České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Jan Žegklitz

Studijní program: Softwarové technologie a management

Obor: Inteligentní systémy

Název tématu: Aplikace algoritmu NEAT pro zpracování zvukových signálů

Pokyny pro vypracování:

1. Prozkoumejte algoritmus pro evoluci neuronových sítí NEAT.
2. Vyberte jeho vhodnou implementaci.
3. Otestujte vlastnosti algoritmu pro generování a filtraci zejména zvukových signálů.
4. Proveďte experimenty s dopřednými i rekurentními sítěmi a různými typy neuronů a synaptických spojů.
5. Modifikujte stávající implementaci tak, aby umožnila interaktivní evoluci generátorů, případně filtrů, podobně jako funguje projekt Picbreeder (picbreeder.org) pro generování obrázků.

Seznam odborné literatury: Dodá vedoucí práce.

Vedoucí bakalářské práce: Ing. Jan Drchal

Platnost zadání: do konce zimního semestru 2011/2012

L.S.

prof. Ing. Vladimír Mařík, DrSc.
vedoucí katedry

prof. Ing. Boris Šimák, CSc.
děkan

V Praze dne 31. 1. 2011

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra kybernetiky

Bakalářská práce

Aplikace algoritmu NEAT pro zpracování zvukových signálů

Jan Žegklitz

Vedoucí práce: Ing. Jan Drchal

Studijní program: Softwarové technologie a management, Bakalářský

Obor: Inteligentní systémy

26. května 2011

Poděkování

Chtěl bych vyjádřit své díky především vedoucímu své práce, Ing. Janu Drchalovi, za podnětné a mnohdy nezbytné rady, připomínky a návrhy, bez kterých bych tuto práci jen těžko dovedl do úspěšného konce.

Dále bych rád chtěl poděkovat své rodině a přátelům, kteří mě podporovali po celou dobu psaní práce a zejména v závěru.

Prohlášení

Prohlašuji, že jsem práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.
Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 26. května 2011 .....

Abstract

In this work I present a non conventional way of signal processing with use of neural networks. In a contrast against existing ways of neural netwok signal processing based on learning algorithms, I utilize the NEAT algorithm. This algorithm along with other modifications, which I present here too, and mainly my computer programe allows a user to easilly evolve his own unit for signal processing, without the need of thorough analysis of the desiread behaviour of such unit.

Abstrakt

V této práci seznámím čtenáře s netradičním způsobem zpracování zvukových signálů za pomoci neuronových sítí. Narozdíl od již existujících způsobů zpracování signálu neuronovými sítěmi, založených na učících se algoritmech, ve své práci využívám algoritmus NEAT. Tento algoritmus s několika dalšími modifikacemi, které v této práci také popíši, a zejména můj program, umožňuje uživateli jednoduše vyvinout vlastní jednotku pro zpracování signálu bez potřeby hluboké analýzy požadovaného chování.

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

1.1 Motivace

Zpracování signálů je v dnešní době úloha velice častá a pronikající do nejrůznějších oblastí techniky, od měření a zpracovávání fyzikálních veličin až po tvorbu hudby či práci se zvukem obecně. Téměř vždy, když je nějaký signál získán, je třeba jej nějakým způsobem zpracovat, nebo je naopak třeba nějaký signál vygenerovat.

Vyvinout jednotku pro zpracování či generování signálu, která se chová co nejblíže představě uživatele, je velice náročný úkol, jehož exaktní řešení může být extrémně náročné, či dokonce nemožné, a nebo je obtížné exaktně definovat požadavky uživatele. Ten totiž nemusí být schopen vyjádřit charakter výsledného signálu, ale pouze ví, který signál je „dobrý“ a odpovídá nebo se blíží jeho představám, a který je „špatný“.

V této práci ukáži jeden ze způsobů, jakým lze tento úkol řešit.

1.2 Struktura práce

Má práce se bude držet následující struktury:

V kapitole 2 specifikuji cíle své práce a uvedu obecné metody týkající se mého problému.

V kapitole 3 analyzuji algoritmus, ze kterého celá má práce vychází a uvedu výsledky v oblastech souvisejících s mou prací. Dále popíši návrh řešení.

V kapitole 4 popíši, jak jsem problém řešil. Popíši zejména modifikace základního algoritmu, které jsem učinil, dále popíši významné programové struktury, použité knihovny a vzájemnou interakci těchto.

V kapitole 5 uvedu jednotlivé experimenty, které jsem na výsledném progamu provedl, jejich metodiku a výsledky.

V 6., poslední, kapitole shrnu cíle a výsledky mé práce a diskutuji možné další pokračování této práce.

Kapitola 2

Popis problému, specifikace cíle

Cílem této práce je vytvořit program, s jehož pomocí bude, i v oblasti zpracování signálů nevzdělaný, uživatel schopen vyvinout jednotku pro zpracování signálů, která se bude chovat tak, jak uživatel požaduje. Těmito jednotkami budou především číslicové filtry, případně generátory, zvuku, realizované neuronovými sítěmi.

2.1 Neuronové sítě jako číslicové filtry

Neuronové sítě [4] jsou využívány především jako klasifikační či regresní modely. Nicméně je lze chápat i jako systém, který určitým způsobem modifikuje svůj vstup. Tento pohled na chování neuronových sítí je velice podobný tomu, jakou úlohu vykonávají číslicové filtry – také modifikují vstup, který je jim předložen. Má-li neuronová síť vhodné nastavení (zejména aktivační funkce), je možné vytvořit takovou topologii, která bude vykonávat stejnou funkci, jako libovolně zvolený FIR či IIR filtr [6]. Navíc, oproti klasickým číslicovým filtrům, neuronové sítě disponují nelineárními členy, které jsou schopny významně ovlivnit svůj výstup a mohou tedy být chápány jako rozšíření FIR a IIR filtrů o nelinearitu.

2.2 Vývoj pomocí genetických algoritmů

Genetické algoritmy [4] patří k tzv. black-box optimalizátorům. Umožňují snadno vytvářet řešení problémů jen na základě znalosti „hodnoty“ řešení či dokonce jen schopnosti porovnat dvě řešení. Díky této schopnosti jsou ideálním nástrojem na řešení problémů, u nichž je obtížné je exaktně charakterizovat. Problém, který jsem zde popsal, je dobře řešitelný (alespoň částečně) genetickými algoritmy.

2.2.1 Interaktivní evoluce

Interaktivní evoluce je takové užití genetického algoritmu, kdy ohodnocování jedinců (a případně i další genetické operátory jako je selekce apod.) nezajišťuje definovaná funkce či pravidlo, ale sám uživatel určí nějakou formou hodnotu – fitness funkci – každého jedince.

Tento přístup se využívá v tzv. evolučním umění, tedy produkování obrazových či zvukových děl. Nicméně princip interaktivní evoluce lze užít kdekoliv, kde ohodnocovací funkce není známa nebo výsledek významně záleží na vůli uživatele, což je i případ mé práce.

Kapitola 3

Analýza a návrh řešení

V této kapitole se zaměřím především na již existující práce související s touto prací, také popíši algoritmus NEAT [9] a možnosti jeho rozšíření na základě souvisejících prací.

3.1 Související práce

3.1.1 Projekt Picbreeder

Projekt Picbreeder [5] je velice zajímavý projekt velmi úzce související s mou prací. Projekt provozuje webové stránky, kde umožňuje uživatelům „vyvíjet“ obrázky, a to za pomoci algoritmu CPPN-NEAT [7]. Ten je rozšířením algoritmu NEAT o možnost různých aktivačních funkcí, kterými jednotlivé neurony mohou disponovat. Uživatelé mohou buď začít s již existujícími obrázky, nebo od úplného začátku. Uživateli je zobrazena sada obrázků a on vybírá ty, které se mu líbí a které budou tvořit rodiče nové generace. Cestou této interaktivní evoluce si uživatel „vyšlechtí“ svůj obrázek.

3.1.2 Digitální filtry

Digitální filtry [6] jsou struktury umožňující provádět filtraci vzorkovaného signálu pomocí matematických operací s tímto signálem. Příkladem těchto filtrů jsou FIR a IIR filtry.

FIR filtry k právě zpracovávanému vzorku přičítají lineární kombinaci několika dalších vzorků, které právě zpracovávanému předcházejí:

$$\begin{aligned} y(n) &= b_0 x(n) + b_1 x(n-1) + ... + b_M x(n-M) = \\ &= \sum_{i=0}^{M} b_i x(n-i) \end{aligned} \tag{3.1}$$

kde $y(n)$ je výstupní vzorek v čase n, b_i je i-tý koeficient filtru a $x(n-i)$ je vstupní vzorek zpožděný o i kroků.

IIR filtry v sobě zahrnují FIR filtry a jsou rozšířeny o lineární kombinaci tentokrát výstupních vzorků, které právě zpracovávanému předcházejí:

$$\begin{aligned} y(n) &= b_0 x(n) + b_1 x(n-1) + ... + b_M x(n-M) + \\ &\quad + a_1 y(n-1) + ... + a_N y(n-N) = \\ &= \sum_{i=0}^{M} b_i x(n-i) + \sum_{i=1}^{N} a_i y(n-i) \end{aligned} \tag{3.2}$$

3.1.3 FIR a IIR sítě

FIR a IIR sítě nebo obecněji sítě s vnitřní pamětí [10] jsou neuronové sítě, které ve své struktuře obsahují FIR či IIR filtry, a to na místě synaptických spojů nebo uvnitř neuronů. Tyto sítě jsou vhodné ke zpracování zvuku. Pro tyto účely existují deterministické algoritmy učení těchto sítí založených na algoritmu backpropagation [4, 10] upraveného pro práci s časovými zpožděními. Lze je využít pro predikci signálu nebo pro jejich spojování. Expresivita těchto sítí je samozřejmě větší než normálních neuronových sítí.

3.2 Algoritmus NEAT

Tento algoritmus [9] je jedním z genetických algoritmů určených k vyvíjení neuronových sítí, který kromě vah synaptických spojů vyvíjí i samotnou topologii sítě. Algoritmus je specifický zejména těmito charakteristikami:

- Algoritmus začíná s populací sestávající se z jedinců o minimální velikosti a topologii, tedy např. síť s jedním vstupem a jedním výstupem sestává právě z jednoho vstupního a jednoho výstupního neuronu.
- Každé strukturální mutaci (přidání synapse nebo neuronu) je přiřazeno unikátní číslo, tzv. innovation number. Na základě těchto čísel lze rozpoznat, kdy jaká mutace vznikla, a lze je díky tomu vzájemně přiřadit, čehož je využito při křížení.
- V průběhu algoritmu je populace rozdělena na několik subpopulací - druhů (species), které obsahují podobné jedince a členové každého druhu do určité míry sdílí hodnotu fitness funkce. To umožňuje, aby nové synaptické spoje, vzniklé při strukturálních mutacích, měly čas na adaptaci svých vah.

Kapitola 4

Realizace

V této části popíši, jaká je struktura programu, jakými fázemi prošel vývoj aplikace a jak jsem použil původní algoritmus. Návod k instalaci a ovládání aplikace je v příloze B.

Rozhodl jsem se postavit program na základě balíku NEAT C++ [8], který implementuje základní algoritmus NEAT, a tedy pracovat v jazyce C++.

Tato implementace kromě implementace algoritmu tak, jak je popsán v [9], obsahuje i tzv. *traits*, což jsou datové struktury přiřazované jednotlivým synapsím a neuronům. Přítomnost této struktury velice usnadnila práci při modifikaci neuronů, o čemž se zmíním níže (viz 4.4.3).

4.1 Způsob tvorby

Na počátku jsem vytvořil aplikaci, která porovnává signály s předem definovaným signálem a uděluje fitness na základě střední kvadratické odchylky. V této fázi měla aplikace jeden jediný prostor pro vykreslení průběhu.

Tuto aplikaci jsem předělal tak, že jsem exaktní ohodnocování nahradil uživatelským, tedy jsem implementoval interaktivní evoluci. V této fázi bylo ohodnocování takové, že uživatel pouze řadil individuály v takovém pořadí, že nejvýše byl nejvíce vyhovující výstup a nejníže nejméně vyhovující.

V dalším kroku jsem předělal systém ohodnocování na takový, jaký je ve finální verzi, tedy matice grafů s průběhy signálu a posuvníky pod každým z nich. Tento design umožňuje lepší porovnání signálů.

Další fáze vývoje obsahovala implementaci toho nejdůležitějšího - FIR synapsí a CPPN režimu (viz 4.4).

V posledním kroku jsem doprogramoval utilitní funkce, konkrétně zápis a čtení zvuků do/ze souborů, přehrávání zvuků přímo z programu a výpočet frekvenčních charakteristik signálů.

4.2 Vývojové prostředky

4.2.1 Vývojové prostředí

Program jsem vyvíjel v operačním systému Ubuntu 10.10 v jazyce C++. Proto jsem zvolil vývojové prostředí Code::Blocks, které je specializované na jazyk C++.

4.2.2 Knihovny

V programu bylo třeba využít několika externích knihoven:

GUI Pro vytvoření grafického uživatelského rozhraní jsem využil multiplatformní knihovny wxWidgets [11].

Zvuk Na přehrávání zvuků přímo z programu jsem využil knihovny SDL, konkrétně její podknihovny SDL_mixer [3].

Zápis a čtení signálů Pro ukládání zvuků do souborů a načítání signálů jsem použil knihovnu libsndfile [1].

Fourierova transformace Pro výpočet frekvenčních charakteristik signálů jsem použil knihovnu FFTW [2].

4.3 Struktura programu

Program je rozčleněn na čtyři základní bloky, jak je vidět na obr. 4.1.

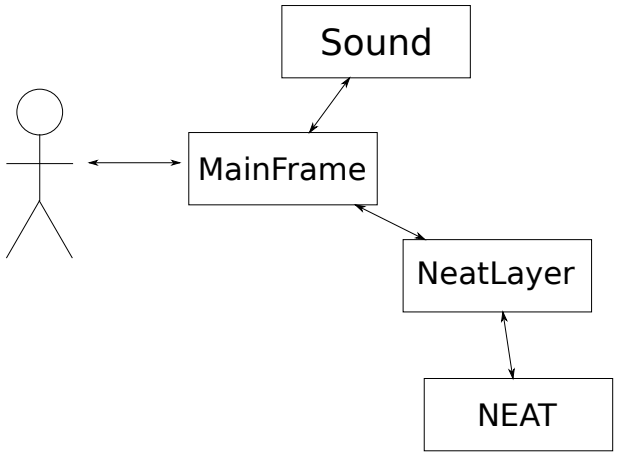
Základem je blok na obrázku označený jako `NEAT`. Ten se skládá z již zmíněné implementace NEAT C++ a jejímž obsahem jsou třídy a funkce realizující algoritmus NEAT, z hlediska jazyka je zabalena do namespace `NEAT`.

Další blok, na obrázku označený jako `Sound`, neobsahuje žádné třídy, ale jen funkce pro přehrávání zvuku a ukládání/načítání zvukových souborů. Tento blok poskytuje interakci s knihovnami SDL a libsndfile (viz 4.2.2).

Další blok, na obrázku označený jako `MainFrame`, se skládá především ze stejnojmenné třídy (a dalších pomocných tříd), která zajišťuje veškeré grafické uživatelské rozhraní. Tento blok je postavený na knihovně wxWidgets (viz 4.2.2). Třída `MainFrame` dědí od třídy `wxFrame` a jako taková představuje hlavní okno programu.

Posledním a nejdůležitějším blokem je blok na obrázku označený jako `NeatLayer`. Jedná se o třídu obsahující jen statické metody a atributy a jedná se o vrstvu mezi `MainFrame` a `NEAT`. Tato třída dohromady integruje data potřebná pro ovládání algoritmu a komunikaci s uživatelem.

Grafické uživatelské rozhraní komunikuje pouze s blokem Sound a třídou `NeatLayer`. Třída `NeatLayer` zase komunikuje pouze s balíkem `NEAT`. Díky tomu nebylo třeba výrazně upravovat strukturu balíku NEAT (pouze úpravy spojené s modifikacemi 4.4.2 a 4.4.3), ale



Obrázek 4.1: Struktura programu z hlediska funkčních bloků a jejich vzájemné interakce. Uživatel (vlevo) interaguje pouze se třídou (blokem) `MainFrame` - uživatelským rozhraním. Ta interaguje pouze se třídou (vrstvou) `NeatLayer` a funkčním blokem Sound. Třída `NeatLayer` interaguje pouze s blokem `NEAT`.

jen operovat na úrovni funkcí a metod, které balík poskytuje, což zapouzdřuje právě třída `NeatLayer`. Tato třída pak poskytuje metody volané při zpracování interakcí uživatele, takže grafické rozhraní je od samotné implementace algoritmu zcela odstíněno.

4.4 Modifikace algoritmu

Algoritmus bylo třeba modifikovat v několika oblastech, které popíši v následujících sekcích.

4.4.1 Úprava pro interaktivní evoluci

Modifikovat algoritmus tak, aby umožňoval interaktivní ohodnocování jedinců, bylo poměrně snadné: běh algoritmu je v momentě, kdy by mělo dojít k ohodnocení, přerušen a data jsou zobrazena uživateli. První koncept ohodnocování spočíval v tom, že uživatel si vygenerované signály jednotlivě zobrazoval a řadil podle toho, jak je s nimi spokojen. Toto se však ukázalo jako poměrně nepraktické rozhraní se zdlouhavým procesem ohodnocování. Proto jsem koncept změnil, a to tak, že všechny signály jsou uživateli zobrazeny naráz a uživatel každému z nich nastaví „procentuální“ hodnotu.

Tato modifikace se nedotkla struktury balíku NEAT, ale byla vyřešena na úrovni vrstvy NeatLayer.

4.4.2 Úprava synaptických spojů

Pro větší expresivitu neuronových sítí jsem se rozhodl implementovat sítě jako FIR sítě (viz 3.1.3). Tato úprava si již vyžádala zásah do původního kódu implementace algoritmu. Bylo třeba upravit kód synaptických spojů, mutačních a křížících operátorů, přidat parametry ovládající tyto operátory a upravit další funkce ovládající běh neuronové sítě.

FIR filtr na synapsi je realizován dvěma vektory: první z nich pracuje jako buffer ukládající potřebný počet předchozích vzorků, druhý z nich obsahuje jednotlivé koeficienty filtru. Výpočet výstupu synapse pak probíhá tak, jak je výstup filtru definován (viz 3.1), tedy průchodem obou vektorů současně, násobením odpovídajících hodnot a sčítáním těchto součinů.

4.4.2.1 Úprava původní implementace

Konkrétní úpravy balíku NEAT byly následující, rozdělené podle upravovaných tříd (souborů):

`Link` (link.h)

- Proměnnou `weight` typu `double` bylo třeba změnit na seznam typu `std::list<double>`. Tento seznam tvoří koeficienty b_i v 3.1.
- Bylo třeba přidat druhý seznam stejného typu nazvaný `buff`, sloužící jako buffer pro předcházející vzorky ($x(n-i)$ v 3.1).
- Poslední úprava se týkala metody `compute_output`. Tu bylo třeba změnit tak, aby provedla následující kroky:
  1. přidání výstupu neuronu, ze kterého synapse vychází, na začátek bufferu
  2. vymazání posledního vzorku v bufferu
  3. výpočet výstupu FIR filtru podle 3.1

Kromě těchto úprav bylo ještě nutné upravit funkce pro zápis a čtení genomů ze souboru, konstruktory a podobné procesní záležitosti.

`Genome` (genome.h)

- Upravil jsem metodu `mutate_link_weights` tak, aby mutaci podléhaly všechny koeficienty filtru.
- Přidal jsem metodu `mutate_link_weights_some`, která na základě předané pravděpodobnosti zmutuje některé z koeficientů filtru. Zároveň bylo nutné zavést další parametr (v souboru neat.h) odpovídající této pravděpodobnosti. Tento parametr se jmenuje `mutate_link_weights_some_prob`.
- Přidal jsem metodu `mutate_link_add_weight`, která přidává jeden koeficient filtru na konec seznamu koeficientů, tedy do filtru bude vstupovat další vzorek. Zároveň bylo nutné zavést další parametr (v souboru neat.h) odpovídající pravděpodobnosti této mutace. Tento parametr se jmenuje `mutate_link_add_weight_prob`.
- Upravil jsem metody `mate_multipoint_avg` a `mate_singlepoint`, které původně průměrují váhy synapsí, tak, aby pracovaly se seznamy. Využil jsem možnosti jazyka C++ a definoval jsem operátory sčítání, násobení a dělení i pro seznamy.

utils.h Přidal jsem soubor s deklaracemi a definicemi operátorů +, * a /, aby křížení, které průměruje váhy, pracovalo stejně i se seznamy koeficientů. Průměrování probíhá jako součet dvou seznamů koeficientů (po prvcích) a dělení dvěma (po prvcích). Pokud jsou seznamy jinak dlouhé, tak ten kratší z nich je doplněn nulami, protože pokud filtr nějaké koeficienty neobsahuje, je to totéž, jako kdyby byly nulové.

4.4.3 Úprava neuronů

Opět pro větší expresivitu jsem se rozhodl převést NEAT na CPPN-NEAT, tedy umožnit, aby neurony mohly disponovat i jinými aktivačními funkcemi, než je sigmoida. Funkce, které jsem přidal, jsou následující:

$$f_1(x) = \begin{cases} a & x > c \\ b & x \leq c \end{cases} \tag{4.1}$$

$$f_2(x) = a \sin(b(x-c)) \tag{4.2}$$

$$f_3(x) = a(x-b)^2 + c \tag{4.3}$$

$$f_4(x) = a \cdot e^{\frac{(x-b)^2}{2c^2}} \tag{4.4}$$

$$f_5(x) = a(x-b) + c \tag{4.5}$$

kde a, b a c jsou parametry napevno nastavené v kódu programu.

Základní implementace byla již částečně připravena na tuto transformaci. Bylo nutno jednak implementovat samotné aktivační funkce, což bylo nejméně obtížné. Dále bylo třeba vyřešit mutační operátory na změnu aktivační funkce neuronu a operátory křížení, aby byly schopny s tímto počítat.

Tuto modifikaci jsem vyřešil využitím již dříve zmíněných *traits*. Každá tato struktura má své číslo, pro každého jedince je samostatná sada těchto struktur a každý gen (synapse nebo neuron) má přiřazenu jednu z těchto sturktur. Toho jsem využil tak, že číslo přiřazené struktury u neuronu je zároveň číslem použité aktivační funkce pro tento neuron. Výhodou je, že mutace typu „změň číslo přiřazené struktury“ byla již v základním algoritmu implementována, a proto stačilo pouze dopsat výběr aktivační funkce na základě čísla struktury.

4.4.3.1 Úprava původní implementace

Konkrétní úpravy balíku NEAT byly následující, rozdělené podle upravovaných tříd (souborů):

`NNode` **(nnode.h)**

- Do výčtového typu `functype` jsem přidal další hodnoty, konkrétně `STEP`, `SINE`, `PARABOLIC`, `GAUSS` a `LINEAR`, které identifikují použité aktivační funkce.
- Do metody `derive_trait` jsem přidal řádek, který zajistí správnou hodnotu proměnné `ftype` podle zadané trait.

neat.h Do tohoto souboru jsem dodefinoval nové aktivační funkce.

`Network` **(network.cpp)** V tomto souboru jsem pouze upravil metodu `activate`, aby při výpočtu hodnoty výstupu neuronu použila správnou aktivační funkci.

`Genome` **(genome.cpp)** Zde jsem pouze upravil metodu `mutate_node_trait` tak, aby při změně trait přiřazené neuronu se zároveň změnil i identifikátor použité funkce, aby souhlasil s novou trait.

Kapitola 5

Experimenty

V experimentech jsem se zaměřil převážně na schopnost algoritmu vytvářet „nové“ sturktury či prvky v signálu, na základě jednoduchého vstupu.

Poznámka k experimentům a jejich výsledkům Výsledky, které jsou zde uvedené a zobrazené, jsou výsledky vždy jednoho ukázkového běhu algoritmu z více běhů. Vzhledem k faktu, že se jedná o interaktivní evoluci, jsou generace hodně rozmanité a často obtížně ohodnotitelné, takže je třeba výsledky brát s ohledem na tento fakt.

Data experimentů Všechna data (vstupní a výstupní signály, konečné populace, parametry algoritmu) jsou i na přiloženém CD ve složce `experiments` (viz příloha A).

5.1 Schopnost vytvořit komplexní signál

V tomto experimentu jsem zkoušel, zda-li je algoritmus schopen, na základě jednoduchého vstupního signálu – jednotkového impulsu (5.1), vytvořit signál, který je netriviální. Netriviálním signálem mám na mysli takový signál, který po celou dobu trvání signálu mění svoji amplitudu, přičemž „nezapočítávám“ počáteční přechodový děj po přiložení vstupního signálu na síť.

$$x(n) = \begin{cases} 1 & \text{pokud } n = 0 \\ 0 & \text{pokud } n \neq 0 \end{cases} \tag{5.1}$$

Tento experiment je rozdělen na několik podexperimentů, které se liší v nastavení algoritmu, především povolení či zakázání FIR filtrů na synapsích nebo aktivačních funkcí na neuronech. Každý z těchto podexperimentů ještě obsahuje dvě části – první jako dopředná síť (se zakázanými rekurentními spoji), druhá s povolenými rekurentními spoji.

Ohodnocování, které je prováděno ručně, jsem se snažil nastavovat tak, abych docílil zajímavého signálu - zvýhodňuji nové struktury v signálu, penalizuji nezajímavé (příliš jednoduché) nebo ty, které jsou v populaci již delší dobu.

Parametry algoritmu uvádím v tabulce 5.1, pričemž některé hodnoty se liší v závislosti nastavení experimentu. Jedná-li se o experiment s dopřednou sítí, jsou parametry recur_prob a recur_only_prob nastaveny na 0,0, jinak jsou takové, jak jsou uvedeny v tabulce. Další změny uvedu vždy u konkrétního typu experimentu.

parametr	**hodnota**
trait_param_mut_prob	0,0
trait_mutation_power	1,0
linktrait_mut_sig	1,0
nodetrait_mut_sig	0,5
weight_mut_power	1,1
recur_prob	0,2
disjoint_coeff	1,1
excess_coeff	1,3
mutdiff_coeff	2,0
compat_thresh	3,5
age_significance	0,8
survival_thresh	0,4
mutate_only_prob	0,2
mutate_random_trait_prob	0,0
mutate_link_trait_prob	0,0
mutate_node_trait_prob	0,1
mutate_link_weights_prob	0,05
mutate_link_weights_some_prob	0,08
mutate_link_add_weight_prob	0,15
mutate_toggle_enable_prob	0,15
mutate_gene_reenable_prob	0,08
mutate_add_node_prob	0,05
mutate_add_link_prob	0,1
interspecies_mate_rate	0,05
mate_multipoint_prob	0,5
mate_multipoint_avg_prob	0,4
mate_singlepoint_prob	0,2
mate_only_prob	0,2
recur_only_prob	0,2
pop_size	14
dropoff_age	25
newlink_tries	20

Tabulka 5.1: V tabulce jsou uvedeny parametry algoritmu během experimentu. Parametry obsahující `prob` jsou pravděpodobnosti. Pro význam jednotlivých parametrů odkazuji na zdrojový kód balíku NEAT C++ (viz příloha A).

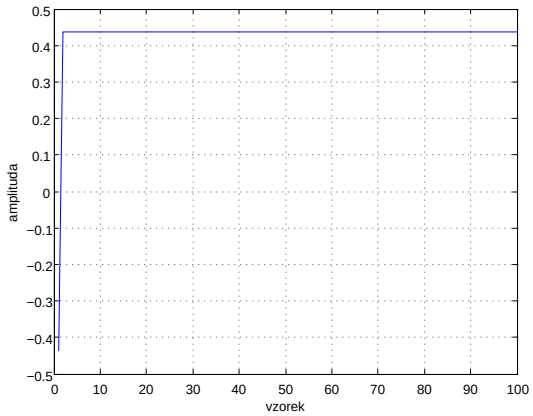
5.1.1 Základní neuronová síť

V tomto nastavení jsem zakázal jak FIR filtry (resp. zakázal jsem mutaci přidání koeficientu na konec filtru), tak aktivační funkce kromě sigmoidy (resp. zakázal jsem mutaci změny aktivační funkce). To znamená, že jsem nastavil parametry `mutate_link_add_weight_prob` (pravděpodobnost přidání koeficientu na konec filtru) a `mutate_node_trait_prob` (pravděpodobnost změny čísla trait u neuronu a tím i změny aktivační funkce) na 0.

5.1.1.1 Dopředná síť

Při zakázání rekurentních spojů by nemělo být možné vytvořit stálý nenulový (nebo nekonstantní) signál, protože bez rekurentních spojů neexistuje způsob, jakým by se v síti signál nebo jeho změny udržely.

Tato úvaha se ukázala být správná, jelikož ani po 55 generacích se signál prakticky nezměnil od výchozího – dojde k jednomu skoku na počátku, odpovídající impulsu vstupního signálu, a dále je výstupní signál konstantní, jak je vidět na obr. 5.1.



Obrázek 5.1: Výstupní signál po přivedení jednotkového impulsu na síť vyvinutou v 55. generaci, se zakázanými FIR synapsemi, ostatními aktivačními funkcemi a rekurentními synapsemi. Jedná se o soubor `generation55_sound0.wav`.

5.1.1.2 Rekurentní síť

U rekurentní sítě by naopak mělo být možné udržet signál v síti a tím vytvořit stálý kmitavý (nebo i jiný) průběh. To se skutečně podařilo a ve 45. generaci se objevil první periodický signál. V 55. generaci jsem došel k signálům vyobrazeným v obr. 5.2.

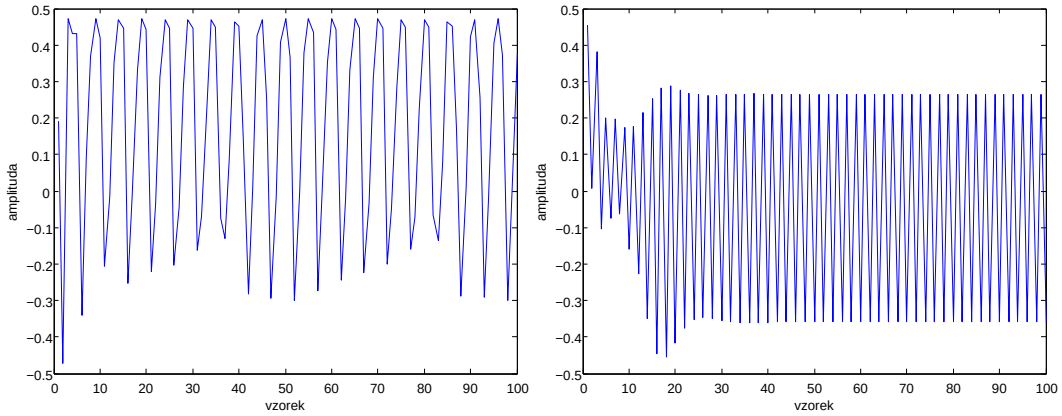
5.1.2 FIR síť

Oproti předchozímu, v tomto nastavení jsem povolil FIR filtry na synapsích – nastavil jsem parametr `mutate_link_add_weight_prob` na hodnotu jako je v tab. 5.1. Aktivační funkce byla možná pouze sigmoida.

5.1.2.1 Dopředná síť

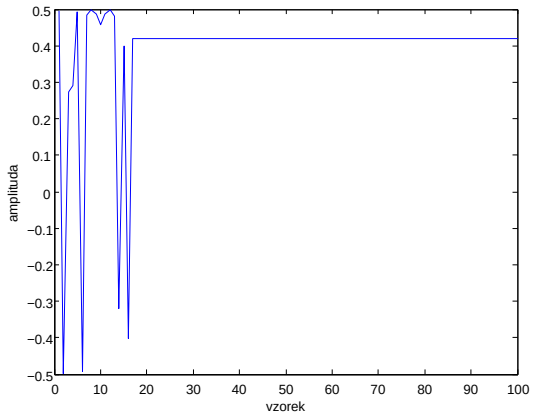
Podobně jako v předchozím případě dopředné sítě, i zde by nemělo dojít k vytvoření stále nekonstantního signálu. Přítomnost FIR filtrů ale může zajistit výraznější zpožďování vzorků a nekonstantnost emulovat na počátku signálu.

Výsledek tohoto experimentu je vidět na obr. 5.3. Z obrázku je patrné, že dochází k jakémusi zakmitání na počátku signálu, ale jelikož je délka FIR filtrů konečná, dále se výstup chová jako v 5.1.1.1.



(a) `generation55_sound0.wav` (b) `generation55_sound9.wav`

Obrázek 5.2: Výstupní signály po přivedení jednotkového impulsu na síť vyvinutou v 55. generaci, se zakázanými FIR synapsemi, ostatními aktivačními funkcemi a s povolenými rekurentními synapsemi.



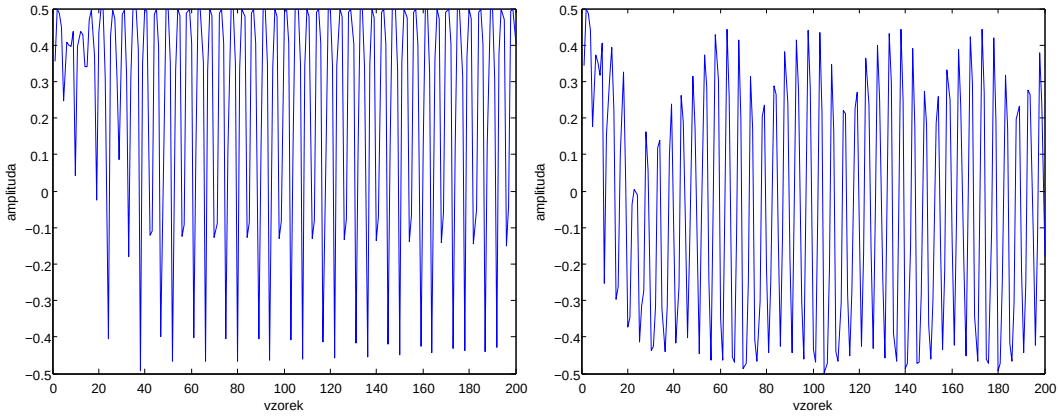
Obrázek 5.3: Výstupní signál po přivedení jednotkového impulsu na síť vyvinutou v 55. generaci, s povolenými FIR synapsemi, zakázanými ostatními aktivačními funkcemi a rekurentními synapsemi. Jedná se o soubor `generation55_sound0.wav`.

5.1.2.2 Rekurentní síť

Jelikož se jedná o rekurentní síť, neměl by být problém vyvinout síť generující stále nekonstatní signál.

Již v 17. generaci se objevil první nekonstantní signál a signály na konci 55. generace jsou na obr. 5.4.

Je vidět, že přítomnost FIR filtrů výrazně napomáhá vytvoření netriviálního signálu, který se objeví mnohem dříve (v 17. generaci) a může být mnohem složitější, než v případě bez FIR filtrů.



(a) `generation55_sound0.wav` (b) `generation55_sound1.wav`

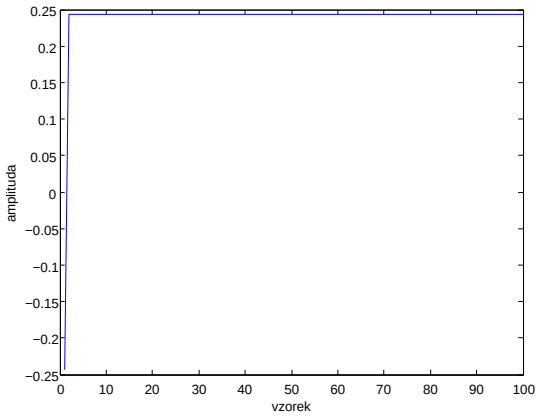
Obrázek 5.4: Výstupní signály po přivedení jednotkového impulsu na síť vyvinutou v 55. generaci, se zakázanými FIR synapsemi, ostatními aktivačními funkcemi a s povolenými rekurentními synapsemi.

5.1.3 CPPN

V tomto nastavení jsem naopak povolil pouze různé aktivační funkce – nastavil jsem parametr `mutate_node_trait_prob` na hodnotu jako je v tab. 5.1 – a zakázal FIR filtry na synapsích.

5.1.3.1 Dopředná síť

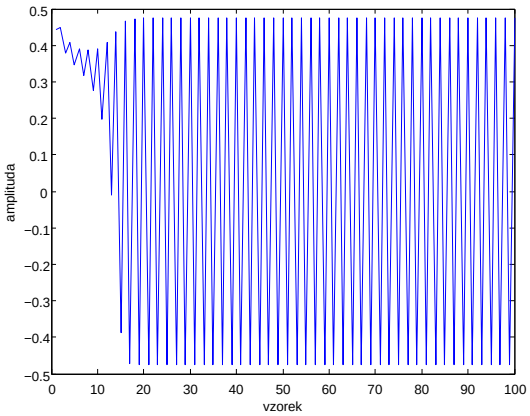
Výsledek tohoto experimentu je vidět v obr. 5.5. Vzhledem k nepřítomnosti rekurentních synapsí a FIR filtrů se výstup chová podobně jako v 5.1.1.1.



Obrázek 5.5: Výstupní signál po přivedení jednotkového impulsu na síť vyvinutou v 55. generaci, zakázanými FIR synapsemi a rekurentními synapsemi a s povolenými ostatními aktivačními funkcemi. Jedná se o soubor `generation55_sound0.wav`.

5.1.3.2 Rekurentní síť

První nekonstantní signál se objevil ve 43. generaci a signál na konci 55. generace je na obr. 5.6.



Obrázek 5.6: Výstupní signál po přivedení jednotkového impulsu na síť vyvinutou v 55. generaci se zakázanými FIR synapsemi, povolenými ostatními aktivačními funkcemi a s povolenými rekurentními synapsemi. Jedná se o soubor `generation55_sound0.wav`.

Je vidět, že přítomnost FIR filtrů napomáhá vytvoření netriviálního signálu více, než přítomnost dalších aktivačních funkcí.

5.1.4 FIR CPPN

V tomto nastavení jsem povolil jak FIR filtry, tak různé aktivační funkce, tedy nastavení takové, jako je v tab. 5.1.

5.1.4.1 Dopředná síť

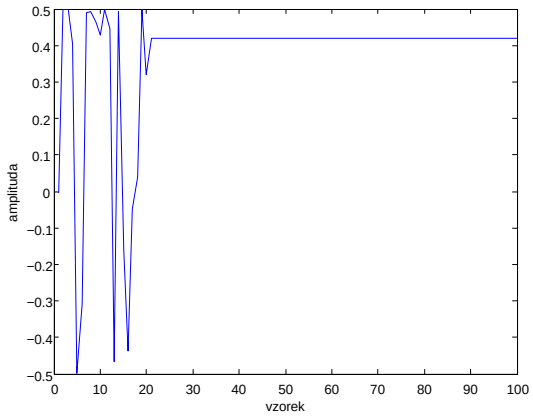
Výsledný výstupní signál po 55. generaci je vidět na obr. 5.7.

Podobně jako v 5.1.2.1, tak i zde dojde pouze k několika zákmitům na počátku signálu a dále je průběh konstantní.

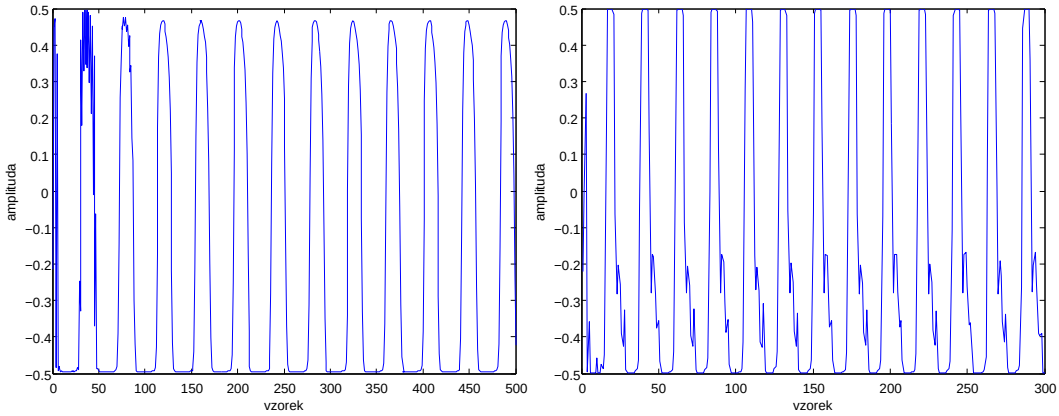
5.1.5 Rekurentní síť

Při povolení všech prostředků, tedy rekurence, FIR synapsí a více aktivačních funkcí pro neurony, by měla být k dispozici největší expresivita.

To se v zásadě potvrdilo už tím, že k prvním netriviálním průběhům došlo už v 16. generaci. Po 55. generaci bylo dosaženo signálů, jejichž průběhy jsou na obr. 5.8.



Obrázek 5.7: Výstupní signál po přivedení jednotkového impulsu na síť vyvinutou v 55. generaci s povolenými FIR synapsemi a ostatními aktivačními funkcemi a zakázanými rekurentními synapsemi. Jedná se o soubor `generation55_sound0.wav`.



(a) `generation55_sound0.wav` (b) `generation55_sound5.wav`

Obrázek 5.8: Výstupní signály po přivedení jednotkového impulsu na síť vyvinutou v 55. generaci s povolenými rekurentními a FIR synapsemi a ostatními aktivačními funkcemi.

5.1.6 Shrnutí experimentu

V experimentu jsem ukázal schopnosti algoritmu a jeho různých nastavení. Je patrné, že zásadní vliv na schopnost vytvářet netriviální struktury mají rekurentní synapse, které přímo umožňují, aby takové struktury vznikly. Dále jsem zjistil, že mnohem větší vliv na expresivitu má přítomnost FIR filtrů nežli nové aktivační funkce.

Z výsledků je tedy vidět, že největší expresivitu poskytuje poslední z uvedených nastavení, tedy povolení FIR filtrů na synapsích, rekurence a více aktivačních funkcí na neuronech.

5.2 Filtrace

V tomto experimentu se pokusím aplikovat můj program na výrobu filtru, nebo spíše zkreslovače, namísto generátoru. Nejdříve zkusím program aplikovat na jednoduchý signál – harmonickou funkci, dále na náhodný signál (šum) a na závěr se pokusím filtrovat reálné signály – tón akustické kytary a lidskou řeč.

V přechozím experimentu jsem zjistil, že největší expresivitu poskytuje nastavení v 5.1.5, tedy povolení FIR synapsí, CPPN režimu (více aktivačních funkcí) a rekurence, a proto v tomto experimentu budu používat toto nastavení. Parametry tohoto nastavení jsou shodné s tab. 5.1.

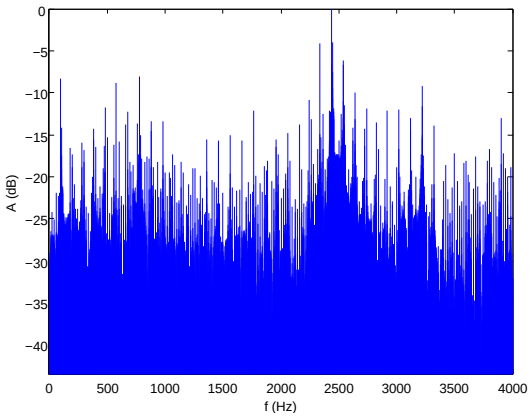
5.2.1 Harmonický signál

Vstupem je jednoduchá harmonická funkce

$$y(n) = sin\left(2\pi f \frac{n}{f_s}\right) \tag{5.2}$$

kde n je číslo vzorku, frekvence $f = 100$ Hz a vzorkovací frekcence $f_s = 8$ kHz.

Výstupní signál po 18. generaci je na obr. 5.9 ve frekvenční oblasti. Je evidentní, že přibyly nové frekvence, které v původním prostě harmonickém signálu nebyly.



Obrázek 5.9: Výstupní signál ve frekvenční oblasti po 18. generaci při harmonickém vstupním signálu o frekvenci 100 Hz. Jedná se o soubor `generation18_sound3.wav`.

5.2.2 Náhodný signál

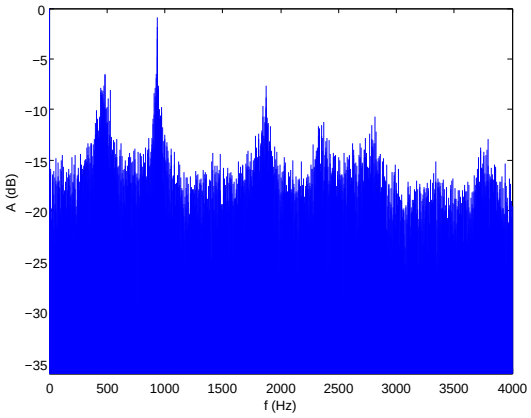
Vstupem je náhodný signál

$$\forall n: \quad y(n) \in (0, 1) \tag{5.3}$$

kde n je číslo vzorku.

Průběh experimentu pro náhodný signál byl poměrně obtížný, protože je třeba vyslechnout každý signál (a zároveň sledovat jejich spektra), což si jednak vyžádalo velké množství

času, a také poslouchání de facto šumu je poněkud nepříjemné. Také nové struktury se neobjevovaly tak rychle jako v ostatních experimentech, a proto zde uvádím výsledek až po 65. generaci, který je na obr. 5.10.



Obrázek 5.10: Výstupní signál ve frekvenční oblasti po 65. generaci při náhodném vstupním signálu. Jedná se o soubor `generation65_sound0.wav`.

Za povšimnutí stojí peaky na 500, 1000 a 2000 Hz, které jsou od sebe vzdáleny vždy o dvojnásobek, tedy o oktávu. I další vyvýšeniny ve spektru jsou na násobcích 500 Hz, tedy se jedná o signál s výraznou periodickou složkou.

5.2.3 Tón kytary

V tomto případě jsem jako vstup načetl reálný signál, jehož obsahem je postupné drnknutí strun E A d g h e_1 a pak drnknutí na všechny struny najednou. Snažil jsem se ohodnocovat výstupy tak, abych dosáhl zajímavého zkreslení.

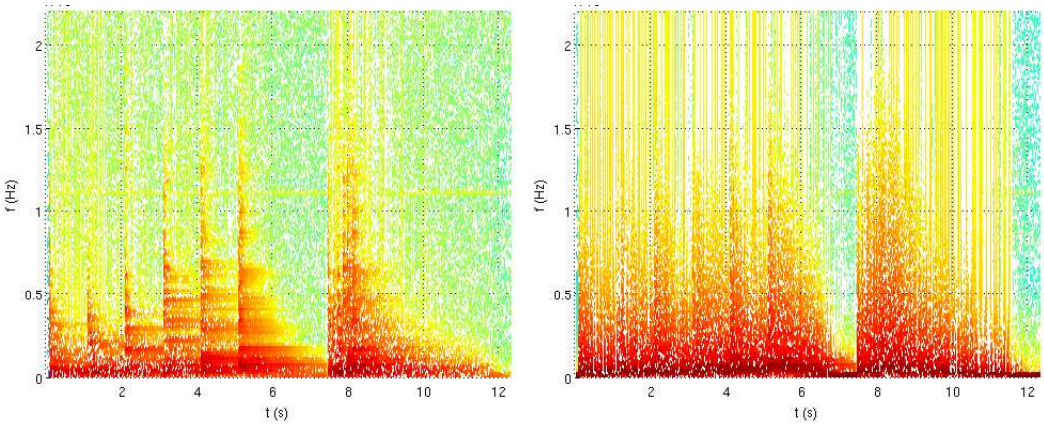
Po 23 generacích jsem dosáhl zkreslení, které se vzdáleně podobá zvuku zkreslené elektrické kytary. Spektrogram vstupního a výstupního signálu je na obr. 5.11. Na první pohled je vidět rovnoměrnější zastoupení vyšších frekvencí a zaplnění „mezer“ mezi spektrálními čarami přítomnými ve vstupním signálu v intervalu 0-750 Hz.

5.2.4 Lidský hlas

V tomto posledním experimentu jsem svému programu podrobil nahrávku svého hlasu, konkrétně promluvy „raz dva tři“. Podobně jako v předchozím případě, i v tomto jsem se snažil vyzískat zajímavé zkreslení.

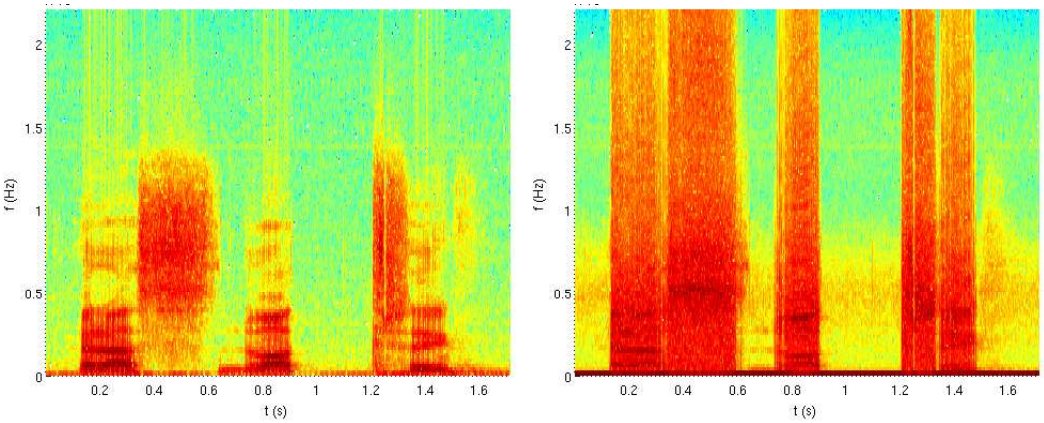
Po 26 generacích jsem vyvinul síť, která produkuje zvuk zkreslený podobně, jako ve vysílačce. Spektrogramy vstupu a výstupu jsou zobrazeny v obr. 5.12.

Na první pohled patrná změna je nárůst intenzity prakticky všech frekvencí, nicméně nedochází k nahrazení řeči šumem, ale její zkreslení do „zašuměné“ podoby, která vyvolává již dříve zmmíněný dojem hovoru přes vysílačku.



(a) Vstupní signál (b) Výstupní signál

Obrázek 5.11: Spektrogram vstupního signálu a výstupního signálu zvuku kytary po 23 generacích. Spektrogram byl vypočítán s délkou okna 256 a překryvem 128 vzorků. Jedná se o soubor `generation23_sound1.wav`.



(a) Vstupní signál (b) Výstupní signál

Obrázek 5.12: Spektrogram vstupního signálu a výstupního signálu lidského hlasu – promluvy „raz dva tři“ – po 26 generacích. Spektrogram byl vypočítán s délkou okna 256 a překryvem 128 vzorků. Jedná se o soubor `generation26_sound7.wav`.

5.2.5 Shrnutí experimentu

V experimentu jsem ukázal schopnosti algoritmu s FIR synapsemi a rekurencí v CPPN režimu při filtraci nebo manipulaci se vstupním signálem. Je patrné, že síť je schopna zásadně změnit frekvenční charakteristiku signálu, a to jak filtračním charakterem – frekvence jsou „odebírány“, tak zkreslujícím charakterem – frekvence jsou „přidávány“.

Vyvinout složitější a specifičtější zkreslení je velice náročné a vzhledem k interaktivitě evoluce, která implikuje poměrně malou velikost populace a tím i lokalitu prohledávání, je obtížné vyvinout zajímavý výstup, a to tím obtížnější, čím složitější je vstupní signál.

Kapitola 6

Závěr

Cílem mé práce bylo vytvořit program, vycházející z implementace algoritmu NEAT, umožňující interaktivní evoluci jednotek pro zpracování signálů, zejména zvukových.

Ukázal jsem, jak je možné využít genetických algoritmů a neuronových sítí v oblasti zpracování signálů, a to konkrétně pomocí algoritmu NEAT. Tento algoritmus jsem prostudoval a analyzoval jeho vlastnosti.

Vybral jsem implementaci algoritmu NEAT – NEAT C++ a podrobně ji prostudoval. Na této implementaci jsem postavil jádro svého programu.

Vytvořil jsem program pro interaktivní evoluci neuronových sítí algoritmem NEAT. Tento program umožňuje uživateli jednoduše vyvíjet neuronové sítě zpracovávající zvukový (ale obecně jakýkoliv reálný) signál bez jakékoliv znalosti problematiky zpracování signálů. Uživatel má k dispozici pohled na výstupní signály v časové i frekvenční oblasti a jednoduchý způsob vyjádření své spokojenosti s výstupy.

Provedl jsem úpravy originální implementace algoritmu, které umožnily pracovat s FIR sítěmi v CPPN režimu namísto klasických neuronových sítí, a tím docílit větší expresivity neuronových sítí vyvinutých v mém programu.

Provedl jsem sérii experimentů, ve kterých jsem zjišťoval schopnosti modifikovaného algoritmu NEAT. Zjistil jsem, že v úloze generování signálů na základě jednotkového impulsu je nejjednodušší cestou k vytvoření nekonstantního výstupního signálu FIR síť s rekurentními spoji v CPPN režimu, což byl výsledek, který jsem očekával. V úloze filtrace či zkreslování signálů jsem zjistil, že sítě jsou schopny jak filtrace – odebírání frekvencí, tak zkreslování – přidávání nových frekvenčních složek. Při aplikaci na reálné signály – zvuk kytary a lidský hlas – se mi dokonce podařilo vyvinout zkreslení, které signál svým způsobem obohacuje.

6.1 Budoucí práce

Mnou vytvořený program rozhodně není dokonalý, a proto zde uvedu několik oblastí, které dávají prostor na další možná vylepšení programu samotného nebo na výzkum algoritmů, které stojí v pozadí.

6.1.1 Asistence při ohodnocování

Jak jsem již několikrát zmínil, nevýhodou interaktivní evoluce je sama interaktivita – populace musí být malá, aby byl člověk schopen rozumně ohodnocovat, a na počátku procesu je pro člověka obtížné rozlišit, kterým „směrem“ se v evoluci vydat, zvláště v úloze zkreslování. Při zkreslování šumu a reálných signálů docházelo k jevu, že velká část populace produkovala signály, které byly pro lidské ucho velice nepříjemné.

Bylo by zajímavé a hodné prozkoumání spojit interaktivní evoluci s předem definovanými pravidly. Tato pravidla by mohla člověku pomoci např. tím, že rovnou zachytí signály, které mají např. příliš vysoké frekvence nebo nemají frekvenční čáry správně vzdálené. Tato pravidla by jednak pomohla tím, že by člověka nezatěžovala „špatnými“ signály a také by umožnila práci s větší populací a tím globálnější prohledávání.

6.1.2 Rozhraní a režimy programu

Uživatelské rozhraní programu rozhodně není ideální a nabízí se v něm spousta prostoru k ladění a vylepšování, ať už z hlediska designu nebo funkcionality.

Další možností rozšíření je přidání druhého režimu, který by nepracoval s interaktivní evolucí, ale s ohodnocováním na bázi porovnávání s předem definovaným signálem, případně pravidly podobně, jak jsem uvedl již v přechozím bodě.

6.1.3 Flexible activaton function

FAF (Flexible Activation Function) [10] je technika, kde aktivační funkce u neuronu není definována napevno, ale je definována pomocí sady kontrolních bodů a samotný průběh funkce je vypočítán z těchto bodů pomocí interpolačních nebo aproximačních algoritmů. Bylo by jistě zajímavé implementovat tuto techniku do algoritmu NEAT a umožnit tak vývoj i samotných aktivačních funkcí na jednotlivých neuronech.

Literatura

[1] CASTRO LOPO, E. libsndfile, 2011. Dostupné z: <http://www.mega-nerd.com/libsndfile/>.

[2] FRIGO, M. – JOHNSON, S. G. The Design and Implementation of FFTW3. *Proceedings of the IEEE*. 2005, 93, 2, s. 216–231. Special issue on "Program Generation, Optimization, and Platform Adaptation".

[3] LANTINGA, S. – PETER, S. – GORDON, R. SDL_mixer 1.2, 2011. Dostupné z: <http://www.libsdl.org/projects/SDL_mixer/>.

[4] RUSSELL, S. – NORVIG, P. *Artificial Intelligence: A Modern Approach*. Upper Saddle River, New Jersey : Prentice Hall, second edition, 2003. ISBN 0137903952.

[5] SECRETAN, J. et al. Picbreeder: Collaborative Interactive Evolution of Images. *Leonardo (Transactions Section)*. 2008, 41, 1, s. 98–99.

[6] SMITH, J. O. *Introduction to Digital Filters with Audio Applications*. http://www.w3k.org/books/ : W3K Publishing, 2007. ISBN 978-0-9745607-1-7.

[7] STANLEY, K. O. Compositional pattern producing networks: A novel abstraction of development. *Genetic Programming and Evolvable Machines*. 2007, 8, s. 131–162. ISSN 1389-2576. Dostupné z: <http://dx.doi.org/10.1007/s10710-007-9028-8>. 10.1007/s10710-007-9028-8.

[8] STANLEY, K. O. – KARPOV, I. V. NEAT C++, 2010. Dostupné z: <http://nn.cs.utexas.edu/?neat-c>.

[9] STANLEY, K. O. – MIIKKULAINEN, R. Evolving Neural Networks through Augmenting Topologies. *The MIT Press Journals*. 2002, 10, 2, s. 99–127.

[10] UNCINI, A. Audio signal processing by neural networks. *Neurocomputing*. 2003, 55, 3-4, s. 593 – 625. ISSN 0925-2312. doi: DOI:10.1016/S0925-2312(03)00395-3. Dostupné z: <http://www.sciencedirect.com/science/article/B6V10-48PDGG0-2/2/42d5a6b5a73925f003997582ad0c4ae3>. Evolving Solution with Neural Networks.

[11] Vývojáři wxWidgets. wxWidgets cross-platform GUI library, 2011. Dostupné z: <http://www.wxwidgets.org/>.

Příloha A

Obsah CD

```
.
|-- experiments/                  složka obsahující data k experimentům (5)
|   |-- exp1/                     data k prvnímu experimentu (5.1)
|   |   |-- CPPNnoFIR/            viz 5.1.3
|   |   |-- FIRCPPN/              viz 5.1.4
|   |   |-- FIRnoCPPN/            viz 5.1.2
|   |   |-- noFIRnoCPPN/          viz 5.1.1
|   |   `-- inputSignal.txt       vstupní signál pro experiment
|   `-- exp2/                     data k druhému experimentu (5.2)
|       |-- guitar/               viz 5.2.3
|       |-- harmonic/             viz 5.2.1
|       |-- noise/                viz 5.2.2
|       |-- voice/                viz 5.2.4
|       `-- exp2.ne               soubor s parametry algoritmu
|-- igfe/                         složka se zdrojovými soubory a aplikací samotnou
|   |-- bin/                      složka se spustitelným souborem pro debugging
|   |-- src/                      složka se zdrojovými soubory
|   |   |-- neat/                 složka se zdrojovými soubory balíku NEAT C++
|   |   `-- ...                   zdrojové soubory
|   |-- igfe                      spustitelný soubor aplikace
|   |-- Makefile                  soubor k progamu make
|   `-- params.ne                 ukázkový soubor s parametry algoritmu
`-- thesis/                       složka se zdrojovými soubory BP
    |-- figures/                  obrázky
    |-- chapters/                 kapitoly a přílohy
    |-- csplainnat.bst
    |-- k336_thesis_macros.sty
    |-- Makefile
    |-- reference.bib             seznam literatury
    |-- Zegklitz-thesis-2011.pdf  text BP
    `-- Zegklitz-thesis-2011.tex  zdrojový soubor BP
```

Příloha B

Instalační a uživatelská příručka

V této příloze poskytnu instrukce k instalaci aplikace a jejímu ovládání.

Poznámka Tato příručka je zaměřena na operační systém, ve kterém jsem aplikaci vyvíjel a pro který je aplikace zkompilována, tedy Ubuntu 10.10 64b, a všechny balíky zde uvedené by měly být k dispozici v základních repozitářích. Nicméně si myslím, že by neměl být problém aplikovat tuto příručku a rozběhnout aplikaci i na jiných linuxových operačních systémech a po notných úpravách i na MS Windows, pro něž potřebné balíky také existují.

B.1 Instalace

B.1.1 Kompilace

Pro kompilaci stačí ve složce `igfe` (viz příloha A) použít příkaz

```
$ make all
```

který zajistí zkompilování a slinkování všech zdrojových kódů. Po zkompilování aplikace, byla-li kompilace úspěšná, bude ve stejné složce spustitelný soubor `igfe`, což je sama aplikace. Navíc ve složce `bin/Debug/` bude spustitelný soubor s týmž jménem, ale zkompilovaný s debugovacími symboly, takže je možné jej pustit v debuggeru.

B.1.1.1 Požadavky

Aby byla kompilace úspěšná, je třeba mít nainstalovány tyto balíky:

- libwxgtk2.8-dev
- fftw-dev
- libsndfile1-dev
- libsdl1.2-dev

- libsdl-mixer1.2-dev

Dále je třeba mít nainstalován kompilátor `g++`, což na debianovském operačním systému zajistí nainstalování balíku build-essential.

B.1.2 Spuštění

Aplikace se spustí spuštěním spustitelného souboru `igfe`. Žádné další soubory samotnou aplikaci netvoří.

B.1.3 Požadavky

Aby bylo možné apliakci spustit, je třeba mít nainstalovány tyto balíky:

- libwxgtk2.8-0
- libsdl-mixer1.2
- fftw2

B.2 Ovládání programu

Progam se spouští spuštěním souboru `igfe`. Po spuštění se zobrazí hlavní okno programu (obr. B.1). Hlavní okno se skládá ze dvou částí. Vlevo se nachází část pro ovládání programu a vpravo (po roztáhnutí okna) je část pro zobrazování signálů.

V následujících částech popíši jednotlivé karty v levé části okna a nakonec pravou část okna.

B.2.1 General

Tato karta slouží k ovládání hlavního běhu evolučního algoritmu a ovládání zobrazení. Následuje popis jednotlivých tlačítek:

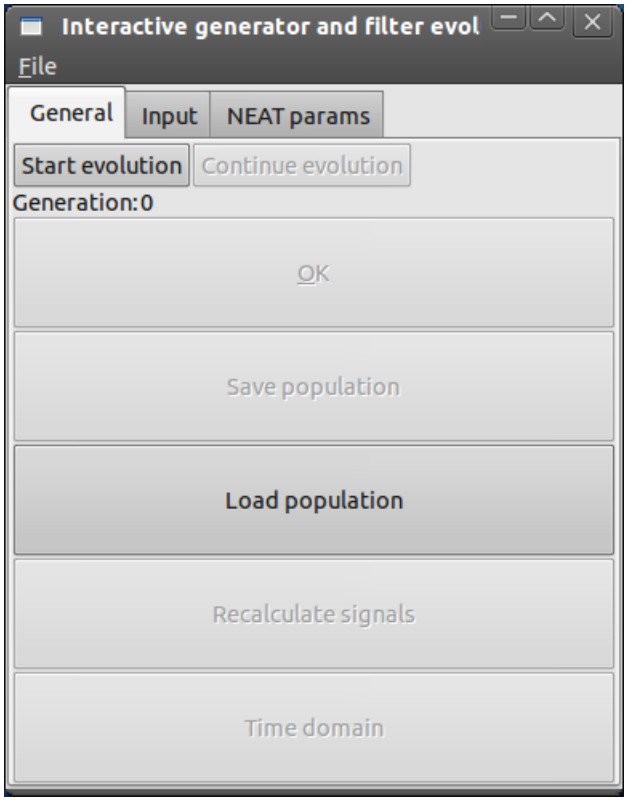
Start evolution Stisk tohoto tlačítka vygeneruje počáteční populaci a zahájí interaktivní evoluci.

Continue evolution Stisk tohoto tlačítka zahájí další generaci evoluce po tom, co již byla udělena fitness.

OK Stiskem tohoto tlačítka je akceptováno rozdělení fitness a je dokončena generace.

Save population Tímto tlačítkem je možné uložit aktuální populaci do souboru.

Load population Tímto je možné načíst dříve uloženou populaci ze souboru.



Obrázek B.1: Hlavní okno programu po spuštění.

Recalculate signals Stiskem tohoto tlačítka dojde k přepočítání všech signálů podle aktuálního vstupního signálu. Přepočítání je nutné vyvolat tímto tlačítkem – při změně vstupního signálu k němu nedochází automaticky.

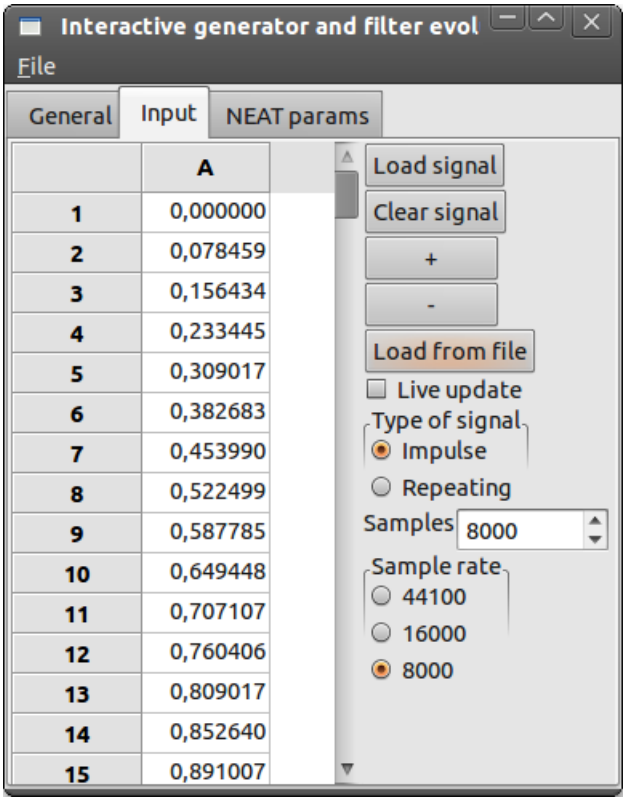
Time domain/Frequency domain Toto tlačítko jednak indikuje, ve které doméně jsou signály zobrazeny (časová, frekvenční), a zároveň jeho stiskem lze tyto domény přepínat. Frekvenční doména zobrazuje fourierovský obraz celého signálu jako celku. Je-li signál dlouhý (řádově stovky tisíc vzorků a více), trvá výpočet frekvenční charakteristiky velice dlouho.

B.2.2 Input

Tato karta (viz obr. B.2) obsahuje prvky k nastavení vstupního signálu a parametrů k interpretaci signálu.

V levé části karty se nachází jednosloupcová tabulka, jejíž jednotlivé buňky reprezentují vzorky signálu. Do buněk je možné vkládat reálná čísla. Buňky lze na konec tabulky přidat stiskem tlačítka `+` a odebrat z konce tabulky stiskem tlačítka `-`. Celý signál lze načíst ze souboru stiskem tlačítka `Load from file`. Načtení ze souboru nastaví, kromě samotných hodnot v tabulce, i vzorkovací frekvenci a délku signálu (viz níže).

Skupina `Type of signal` určuje, zda-li zadaný vstupní signál bude na vstupu sítě pouze jednou a po jeho konci budou na síť přiváděny nuly (volba `Impulse`), nebo se signál bude



Obrázek B.2: Nastavení vstupního signálu.

opakovat (volba `Repeating`).

Pole `Samples` určuje, kolik vzorků bude síť generovat. Je-li toto číslo menší, než je délka signálu, pak volby popsané v předchozím odstavci nemají vliv na výstup. Pokud je toto číslo delší, tak je se signálem nakládáno podle volby popsané v předchozím odstavci.

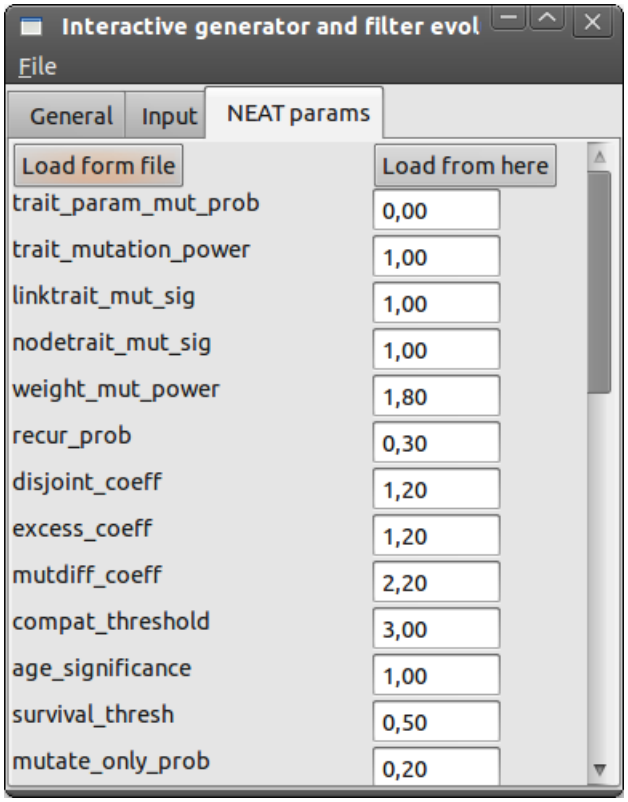
Skupina `Sample rate` určuje, jaká vzorkovací frekvence bude zvolena při přehrávání vygenerovaných signálu, neboli jak budou signály interpretovány (i při ukládání vygenerovaných signálů do souboru).

Tlačítka `Load signal` a `Clear signal` Signál, který je v této kartě nadefinován, není automaticky použit jako vstupní signál pro jednotlivé sítě. Je nejdříve třeba signál potvrdit stiskem tlačítka `Load signal` – tímto je signál načten do evolučního algoritmu jako vstupní signál. Naopak, stiskem tlačítka `Clear signal` je signál, který je uložený v evolučním algoritmu, vymazán.

Zaškrtnutí pole `Live update` zajistí, že kdykoliv dojde ke změně signálu v této kartě (změna hodnoty, přidání/odebrání buňky v tabulce), je provedeno totéž, jako kdyby bylo stisknuto tlačítko `Load signal`.

B.2.3 NEAT params

Tato karta (viz obr. B.3) slouží k nastavení parametrů agoritmu NEAT. Tyto parametry lze zadat buď ručně a do algoritmu načíst tlačítkem `Load from here`, anebo načíst ze souboru tlačítkem `Load from file`.



Obrázek B.3: Nastavení parametrů algoritmu.

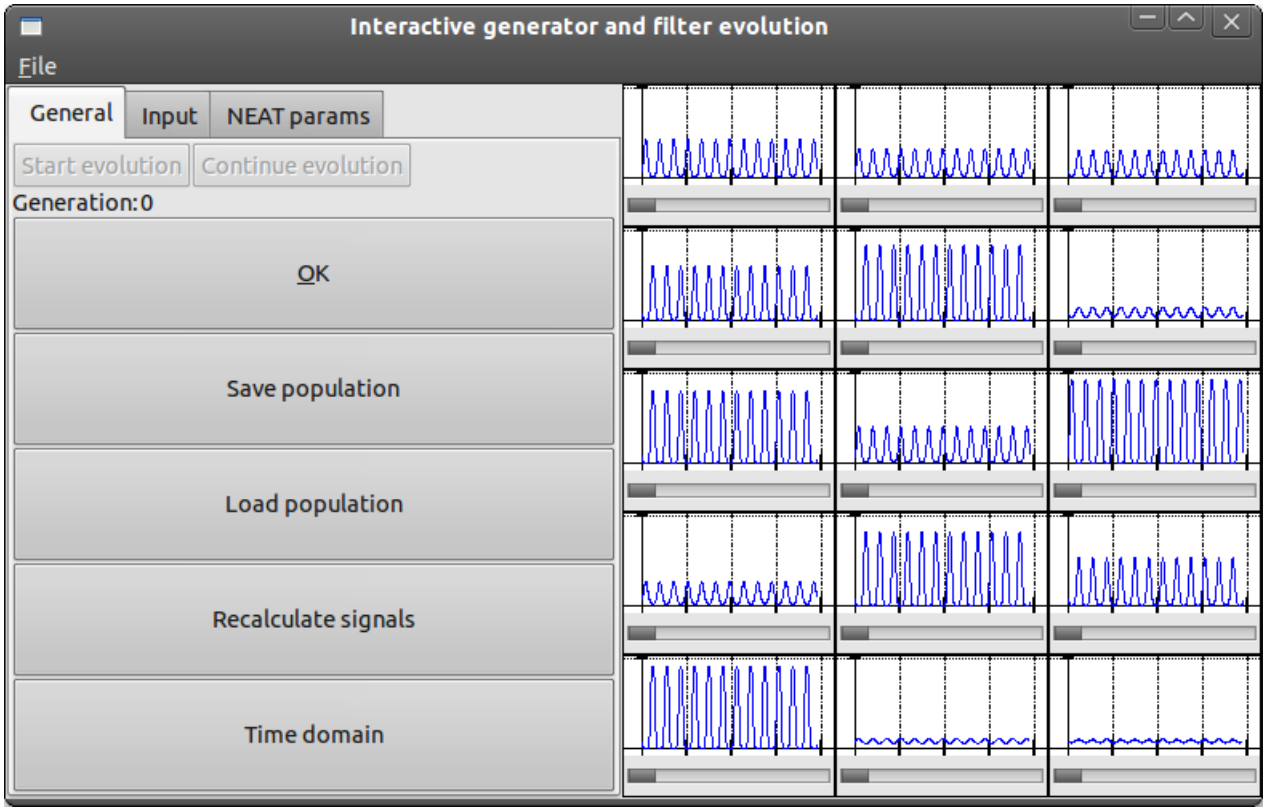
Z praktického hlediska je důležitý parametr `print_every`. Je-li hodnota tohoto parametru n, tak každou n-tou generaci je populace uložena do aktuální složky do souboru `gen_X.gen`, kde `X` je číslo generace.

Poznámka Pro běh evolučního algoritmu je **nutné**, aby byly načteny nějaké parametry. Nebude-li tak, hrozí pád programu.

B.2.4 Pravá část okna

Jsou-li nějaké vygenerované signály, tak se zobrazí v matici grafů v pravé části okna jako na obr. B.4. Každý graf má pod sebou posuvník, jehož nastavením se určuje hodnota fitness pro síť, která vygenerovala příslušný signál.

Kliknutím levým tlačítkem na graf se přehraje příslušný signál. Kliknutím pravým tlačítkem se vyvolá kontextové menu se dvěma položkami:



Obrázek B.4: Matice grafů průběhů výstupních signálů.

Play Volbou této položky je také přehrán signál stejně, jako kliknutím levým tlačítkem na signál.

Export Volbou této položky je možné uložit signál do souboru ve formátu WAV.