel Infos]
; Each entry: Ch<Channel number>=<Name>,<Reference channel name>,
; <Resolution in "Unit">,<Unit>, Future extensions..
; Fields are delimited by commas, some fields might be omitted (empty).
; Commas in channel names are coded as "\1".
Ch1=Fp1,,0.1,µV
Ch2=Fp2,,0.1,µV
Ch3=F3,,0.1,µV
Ch4=F4,,0.1,µV
Ch5=C3,,0.1,µV
Ch6=C4,,0.1,µV
Ch7=P3,,0.1,µV
Ch8=P4,,0.1,µV
Ch9=O1,,0.1,µV
Ch10=O2,,0.1,µV
Ch11=F7,,0.1,µV
Ch12=F8,,0.1,µV
Ch13=T7,,0.1,µV
Ch14=T8,,0.1,µV
Ch15=P7,,0.1,µV
Ch16=P8,,0.1,µV
Ch17=Fz,,0.1,µV
Ch18=Pz,,0.1,µV
Ch19=Cz,,0.1,µV
```

Obrázek 3.2: Ukázka Brain Vision hlavičkového souboru (VHDR)

```
Brain Vision Data Exchange Marker File, Version 1.0

[Common Infos]
Codepage=UTF-8
DataFile=dominik.eeg

[Marker Infos]
; Each entry: Mk<Marker number>=<Type>,<Description>,<Position in data points>,
; <Size in data points>, <Channel number (0 = marker is related to all channels)>
; Fields are delimited by commas, some fields might be omitted (empty).
; Commas in type or description text are coded as "\1".
Mk1=New Segment,,1,1,0,20110316131240336311
Mk2=Stimulus,S  4,16268,1,0
Mk3=Stimulus,S  2,17264,1,0
Mk4=Stimulus,S  4,18261,1,0
Mk5=Stimulus,S  4,19258,1,0
Mk6=Stimulus,S  4,20255,1,0
Mk7=Stimulus,S  4,21252,1,0
Mk8=Stimulus,S  4,22249,1,0
.
.
. //Další záznamy
```

Obrázek 3.3: Ukázka souboru se záznamy markerů (VMRK)

3.3 Architektura projektu

Program používá třívrstvou architekturu pro oddělení dat, aplikační logiky a grafického uživatelského rozhraní. Zjednodušený model architektury je k dispozici v přílohách (Příloha 1). Ne každý blok představuje jen jednu třídu, aby byl obrázek přehlednější jsou některé třídy sloučeny do funkčních celků a ty méně podstatné třídy zobrazeny nejsou (například zdroj ikonek).

3.4 Předzpracování signálu

3.4.1 Segmentace

Načtený signál je potřeba rozdělit na epochy, k tomu slouží segmentace. Epochy začínají obvykle chvíli před markerem, tento krátký úsek se použije pro korekci baseline, protože signál epochy bývá často posunut na ose y i o stovky μV a korekce baseline jej srovná do okolý 0 μV. Provede se tak, že se spočítá průměrná hodnota napětí na úseku pro korekci baseline a ta se odečte od celého signálu. Délka epochy se volí podle toho jaké komponenty hledáme (jak dlouho po stimulu se nacházejí). Epochy se tvoří vždy kolem jednoho typu markeru (stimulu).

3.4.2 Filtry

Pro odstranění epoch obsahujících artefakty a šumu ze signálu je možné požít některý z filtrů zmíněných v podkapitole 2.4. Pro FIR filtr byla použita implementace z knihovny EEGDSP vyvíjené na KIV, bohužel zatím není k dispozici žádná dokumentace s popisem jeho implementace (kromě komentářů v kódu).

3.5 Atomizace

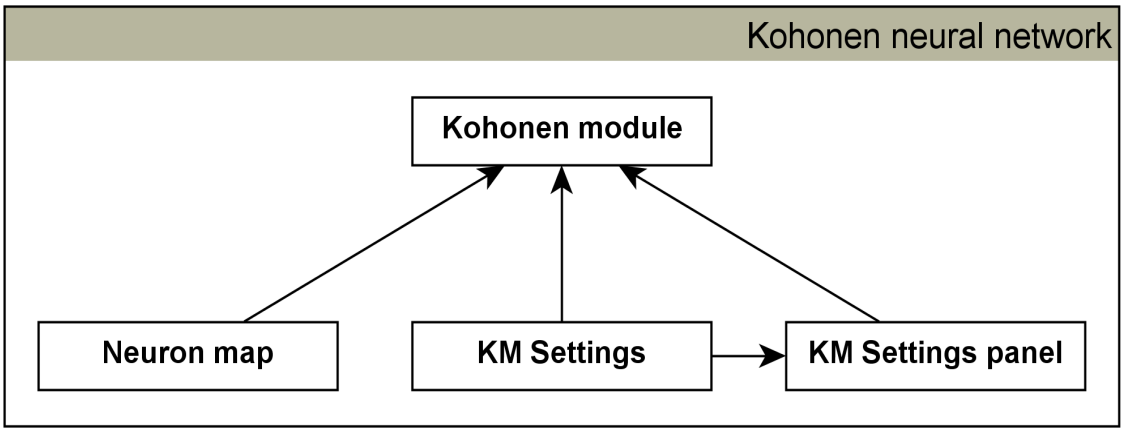
Aby bylo možno v signálu hledat ERP komponenty je potřeba ho dekomponovat na atomy, které budou aproximovat výrazné části signálu epochy. Jejich součet vytvoří aproximaci naměřeného signálu.

Pro atomizaci je použit algoritmus matching pursuit, byl zvolen proto, že se snaží v každé iteraci odečíst od signálu nejvýraznější část, a ta by mohla být jednou z hledaných ERP komponent. K dispozici jsou dvě varianty algoritmu:

- **Obecný Matching pursuit** – více o principu v podkapitole 2.5 a v [2] a [4].
- **Genetický Matching pursuit** – můj kolega Vít Bábel jako svoji bakalářskou práci implementoval matching pursuit pomocí genetických algorimů. Více informací v [7]

3.6 Návrh Kohonenovi neuronové sítě

Neuronovou síť jsem navrhl podle popisu v podkapitole 2.8. Je navržená univerzálně a s menšími úpravami by jí šlo použít pro klasifikaci libovolných vektorů reálných čísel. Architekturu modulu Kohonenovi neuronové sítě si je možno prohlédnout na obrázku 3.4. Jednotlivé třídy jsou popsány v implementační části na straně 29.

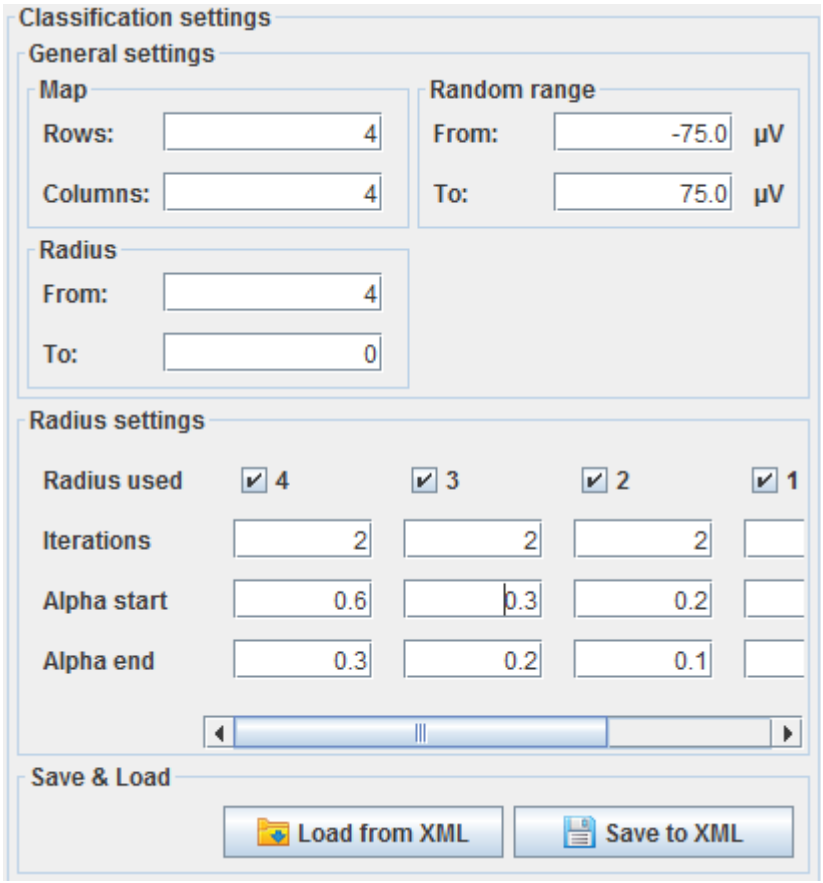


Obrázek 3.4: Architektura modulu Kohonenovi neuronové sítě

Zvolená topologická struktura je dvourozměrná mapa a okolí které se bude učit (radius) je čtvercové, protože implementace je jednoduší oproti hexagonu (nemusí se přepočítávat indexy).

Modul poskytuje i možnost grafického nastavení parametrů klasifikace, k tomuto účelu slouží panel (Obrázek 3.5) obsahující ovládací prvky, ten se dá snadno umístit do grafického rozhraní.

Použití je popsáno v uživatelské příručce v příloze.

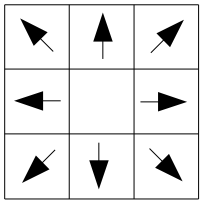


Obrázek 3.5: Panel s nastavením neuronové sítě

3.7 Hledání komponent

Pomocí zvoleného kritéria pro srovnání signálů z podkapitoly 2.10 můžeme porovnávat váhy neuronů mezi sebou a atomy s jejich vahami. Normalizovaná euklidovská vzdálenost se ukázala jako nevhodný způsob porovnání (neakceptuje změnu amplitudy), kosinová nebo korelační metoda je pro porovnávání mnohem lepší.

Každý neuron je zobrazen jako matice devíti čtverců, které představují jeho podobnost k okolním neuronům (čtverce po obvodu) a podobnost jeho vah k atomům, které obsahuje (čtverec uprostřed) (Obrázek 3.6). Podobnost je vyjádřena barvou od červené (odlišné) přes žlutou po zelenou (stejné). Šedý střed značí, že shluk neuronu neobsahuje žádné atomy.



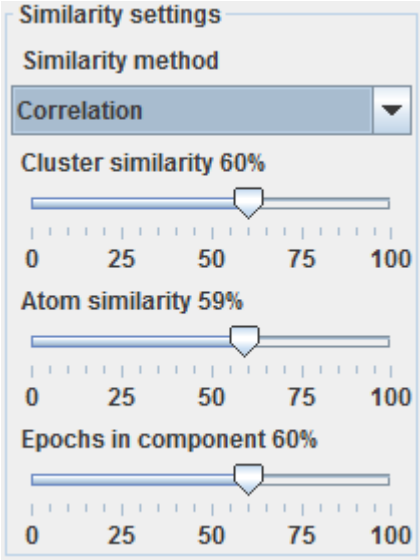
Obrázek 3.6: Vizualizace neuronu

Neurony jsou rozmístěny ve dvourozměrné mapě (Obrázek 3.7). Jak bylo napsáno v podkapitole 2.9 nad každým neuronem se vytvořil shluk jemu podobných atomů, ale protože během učení se kromě vítězného neuronu učí i jeho sousedé tak nám může vzniknout shluk podobných atomů nad několika sousedními neurony.

Obrázek 3.7: Mapa neuronů

3.7.1 Prohledání mapy shluků

Uživatel nejprve nastaví parametry pro tvorbu komponent a to minimální podobnost neuronů mezi sebou, minimální podobnost atomů neuronu s jeho vahami a minimální zastoupení epoch ve shluku, aby byl považován za možnou komponentu (Obrázek 3.8). Zvolené prahy porovnání jsou v procentech.



Obrázek 3.8: Nastavení podobnosti

Algoritmus prohledání

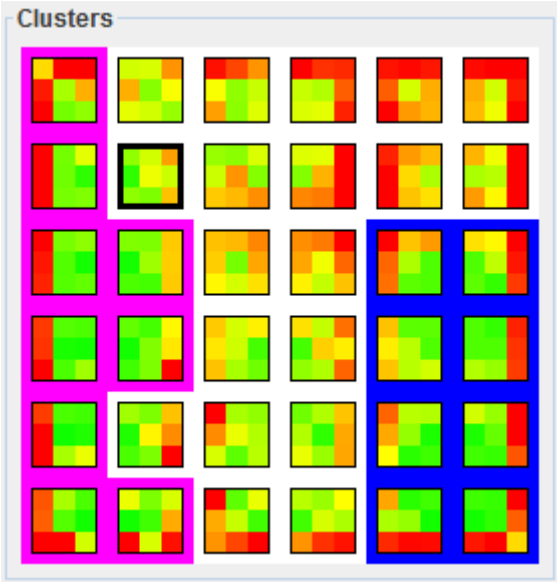
```
Inicializace pole komponent int[výška_mapy][šířka_mapy] komponenty na 0
int současná_komponenta = 0
pole neuronů neuron[][] neurony
//Uživatel nastaví
double N //Podobnost neuronů mezi sebou
double A //Podobnost atomů neuronu s jeho váhami

FUNCTION najdi_komponenty()
BEGIN
      //Každý shluk v poli komponenty označí číslem komponenty kam patří
      FOR i = 0 TO šířka_mapy DO
      BEGIN
            FOR j = 0 TO výška_mapy DO
            BEGIN
                  přidej_komponentu(i, j, ++současná_komponenta)
            END
      END
END

//Rekurzivní funkce pro přidání komponenty
FUNCTION přidej_komponentu(int řádek, int sloupec, int číslo_komponenty)
BEGIN
      IF řádek < 0 OR sloupec < 0 OR  řádek >  výška_mapy OR
      sloupec > šířka_mapy THEN
      BEGIN
            RETURN
      END
      IF neurony.podobnost_atomů > A THEN
      BEGIN
            IF komponenty[řádek][sloupec] = 0 THEN //Ještě není v komp.
            BEGIN
                  komponenty[řádek][sloupec] =  číslo_komponenty
                  FOREACH neuron soused IN neurony AROUND neurony[řádek]
                  [sloupec] DO
                  BEGIN
                        IF soused COMPARE TO neurony neurony[řádek]
                        [sloupec] > N THEN
                        BEGIN
                              přidej_komponentu(soused.řádek,
                                                soused.sloupec,
                                                číslo_komponenty)
                        END
                  END
            END
      END
      ELSE
      BEGIN
            komponenty[řádek][sloupec] = -1 // Není komponenta
      END
END
```

3.7.2 Přijmutí nalezených komponent

Po dokončení prohledání je nutné ještě odstranit ty komponenty, které nemají dostatečné zastoupení epoch. Pro každý neuron se vytvoří seznam epoch jejichž atomy obsahuje. Jeho délka se vydělí celkovým počtem epoch ve skupině extrahovaných epoch a tak získáme procentuální zastoupení epoch v dané komponentě. To pak porovnáme s uživatelem definovaným prahem pro zastoupení epoch a pokud je menší komponentu odstraníme.

Je nutné podotknout, že takto vzniklé komponenty jsou pouze komponenty potenciální a záleží na uživateli, zda je bude opravdu za komponenty považovat. Při nevhodném nastavení prahů nemusí nalezené komponenty vůbec připomínat hledané ERP komponenty.

Nalezené komponenty jsou na mapě neuronů podbarveny (Obrázek 3.9).



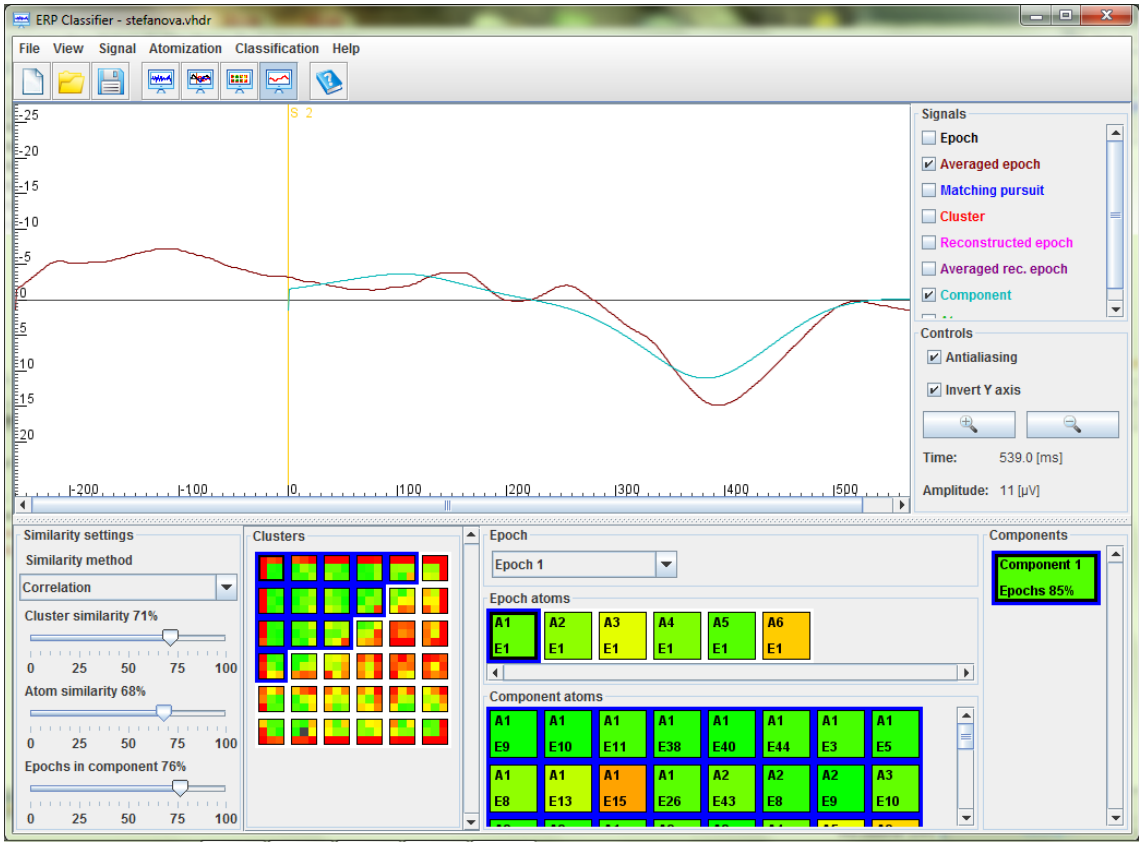
Obrázek 3.9: Nalezené komponenty

3.8 ART2

V rámci jedné semestrální práce na ZSWI byla vytvořena neuronová síť ART2, bohužel ale stále není úplně dokončená. Já jsem pro ni udělal GUI pro zobrazení výsledků podobně jako pro Kohonenovu síť. Je možné jí také použít pro klasifikaci a hledání ERP komponent. Já se zde věnovat popisu jak funguje nebudu a případní zájemci se mohou podívat do [5], v části s výsledky se nachází jeden výsledek ART2 pro srovnání s Kohonenovou sítí.

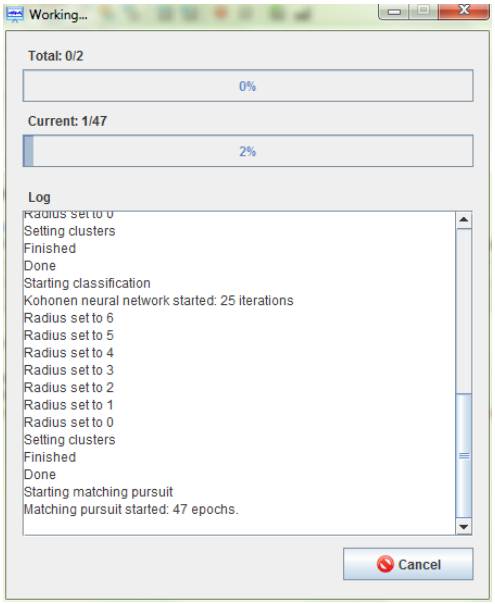
3.9 Vizualizace

Pro přehledné zobrazení výsledků jsem vytvořil grafické uživatelské rozhraní. GUI je vytvořeno pomocí knihovny Swing z Javy. Program má jedno hlavní okno na kterém se zobrazují výsledky, rozpracovaná data a nastavení. Na obrázku 3.10 je vidět hlavní okno aplikace se zobrazenou tvorbou komponent.



Obrázek 3.10: Hlavní okno aplikace

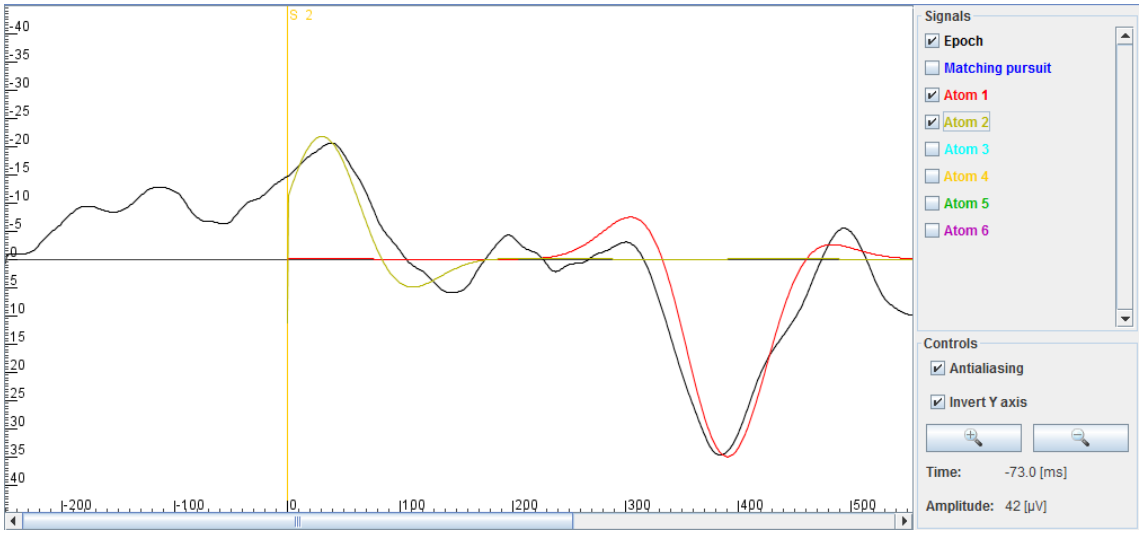
Dále pak menší okno s průběhem výpočtu, které se zobrazí během časově náročných operací a ukazuje kolik práce je již hotovo a kolik zbývá (Obrázek 3.11). Detailní popis jednotlivých částí GUI a jeho použití je v uživatelské příručce v příloze.



Obrázek 3.11: Okno s průběhem výpočtu

3.9.1 Zobrazení signálů

Pro zobrazení signálů jsem vytvořil univerzální panel, který umožňuje snadné použití na různých místech v programu. Jeho použití je snadné a je možné ho velmi snadno upravit. Tento panel v základu disponuje schopnostmi pro přidání, odebrání a změnu signálů, jejich přepínání, vykreslení os a pozičního kříže, přibližování a oddalování signálu, vyhlazování a přepínání několika různých signálů a jejich současné zobrazení v různých barvách (Obrázek 3.12).



Obrázek 3.12: Panel pro zobrazení signálu

3.10 Implementace

Obrázek architektury je v příloze číslo 1. Podrobnější informace o jednotlivých třídách a jejich metodách se nacházejí v JavaDocu na přiloženém CD.

3.10.1 Programovací jazyk

Pro vývoj projektu byla zvolena Java, se kterou jsem mám větší zkušenosti, měl jsem k dispozici knihovnu pro matching pursuit v Javě a základ projektu vytvořený v rámci semestrální práce na ZSWI (základy softwarového inženýrství). Tvorba složitých grafických rozhraní, která program potřebuje pro vizualizaci výsledku je v Javě mnohem jednodušší než například v C++. Java je také multiplatformní a tak je program možné spustit pod různými operačními systémy. Vývoj probíhal ve vývojovém prostředí NetBeans, které umožňuje snadné napojení na repositář.

3.10.2 Datová vrstva

Obsahuje třídy pro načtení a uchování dat v rámci programu, dále umožňuje ukládat rozpracované výsledky do *XML* souboru.

Třídy Vhdr, Vmrk, Eeg

Obstarávají načtení dat z odpovídajících vstupních souborů.

Třída Loader
Pomocí tříd *Vhdr*, *Vmrk* a *Eeg* načte vstupní data umožní k nim přistupovat, aby se mohli nahrát do datových tříd programu.

Třída Channel
Tato třída uchovává informace o signálu měřeného na jedné elektrodě (kanálu) a umožňuje k nim přistupovat. Jsou zde uloženy jak samotná data signálu, tak i jméno kanálu, jednotky měření a jejich rozlišení.

Třída Marker
Obsahuje informace o markeru (stimulu) a umožňuje k nim přistupovat. Jsou zde uloženy informace v kterém kanálu se marker nachází, jeho typ, popis, velikost (délka stimulu) a pozice v signálu.

Třída DataSheet
Tato třída sdružuje data naměřená během jednoho měření. Obsahuje všechny měřené kanály a markery, datum měření, vzorkovací frekvenci a cestu k datovému souboru.

Třída Atom
Reprezentuje Gaborův atom, obsahuje jeho parametry a zrekonstruovaný signál. Dědí od třídy *Pattern*, která je vstupní třídou klasifikace a umožňuje nastavit každému atomu shluk, do kterého jej klasifikuje neuronová síť.

Třída Epoch
Představuje jednu epochu extrahovanou ze signálu v okolí jednoho markeru. Obsahuje extrahovaný signál epochy, začátek a konec baseline pro korekci pozice signálu, číslo kanálu ze kterého pochází, marker, pole atomů po atomizaci matching pursuitem a sumu signálů těchto atomů jako aproximaci signálu epochy.

Třída EpochGroup
Sdružuje skupinu epoch (například všech epoch extrahovaných z jednoho kanálu v okolí jednoho markeru). Umožňuje přistupovat k datům jednotlivých epoch a obsahuje některé metody pro práci s nimi – průměrování a filtry. Dále obsahuje neuronovou sítí vytvořené váhové vektory pro jednotlivé shluky atomů, vzorkovací frekvenci signálu, informaci o tom zda byl proveden matching pursuit a šířku mapy aby bylo možné zjistit jak vypadá topologická mapa shluků.

Třída XMLEpochGroup
Vzhledem k tomu, že předchozí datové třídy byly navrženy v projektu na ZSWI, tak nikdo nepředpokládal potřebu ukládat rozpracované výsledky, proto nejsou vhodné pro přímou serializaci do *XML*. Pomocí standardní knihovny Javy *JAXB* (Java Architecture for *XML* Binding) lze provést uložení (serializaci) a načtení (de-serializaci) instancí objektů do souboru *XML*, je ale nutné tomu trochu přizpůsobit návrh těchto tříd. Proto jsem vytvořil tuto jednoduchou třídu

a její soukromé vnitřní třídy (*XMLEpoch*, *XMLAtom*, *XMLMarker* a *XMLCluster*) aby umožnila serializaci požít bez zásahů do již hotové datové vrstvy.

Obsahuje metody pro snadný převod mezi *EpochGroup* a *XMLEpochGroup* a samotnou serializaci a de-serializaci. K dispozici je i *XSD* schéma pro validaci uložených *XML* souborů.

3.10.3 Aplikační vrstva

Je rozdělena na několik balíků:

- **application** – Obsahuje řídící třídy programu.
- **application.algorithms** – Obsahuje jednoduché třídy s univerzálními metodami a algoritmy pro použití kdekoliv v programu.
- **application.modules** – Obsahuje moduly pro matching pursuit a neuronové sítě. Budou popsány níže.

3.10.4 Moduly obecně

Základ modulů přestavují abstraktní třídy pro jednotlivé typy modulů. Mají usnadnit případnou tvorbu nových algoritmů pro atomizaci a klasifikaci. Více informací o struktuře modulů pro matching pursuit najdete v [7].

Třída Pattern

Vzorek pro klasifikaci, obsahuje klasifikovaný vektor (signál) a umožňuje nastavit shluk do kterého patří (provede neuronová síť). Od *Patternu* dědí třída atom a jako klasifikovaný vektor je použit signál atomu.

Třída AbstractModule

Šablona pro tvorbu modulu, dědí od třídy *Observable*, aby GUI mohlo být informováno o postupu modulu. Obsahuje hlavičky základních metod pro práci s modulem (spuštění výpočtu, resetování do výchozího nastavení, zjištění typu modulu, jeho jména, panelu s nastavením, uložení nastavení, zastavení běhu, zjištění zda je modul možno spustit, zda běží a jestli už skončil).

Třída AbstractMpModule

Šablona pro tvorbu modulů pro matching pursuit, dědí od třídy *AbstractModule*. Obsahuje metody pro práci s modulem matching pursuit, implementuje některé abstraktní metody *AbstractModule* a přidává vlastní metody a hlavičky metod pro implementaci v konkrétních modulech (nastavení vstupních signálů, získání vytvořených atomů a jejich parametrů).

Třída MatchingPursuit

Rohraní pro různé typy algoritmů matching pursuit.

Třída AbstractClassificationModule
Šablona pro tvorbu modulů pro klasifikaci, dědí od třídy *AbstractModule*. Obsahuje metody pro práci s modulem klasifikace. Implementuje některé abstraktní metody *AbstractModule* a přidává vlastní metody a hlavičky metod pro implementaci v konkrétních modulech (nastavení, získání a kontrola klasifikovaných vzorků, získání vah neuronů a získání počtu vytvořených shluků).

3.10.5 Modul pro obecný matching pursuit
Obsahuje obalující třídy pro obecný matching pursuit z knihovny EEGDSP, informace o implementaci ve [4].

3.10.6 Modul pro Genetický matching pursuit
Bakalářská práce mého kolegy Víta Bábela, více informací v [7].

3.10.7 Modul pro ART2
Ze ZSWI použitá neuronová síť ART2, kterou dělal Vít Bábel. Byla trochu upravena aby vyhovovala novým požadavkům (klasifikace podle signálu místo parametrů atomu). Více informací o implementaci v JavaDocu a o principu klasifikace v [5].

3.10.8 Modul pro Kohonenovu neuronovou síť
Modul jsem implementoval podle popisu v podkapitolách 2.8 a 3.6. Náročnost výpočtu Kohonenovo neuronové sítě je lineární vzhledem k počtu neuronů, délce klasifikovaného vektoru, počtu vektorů a celkovému počtu iterací.

Třída KohonenModule
Hlavní třída modulu, slouží pro nastavení vstupních vektorů, nastavení parametrů klasifikace (pomocí *JPanelu* s grafickým nastavením), spuštění klasifikace a získání výsledků. Dědí od *AbstractClassificationModule*.

Třída NeuronMap
Představuje mapu neuronů, umožňuje vytvořit dvourozměrnou mapu neuronů s vybranou výškou a šířkou a inicializovat váhy neuronů na náhodné hodnoty v zadaném rozsahu. Mapa umí najít neuron, kterému se zadaný vektor podobá nejvíce, zařadit jej do jeho shluku a aktualizovat váhy daného neuronu a jeho okolí podle zvoleného radiusu.

Třída KMSettings
Uchovává nastavení modulu pro Kohonennovu neuronovou síť (velikost mapy, rozsah náhodných hodnot ve vahách, rozsah radiusu a nastavení jednotlivých radiusů – počet iterací, počáteční učení a a koncové učení a). Slouží také pro uložení nastavení do *XML* serializací pomocí Java knihovny *JAXB*.

Třída KMSettingsPN
Třída pro nastavení modulu Kohonenovi neuronové sítě. Dědí od JPanelu a umožňuje přehledné nastavení v grafickém rozhraní. Stará se také o serializaci a de-serializaci *KMSettings* do a z *XML*. Kontrola správnosti je prováděna pouze programově bez použití *XSD* schématu.

3.10.9 Grafické uživatelské rozhraní

Vzhledem k rozsahu GUI a počtu tříd odkážu s popisem jednotlivých tříd a jejich metod na přiložený JavaDoc na CD.

4 Dosažené výsledky

V této kapitole bude prezentováno několik zpracovaných naměřených dat. U prvního bude přesně popsán postup zpracování a nastavení s jakým bylo těchto výsledků dosaženo. Při pokusu o zopakování zpracování se mohou výsledky lišit, protože jak Genetický matching pursuit tak Kohonenova neuronová síť vnáší do svých výpočtu prvek náhody (váhy v síti jsou na počátku nastaveny na náhodné hodnoty).

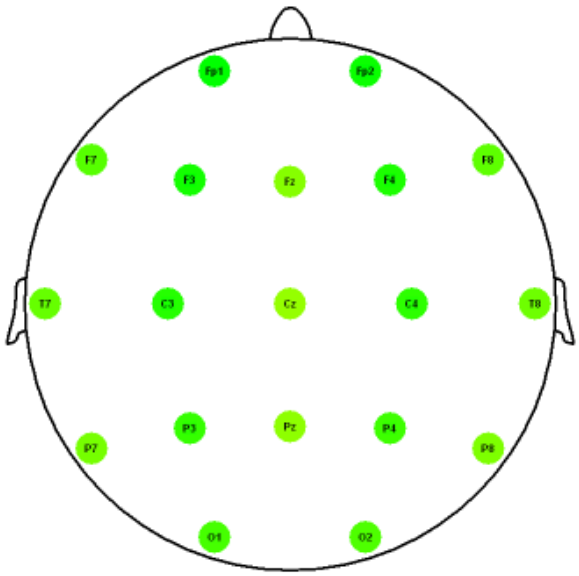
4.1 Zpracovaná měření

Data pocházejí z měření v neuroinformatické laboratoři na KIVu. První dvě osoby jsem neměřil já, na měření druhých dvou jsem se podílel. Jako stimuly sloužily 3 barevně blikající diody (červená, žlutá a zelená), pravděpodobnost výskytu viz tabluka 4.1. Měřené osoby se na blikající diody měli soustředit, alternativou by bylo nechat je dívat se například na video a diody by pak vnímali pouze podvědomě (komponenty budou pak méně výrazné).

Marker	Barva	Pravděpodobnost výskytu
S4	Zelená	80%
S2	Červená	15%
S1	Žlutá	5%

Tabulka 4.1: Pravděpodobnost výskytu stimulů

Při měření bylo použito 19 elektrod rozmístěných po celé hlavě (Obrázek 4.1) [2]. Detekci komponent budu provádět na středové části hlavy (*Fz, Cz a Pz*) a na zadní části hlavy (*O1 a O2*).



Obrázek 4.1: Rozmístění elektrod

4.1.1 Osoba 1 – kanál FZ, komponenta P3

Jeden z nejlépe naměřených signálů, kde je ERP komponenta P3 vidět dobře i okem. Dále je zde zastoupena i ERP komponenta N1 a N2, ale již nejsou tak výrazné a jejich pozice a amplituda se dost liší v rámci jednotlivých signálů a tak splývají s mnohem výraznější P3. Na průměru všech epoch jsou trochu vidět.

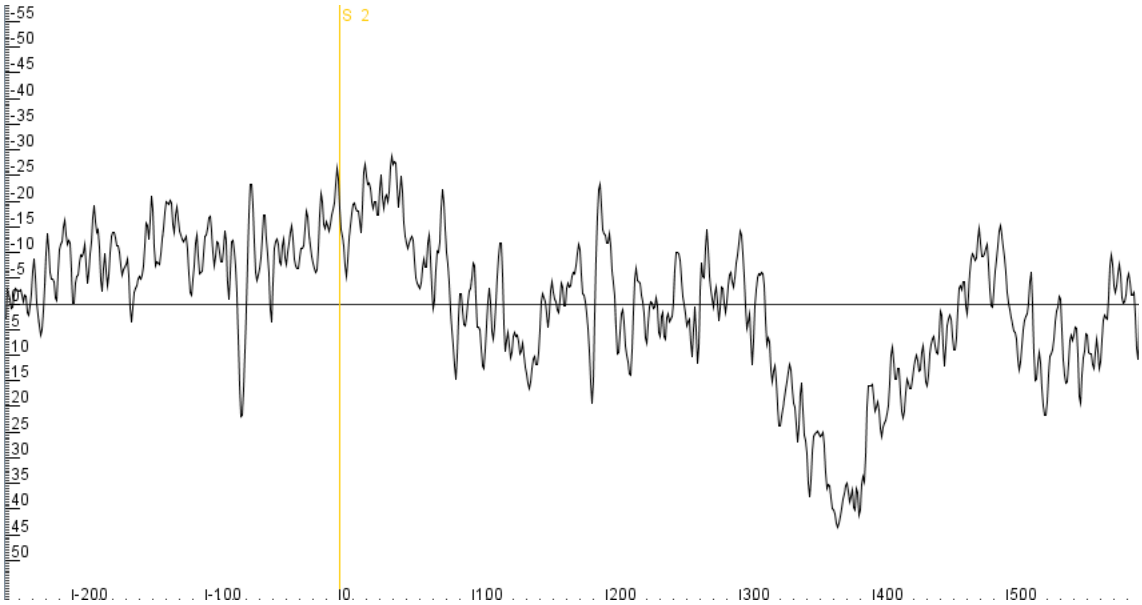
Epochy

Pro segmentaci na epochy jsem zvolil následující parametry (Tabulka 4.2). Poslední epochu jsem ručně odstranil, protože neobsahovala žádný signál. Segmentací tedy vzniklo 46 epoch, které budou dále zpracovány.

Začátek epochy	Konec epochy	Začátek úseku pro baseline	Konec úseku pro baseline	Typ markeru	Kanál
-250 ms	600 ms	-250 ms	0 ms	Target (S2)	FZ

Tabulka 4.2: Vlastnosti epoch

Na obrázku 4.2 si můžete prohlédnout první epochu před filtrací, je na ní dobře vidět velké množství nežádoucího šumu, ale i ERP komponenta P3 (negativní vlna ve zhruba 360 ms po markeru).



Obrázek 4.2: Epocha před filtrací

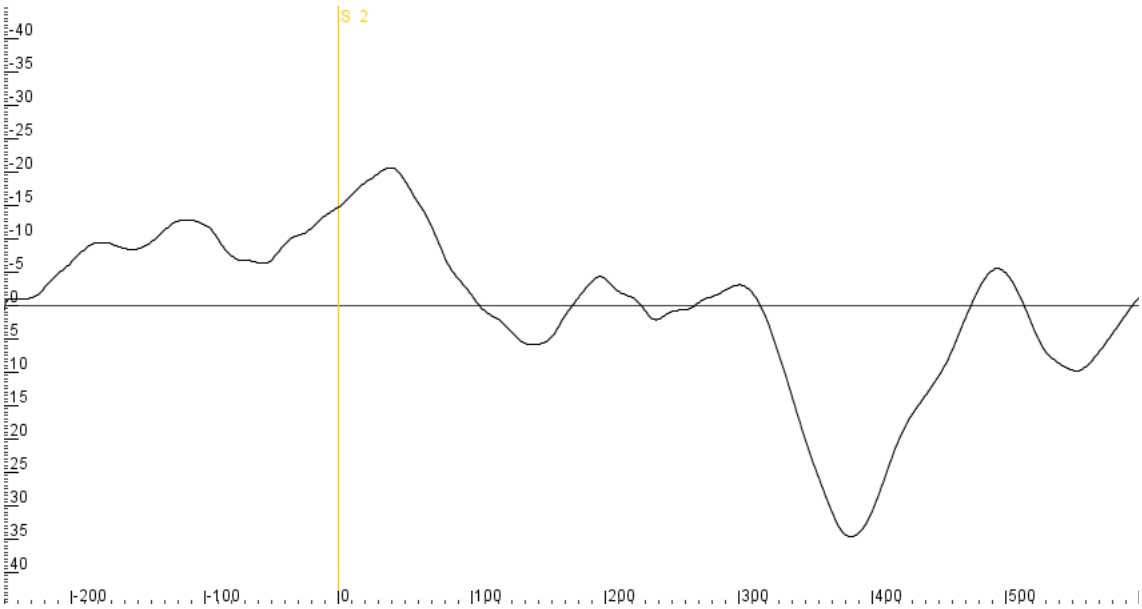
Filtry

Filtry byly použity v následujícím pořadí: prahový, pohyblivé okénko, FIR a průměrování signálu. Nastavení filtrů je v tabulce 4.3.

Prahový filtr		FIR filtr	
Minimum	-100 µV	Řád	20
Maximum	100 µV	Typ filtru	Dolní pásmová propusť
Filtr pohybujícím se okénkem		Typ okénka	Čtvercové
Krok	10 ms	Dolní frekvence	-
Délka	75 ms	Horní frekvence	30 Hz
Maximální rozdíl	75 µV	Průměrování epoch	
Filtr pro průměrování signálu		Počet	Nepoužito
Počet hodnot	41	Pořadí	
Počet iterací	1	Prahový, okénko, FIR, průměrování signálu	

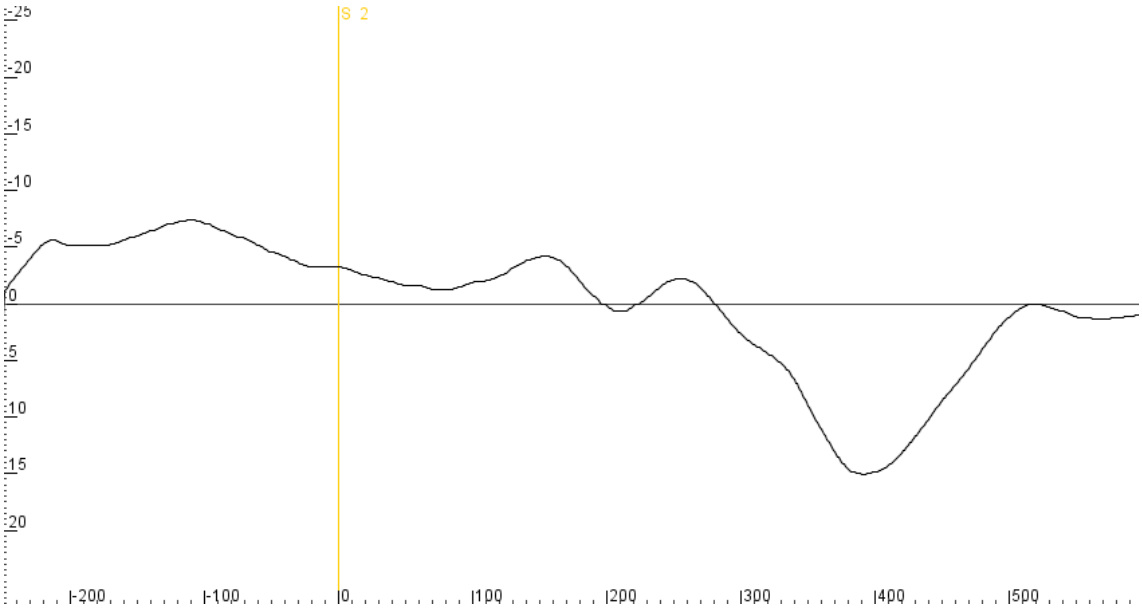
Tabulka 4.3: Nastavení filtrů

Na obrázku 4.3 je vidět první epocha po použití výše zmíněných filtrů. Jak je vidět povedlo se odstranit šum, ale zároveň některé detaily nejsou tak výrazné. Došlo k zmenšení amplitudy P3 komponenty asi o 10 µV a její posunutí o 20 ms dál v čase.



Obrázek 4.3: Epocha po filtraci

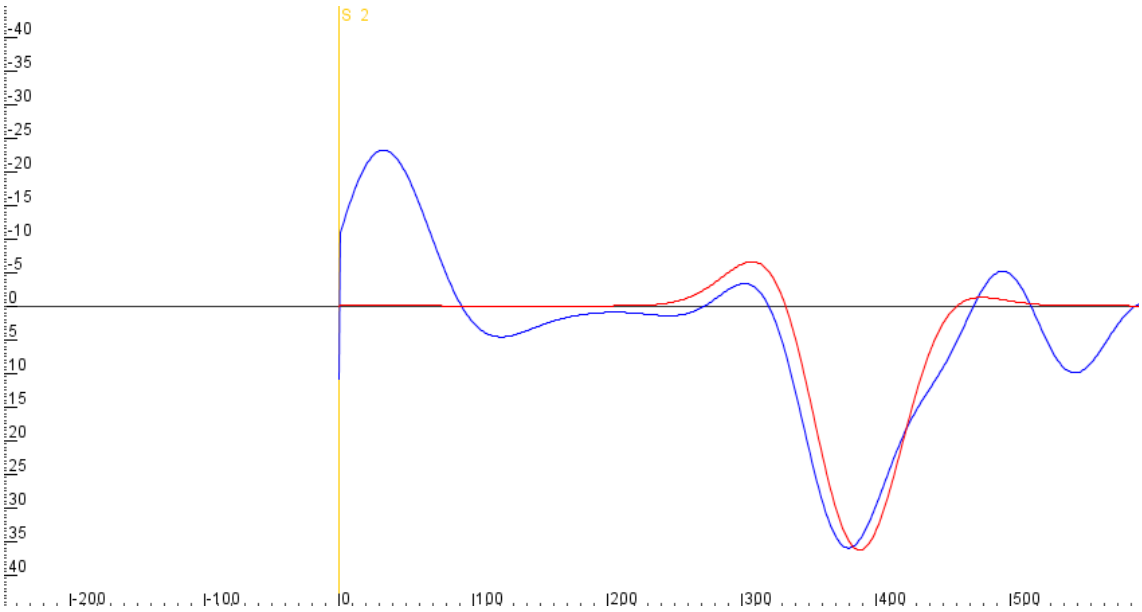
Pokud jako metodu hledání ERP komponent použijeme průměrování epoch získáme průměrný signál všech 46 epoch (Obrázek 4.4). Na průměru je dobře patrná P3 komponenta a i náznak komponent N1 a N2, které jsou ale málo výrazné a dost posunuté dále v čase.



Obrázek 4.4: Průměr všech 46 epoch

Atomizace

Pro detekci v jednotlivých epochách je napřed rozložíme na atomy. Pro atomizaci použijeme genetický matching pursuit v módu *partial FFT acceleration*. Počet vytvořených atomů bude stačit 6, je nepravděpodobné, že by se komponenta nacházela v dalších atomech. Na obrázku 4.5 je vidět aproximace první epochy matching pursuitem jako součet jejích 6 atomů (modrý signál) a první atom, který dobře pokryl komponentu P3 (červená).



Obrázek 4.5: Atomizace matching pursuitem

Klasifikace Kohonenovou neuronovou sítí
Nastavení klasifikace bylo následující (Tabulka 4.4).

Velikost mapy		Náhodné váhy		Radius	
Šířka	4	Od	-75 µV	Od	4
Výška	4	Do	75 µV	Do	0
Nastavení radiusů					
Radius	4	3	2	1	0
Počet iterací	2	2	2	5	10
Počáteční učení α	0,6	0,3	0,2	0,1	0,01
Koncové učená α	0,3	0,2	0,1	0,01	0,001

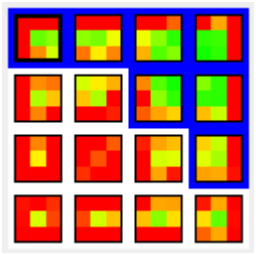
Tabulka 4.4: Nastavení klasifikace

Tvorba komponent
Pro tvorbu komponent jsem zvolil následující kritéria (Tabulka 4.5).

Metoda porovnání	Minimální podobnost shluků	Minimální podobnost atomů ve shluku	Minimální zastoupení epoch v komponentě
Korelace	52%	68%	90%

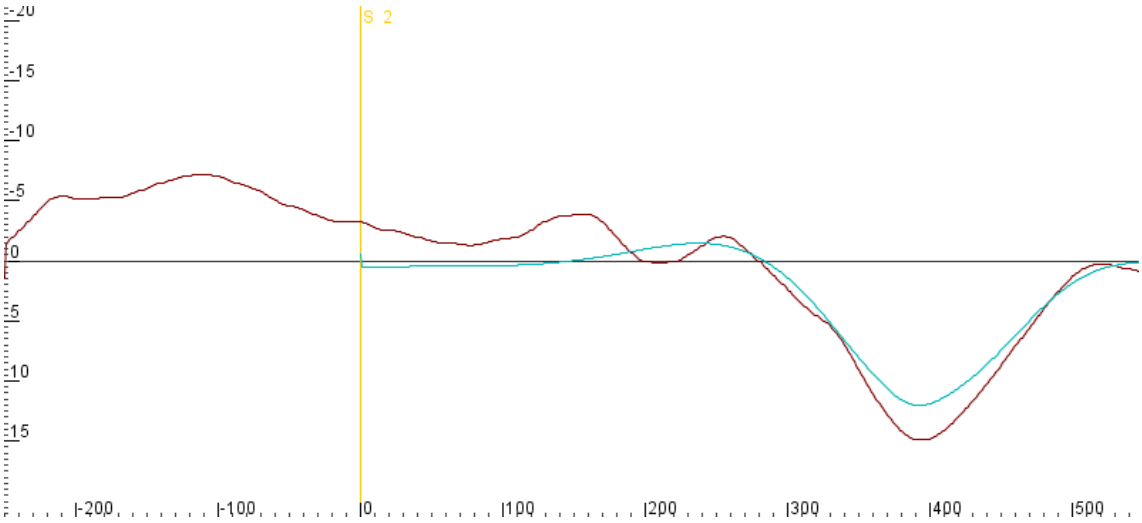
Tabulka 4.5: Kritéria tvorby komponent

Nalezená P3 komponenta
Komponenta byla vytvořena z několika sousedních shluků (Obrázek 4.6).



Obrázek 4.6: Mapa s nalezenou komponentou

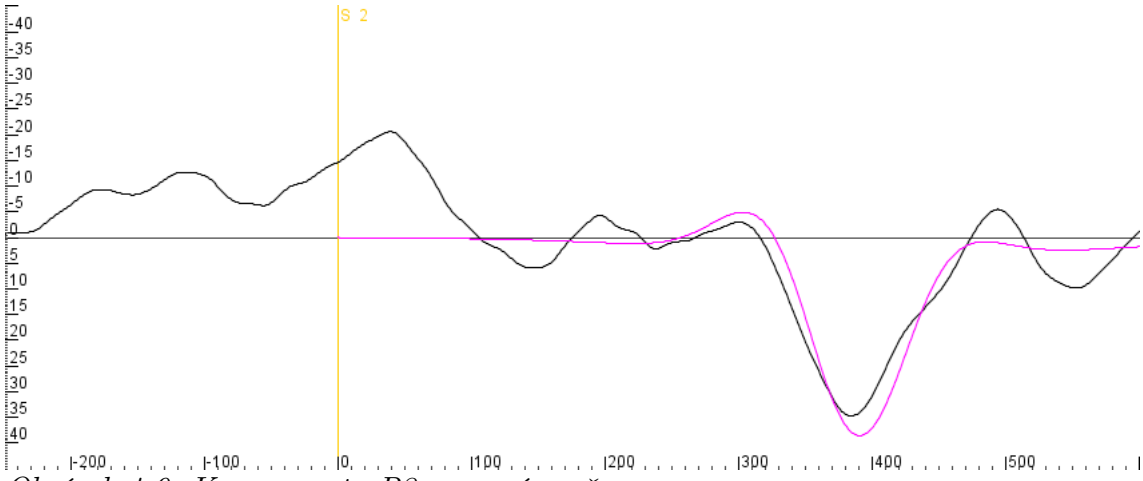
Nalezená P3 komponenta je trochu posunutá z obvyklých 300 ms do 400 ms (o minimálně 20 ms ji posunul FIR filtr 20 řádu). Také trochu splývá s N1 a N2 viz obrázek 4.7. Hnědý signál představuje zprůměrovaný signál všech epoch a světle modrý nalezenou komponentu (vlna byla vytvořena jako průměr ze signálů všech atomů, které byly do komponenty zahrnuty. Na průměrovaném signálu epoch je jsou dobře vidět i komponenty N1 a N2, bohužel nebyly v jednotlivých epochách dost výrazné a podobné na to, aby se jim podobaly matching pursuitem vytvořené atomy.



Obrázek 4.7: Nalezená P3 komponenta

V první epoše byly za komponentu P3 označeny dva atomy (Obrázek 4.8). Jejich součet je vidět na obrázku 4.9, ten představuje výslednou P3 komponentu v první epoše.

Obrázek 4.8: Atomy tvořící komponentu P3



Obrázek 4.9: Komponenta P3 v první epoše

V následující tabulce (4.6) je porovnání neuronovou sítí nalezených komponent s realitou. P3 komponenta byla detekována ve 43 epochách z 46. Ve dvou z těchto epoch komponenta P3 nebyla (chybná detekce) a ve třech epochách nebyla komponenta P3 detekována, i když obsažena byla. V jedné epoše byl za

P3 komponentu označen špatný atom, který evidentně P3 komponentou být nemohl, ale epocha ji obsahovala a tak by se to mohlo jevit jako správná detekce, což není. Počet správných detekcí je tedy 40 ze 46 epoch (87%).

Počet epoch	Detekována komponenta	Nedetekována komponenta	Detekce když neobsahuje	Nedetekováno když obsahuje	Sporně detekováno
46	43	3	2	3	1

Tabulka 4.6: Nalezené komponenty

Porovnání s ART2

Použil jsem výchozí nastavení a jen zvýšil počet iterací a epoch učení na 10. ART2 není úplně odladěná, výsledky zde uvádím jen pro srovnání (Tabulka 4.7). Počet správných detekcí je tedy 38 ze 46 epoch (82%).

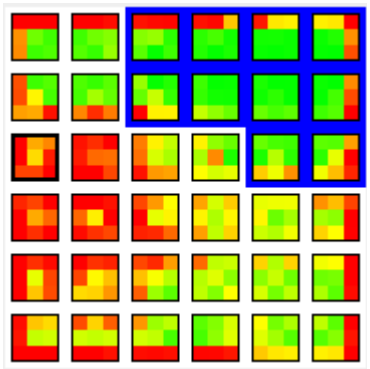
Počet epoch	Detekována komponenta	Nedetekována komponenta	Detekce když neobsahuje	Nedetekováno když obsahuje	Sporně detekováno
46	37	9	1	8	5

Tabulka 4.7: Porovnání s ART2

4.1.2 Osoba 2 – kanál O1, komponenty N1 a N2

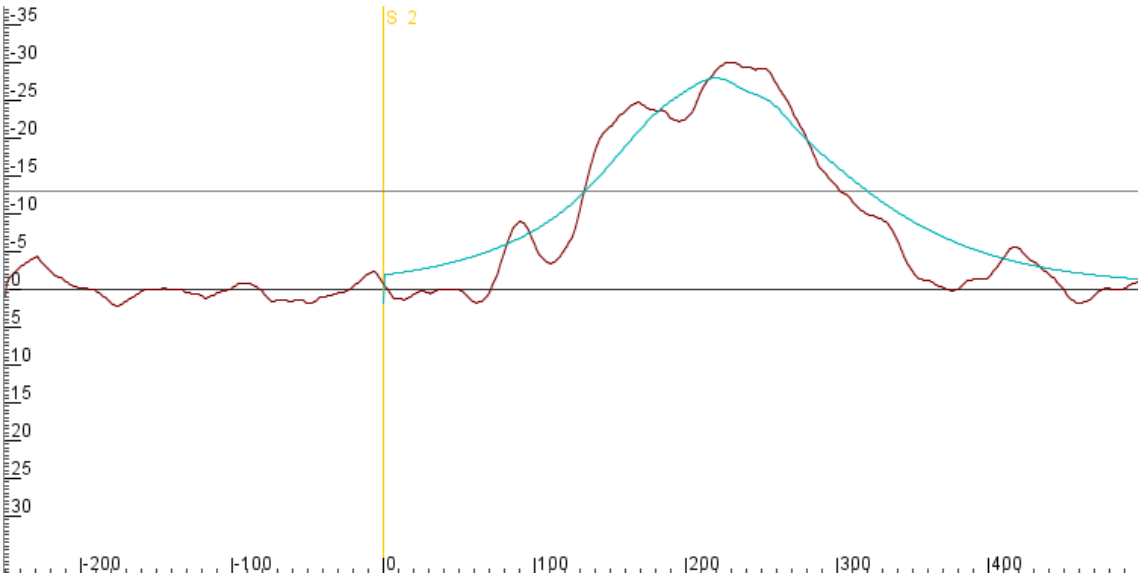
Tato data nejsou moc dobře naměřená, ale jsou zde celkem výrazné komponenty N1 a N2, bohužel jsou blízko u sebe a tak splývají často v jeden vrchol.

Byly použity filtry se stejným nastavením jako v prvním případě, kromě průměrování signálu, které použito nebylo. Tak vzniklo 34 epoch a každá z nich pak byla rozložena matching pursuitem na 10 atomů. Klasifikace proběhla s výchozím nastavením. Po nastavení podobnosti (shluky 85%, atomy 51%) byla nalezena hledaná komponenta, podbarvená modře na obrázku 4.10.



Obrázek 4.10: Mapa s nalezenou komponentou

Na obrázku 4.11 jsou vidět nalezené komponenty N1 a N2, které byly obě spojeny do jedné a jsou hodně posunuty dále v čase (světle modrá). Na průměru všech epoch (hnědá) jsou více patrné. Tato detekce již není příliš ideální, protože pozice komponent a jejich amplituda se v signálu jednotlivých epoch velmi mění.



Obrázek 4.11: Nalezené komponenty N1 a N2

Úspěšnost detekce byla i v tomto případě celkem vysoká, podrobné výsledky viz tabulka 4.8. Správně byla komponenta detekována v 88% epoch, ale je nutné si uvědomit, že se nejedná jen o jednu komponentu, ale dvě spojené komponenty (a pravděpodobně i několik menších artefaktů, které se filtrům nepodařilo odstranit). Tato metoda tedy nedokázala v tomto případě správně komponenty oddělit, ale když se člověk podívá na jednotlivé epochy sám, tak to ve většině z nich v tomto případě také nedokáže.

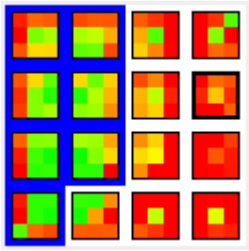
Počet epoch	Detekována komponenta	Nedetekována komponenta	Detekce když neobsahuje	Nedetekováno když obsahuje	Sporně detekováno
34	32	2	1	2	1

Tabulka 4.8: Nalezené komponenty

4.1.3 Osoba 3 – kanál PZ, komponenta P3

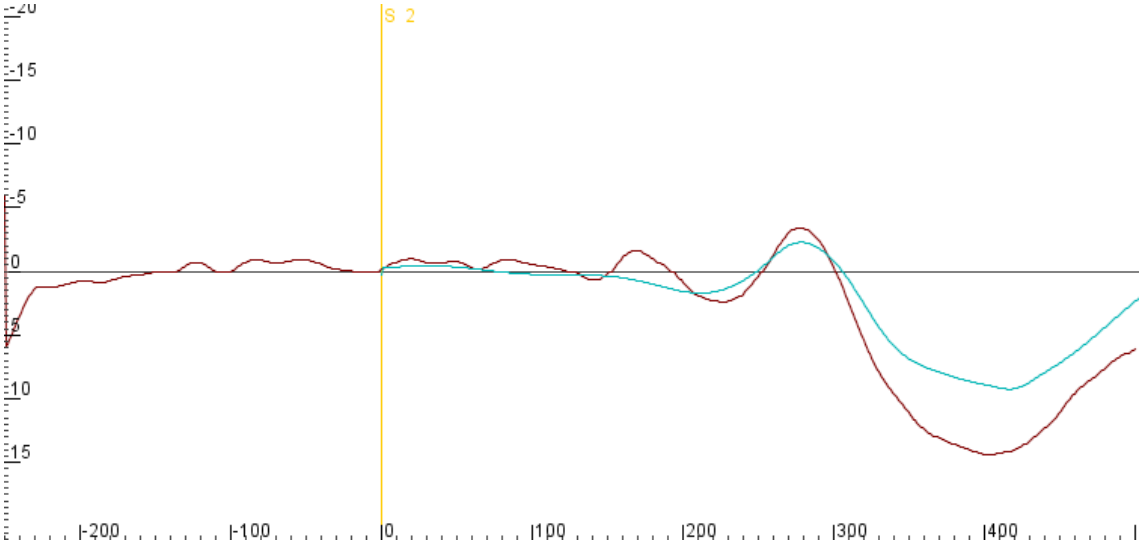
Tato data jsou celkem dobře naměřená, a je zde celkem výrazná komponenta P3, která je spojená s obloukem možné N1 nebo N2.

Byly použity filtry se stejným nastavením jako v prvním případě, kromě průměrování signálu, které použito nebylo a filtr s pohybujícím se okénkem byl nastaven na 25 ms délka okénka a rozdíl 40 μV. Tak vzniklo 50 epoch a každá z nich pak byla rozložena matching pursuitem na 6 atomů. Klasifikace proběhla s nastavením jako v prvním případě. Po nastavení podobnosti (shluky 59%, atomy 57%) byla nalezena hledaná komponenta, podbarvená modře na obrázku 4.12.



Obrázek 4.12: Mapa s nalezenou komponentou

Na obrázku 4.13 je vidět nalezená komponenta P3 (světle modrá), je vidět jak je spojená s předchozí zápornou komponentou pravděpodobně N2. Na průměru všech epoch (hnědá) jsou více patrné N1, N2 i P3.



Obrázek 4.13: Nalezené komponenta P3

Úspěšnost detekce byla i v tomto případě celkem vysoká, podrobné výsledky viz tabulka 4.9. Správně byla komponenta detekována v 80% epoch, ale je nutné si uvědomit, že se nejedná jen o jednu komponentu, ale dvě spojené komponenty. Tato metoda tedy nedokázala v tomto případě správně komponenty oddělit,

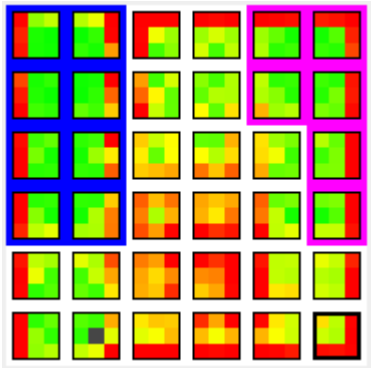
protože atomy vytvořené matching pursuitem procházejí často více komponentami. V tomto případě člověk oddělit tyto komponenty vizuálně dokáže.

Počet epoch	Detekována komponenta	Nedetekována komponenta	Detekce když neobsahuje	Nedetekováno když obsahuje	Sporně detekováno
50	41	9	3	4	3

Tabulka 4.9: Nalezené komponenty

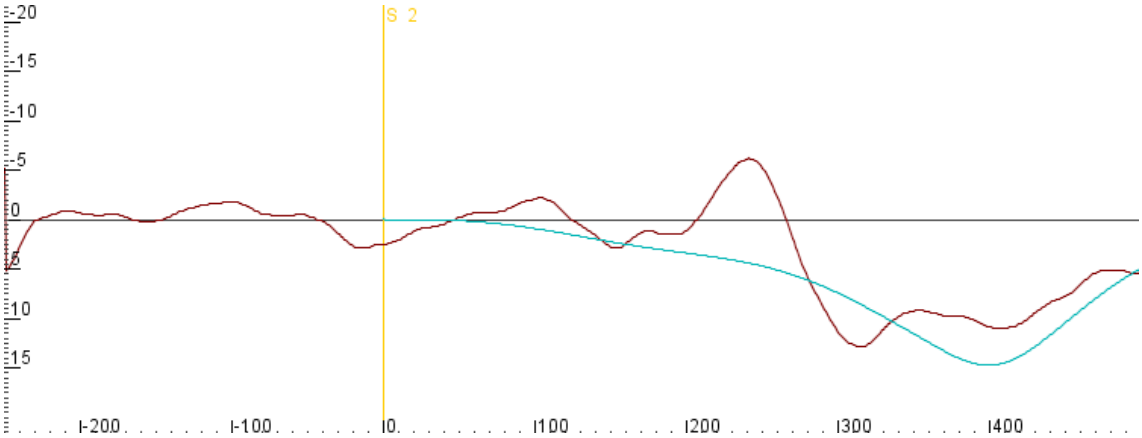
4.1.4 Osoba 4 – kanál PZ, komponenty N1 a P3

Tato data jsou dobře naměřená, a jsou to jediná data ve kterých se podařilo oddělit komponenty N1 (fialová) a P3 (modrá) viz obrázek 4.14.

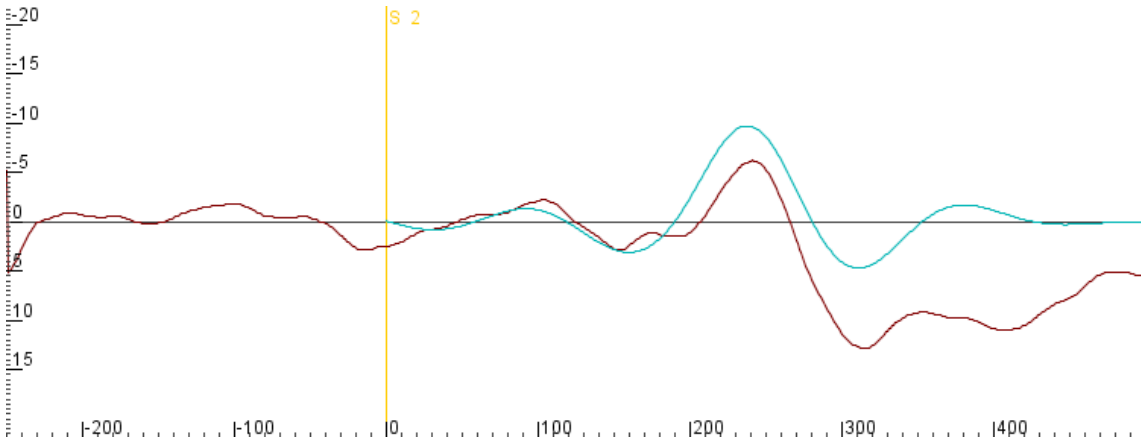


Obrázek 4.14: Mapa s nalezenou komponentou

Na obrázku 4.15 je vidět nalezená komponenta P3 a na obrázku 4.16 je vidět komponenta N1 (zde se může jednat i o N2 pokud bychom za N1 označili malý vrchol na začátku).



Obrázek 4.15: Nalezené komponenta P3



Obrázek 4.16: Nalezená komponenta N1

Nastavení filtrů bylo jako v předchozím případě a klasifikace byla ve výchozím nastavení.

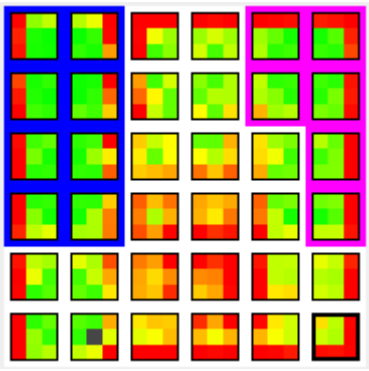
Úspěšnost detekce již nebyla tak vysoká, podrobné výsledky viz tabulka Chyba: zdroj odkazu nenalezen. Správně byla komponenta P3 detekována v 68% epoch a N1 v 75% epoch. Jedná se, ale o jediný případ kdy se mi podařilo komponenty oddělit.

	Počet epoch	Detekována komponenta	Nedetekována komponenta	Detekce když neobsahuje	Nedetekováno když obsahuje	Sporně detekováno
P3	40	29	11	1	11	1
N1	40	41	9	0	8	2

Tabulka 4.10: Nalezené komponenty

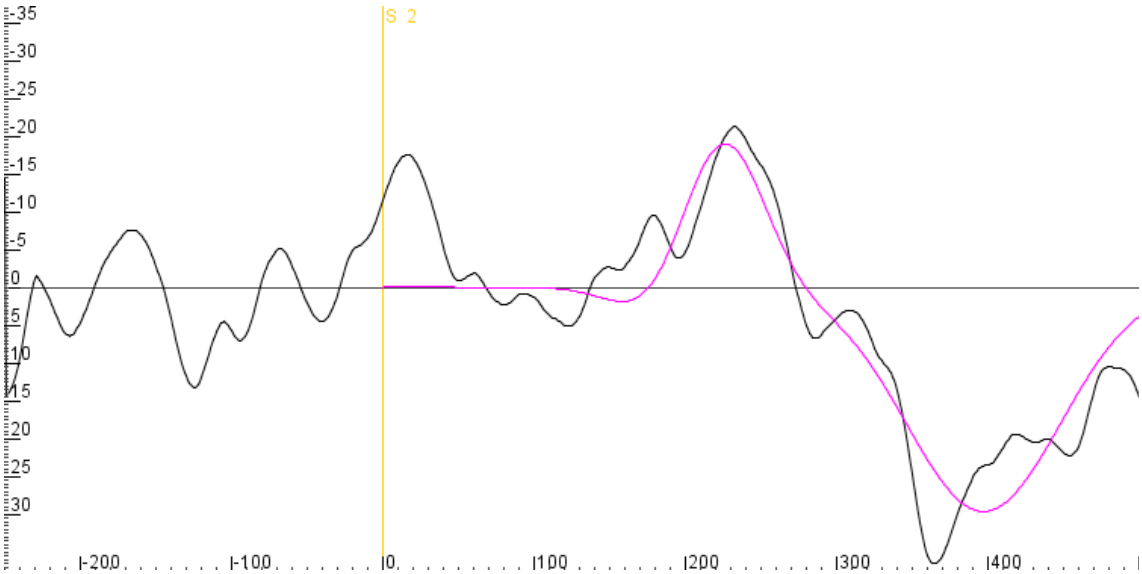
4.2 Použití této metody jako filtru

Tato metoda by se dala použít jako filtr pro odstranění nežádoucích částí signálu jako je šum, artefakty a ostatní vlivy, které se v signálu nacházejí, ale nemají pravidelný výskyt. Neuronová síť snadno najde ty atomy, které se opakují ve větším množství epoch a z těch je pak možné zrekonstruovat zpět celé epochy bez těchto nežádoucích částí. Na obrázku 4.17 jsou vyznačeny shluky podobných atomů nacházejících se ve většině epoch.



Obrázek 4.17: Shluky nejčastějších atomů

Na následujícím obrázku (4.18) je vidět epochu zrekonstruovanou ze dvou atomů (růžová barva) patřících do výše zmíněných shluků. Po takovémto předzpracování by bylo možné hledat komponenty v signálu dalšími metodami a tak dále zlepšit jejich detekci.



Obrázek 4.18: Zrekonstruovaná epocha

4.3 Shrnutí výsledků

U osoby 1, 3 a 4 se mi podařilo úspěšně najít komponentu P3, u osoby 1 a 3 se nepodařilo najít komponentu N1, protože byla málo výrazná a atomy aproximující komponentu P3 jí často procházely také, a tak nebylo možné je od sebe oddělit. U osoby 4 se naopak toto oddělení komponent podařilo, i když procento detekce již nebylo tak vysoké. U osoby 2 jsem se pokusil detekovat vizuální komponentu N1 a N2 na zadní části hlavy, kde se nachází zrakové centrum a kde by měla být výraznější. Bohužel tomu tak nebylo a obě komponenty splývali do sebe a ani nevypadají moc dobře (v naměřených datech je jde špatně vidět i očima a odlišit je od sebe často ani nejde).

Výsledky testování na signálu, který neobsahuje žádné ERP komponenty zde neuvádím, protože přesně podle očekávání v něm nebyly žádné nalezeny. K falešné detekci tedy nedochází, protože program kontroluje zastoupení nalezených komponent v epochách a pokud je nízké, tak je za komponenty nepovažuje.

Pro zlepšení detekce bude nutné vylepšit kvalitu filtrů (například použít jinou knihovnu, nebo použít jejich implementaci z ERPLAB Toolboxu pro MATLAB). Lepší filtry budou méně vyhlazovat ze signálu detaily a nebudou tolik měnit jeho časový průběh (což dělá aktuální implementace FIR filtru). Kromě vylepšení filtrů by stálo za vyzkoušení vyměnit algoritmus matching pursuit za nějakou jinou metodu rozkladu signálu. Změnou funkcí ve slovníku matching pursuitu by se mohlo také dosáhnou vylepšení tvorby atomů, aby lépe odpovídaly hledaným komponentám a odstranil se problém kdy atom prochází více než jednou komponentou.

Dále se nabízí použití této metody jako filtru jak je rozebráno v předchozí podkapitole. V takto filtrovaných epochách jsou mnohem lépe vidět ERP komponenty a nevadí ani skutečnost, že některý atom aproximoval více komponent najednou, nebo že komponentu tvoří více atomů.

5 Závěr

Jako metodu pro extrakci příznaků z EEG signálu jsem si vybral algoritmus matching pursuit, který se jevil jako nejvhodnější, protože se snaží atomy aproximovat nejvýraznější části signálu a ty by měly být tvořeny právě ERP komponentami. Použil jsem matching pursuit z knihovny EEGDSP vyvíjený na katedře informatiky a výpočetní techniky (KIV) a matching pursuit implementovaný genetickými algoritmy vytvořený Vítem Bábelem v rámci jeho bakalářské práce [7]. Zde vznikl první problém, a to že matching pursuit často vytvoří atom, který kromě hledané komponenty aproximuje i nějakou jinou část signálu, nebo dokonce jedním atomem aproximuje více komponent, proto jsou atomy tvořící jednu ERP komponentu často dost odlišné. Z tohoto důvodu je hledání správných atomů, které tvoří ERP komponenty obtížné. Vycházel jsem z předpokladu, že si tyto atomy, ale budou alespoň podobné, což se i potvrdilo.

Pro hledání podobných atomů jsem implementoval Kohonenovu neuronovou síť, ta má výhodu, že k učení nepotřebuje učitele a nemusí znát vzorové výstupy. Další výhodou je, že organizuje vzniklé shluky atomů do dvourozměrné mapy, kde shluky podobných atomů budou vedle sebe a to značně usnadní hledání atomů tvořících komponenty. Implementoval jsem algoritmus, který pomocí několika způsobů pro porovnání podobnosti signálů najde ve vzniklé mapě shluky sobě podobných neuronů.

Pro přehledné zobrazení výsledků a manipulaci s nimi jsem vytvořil grafické uživatelské rozhraní. To umožňuje pohodlné zpracování dat od jejich načtení, přes použití filtrů, atomizaci matching pursuitem, klasifikaci vzniklých atomů, po zobrazení nalezených komponent. Dále je možné rozpracované výsledky i ukládat a načítat.

Vytvořený program jsem otestoval na datech naměřených v neuroinformatické laboratoři na KIVu. Více informací o výsledcích v předchozí kapitole.

Tuto metodu pro hledání ERP komponent v signálu je možno celkem úspěšně použít pro nalezení jedné nejvýraznější komponenty v signálu. Pokud je komponent více a nejsou tak výrazné, tak jsou velmi často spojeny do sebe, nebo vyhlazeny filtrací, tato metoda je pak nedokáže najít. Dalším problémem je již výše zmíněná odlišnost jednotlivých atomů, které aproximují jednu ERP komponentu. Lepší tvorby atomů by se dalo dosáhnout například upravením slovníku funkcí, které algoritmus matching pursuit používá.

Jako další možnost aplikace této metody se jeví její použití jako filtru, protože neuronová síť snadno najde ty atomy, které se opakují ve většině epoch a z těch se pak dají zrekonstruovat epochy bez šumu a artefaktů.

Seznam použité literatury

[1] Luck, Stephen J. *An introduction to the event-related potential technique*. 1. vydání. MIT Press, London 2005. ISBN 978-0-262-62196-0

[2] Tomáš Řondík. *Metody zpracování ERP signálů*. Diplomová práce. Katedra informatiky a výpočetní techniky ZČU v Plzni, 2010

[3] Candace Markley, Steve Luck, Javier Lopez-Calderon. *ERPLAB ToolBox User's Manual*. [online] 2011, dostupný na http://erpinfo.org/erplab/erplab-documentation/manual

[4] Petr Kellnhofer. *Knihovna pro zpracování signálů*. Katedra informatiky a výpočetní techniky ZČU v Plzni, 2011

[5] Laurene Fausett. *Fundamentals of Neural Networks : architectures, algorithms, and applications*. 1. vydání. Prentice Hall, 1994. ISBN 0-13-334186-0

[6] Ali Darvishi. *Translation Invariant Approach for Measuring Similarity of Signals*. Department of Computer and Electrical Engineering Babol Noushirvani University of Technology, 2009

[7] Vít Bábel. *Implementace algoritmu matching pursuit s využitím genetických algoritmů*. Bakalářská práce. Katedra informatiky a výpočetní techniky ZČU v Plzni, 2012

Seznam obrázků

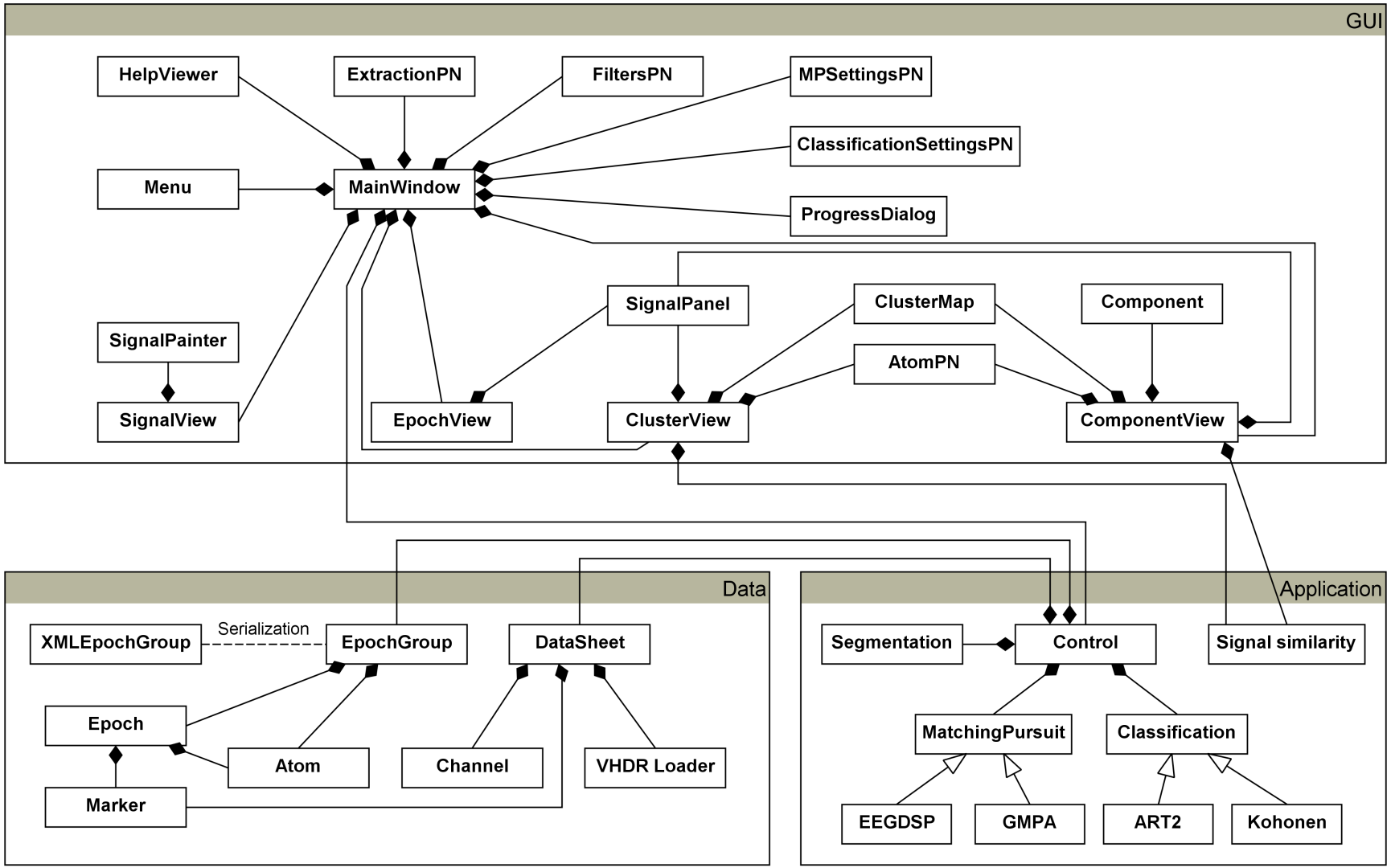
Přehled zkratek

EEG	Elektroencefalografie
ERP	Event-related potential, evokovaný potenciál
N1, N2 a P3	ERP komponenty
FIR	Diskrétní lineární filtr s konečnou impulzní odezvou (finite impulse response)
SOM	Self organizing map (samoorganizující mapa)
EEGDSP	Knihovna pro zpracování signálů vyvíjená na KIVu
KIV	Katedra informatiky a výpočetní techniky
GUI	Grafické uživatelské rozhraní
ART2	Adaptative Resonance Theory, neuronová síť
XML	Extensible markup language, značkovací jazyk
JAXB	Java Architecture for *XML* Binding, knihovna pro práci s XML
XSD	XML Schema Definition, informace o struktuře XML souboru
MATLAB	Matrix Laboratory, matematický software
ERPLAB	Plugin do MATLABu pro práci s ERP a EEG

Obsah CD

- **build** – Adresář obsahuje spustitelnou verzi programu.
- **data** – Adresář obsahuje naměřená data.
- **doc** – Adresář obsahuje pdf s textem bakalářské práce, zdrojový dokument odt pro OpenOffice, použité obrázky a vygenerovanou programátorskou dokumentaci JavaDoc.
- **src** – Adresář obsahuje zdrojové soubory pro překlad programu ve formě projektu pro vývojové prostředí NetBeans. Nachází se zde i soubor *build.xml* pro překlad programu z příkazové řádky. Pro vytvoření spustitelného jar souboru je možno použít příkaz `ant jar`.

Přílohy



Příloha 1: Zjednodušený model architektury projektu

Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

ERP-CLASSIFIER
UŽIVATELSKÁ PŘÍRUČKA

Plzeň, 2012 Jan Strejc

Obsah

6 Program

6.1 Překlad

Pro překlad jsou k dispozici zdrojové soubory a Ant skript v souboru *build.xml*. Pro vytvoření spustitelného *.jar* souboru použijte cíl *jar*. Je také přiložena celá složka s projektem z NetBeans a tak je možné program snadno upravit a přeložit pomocí NetBeans.

6.2 Instalace

Program není nutné instalovat, je možné nakopírovat do vlastního počítače a spouštět jej odtamtud. Na přiloženém CD je k dispozici přeložená verze programu pro Javu 1.7. Kromě souboru *erp-classifier.jar* je nutné do stejné složky nakopírovat složku lib s knihovnami.

6.3 Požadavky

Program pro svůj běh vyžaduje nainstalovanou aktuální verzi Javy 1.7 (mělo by být možné spustit jej na starší verzi 1.6, ale bezchybná funkčnost není zaručena a bude nutné provést rekompilaci). Pro bezproblémový chod je doporučeno mít alespoň 1 GB volné paměti RAM.

Hlavní platformou pro použití je MS Windows, ale program je možné používat i na Linuxu, nebo Mac Os (testováno na Windows 7 a Ubuntu 12.04).

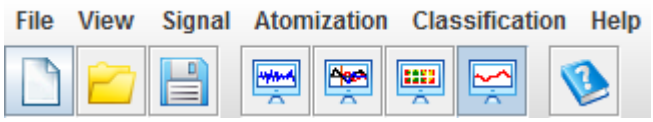
6.4 Spuštění

Pokud je správně nainstalována Java tak je možné spouštět program dvojklikem na soubor *erp-classifier.jar*. Jinak lze spuuštění provést z příkazové řádky příkazem *java -jar erp-classifier.jar*.

7 Uživatelské rozhraní

7.1 Menu

Menu slouží jako hlavní ovládací prvek aplikace a jsou z něj přístupné všechny funkce, ty nejčastější jsou umístěny na toolbaru pod menu (Obrázek 7.1). Aktuálně nepoužitelné položky menu jsou šedivé a nejde je použít.



Obrázek 7.1: Menu a toolbar

Seznam jednotlivých položek menu:

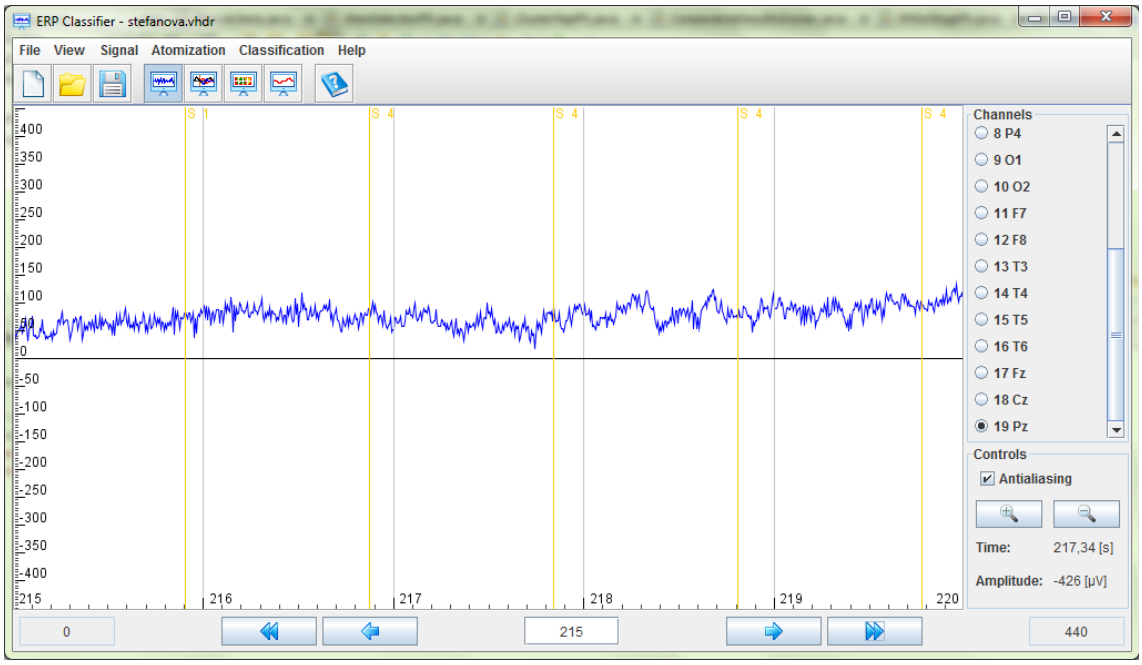
- **File**
  - **New Project** – vytvoří nový projekt z VHDR souboru.
  - **Open Project** – otevře uložený projekt z XML souboru.
  - **Save Project** – uloží rozpracovaný projekt do XML souboru.
  - **Exit** – ukončí aplikaci.
- **View**
  - **View Signal** – zobrazí načtený neupravený signál.
  - **View Epochs and Atoms** – zobrazí extrahované epochy a jejich atomy vytvořené matching pursuitem.
  - **View Clusters** – zobrazí neuronovou sítí vytvořené shluky atomů.
  - **View Components** – zobrazí okno s tvorbou komponent.
- **Signal**
  - **Epoch Extraction** – zobrazí panel pro extrakci epoch.
  - **Epoch Averaging** – zobrazí panel s filtry a přepne na záložku pro průměrování epoch.
  - **Treshold Fitrer** – zobrazí panel s filtry a přepne na záložku pro prahový fitr.
  - **FIR Fitrer** – zobrazí panel s filtry a přepne na záložku pro FIR filtr.
  - **Moving Window Fitrer** – zobrazí panel s filtry a přepne na záložku pro filtr pohybujícím se okénkem.

- **Signal Averaging Fitrer** – zobrazí panel s filtry a přepne na záložku pro filtr na průměrování signálu.

- **Atomization**
  - **GMPA** – přepnutí modulu atomizace na Genetický matching pursuit.
  - **MPFFT** – přepnutí modulu atomizace na obyčejný matching pursuit z EEGDSP.
  - **Settings** – nastavení aktuálně vybraného modulu atomizace.
  - **Atomize** – spustí atomizaci aktuálně vybraným modulem.
- **Classification**
  - **Kohonen neural network** – přepnutí modulu klasifikace na Kohonenovu neuronovou síť.
  - **ART2 neural network** – přepnutí modulu klasifikace na ART2.
  - **Settings** – nastavení aktuálně vybraného modulu klasifikace.
  - **Classify** – spustí klasikaci aktuálně vybraným modulem.
  - **Atomize and Classify** – spustí atomizaci a pak klasikaci aktuálně vybranými moduly.
- **Help**
  - **Help** – zobrazí nápovědu k programu.

7.2 Načtení signálu

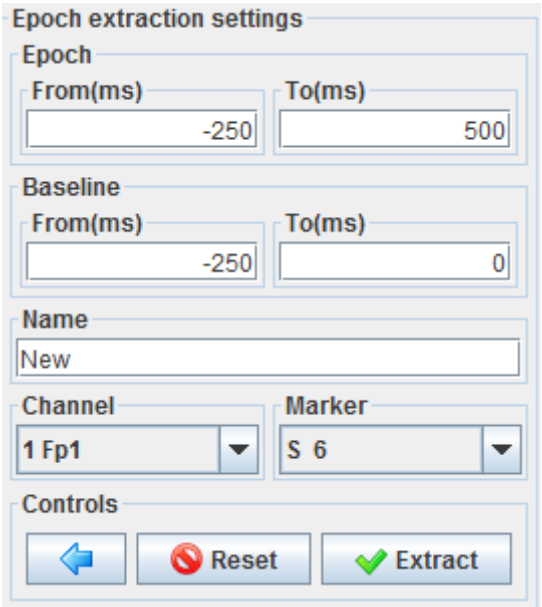
Pomocí *File* → *New Project* vybereme VHDR soubor, který chceme načíst. Poté se nám objeví okno s načteným signálem, později je možné signál zobrazit pomocí *View* → *Signal* (Obrázek 7.2). V pravé části je možno si vybrat kanál, který chceme zobrazit dole se můžeme pomocí šipek posouvat v signálu.



Obrázek 7.2: Zobrazení signálu

7.3 Extrakce epoch

Pro extrakci epoch musí být načtený signál. Pro nastavení extrakce klikneme na *Signal* → *Epoch Extraction* a zobrazí se nám následující panel (Obrázek 7.3).



Obrázek 7.3: Extrakce epoch

První dvě políčka určují kde bude epocha začínat a končit (vzhledem k markeru), druhá dvě určují úsek epochy, který bude použit pro korekci baseline. Skupinu je možné pojmenovat zadáním jména do políčka *Name*. Políčko *Channel* slouží pro výběr kanálu (elektrody), jehož signál se použije a políčko Marker pro výběr markeru (stimulu), od kterého chceme epochy vytvořit.

Extrakci spustíme tlačítkem *Extract*, tlačítko *Reset* vrátí hodnoty do původního stavu a šipka zpět panel schová.

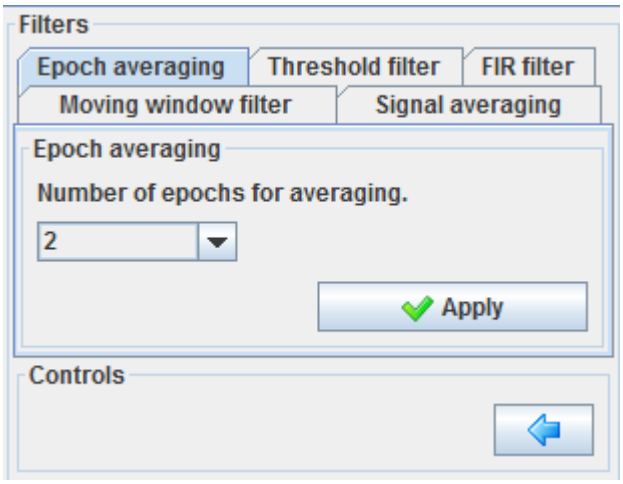
Po dokončení extrakce se zobrazí okno s epochami (Obrázek 7.4). V dolní části se nachází ovládání pro přepínání epoch a možnost odebrání epochy, která se nám nelíbí. V pravo je seznam zobrazených signálů (zde později přibudou atomy a signál epochy aproximovaný matching pursuitem) a pod ním je ovládání grafu (zoom, vyhlazování, možnost otočit osu Y a informace aktuální pozici v signálu). Svislá oranžová čára na grafu představuje marker, osa Y představuje napětí v µV a osa X čas v milisekundách.

Obrázek 7.4: Zobrazení epoch

7.4 Filtry a průměrování epoch

7.4.1 Průměrování epoch

Pro průměrování epoch vybereme *Signal* → *Epoch Averaging*. Předem musíme mít vytvořeny epochy a pak se nám zobrazí následující panel (Obrázek 7.5).

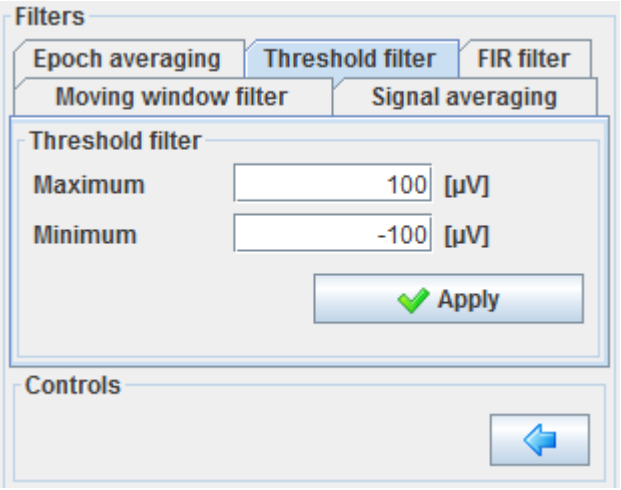


Obrázek 7.5: Průměrování epoch

Zde je možné nastavit kolik epoch má být průměrováno. Tlačítko *Apply* provede průměrování.

7.4.2 Prahový filtr

Pro prahový filtr vybereme *Signal* → *Treshold Filter*. Předem musíme mít vytvořeny epochy a pak se nám zobrazí následující panel (Obrázek 7.6).

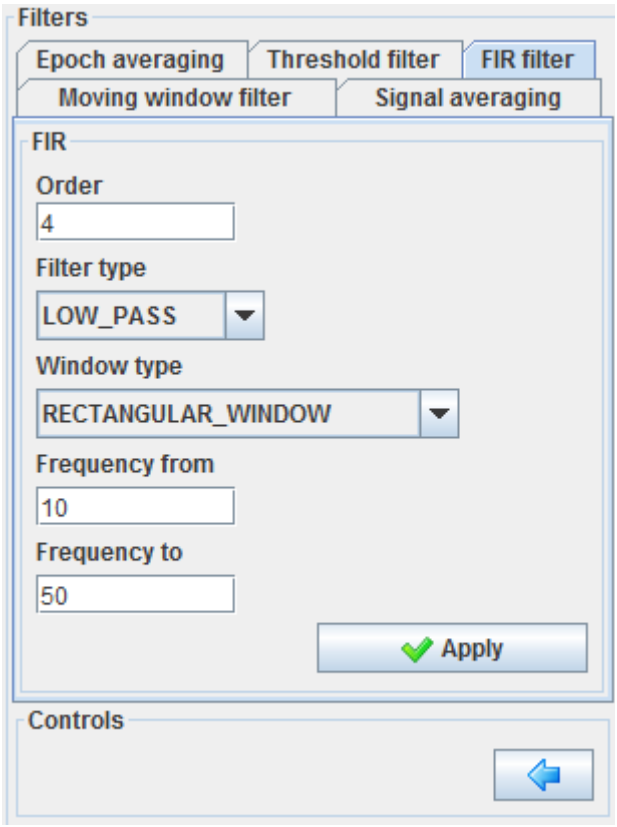


Obrázek 7.6: Prahový filtr

Zde je možno nastavit minimální a maximální hranici, která je přípustná pro signál v epochách, po spuštění tlačítkem *Apply* filtr odstraní ty epochy, které se do tohoto limitu nevejdou.

7.4.3 FIR filtr

Pro FIR filtr vybereme *Signal* → *FIR Filter*. Předem musíme mít vytvořeny epochy a pak se nám zobrazí následující panel (Obrázek 7.7).

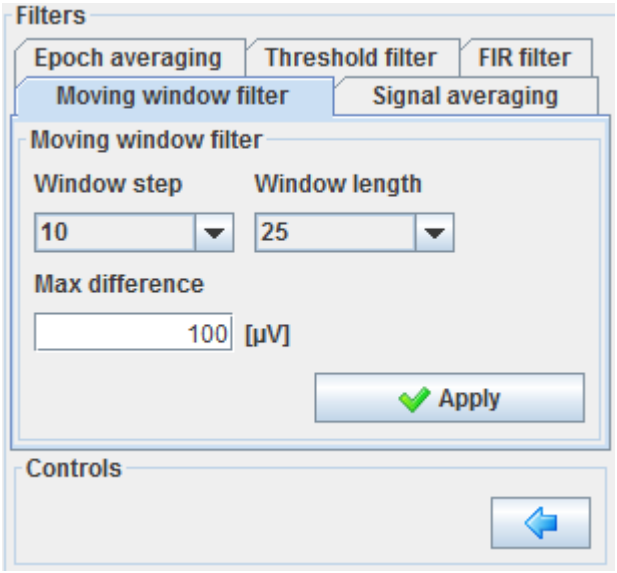


Obrázek 7.7: FIR filtr

Zde je možno nastavit řád filtru (vyšší řád znamená lepší filtraci, ale způsobí posunutí fáze signálu o řád filtru), typ filtru, typ okénka a dolní a horní frekvenci pro jednotlivé filtry. Tlačítkem *Apply* spustíme filtraci.

7.4.4 Filtr pohybujícím se okénkem

Pro filtr pohybujícím se okénkem vybereme *Signal* → *Moving Window Filter*. Předem musíme mít vytvořeny epochy a pak se nám zobrazí následující panel (Obrázek 7.8).

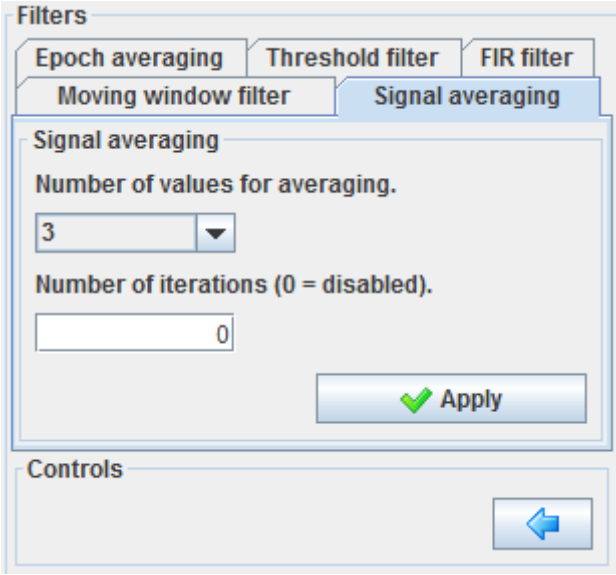


Obrázek 7.8: Filtr pohybujícím se okénkem

Zde můžeme nastavit krok po jakém se bude okénko posouvat, jeho délku a maximální rozdíl hodnot signálu v okénku. Po spuštění tlačítkem *Apply* filtr odstraní ty epochy, které se do tohoto limitu nevejdou.

7.4.5 Filtr pro průměrování signálu

Pro filtr průměrování signálu vybereme *Signal* → Signal Averaging *Filter*. Předem musíme mít vytvořeny epochy a pak se nám zobrazí následující panel (Obrázek 7.9).

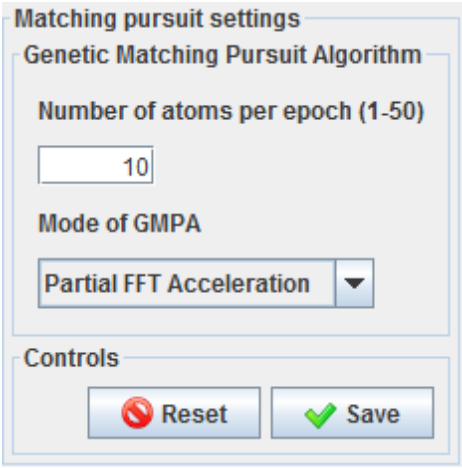


Obrázek 7.9: Filtr pro průměrování signálu

Zde si můžeme vybrat počet hodnot pro průměrování a počet iterací filtru. Nastavení příliš velkého počtu hodnot a iterací může ze signálu odstranit důležité detaily. Tlačítkem *Apply* spustíme filtraci.

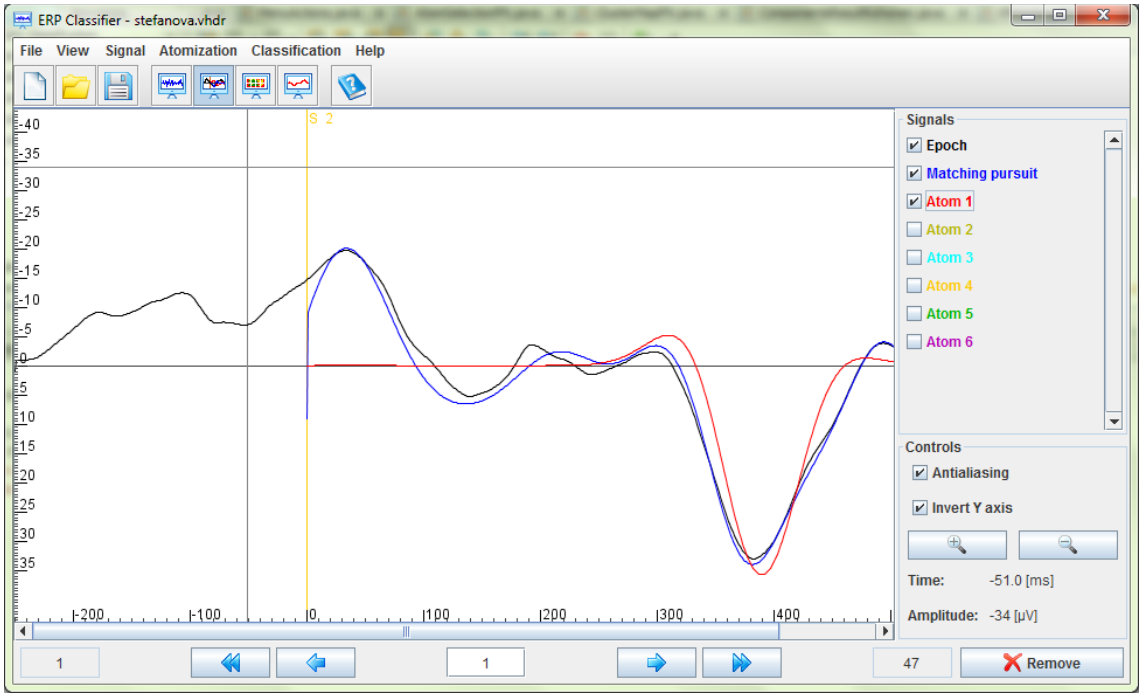
7.5 Atomizace

V menu *Atomization* můžeme vybrat, který modul pro atomizaci chceme použít. Pak si můžeme zobrazit jeho nastavení. Na obrázku 7.10 je nastavení pro Genetický matching pursuit, zde je k dispozici nastavení počtu atomů na epochu a mód matching pursuitu.



Obrázek 7.10: Nastavení matching pursuitu

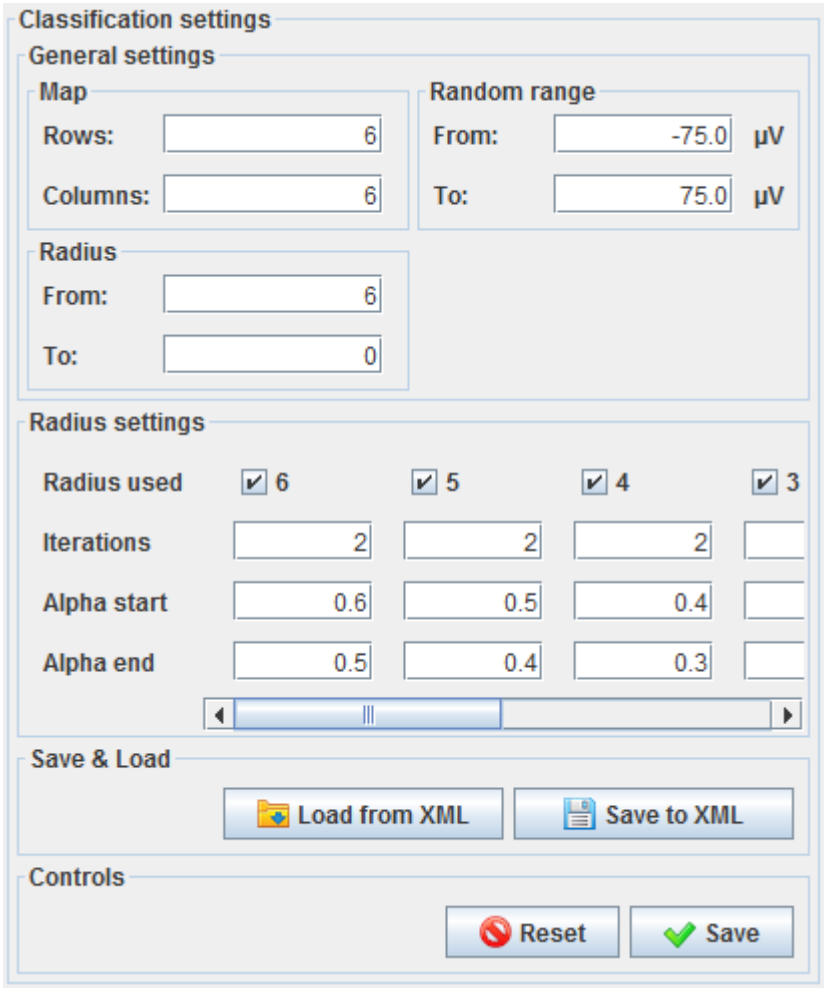
Tlačítkem *Save* uložíme nastavení. V menu pak nejdeme ještě položku *Atomize*, která spustí tvorbu atomů. Na obrázku 7.11 jsou vidět výsledné atomy, ovládání se shoduje s dříve uvedeným oknem pro zobrazení epoch.



Obrázek 7.11: Zobrazení výsledných atomů

7.6 Klasifikace

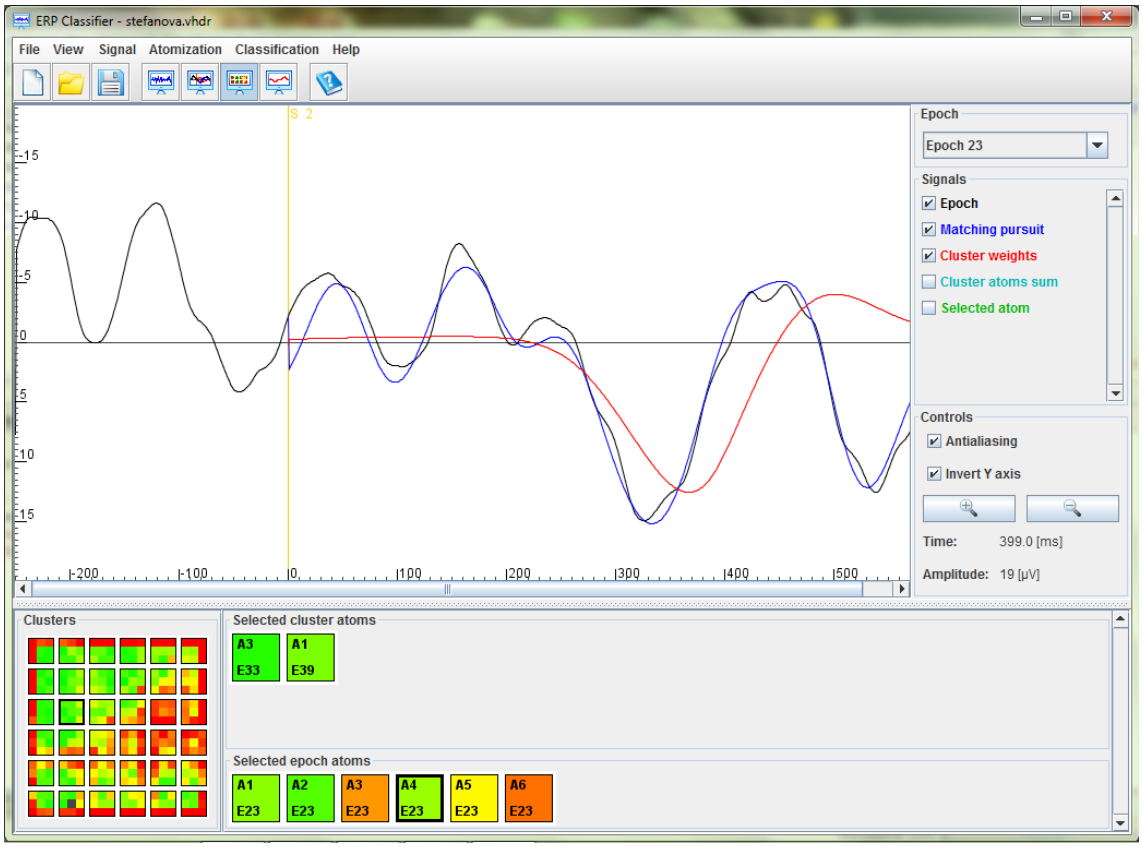
V menu *Classification* si můžeme vybrat jaký modul pro klasifikaci budeme používat. Pak si můžeme zobrazit jeho nastavení. Na obrázku 7.12 je vidět natavení pro Kohonenovu neuronovou síť. Ve vrchní části možno nastavit velikost mapy neuronů, rozsah náhodných hodnot pro váhy sítě a použité radiusy pro aktualizaci vah neuronů. Ve středu se nachází nastavení jednotlivých rasiusů (počet iterací, počáteční a koncov= učení). Dole je možno uložit a nahrát nastavení sítě do a z XML.



Obrázek 7.12: Nastavení Kohonenovi neuronové sítě

Stisknutím tlačítka Save uložíme nastavení. V menu pak nejdeme ještě položku *Classify* pro spuštění klasifikace a položku *Atomize and Classify* pro sekvenční spuštění obou výpočtů.

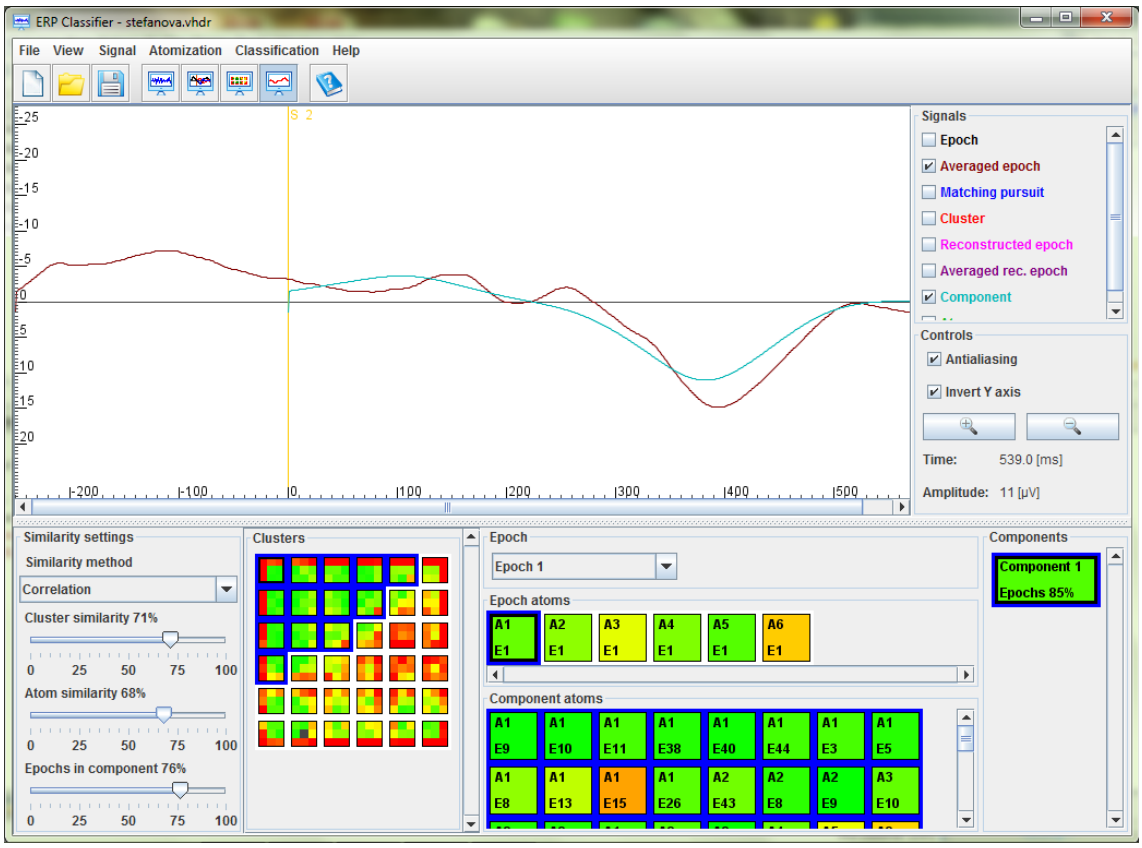
Na obrázku 7.13 je okno pro zobrazení výsledků klasifikace. V pravé části jsou k dispozici různé signály a výběr epochy. V levém dolním rohu je mapa neuronů po kliknutí na některý neuron (jeho shluk) se zobrazí signál jeho vah a průměr signálu shluku jeho atomů. Uprostřed dole jsou potom tlačítka pro zobrazení jednotlivých atomů, nahoře jsou ty z vybraného shluku a dole ty z vybrané epochy. U neuronů barvy zobrazují podobnost vah neuronů jejich sousedům (osm čtverců po obvodu tlačítka) a podobnost atomů jeho shluku s jeho vahami (prostřední čtverec), šedivá znamená že shluk neobsahuje žádné atomy.



Obrázek 7.13: Zobrazení výsledků klasifikace

7.7 Tvorba komponent

Po dokončení klasifikace je k dispozici i okno pro tvorbu komponent z *View* → *Components* (Obrázek 7.14). V pravé části se jsou k dispozici různé signály. V levém spodní rohu je nastavení kritérií pro tvorbu komponent (typ porovnání, podobnost shluků, podobnost atomů shluku a minimální zastoupení epoch v komponentě). Vedle se nachází mapa shluků, na které jsou podbarveny ty, co patří do nějaké komponenty. Vpravo dole je seznam nalezených komponent, po stisknutí se zobrazí uprostřed její atomy a na grafu signálů tvar této komponenty. Uprostřed je možno přepínat zobrazenou epochu a prohlížet atomy vybrané epochy a komponenty.



Obrázek 7.14: Tvorba komponent