ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – ВАРНА

Факултет по изчислителна техника и автоматизация

Катедра „Компютърни науки и технологии“

ДИПЛОМНА РАБОТА

на

Иван Даниелов Иванов, спец. КСТ, Фак. № 096023

Тема: „Разработка на приложение за предсказване на мелодии“

Варна, 2013 г.

Ръководител: **доц. д-р инж. Анатолий Антонов**

Съдържание

1. Въведение - Необходимост от решаване на дипломната задача

1.1. Увод

Записването, съхранението, възпроизвеждането и обработването на звукова информация са дейности, които са неделима част от днешния технологичен свят. Точно те са повлияли най-силно за оформянето на съвременния облик на музикалното изкуство. Днес все по-рядко се слуша музика на живо, изпълнявана от хора. Вместо това, музиката се записва и след това се изпълнява от някакво устройство (най-често компютърна система). Дори изпълненията на живо много често биват съпроводени от системи за усилване на звука, което също е отражение на развитието на технологиите в сферата на музиката.

В днешно време всеки човек има достъп до устройство за запис или възпроизвеждане на музика. Тези устройства са развити до съвършенство, като най-добрите такива изпълняват функциите си с отклонения, недоловими за човешкото ухо.

Съхранението на музика също не представлява проблем. Едно съвременно записващо устройство, с обем 1 терабайт, може да съхрани запис с времетраене 2 години и задоволително качество на звука.

Обработването на звукова информация от своя страна се е развило значително, особено в последните 30 години. Усъвършенстването на компютърните системи и увеличаването на изчислителната им мощ е довело до способността им да изпълняват много сложни и тежки алгоритми за обработка на дигиталните звукови записи. В резултат на това, днес е възможно да се създадат всякакви звуци само с помощта на компютър. Това развитие на технологиите е довело и до създаването на съвсем нови жанрове в музиката.

Въпреки главоломният технологичен прогрес в областта на музиката в последните десетилетия, човекът все още е неделима част от процеса на създаването ѝ. За да бъде създадена нова песен, е необходимо първо тя да бъде композирана от човек. Въпреки че този принцип е, може би, в същността на създаването на последователностите от звуци, които наричаме музика, той влага и известни ограничения.

На една компютърна система, по правило, не е присъща мисловна дейност. Такава система не би могла да създаде музикално произведение, в което да има вложена емоция, както би го направил човек. Въпреки това, изчислителното устройство може да прихване моделът на последователностите в дадено готово произведение. На база на този модел може да бъде създадена нова последователност с надеждата да се запази емоцията, заложена в оригинала и усещането за музика. Процесът на генериране на нова последователност от звуци,

на базата на предварително зададена такава, ще наричаме **предсказване на мелодия.**

Смисълът на разработката на системата за предсказване на мелодии, която е предмет на разглеждане на текущата дипломна работа, е да представи възможно решение на проблема за създаване на музика с минимална човешка намеса.

1.2. Проблемът за предсказване на мелодии

Компютърното предсказване на мелодии е проблем, чието решение би било в полза на потребители от различни сфери. То има потенциала да улесни създаването на еднотипна, но разнообразна музика, без ограничения в продължителността на просвирване. Предимството в случая е ограничаването на влиянието на човешкия фактор в процеса на създаване на музикалния продукт.

Предсказването на мелодии се базира на процеса на създаване на музика по зададен модел, без да съществуват ограничения в цялостната структура на новата композиция. Според тази дефиниция, можем да направим извод, че предсказването на мелодия е много близко по същност до импровизирането. Импровизираната музика в повечето случаи е по-разнообразна от предварително композираната такава и същевременно не изисква голяма предварителна подготовка (процес на композиране). Това я прави по-приложима в някои случаи, особено когато е необходимо да се изпълнява еднородна музика през продължителен период от време.

Същевременно, извършването на тази дейност от компютърна система би намалило до минимум необходимостта от участието на човек в цялостния процес на създаване на музикалния продукт. Погледнато от икономическа гледна точка, това би намалило ресурсните изисквания за изпълнение на процеса. За него ще бъде необходима единствено компютърна система, оборудвана с необходимия софтуер и подходящи периферни устройства (звукова карта, тонколони и др.). За сравнение, закупуването на готови музикални продукти или наемането на музиканти би било свързано със значително по-големи финансови разходи на заинтересованата страна.

Компютърното предсказване на мелодии е приложимо в множество отрасли. Някои от тях са:

- Туризъм – озвучаване на хотели, ресторанти и др.
- Търговия – озвучаване на търговски обекти.
- Транспорт – озвучаване на обществени превозни средства и станции.
- Изкуство – кино продукции, озвучаване на галерии и др.

То също може да се използва и за озвучаване на лекарски кабинети, обредни зали или навсякъде, където може да се добави фоново музикално озвучение.

Основни трудности при решаване на проблема за компютърното предсказване на мелодии са:

- Представяне на данните от оригиналната мелодия в подходящ за анализиране вид и определяне на нейните характеристики.
- Разпознаване на зависимостите в мелодията и изграждане на модел, следващ с достатъчна точност оригиналните входни данни.
- Алгоритъмът за предсказване трябва да бъде такъв, че получената мелодия да отговаря на представите за музика.
- Представяне на получените данни в подходящ за слушане вид.

1.3. Обзор на съществуващи решения

В научните среди съществува терминът **компютърна музика**[1]. Той дефинира приложението на компютърните технологии в процеса на създаване и композиране на музика. Компютърната музика се базира на връзката между теорията на музиката и математиката. Първият компютър в света, който е изпълнявал музика, се нарича CSIRAC (Council for Scientific and Industrial Research Automatic Computer). Той е бил разработен от австралийците Тревър Пирси и Мастън Биърд. Математикът Джеоф Хил програмирал CSIRAC да изпълнява популярни мелодии от началото на 50-те години на 20-ти век. Най-старите познати записи на компютърно-генерирана музика са били изпълнявани от компютърът, разработен в Манчестърския университет, Ferranti Mark 1. Музиката била програмирана от Кристофър Стрейчи.

С развитието на компютърните технологии се усъвършенства и компютърната музика. Голям дял от нея заема изучаването на **машинната импровизация**. Този проблем включва в себе си, както принципите на компютърната музика, така и използването на принципи от машинното обучение и съпоставянето на образи. Това прави процесът на изучаване на машинната импровизация тясно свързан с изучаването на алгоритмите за машинно обучение, като най-често прилаганите такива са **невронните мрежи.**

Обширни изследвания в областта на композирането на музика чрез предсказване с помощта на невронни мрежи е направил професорът от университета в Колорадо Майкъл Мозер[2]. Първоначалният подход при предсказването на музика е бил да се изгради таблица на преходите, на принципа на Марковската верига, която определя вероятността на следващата нота като функция от текущия контекст. Мозер надгражда този метод, използвайки рекурентна автоматично предсказваща конекционистка мрежа (на англ. Recurrent Auto-Predictive Connectionist Network), която е наречена CONCERT. Нейната същност се състои във взаимодействието на психологично представените височина, продължителност и хармонична структура на музикалните елементи. Принципът ѝ на работа е следният: Първо мрежата се обучава върху поредица от прости мелодии, от които се извличат регулярности на нотни и фразови прогресии. Това са мелодичните и стилови ограничения. След това се извличат

височина, продължителност и хармонична структура, които се основават на психологични изследвания на човешкото възприятие.

Изследванията на Мозер се развиват върху т. нар. **статистическо стилово моделиране**. То се базира на изграждането на математически модел на музикалната платформа, който прихваща важни стилови особености от данни[1]. Статистическите подходи се използват за улавяне на повтаряемостите, в смисъл на повтарящи се мотиви и тенденции, които по-късно се съчетават, за да генерират нови музикални данни. Стиловото смесване може да бъде реализирано чрез анализиране на база от данни, съдържаща множество музикални примери от различни стилове. Машинното импровизиране се строи на базата на голяма музикална традиция от статистическо моделиране, която започва със създаването през 1957 г. на първото компютърно композирано класическо музикално произведение – „Сюита ILLIAC за струнен квартет“ (на англ. “Illiac Suite for String Quartet”) на Хилър и Исааксон, както и използваните от Янис Ксенакис вериги на Марков и стохастични процеси. Модерните методи включват компресия без загуба на информация на инкрементално парсиране, предсказващо суфиксно дърво и търсене на низове от т. нар. алгоритъм на факторния оракул (принципно факторният оракул е краен автомат, построен в линейно време и пространство в инкрементална насока).

Съществуват множество реализации на системи за машинно импровизиране. Първата такава система, използваща вериги на Марков и стилово моделиращи техники, е, разработеният от Франсоа Паше за Sony CSL, „Continuator“. Тази система, създадена в Париж през 2002 г., е базирана на нереално времево стилово моделиране. Внедряване на машинна импровизация, използваща факторен оракул, в Matlab, може да бъде намерена като част от пакета „Computer Audition Toolbox“. Друг забележителен продукт е OMax, който е софтуерна среда, разработена от IRCAM[4]. Тя е базирана на изследвания в областта на стиловото моделиране от Джерард Асеяг и Шломо Дъбнов, както и изследвания на импровизация чрез компютър от Дж. Асеяг, М. Кемълиър и Г. Блок (т. нар. OMax братя) в музикалната представителна група на IRCAM.

По темата за компютърно импровизиране или предсказване на музика са направени голям брой изследвания. Такива са направили Майкъл Мозер, Даръл Конклин и Ян Уитън, Леонело Тарабела и др. Съществуват и организации занимаващи се тясно с компютърната музика. Такава е Международната асоциация за компютърна музика - ICMC[3] (International Computer Music Association), която представлява международно сдружение на хора и институции, които се занимават с техническите, креативните и изпълнителските аспекти на компютърната музика. Тя служи на композитори, инженери, изследователи и музиканти, които са заинтересовани от съчетаването на музика и технологии в едно. Друга такава организация е IRCAM[4] (Institut de Recherche et Coordination Acoustique/Musique). Това е френски научен институт за изследвания в областта

на музиката. Организация, която също се развива в областта на компютърната музика, е Обществото за акустична електронна музика в Съединените щати - SEAMUS (Society for Electro Acoustic Music in the United States).

1.4. Средства за изграждане и управление на невронни мрежи

Невронните мрежи, които ще бъдат подробно разгледани в раздел **2.1**, представляват модели за обработка на информация, вдъхновени от изучаването на биоелектричните мрежи в мозъка на човека и животните, образувани от неврони и техните синапси[5]. Една невронна мрежа може да бъде представена абстрактно като граф, като отделните неврони се представят като възли, а синапсите, свързващи невроните като дъги между възлите. Основното свойство на невронната мрежа е нейната способност за самообучение. Невронна мрежа може да бъде приложена в различни среди. Тя може да се изгради, както софтуерно, така и хардуерно.

Софтуерната реализация на невронни мрежи най-често се изпълнява чрез алгоритмични програмни езици. Предпочитани такива са C, C++, Java, C# и др. Предимството на тези езици е, че с тяхна помощ лесно може да се опише абстрактният алгоритъм на мрежата, тъй като те са от високо ниво. Същевременно тези езици, при добро организиране на кода и ресурсите от страна на програмиста, водят до задоволително бързодействие на невронната мрежа. От друга страна, използването на език от ниско ниво като Асемблер е възможно да доведе до по-голямо бързодействие на мрежата. Проблемът в този случай е трудното описване на алгоритъма, както и необходимостта от достатъчно квалифициран програмист, който да е наясно със специфичните особености на конкретната апаратна конфигурация, за която се разработва невронната мрежа.

Освен езикът, на който се разработва мрежата, от значение е и парадигмата на програмиране, която ще бъде използвана. При обектно-ориентираното програмиране описанието на невронната мрежа е подходящо да бъде представено като набор от обекти, инстанции на класове (например клас Неврон, клас Синапс и др.). Този стил на програмиране води до по-прегледен код и по-интуитивно структуриране на абстрактния модел. Недостатък на обектно-ориентирания подход е, че се внася забавяне в бързодействието на алгоритъма, тъй като се извършват множество операции с големи масиви от данни, голяма част от които са служебни, а също и че се правят множество достъпи до оперативната памет.

Друг подход е използването на процедурен модел за изграждане на алгоритъма. В този случай невронната мрежа се построява като последователност от команди и процедури, които дословно следват описанието на абстрактния алгоритъм. Абстрактната структура на мрежата може да се представи като многомерен масив от числови данни. Този подход води до по-голямо бързодействие на мрежата, в сравнение с обектно-ориентирания. Недостатъкът е,

че структурата на мрежата не е интуитивно представена. Вероятността от допускане на грешки и нарушаване на сигурността е по-голяма.

При управлението на мрежа е важно да се вземат предвид нейните характеристики. Има различни видове невронни мрежи, които се класифицират по признаците, разгледани в точка **2.1.1**. Всеки вид е предназначен за решаване на различен тип проблем. Обикновено различните видове невронни мрежи се характеризират с различна структура и поведение. При избор на вид невронна мрежа е важно да се определи типът на проблема, който ще бъде решаван от нея. Примерно, при решаване на проблема за предсказване на времеви серии е подходящо да се използва **невронна мрежа с обратно разпространение на грешката**.

Важно е също да се определи броят на невроните в мрежата. Колкото по-голям е той, толкова по-мощна се явява мрежата (може да обхване по-голямо пространство от състояния). Проблемът при покачване броя на невроните е повишаването на сложността на системата, а от там и значителното намаляване на бързодействието ѝ.

Определянето на входните и изходни параметри на системата с невронна мрежа също е основен момент от нейното проектиране. Обикновено входът на невронната мрежа представлява вектор от числови данни. Това води до необходимостта за представяне на качествените данни в количествени (например една нота да се представи като набор от числови характеристики). Изходът на системата може да бъде единична стойност (например булева стойност – 0 или 1), както и вектор от числови данни.

Възможно е да се определят и други допълнителни параметри, описващи характеристиките на невронната мрежа, в зависимост от нейният вид, структура и предназначение.

Предимствата на невронните мрежи пред другите модели за машинно обучение са:

- Лесно е да се осъзнае принципът на работа.
- Големият брой на научните изследвания в областта са довели до голямо усъвършенстване невронните мрежи.
- Невронните мрежи имат голямо приложение в множество сфери.
- Съществуват множество разработки и библиотеки за работа с невронни мрежи.

Недостатъци:

- По бавни и трудоемки са в сравнение с алтернативни решения като метод на опорните вектори, дърво на решенията, регресионен анализ и др.

- Трудно се обучават и е необходима прецизна настройка на параметрите.

1.5. Обучение и изпълнение

Както беше обяснено, невронните мрежи са изградени от два основни компонента – неврони и свързващи ги синапси. Обикновено всяка връзка (синапс) има някакво тегло - числова стойност. Процесът на обучение на мрежата представлява настройване на множеството от теглата на връзките между невроните, така че при определен входен вектор да се получи подходящ изходен такъв.

Два са основните метода за обучение на невронна мрежа – с учител и без учител. Ще наблегнем върху обучението с учител, тъй като невронната мрежа, разработена в текущата дипломна работа, се обучава именно на този принцип. То се извършва по следния начин: В началото на обучителния процес теглата на синапсите са зададени с произволни стойности. Първо се задава конкретен входен вектор на мрежата. На изхода на мрежата се получава вектор от стойности, определени от теглата на синапсите. В ролята на учител се явява вектор от стойностите, които е трябвало да се получат на изхода при дадения входен вектор. Изходният и примерният вектор се сравняват и, на база на тяхната разлика, теглата на връзките се променят в желаната посока с много малка стъпка. Този процес се повтаря многократно, докато разликата между получаваните резултати и правилните такива не се снижи до минимум. Същинският алгоритъм за обучение на невронна мрежа с обратно разпространение на грешката е разгледан в точка **2.1.3**.

Принципът на обучение с учител е добре приложим в случаите, когато решаваме задача за предсказване на времеви серии. При тях имаме наличен набор от елементи. Тези елементи са последователно разпределени във времето. Обучението на мрежата става, като на входа ѝ се зададат началните елементи на набора, а полученият изход се сравни със следващите. Точно такъв е случаят при предсказването на мелодии. При обучението на мрежата е важно добре да се определят входовете и очакваните изходи. Принципът на обучение на невронната мрежа за предсказване на мелодии е подробно описан в точка **3.3.**

След етапа на обучение на мрежата, идва етапът на изпълнение (предсказване). Тук мрежата вече е с настроени тегла на синапсите и е готова за същинската работа, за която е предназначена. Обикновено при предсказване изпълнението започва като на входа на мрежата се постави последният вектор от известните вече данни. На базата на предишния опит, невронната мрежа извежда изход, който най-добре следва зависимостите при предходните серии.

2. Въведение - Теоретична част

2.1. Невронни мрежи

В тази глава ще бъдат разгледани по-подробно видовете невронни мрежи и тяхното предназначение. Ще бъде наблегнато на невронните мрежи с обратно разпространение на грешката.

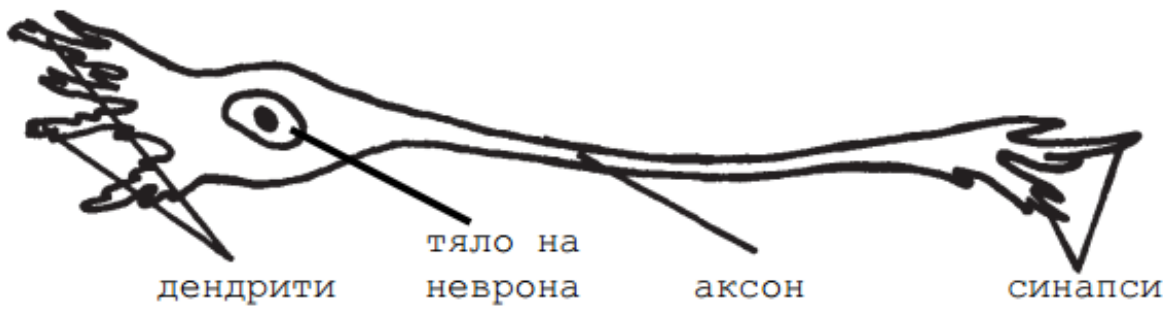
Същност и видове

В изкуствения интелект[6] съществуват два основни подхода за представяне и използване на знанията – символен и конекционистки. Най-съществената разлика между тях е свързана с отношението им към хипотезата за представянето на знанията, формулирана от Брайън Смит през 1982 г. Същността на тази хипотеза е, че всяка система, която проявява интелигентно поведение, включва в състава си две обособени части:

- База знания, която съдържа в кодиран вид знанията, достъпни на системата.
- Машина за извод (интерпретатор на знанията), която обработва символите (текстовете) от база знания с цел пораждане на интелигентно поведение.

Тази хипотеза се приема от привържениците на символния (знаков, текстов) подход и се отхвърля от привържениците на конекционисткия подход, които предлагат методи и архитектури за представяне на знания и решаване на интелектуални задачи, които се основават на идеи, свързани с начина на запазване и използване на знанията в човешкия мозък.

Невронните мрежи (по-точно, изкуствените невронни мрежи) са най-типичен и най-развит представител на конекционисткия подход. Основните градивни елементи на (изкуствените) невронни мрежи са силно опростени модели на биологичните неврони, от които е съставен човешкият мозък.



Фигура 1. *Стилизиран модел (опростена схема) на биологичен неврон.* [6]

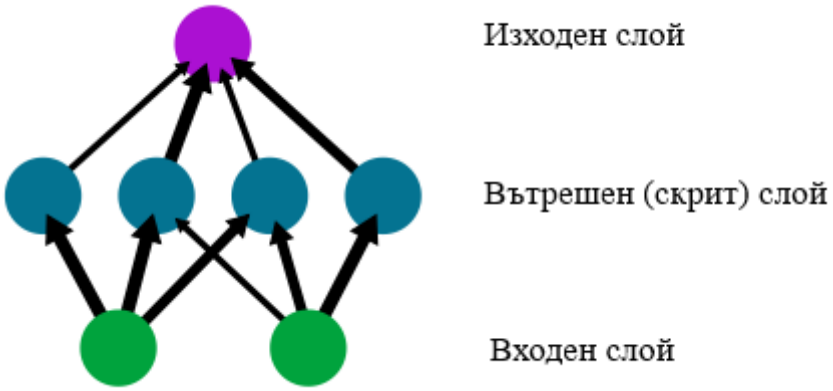
- Дендрити - разклонени структури, които осигуряват сензорен вход към тялото на неврона (входни устройства на неврона).

- Тяло на неврона - сумира мембранните потенциали между синапсите и дендритите и изпраща по аксона активиращо напрежение с определен размер.
- Аксон - провежда електрическия сигнал от тялото на неврона до неговите синапси.
- Синапси - трансформират активиращото напрежение, получено по аксона, в електрически сигнали (импулси), които възбуждат или потискат активността на съседните неврони (изходни устройства на неврона).

Всяка (изкуствена) невронна мрежа е система от обработващи елементи (processing units), които са силно опростени модели на биологическите неврони и затова често условно се наричат (изкуствени) неврони. Обработващите елементи са свързани помежду си с връзки с определени тегла. Всеки обработващ елемент преобразува входните стойности (входните сигнали), които получава, и формира съответна изходна стойност (изходен сигнал), която разпространява към елементите, с които е свързан. Това преобразуване се извършва на две стъпки. Най-напред се формира сумарният входен сигнал за дадения елемент, като за целта отделните входни стойности (сигнали) се умножават по теглата на връзките, по които се получават, и резултатите се събират. След това обработващият елемент използва специфичната за мрежата активационна функция (функция на изхода), която трансформира получения сумарен вход в изхода (изходния сигнал) на този елемент.

Специалистите определят пет основни характеристики на невронните мрежи, които обикновено се използват за класификационни признаци при определянето на типовете модели на невронни мрежи. Тези характеристики са: **топологията на мрежата, параметрите на обработващите елементи, типът на входните стойности, използваното обучаващо правило и методът за обучение на мрежата.**

Според **топологията на мрежата**, най-често срещани са невронните мрежи, съставени от няколко обособени (последователни) слоя от елементи (неврони), при които елементите от най-ниския слой играят ролята на входни устройства на мрежата (те възприемат сигнали от външната среда), а елементите от най-горния слой играят ролята на изходни устройства на мрежата (те извеждат резултата от работата на мрежата, който по същество се получава въз основа на входните сигнали и теглата на връзките в системата). Често при тези невронни мрежи връзките са еднопосочни и свързват елементите от един слой с елементи от слоя, разположен непосредствено над него. В зависимост от броя на слоевете в мрежата се говори за двуслойни невронни мрежи (при тях има само входен и изходен слой



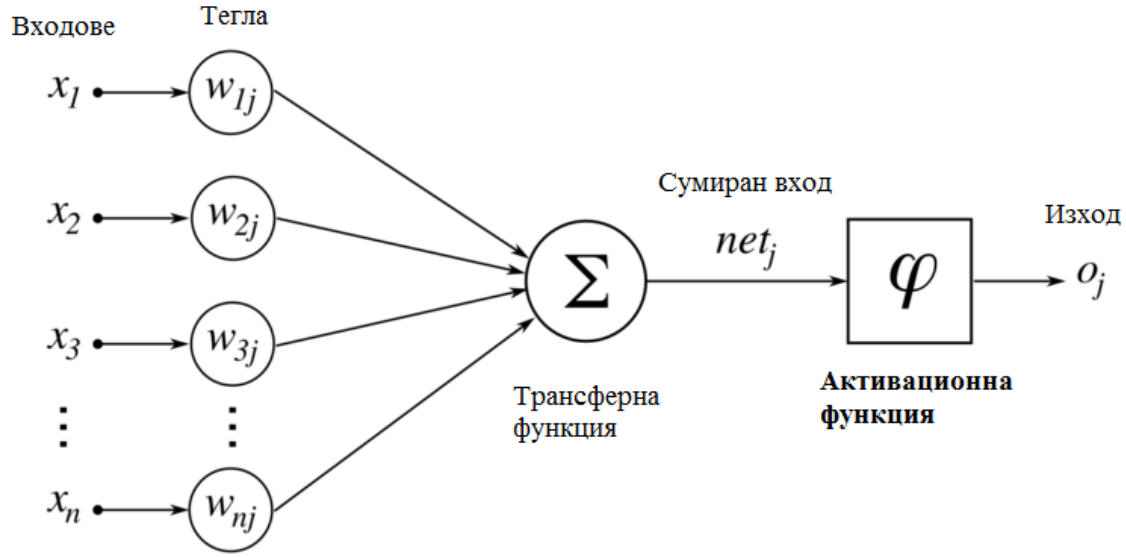
Фигура 2. *Опростен модел на трислойна невронна мрежа.*

и липсва т. нар. вътрешен или скрит слой) и многослойни невронни мрежи (при тях има поне един скрит слой).

На **Фигура 2** е изобразена схема на проста трислойна невронна мрежа. Обикновено за удобство се счита, че всеки елемент от i-я слой е свързан с всеки елемент от (i+1)-я слой, но е възможно теглата на някои връзки да са нули.

Невронните мрежи могат да се класифицират и по **параметрите на обработващите елементи.** Тук се включват теглата и посоката на връзките, видът на активационната функция и др.

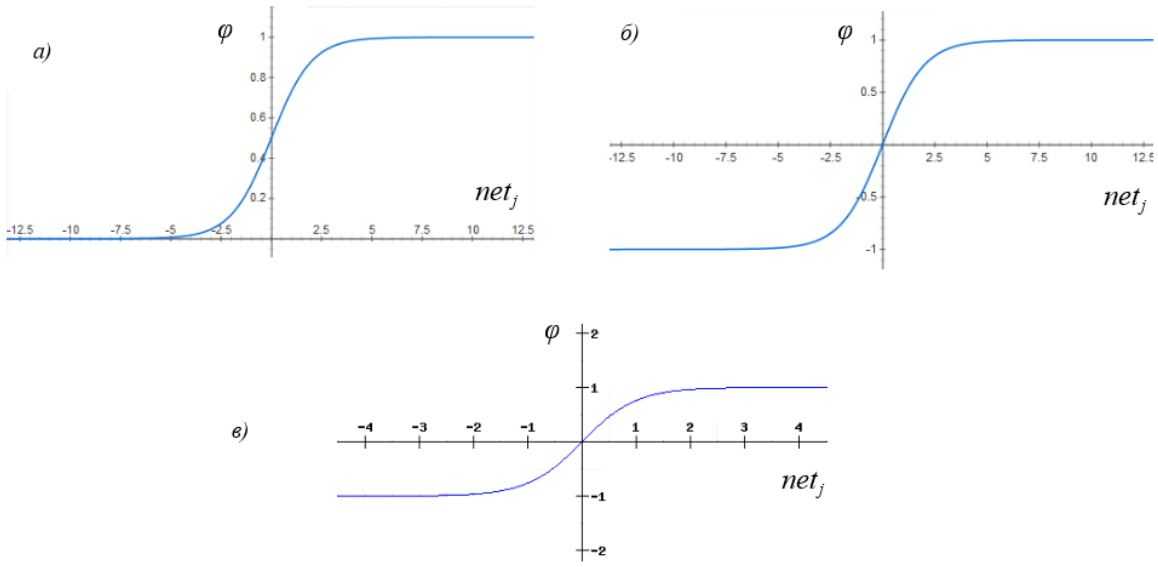
Както беше посочено, при повечето невронни мрежи връзките са еднопосочни и са ориентирани от елементите от даден слой към елементите от слоя, разположен непосредствено над него. Съществуват обаче и невронни мрежи (такива са например мрежите на Хопфийлд), при които всеки елемент е свързан с двупосочни връзки с всички свои съседи. При тези мрежи понятието слой до голяма степен губи своя смисъл.



Фигура 3. *Модел на изкуствен неврон*

На **Фигура 3** е дадена схема на изкуствен неврон. Всеки вход x_i се умножава по съответстващото му тегло w_{ij}. След това трансферната функция сумира получените стойности, от където получаваме net_j. Активационната функция е от основно значение за работата на невронната мрежа. Някои от най-често използваните активационни функции са:

- $\varphi = \frac{1}{1+e^{-net_j}}$, чиято графика е изобразена на **Фигура 4, а)**.
- $\varphi = \frac{1-e^{-net_j}}{1+e^{-net_j}}$, чиято графика е изобразена на **Фигура 4, б)**.
- $\varphi = \frac{e^{net_j}-e^{-net_j}}{e^{net_j}+e^{-net_j}}$, чиято графика е изобразена на **Фигура 4, в)**.



Фигура 4. *Графики на активационни функции*

Входните стойности на мрежата (т. е. сигналите, които получават елементите от входния слой) могат да бъдат двоични (0 или 1) или аналогови (реални числа).

Най-често невронните мрежи се използват за решаване на задачи, свързани с разпознаване (класификация). При това създаването на невронна мрежа, предназначена за решаване на дадена задача, обикновено се извършва по следната обща схема. Най-напред се определя топологията на мрежата, т. е. броят на слоевете и броят на елементите във всеки слой. Броят на елементите от входния слой се определя от обема на входните данни, а броят на елементите от изходния - от броя на разпознаваните класове. Размерите на скритите слоеве, които обикновено са нула, един или два на брой, най-често се определят итеративно в зависимост от конкретната задача. След определянето на топологията на мрежата се преминава към нейното обучаване, т. е. към определянето на подходящи

стойности на теглата на връзките между елементите. За целта най-често първоначално се задават случайни стойности на търсените тегла, след което многократно се изпълнява следната процедура. Избира се пример от обучаващото множество. Този пример се състои от вектор от входни стойности и съответен вектор от правилни (желателни, очаквани) изходни стойности на мрежата. След това се определят действителният изход на мрежата за разглеждания входен вектор и разликата между желателния и действителния изход. Накрая се променят текущите тегла на връзките така, че новият изход да бъде по-близък до желателния. Правилото, по което се променят теглата на връзките, се нарича **обучаващо правило** (обучаващ алгоритъм) на мрежата.

Както беше споменато в точка **1.5**, съществуват два основни **метода на обучение** на невронни мрежи - обучение с учител (надзиравано обучение, supervised learning) и обучение без учител (самообучение, ненадзиравано обучение, unsupervised learning). При обучението с учител се следи разликата между получения и очаквания изход на мрежата (учителят задава очаквания изход на мрежата) и итеративно се извършват корекции на теглата на връзките съгласно избраното обучаващо правило. При самообучението липсва учител, т. е. липсват предварителни данни за правилния изход на мрежата. Теглата на връзките се настройват така, че представянето на данните в мрежата да е най-добро съгласно използвания (зададения) критерий за качеството на представянето.

В зависимост от използвания метод на обучение невронните мрежи се разделят на две групи: невронни мрежи, обучавани с учител, и самообучаващи се невронни мрежи.

Предимства на невронните мрежи

Своята сила невронните мрежи черпят от разклонената обработка на информацията и от способността за самообучение за създаване на обобщение. [7] Под термина обобщение се разбира способността за получаване на обоснован резултат на основата на данни, които не са се срещали в процеса на обучение. Тези свойства позволяват на невронните мрежи да решават сложни (мащабни) задачи, които за даденият момент се считат трудно разрешими. На практика, обаче, автономната работа на невронните мрежи не може да обезпечава готови решения. Необходимо е те да се интегрират в сложни системи. В частност комплексна задача може да се раздели на последователности (относително прости), част от които може да се решават от невронната мрежа. За създаването на компютърна архитектура, която да бъде способна да имитира човешкият мозък (ако е възможна изобщо) ще трябва да се измине дълъг и труден път. Използването на невронните мрежи обезпечава следните полезни свойства на системата:

1. Нелинейност – изкуствените неврони могат да бъдат линейни и нелинейни. Невронните мрежи, построени от съединени нелинейни неврони се явяват нелинейни. Тази нелинейност е от особен вид, тъй като е разпределена по

мрежата. Нелинейността е много важно свойство особено ако физическият механизъм отговарящ за формиране на входният сигнал също е нелинеен (например човешката реч).

2. Отразяване на входната информация в изходна – една от популярните парадигми се явява обучение с учител. Това включва изменение на синаптическите тегла на основа на избор на маркирани учебни примери. Всеки пример се състои от входен сигнал и съответен на него желан отговор. От това множество случайни образи се избира пример, а невронната мрежа модифицира синаптичните тегла за минимизация на разклонение на желания изходен сигнал и формулираната мрежа съгласно избран статистически критерии. При това собствено се модифицират свободни параметри на невронната мрежа. По-рано използваните примери в последствие могат да се прилагат отново, но вече в друг ред. Това обучение се провежда до тогава, докато изменението на синаптичните тегла не стане незначително. По този начин невронната мрежа се обучава на примери, съставяйки таблици на съответният вход и изход за конкретна задача. Този подход наподобява непараметричното статистическо обучение. Това направление в статистиката има работа с оценките, които не са свързани с конкретни модели или от биологична гледна точка, с обучението с нула. Тук термина „непараметрично” се използва за акцентиране на това, че от начало не съществува никакъв предопределен статистически модел за входни данни. За пример ще разгледаме задачата за класификация на образите. В нея се изисква да се съотнесе входния сигнал, представляващ физически обект или събитие с някои предопределени категории (класове). При непараметричният подход при решаването на тази задача трябва да се оцени рамка за решение в пространството на входния сигнал на основата на избора на примери. При това не се използва никакъв вероятностен модел на разпределение. Аналогичен подход се прилага в параметричното обучение с учител. Това още веднъж подчертава успоредността между отразяването на входните сигнали в изходни, осъществено от невронната мрежа и непараметричното статистическо обучение.

3. Адаптивност – невронните мрежи имат способността да адаптират своите синаптични тегла към околната среда. В частност, невронните мрежи обучени да работят в една среда могат лесно да бъдат пробудени да работят в условия с незначителни колебания на параметрите на средата. При работа в нестационарна среда, където статистически се изменя с течение на времето, могат да бъдат създадени невронни мрежи, изменящи синаптичното си тегло в реално време. За да се използват всички достойнства на адаптивността, основните параметри на системата трябва да са достатъчно стабилни, за да работят, неотчитайки външните въздействия и да бъдат достатъчно гъвкави, обезпечавайки реакцията на съществено изменение на средата. Тази задача обикновено се нарича Дилема на стабилност – пластичност.

4. Очевидност на отговора – В контекста на задачите за класификация на образи може да се разработи невронна мрежа, събираща информация не само за определен конкретен клас, но и за увеличение на достоверността на взетото решение. Тази информация може да се използва за изключване на съмнителни решения, което повишава продуктивността на невронната мрежа.

5. Контекстна информация – Знанията се представят в самата структура на невронната мрежа с помощта на нейното състояние на активност. Всеки неврон от невронната мрежа може да бъде подложен на влияние от всички останали неврони. Като следствие съществуват невронни мрежи свързани с конкретна информация.

6. Отказоустойивост – Невронните мрежи облечени във форма на електроника са потенциално отказоустойчиви, т.е. при неблагоприятни условия тяхната производителност пада незначително. Например ако е повреден някой от невроните или неговата връзка, извличането на запаметената информация се затруднява. Отчитайки разпределителната характеристика на съхранение на информацията в невронните мрежи може да се потвърди, че само сериозни повреди в структурата на невронните мрежи съществено ще повлияе на нейната работоспособност. Затова понижаването на качеството на работа на невронната мрежа протича бавно. Незначителното повреждане на структурата никога не оказва катастрофални последствия.

7. Мащабируемост (VLSI Implementability) – Успоредната структура на невронните мрежи потенциално ускорява решението на някои задачи и обезпечава мащабността на невронните мрежи в рамките на технологията VLSI (very-large-scale-integrated). Едно от предимствата ѝ е възможността да се представи достатъчно сложно поведение чрез йерархична структура.

8. Еднообразие на анализа и проектирането – Невронните мрежи са универсален механизъм за обработка на информация, т.е. едно и също проектно решение на невронната мрежа може да се използва в много различни области. Това свойство се проявява по няколко начина:

- Невроните в една или друга форма са стандартни съставни части на всяка невронна мрежа;
- Тази общност позволява да се използват едни и същи теории и алгоритми за обучение в различни невронно мрежови приложения;
- Модулната мрежа може да бъде построена на основата на интеграция на цели модули.

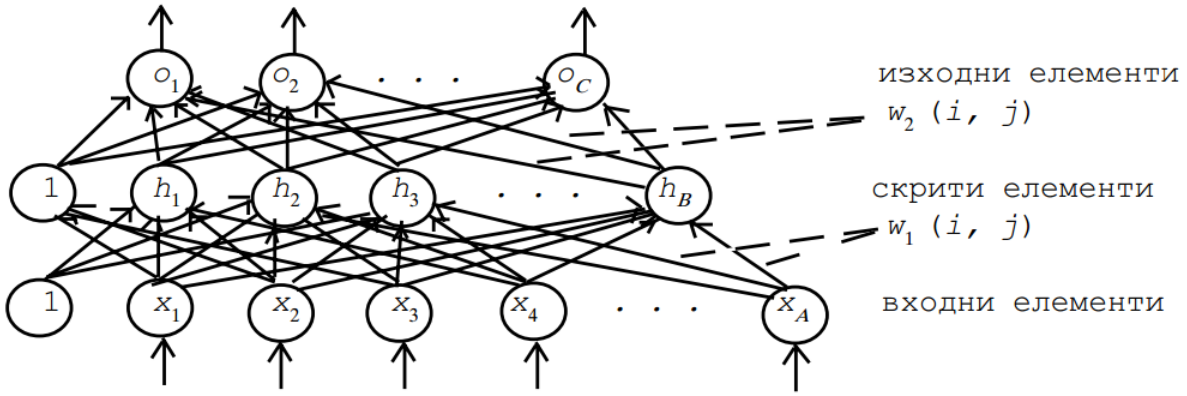
9. Аналогия с невробиологията – Построяването на невронни връзки се определя чрез аналогията с човешкият мозък, който се явява живо доказателство за това, че отказоустойчиви успоредни изчисления, не само физически са реализуеми, но и се явяват бърз и мощен инструмент за решаване на различни задачи. Невробиологията разглежда изкуствените невронни връзки като средство

за моделиране на физически явления. Невробиологичният модел въплътен в нервно-морфните контури дава надежда за това, че физическото разбиране на невробиологичните структури може да окаже съществено влияние в областта на електрониката и технологиите VLSI.

Невронни мрежи с обратно разпространение на грешката

Това са невронни мрежи с поне един скрит слой, при които съществуват връзки от всеки елемент от даден слой към всеки елемент от непосредствено следващия слой.

Ще разгледаме алгоритъмът[6] на работа на примерна невронна мрежа с обратно разпространение като тази, изобразена на **Фигура 5**.



Фигура 5. *Примерна невронна мрежа с обратно разпространение, която има един скрит слой.* [6]

В конкретния случай за активационна функция на невроните в мрежата ще използваме следната:

$$\varphi = \frac{1}{1+e^{-net_j}}$$

Където net_j е сумата на всички x_i, умножени със съответните тегла. Така при $net_j = 0$ се получава стойност $\varphi = 0.5$. Изобщо φ има реална стойност между 0 и 1.

Първоначално се задават случайни начални стойности на търсените тегла, които се коригират при проверката на действието (резултатите от работата) на мрежата върху всеки от обучаващите примери. Работата върху всеки пример се извършва на два етапа, които се наричат съответно прав и обратен пас. При правия пас се получава изходът на мрежата, съответен на текущите стойности на търсените тегла. При обратния пас получените изходни стойности се сравняват с правилния изход за съответния пример и се коригират стойностите на теглата на връзките $w_2(i, j)$, стойностите на елементите от скрития слой h_i и теглата на връзките $w_1(i, j)$.

Точна формулировка на алгоритъма[6]**:**

Дадено: Множество от обучаващи примери (наредени двойки $(\vec{x}, \vec{y})$ от съответни входни и изходни стойности на мрежата).

Търси се: подходящи стойности $\vec{w}_1$ и $\vec{w}_2$ на теглата на връзките в трислойна невронна мрежа, която по дадени входни стойности от обучаващото множество връща съответните изходни стойности (тя работи правилно върху даденото множество от обучаващи примери).

Изчислителна схема:

1. Нека A е броят на елементите от входния слой (съответен на дължината на входните вектори от обучаващите примери) и C е броят на елементите от изходния слой (съответен на дължината на изходните вектори от обучаващите примери). Избира се подходяща стойност на B, равна на броя на елементите от скритият слой. За целта има многобройни резултати, които могат да се използват. Сравнително голям скрит слой дава по-голяма мощност на мрежата, но тя не винаги е желана. Примери за успешно използвани топологии при сравнително големи мрежи са 203-80-26, 960-9-45, 459-24-24-1 (числата са броя на елементите в съответния слой) и др. Тук за удобство във входния и скритият слой се добавя още по един, допълнителен елемент с постоянна стойност 1. Така индексите на елементите от входния слой приемат стойности от 0 до A, тази на елементите от скритият слой - от 0 до B, а в изходния слой - стойности от 1 до C.

Въвеждаме следните означения:

- x_i - стойностите (активационните равнища) на елементите от входния слой (при това винаги $x_0 = 1$).
- h_i - стойностите (активационните равнища) на елементите от скритият слой (при това винаги $h_0 = 1$).
- o_i - стойностите (активационните равнища) на елементите от изходния слой.
- $w_1(i, j)$ - теглата на връзките между елементите от входния и скритият слой (i - индекс на елемента от входния слой, j - индекс на елемента от скритият слой).
- $w_2(i, j)$ - теглата на връзките между елементите от скритият и изходния слой (i - индекс на елемента от скритият слой, j - индекс на елемента от изходния слой).

2. Инициализират се теглата на връзките в мрежата. Всяко тегло получава случайна стойност в интервала (-0.1, 0.1):

$w_1(i, j) = random(-0.1, 0.1)$ за $i = 0,1,..., A$ и $j = 1,2,..., B$;

$w_2(i, j) = random(-0.1, 0.1)$ за $i = 0,1,..., B$ и $j = 1,2,..., C$.

3. Инициализират се стойностите (активационните равнища) на допълнителните елементи: $x_0 = 1.0$; $h_0 = 1.0$. Тези стойности не се променят при следващото изпълнение на алгоритъма.

4. Избира се обучаващ пример, т. е. двойка $(\vec{x}, \vec{y})$, където $\vec{x}$ е входен вектор и $\vec{y}$ - съответен изходен вектор. Съответните стойности от $\vec{x}$ се присвояват на елементите от входния слой на мрежата.

5. Разпространяват се активационните стойности от входния към скрития слой, като за целта се използва активационна функция от посочения по-горе вид:

$$h_i = \frac{1}{1+\exp(-\sum_{i=0}^{A} w_1(i,j)x_i)}, j = 1,2,...,B$$

6. Разпространяват се активационните стойности от скрития към изходния слой:

$$o_j = \frac{1}{1+\exp(-\sum_{i=0}^{B} w_2(i,j)h_i)}, j = 1,2,...,C$$

7. Пресмятат се грешките, получени при елементите от изходния слой. Нека означим тези грешки с $\delta_2(j)$; всяка от тях се пресмята с помощта на изхода на мрежата o_j и правилния изход от обучаващия пример y_j:

$$\delta_2(j) = o_j(1-o_j)(y_j - o_j), j = 1,2,...,C$$

8. Пресмятат се грешките на елементите от скрития слой. Нека означим тези грешки с $\delta_1(j)$:

$$\delta_1(j) = h_j(1-h_j)\sum_{i=1}^{C}\delta(i)w_2(j,i), j = 1,2,...,B$$

9. Уточняват се стойностите на теглата на връзките между скрития и изходния слой:

$$w_2(i,j) = w_2(i,j) + \Delta w_2(i,j),$$
$$\Delta w_2(i,j) = \eta\delta_2(j)h_i; i = 0,1,...,B; j = 1,2,..,C.$$

Забележка. Стойността на скаларния коефициент η определя скоростта на доближаване до търсените стойности на теглата и изборът ѝ зависи от спецификата на решаваната задача. Една подходяща стойност е $\eta = 0.35$.

10. Уточняват се стойностите на теглата на връзките между входния и скрития слой:

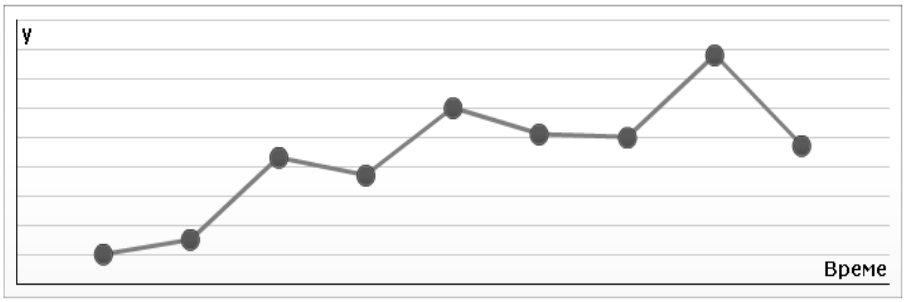
$$w_1(i,j) = w_1(i,j) + \Delta w_1(i,j),$$
$$\Delta w_1(i,j) = \eta\delta_1(j)x_i; i = 0,1,...,A; j = 1,2,..,B.$$

11. Преминава се към стъпка 4 и се повтарят действията от стъпки 4 - 10 за всички останали обучаващи примери.

Сходимост на алгоритъма: Няма теорема за сходимост на представения алгоритъм. Проблемът е свързан с това, че при него по същество се цели минимизация на грешките $\delta_2(j)$ и е възможно да се достигне до локален, а не до търсения глобален минимум върху повърхнината над пространството от теглата, дефинирана от специална функция на грешката (която няма да разглеждаме). В тези случаи алгоритъмът не е сходящ, но такива ситуации могат да бъдат откривани, а съществуват и методи, които позволяват те да бъдат избягвани чрез подходяща модификация на изчислителната схема на алгоритъма. Следователно не възникват реални проблеми със сходимостта на разгледания алгоритъм. Проблемите при него са свързани с ниската му скорост на сходимост и следователно с ниската скорост на обучение на невронната мрежа, което означава, че са необходими голям брой обучаващи примери. Въпреки това невронните мрежи с обратно разпространение се използват широко поради голямата им мощност.

Управление на процеса на предсказване

Невронните мрежи и в частност мрежите с обратно разпространение могат да се използват за предсказване на времеви серии. Времевите серии представляват последователности от вектори или скалари, които зависят от времето. Примери за времеви серии са годишните енергийни разходи на град, климатичните температурни стойности за месец на дадена територия и др. За времева серия може да се приеме и мелодията, ако нотите се представят като вектори от техните характеристики, тъй като тези ноти са последователно разпределени във времето. Времевите серии могат да бъдат дискретни и аналогови. Пример за графично представяне на времева серия е показан на **Фигура 6**.



Фигура 6. *Времева серия.*

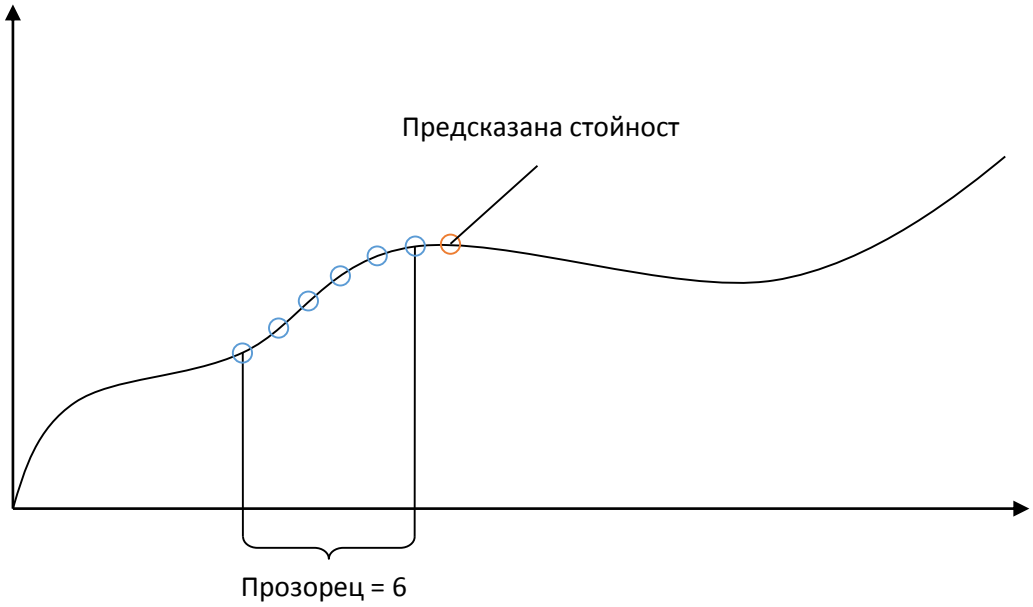
Времевите серии зависят от множество фактори. Например, продажбите на едно списание могат да бъдат понижени, ако в дадения период има лоши метеорологични условия (дъжд, сняг и др.). Тази на пръв поглед слаба връзка може да доведе до осезаемо изменение в стойностите на времевата серия на продажбите. Да се следи влиянието на всички възможни фактори, които

потенциално могат да повлияят на дадена времева серия, е трудоемка, дори невъзможна задача за човек. Тук на помощ идват невронните мрежи, които лесно откриват зависимости във времевите серии. При намиране на повторяемости в историческата серия, невронната мрежа настройва по такъв начин теглата на своите синапси, така че при подаването на сходни входни данни, мрежата да повтори поведението на серията. Невронните мрежи са много гъвкави в това отношение. Без значение дали стойностите ще бъдат много по-големи от предварително проучените или по-друг начин да се отличават от тях, мрежата ще повтори времевата серия със същото поведение, но с пропорционални отклонения.

Невронните мрежи могат да класифицират, предсказват, констатират зависимости (например картелни споразумения между търговци) и извършват други действия с времеви серии. В тази точка ще наблегнем на процеса на предсказване.

За да обясним методът, който се използва за предсказване на времеви серии, ще използваме пример, в който времевата серия съдържа единични стойности. Да вземем например акциите на дадена фирма. Можем да вземем цените на тези акции в различни моменти от време. За предпочитане е примерите да бъдат голям брой, за да може мрежата да има достатъчна база за обучение. Наредени последователно, стойностите образуват вектор от числени данни. Това от което се интересуваме е как ще се движат цените в някакъв бъдещ период от време.

Трябва да определим входния и изходния вектор на невронната мрежа, които ще играят работа на обучаващи двойки $(\vec{x}, \vec{y})$. За вход можем да вземам определен брой елемента от времевата серия (например 6) с надеждата, че невронната мрежа ще има достатъчно информативен „поглед назад“ в серията. Броят на елементите от времевата серия, които ще предаваме като вход на невронната мрежа, се нарича **прозорец** (**Фигура 7**). На изхода ще очакваме елемента, който трябва да следва избраните шест елемента за входа. В процеса на **обучение** ще сравняваме стойността на получения елемент със стойността на съответстващия му реален такъв. Обучението става на гореописания принцип на обучение с учител. На база на разликата между получената и очакваната стойност на изходния елемент се настройват теглата на връзките между невроните в мрежата. При обучението, след всяка итерация (завършен цикъл „предсказване на стойност - корекция“), прозорецът се измества с един елемент напред във времевата серия. Така елементът, който е бил изходен в предишната итерация, става последен елемент от прозореца на входните елементи, следващият го – нов изходен елемент, върху който ще се обучава мрежата. Това придвижване на прозореца става, докато последният елемент от историческата серия стане част от прозореца. В този момент започва процесът на **предсказване**. За изходния елемент вече не

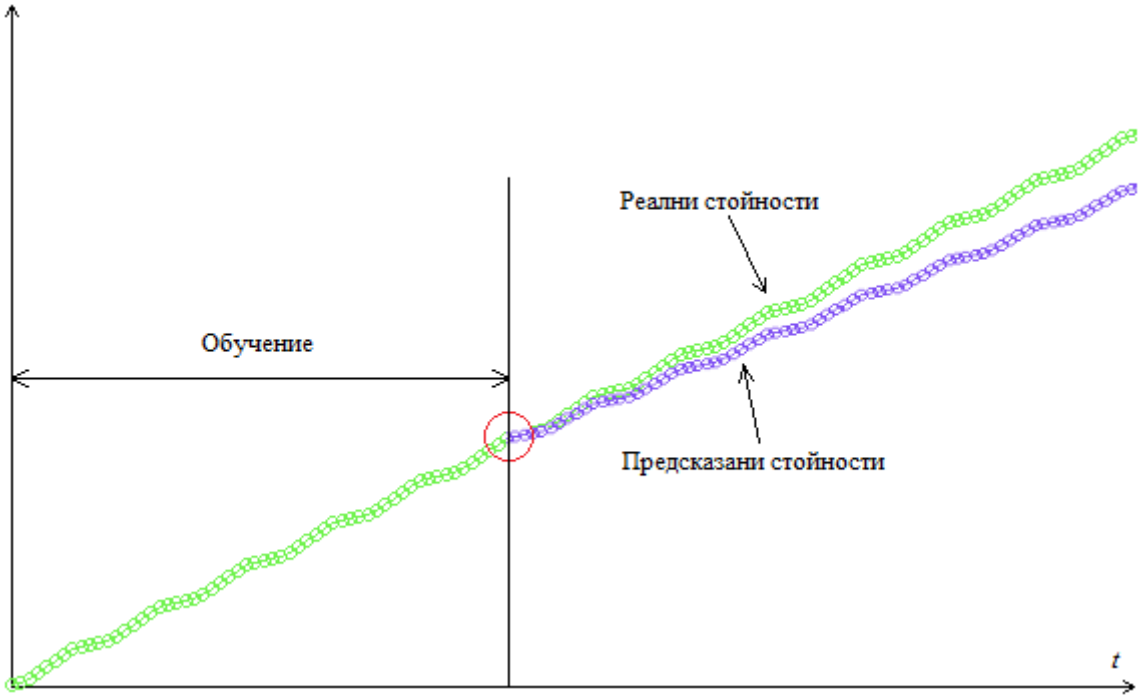


Фигура 7. *Прозорец при предсказване на времеви серии*

съществува очаквана стойност и следователно няма с какво да бъде сравнен. Новите изходни стойности се натрупват във времевата серия като предсказание за бъдещите елементи. Прозорецът продължава да се придвижва с по един елемент напред на итерация. Няма ограничение за продължителността на процеса на предсказване. Важно е да се отбележи, че колкото по-напред в бъдещето придвижваме предсказанието, толкова по-неточно става то. Предсказаните стойности са най-акуратни в областта, където прозорецът все още съдържа стойности, които са част от оригиналната времева серия (т. е. реални).

За тестване на точността на предсказване на невронната мрежа може да се направи следният експеримент. Вземаме времева серия от предишен период от време (например преди една година). Когато обучим мрежата, започваме да предсказваме нови стойности. Тъй като използваме стари данни, можем да проверим точността на предсказаните стойности, като ги сравним с известните такива да периода, за който предсказваме (примерно текущата година).

На **Фигура 8** е показана графика на предсказаните и реалните стойности на дадени тестови данни. Данните са фиктивни и служат единствено за целите на теста. Умишлено е зададен елементарен модел на повтарящо се поведение (под формата на леки подскоци в графиката), за да се проследи как невронната мрежа следва поведението на времевата серия. Периодът на обучение започва от началото на графиката и свършва в момента на разделянето на двете графики (отбелязан с червена окръжност на фигурата). Тъй като данните за обучение са твърде малко количество, предсказаните стойности не са съвсем точни и на това се дължи разделянето на двете графики. Въпреки това, ясно се вижда, че мрежата



Фигура 8. *Тестване на точността на предсказване на времеви серии от невронна мрежа с обратно разпространение на грешката*

добре повтаря поведението на оригиналната графика. Тестът от фигурата е извършен върху невронната мрежа с обратно разпространение, която е разработена за текущото приложение за предсказване на мелодии. Това показва, че такава невронна мрежа има голяма гъвкавост, тъй като е приложима за решаване на проблеми от в различни области.

За описания принцип на предсказване можем да направим следните изводи: Предсказаните стойности не следват някаква обща структурна схема. Те се генерират една по една, като невронната мрежа извършва действията без да съхранява предишните си състояния. Точността на предсказаните стойности намалява с отдалечаване от реалните такива. Колкото повече реални данни имаме, толкова по-точно ще бъде предсказанието.

Значение имат следните предпоставки:

- Големина на прозореца – Колкото по-голям е прозорецът, толкова по-гъвкаво ще бъде предсказанието. Невронната мрежа ще се обучава върху по-разнообразен набор от поведения, като ще има по-обширен поглед над тях. Проблем: при увеличаване ширината на прозореца се увеличава броят на елементите на входния вектор на невронната мрежа, което води до намаляване на бързодействието ѝ. Най-добре е на принципа на пробата и грешката да се определи най-подходящата големина на прозореца, за целите на решаването на конкретния проблем.

- Дължина на времевата серия – Колкото повече елементи съдържа серията, толкова по-точно ще бъде предсказанието. Невронните мрежи се нуждаят от много на брой итерации на обучение, за да настроят успешно теглата на синапсите си. Има случаи, в които е невъзможно да увеличим времевата серия с реални данни (нямаме достатъчно информация за разглеждания обект). В такива случаи, за да обучим мрежата, можем да свържем края на серията с нейното начало. По този начин, когато прозорецът стигне до края на историческата серия, невронната мрежа няма да влезе в етап на предсказване, а ще продължи обучението като изходен елемент ще бъде първият елемент от серията. Това повтаряне на една и съща серия може да се извърши многократно, докато не се уверим, че теглата са настроени правилно. Такъв подход на обучение се използва за невронната мрежа в приложението за предсказване на музика, разгледано в **3-та глава.**
- Брой скрити слоеве в невронната мрежа и брой на невроните в тях – Броят на скритите слоеве е решаващ за правилната работа на невронната мрежа. Прекалено малко на брой слоеве или неврони в тях могат да намалят значително мощността на мрежата, а с това и точността на предсказването. Ако обаче броят им е прекалено голям, това може сериозно да натовари системата, което ще доведе до значително намаляване на бързодействието ѝ. Определянето на броя на скритите слоеве и броя на невроните в тях отново се определя на принципа на пробата и грешката. Тази настройка е специфична за различните проблеми, които ще се решават.

Данните, предсказани от невронна мрежа, която е настроена правилно, в повечето случаи наподобяват реалните бъдещи резултати. На това се дължи и голямата популярност на невронните мрежи в сферата на финансите, статистиката, метеорологията и др. Въпреки това, тези данни не винаги отговарят на истината, тъй като не отчитат множеството произволни фактори, които влияят на разглежданите обекти в реална среда. Предсказването чрез невронни мрежи може да се използва като статистически ориентир за времевите серии в бъдещ период. Неговото приложение, върху което ще наблегнем, обаче, е предсказването на мелодии и как може невронната мрежа да прихване поведението на мелодия като най-обикновена многомерна времева серия.

2.2. Цифрово съхранение и обработка на музика

Историята на цифровите музикални записи започва с изобретяването на импулсно-кодовата модулация от английския учен Алек Рийвс през 1937 г.[8] Тази модулация на аналоговите сигнали първоначално е била предназначена за

използване в телекомуникациите. В последствие тя става приложима за комерсиалното разпространение и запис на звукова информация. Първите комерсиални записи са направени от Японската телекомуникационна корпорация (NHK) и Nippon Columbia, известна също като Denon. Днес дигиталното представяне на музиката е на много високо ниво и е неделима част от ежедневието на голяма част от хората на Земята.

Формати и компоненти на потоци за запис на музика

Съществуват две основни направления в представянето на музика в дигитални формати. **Първото направление** представя музиката като дигитализиран звуков сигнал. При форматите, следващи това направление, музиката представлява реален записан или генериран звук. При запис на реално изпълнение, резултатът от проиграването на данните в записа се доближава до оригинала, в зависимост от качеството на записващото устройство, разделителната способност на формата, както и други фактори.

За запис на звука се използва устройство - микрофон. Микрофонът преобразува звуковите вълни в аналогов електрически сигнал. За да се създаде цифров запис е необходимо аналоговият сигнал от микрофона да премине през аналогово-цифров преобразувател (АЦП). Получените цифрови данни се манипулират от компютърна система. В зависимост от това, какъв формат е избран за съхранение на музиката, се избира определен алгоритъм за преобразуване на чистите цифрови данни в определените от желания формат такива. Устройството или компютърната програма, която извършва преобразуването на потока от първични цифрови данни в желания формат, се нарича **звуков кодек** (от англ. Codec, coder-decoder).

Най-известните цифрови формати от това направление за запис и съхранение на музика са:

- **WAV** (WAVE, от англ. Вълна)[9] – Това е формат, разработен от Microsoft и IBM през 1991 г. Той е представител на т. нар. RIFF (Resource Interchange File Format) файлови формати. Служи за съхранение на потоци от битове на персонални компютри (PC). Този формат съхранява дигиталния звуков запис в чист и некомпресиран вид. Благодарение на това, няма промяна в качеството на звука (т. нар. CD качество). Недостатък на този файлов формат е големият размер на файловете – до около 10 мегабайта за минута[10]. WAV файловете могат също да съдържат данни, кодирани с различни кодеци, за да се намали големината на файла (например GSM или MP3 кодеци). Този формат рядко се използва за трансфер на музика в интернет пространството, поради големината на файловете. Благодарение на това, че има опростена структура и не внася загуби в качеството на

звука, този формат широко се използва при звукозапис и звукообработка.

- **MP3** (MPEG Audio Layer 3)[11] – Това е патентован звуков формат, който използва компресия със загуби. Той е най-популярният формат за съхранение и стрийминг на потребителски звукови данни. Освен това е най-поддържаният формат за дигитални устройства като CD плейъри, DVD плейъри и др. Това е форматът, в който се съхраняват по-голямата част от музикалните продукти в интернет. MP3 е аудио формат, създаден от групата MPEG (Motion Pictures Expert Group - група на кинематографичните експерти) към институцията Фраунхофер и е пуснат за експлоатация през 1995 г. Алгоритъмът в основата на формата използва метод за компресиране на данните със загуба. Той използва като преимущество границите на чуваемост на човешкото ухо. Този алгоритъм използва принципът на аудиторно маскиране, според който тон може да бъде заглушен от друг тон с по-ниска честота. Алгоритъмът за кодиране на звука в MP3 формат използва психоакустични принципи за компресиране на данните, така че загубите в качеството да останат незабелязани от слушателя. Качеството на компресирания звук може да варира, в зависимост от побитовата скорост (bit rate), с която е кодиран потокът от данни.
- **OGG** - Свободен контейнерен формат с отворен код, който поддържа множество кодеци, най-популярен от които е звуковият кодек Vorbis. Vorbis предлага по-добра компресия от MP3, но е по-малко популярен. Идеята на разработчиците на OGG формата от Xiph.Org Foundation е била да създадат висококачествен отворен мултимедиен формат, който не е ограничен от никакви софтуерни патенти.
- **WMA** (Windows Media Audio)[12] - Това е формат за компресия на звук, разработен от Microsoft. Името е събирателно за файловия формат, както и за различните кодеци. Форматът е комерсиална разработка и е част от колекцията Windows Media мултимедийни формати. WMA се състои от четири кодека. Оригиналният WMA се явява конкуренция на MP3 и RealAudio. Днес е сред най-популярните формати, на едно с MP3 и MPEG-4 AAC.
- **FLAC** (Free Lossless Audio Codec)[13] – Формат и кодек за беззагубно компресиране на звукови потоци. Звукът се компресира, за да се намали големината на файловете, но без промяна в качеството. Големината на записа може да се намали до 50% - 60% от оригиналната. FLAC форматът е с отворен безвъзмезден лиценз и може да се използва свободно. Този формат не се поддържа масово от портативни звукови устройства и аудио системи, за разлика от MP3.

Въпреки това, той е най-разпространеният формат с беззагубна компресия, поддържан от множество хардуерно устройства.

Второто направление за представяне на музика в дигитално състояние се базира на напълно различен принцип от гореописаното. Докато първото направление разглежда музиката като обикновен звук, това разглежда музиката по-скоро като последователност от тонове. Тук форматите на данните представляват набори от инструкции към компютърната система, които описват музикалните характеристики на тоновете, вместо да описват поведението на звуковите вълни. Съдържанието на файл в такъв формат има смисъла на нотен текст.

Формати, които следват това направление, рядко се използват за слушане на музика. Те служат по-скоро за съхранение на лесно разчетими музикални композиции. Най-често се използват от музиканти, композитори и хора, изявяващи интерес към структурата на музикалното съдържание.

Предимството на тези формати е в това, че представят музиката в лесно разчетим вид. Файловете в този формат принципно са много по-малки, от колкото файловете, съдържащи реални аудио данни.

Недостатък се явява това, че музиката, записана по-този начин, не представлява реално записан звук, а се генерира на момента от прочитащото устройство. Това ограничава формата от възможност за представяне, примерно, на вокални изпълнения.

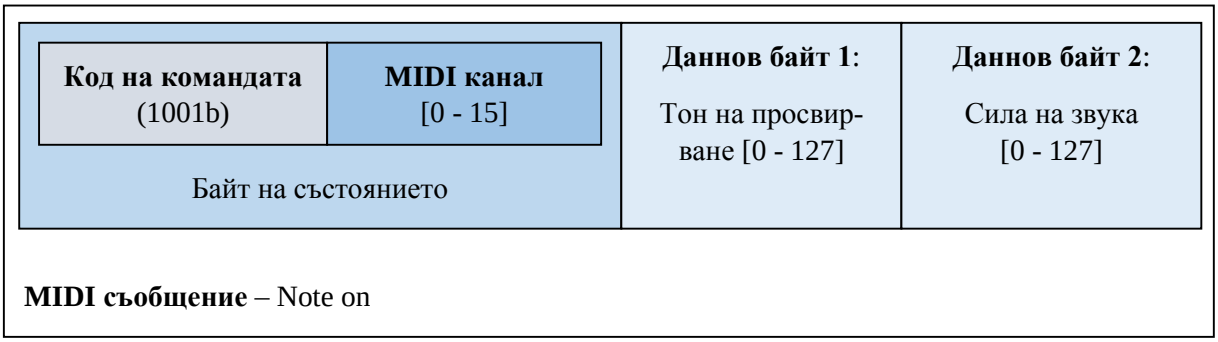
Благодарение на простотата на представяне на музикалната информация във форматите от това направление, те се явяват удобно средство за изпробване на системата за предсказване на мелодии. За предсказването на мелодия е необходимо първоначално да се извлекат характеристиките на нотите от оригиналната мелодия. Това се явява много сложна задача при файлове като WAV, MP3 и др., тъй като е необходимо прилагането на сложни алгоритми за тяхното определяне. Освен това, резултатите от работата на тези алгоритми е възможно да не бъдат коректни, поради шумовете, които се съхраняват в тези формати, заедно със значещата информация. При файлове от второто направление задачата се свежда до прочитане и структуриране на данните, записани във файла. Това е причината в приложението, разработено за дипломната работа, да се използва файлов формат от второто направление и по-точно **MIDI форматът**, който е най-разпространеният такъв.

MIDI стандарт

MIDI (Musical Instrument Digital Interface) е обширен стандарт, описващ протокол за комуникация между устройства, дигитален интерфейс, файлов формат и др. Идеята, заложена в стандарта, е да се създаде общоприет метод за свързване на компютърните системи с музикални инструменти и устройства.

Общото между всички музикални инструменти е, че под действието на човек инструментът издава определен звук. Този звук има определени характеристики (тон, продължителност, сила и др.). Копирайки на тези характеристики, подобен звук може да се изпълни от различен инструмент. Например, ако запишем характеристиките на звуците на една мелодия, изсвирена на пияно, след това можем да изпълним същата мелодия, но на китара като следваме въпросните характеристики.

В този текст основно ще разгледаме структурата на MIDI протокола, тъй като той играе ключова роля в приложението за предсказване на мелодии. Този протокол служи за комуникация между музикални инструменти и компютри, но също така описва и структурата на MIDI файловете. Основните градивни единици на протокола са **MIDI съобщенията** (MIDI message). На **Фигура 9** е изобразена



Фигура 9. *Структура на примерно MIDI съобщение*

схема на такова съобщение.

Едно такова съобщение представлява низ от байтове[14]. В MIDI има множество дефинирани съобщения. Някои от тях са изградени само от 1 байт. Други съдържат 2 байта, за разлика от трети, които са изградени от 3 байта. Има дори такова съобщение, което има неограничен брой байтове. Това което е общо между всички тези съобщения е първия байт от низа - **байта на състоянието** (Status byte). Този байт е единствения в съобщението, който има стойност 1 на най-старшия си бит. Всички останали байтове имат 0 на най-старшата позиция.

Байтът на състоянието е разделен на две части по 4 бита. Лявата страна съхранява кода на **MIDI командата** (MIDI command), която ще се изпълнява. MIDI командата представлява инструкция към изпълняващото устройство за изпълнение на конкретна процедура. Двете най-често срещане команди са:

- **Note on** (включена нота) – Оповестява началото на изпълнение на нота. Може да се интерпретира като натискане на клавиш от пианото. Кодът на командата е 1001b.
- **Note off** (изключена нота) – Оповестява край на изпълнение на нота. Може да се интерпретира като отпускане на клавиш от пианото. Кодът на командата е 1000b.

Дясната половина от байта на състоянието съхранява **MIDI канала** (MIDI channel), на който ще се изпълни командата. Всяко MIDI устройство или програма съдържа 16 на брой канала. Всеки канал може да се разгледа като независимо от останалите канали пространство за изпълнение на MIDI команди. Най-лесно можем да си представим MIDI каналите като отделни инструменти в песен. Те звучат едновременно по време на изпълнение на песента. Всеки канал е отделна опашка, на която се изпълняват командите, предназначени за него.

Както беше уточнено, има голям брой разнообразни MIDI съобщения дефинирани в стандарта. Ще се спрем само на **Note on** и **Note off**, тъй като те са единствените MIDI съобщения, които се използват в приложението за предсказване на мелодии.

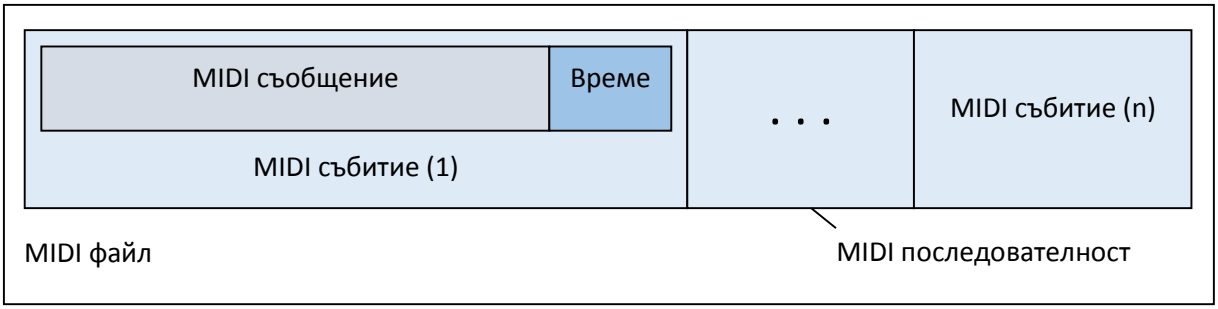
На **Фигура 9** е изобразена структурата на Note on съобщение. Note off има подобна структура на Note on съобщението, с изключение на байта на състоянието. И двете съобщения имат по два даннови байта след статус байта. Първият даннов байт съхранява тона на нотата (Note Number), който ще бъде изсвирена (или спряна). Възможните тонове, които могат да се изсвирят в MIDI формата са 128 на брой, започвайки от До в нулева октава (код 0) и стигайки до Сол в 10-та октава (код 127)[16].

Вторият даннов байт служи за обозначаване на силата на звука на нотата (Velocity). Този байт се интерпретира различно от различните устройства. Възможно е дори да не се вземе под внимание. При Note off съобщението този байт определя колко бързо ще бъде отпусната нотата.

MIDI поддържа два основни режима на работа – **режим в реално време и режим на възпроизвеждане.**

Режимът в реално време е предназначен основно за живи изпълнения и стрийминг. Пример за това е, когато музикант свири на клавишите на електронно пиано. В този случай дигиталният интерфейс приема данните, получавани от клавиатурата и ги предава като MIDI съобщения към **синтезатора** (Synthesizer). Синтезаторът е устройство за генериране на звуци[15]. То приема MIDI съобщения на входа си и извежда звукова информация на изхода. Синтезаторът обикновено има заложена библиотека с готови звуци, които отговарят на различни инструменти. Според информацията, описана във входните данни, синтезаторът комбинира по такъв начин звуци от своята библиотека, така че да се повтори мелодията заложена от пианиста. При режима в реално време се обработват единствено MIDI съобщения. Тук няма времеви събития, тъй като всичко се случва в реално време.

За разлика от режима в реално време, при **режима на възпроизвеждане** музикалната информация е предварително записана в т. нар. MIDI последователности (Sequences). MIDI последователностите се записват във файлове с разширение "**.mid**" (ще ги наричаме **MIDI файлове**). На **Фигура 10** е изобразена опростена схема на MIDI файл. При предходния режим нямаше нужда



Фигура 10. *Структура на MIDI файл*

да влагаме информация за времето на изпълнение на MIDI съобщенията, тъй като всяко такова се изпълняваше в реално време. При режима на възпроизвеждане, обаче, е важно да уточним продължителността на всяка нота и времевите характеристики на всяко едно друго съобщение, което ще се възпроизвежда. За целта всяко MIDI съобщение се капсулира в **MIDI събитие** (MIDI event). MIDI събитията съдържат два компонента – MIDI съобщение и времева информация. Времето се представя в брой отброявания (Ticks). Стойността на времевата променлива означава броят на отброяванията от началото на MIDI последователността. Под отброявания се има предвид равномерни времеви интервали определени от темпото, заложено в MIDI файла, на принципа на метронома. Принципно, темпото, заедно с други глобални настройки, се задава в началото на последователността (файла), т. е. преди първото събитие.

MIDI последователностите се изпълняват от устройство или програма, наречено **секвенсер** (Sequencer). То приема MIDI събитията и в определените от времевите променливи моменти изпълнява приетите съобщения. В днешно време всяка популярна операционна система е снабдена със софтуерен секвенсер за MIDI. В Windows операционните системи, MIDI секвенсер е включен в пакета DirectX.

Темата за MIDI стандарта е изключително обширна и няма да бъде разглеждана в детайли, тъй като разработката за текущата дипломна работа използва само малка част от възможностите му. Няма да наблягаме на точното представяне на двоичните данни в MIDI файловете, тъй като има разработени програмни библиотеки за работа с тях, които оставят този проблем извън полезрението на програмиста. Пакетът от инструменти за работа с MIDI, използван за текущата реализация на приложението на езика Java, се нарича "**javax.sound.midi**"[15] и е разработена от Oracle.

Извличане на музикални данни от MIDI серии

В тази точка по-подробно ще разгледаме пакета "**javax.sound.midi**" и какви функционалности, необходими за реализирането на приложението за предсказване на мелодии, предоставя той. [15] Този пакет е част от обширния пакет "**javax.sound**", който предоставя приложно-програмен интерфейс (API) на Java

програмистите за работа със звукова информация. Пакетът за работа с MIDI представлява класова структура, която се доближава абстрактно до гореописаната структура на MIDI протокола. Някои от основните класове са:

- **MidiSystem** – Този клас дава достъп на програмиста до инсталираните на машината MIDI ресурси, включително устройства като синтезатори, секвенсери и MIDI входни и изходни портове.
- **MidiMessage** – Абстрактен клас, описващ първично (raw) MIDI съобщение. Неговите наследници класифицират съобщенията на кратки съобщения (ShortMessages), част от които са Note on и Note off, системни съобщения (SysexMessages) и мета съобщения (MetaMessages).
- **MidiEvent** – Описва MIDI събитие по начина, по който то се съхранява в MIDI последователност (файл). За разлика от чистите съобщения, обаче, времевата променлива в обектите от този тип означава броя на отброяванията от изпълнението на миналото събитие, а не от началото на последователността.
- **Sequence** – Описва MIDI последователност.
- **Track** – Това е независим поток от инструкции в рамките на една последователност. Това е допълнителна абстракция към модела на MIDI протокола и не се припокрива с MIDI каналите. В една писта (Track) може да има събития, предназначени за различни канали.
- **Sequencer** – Описва модел за работа със секвенсери. Има методи за пускане на последователност, спиране и др.

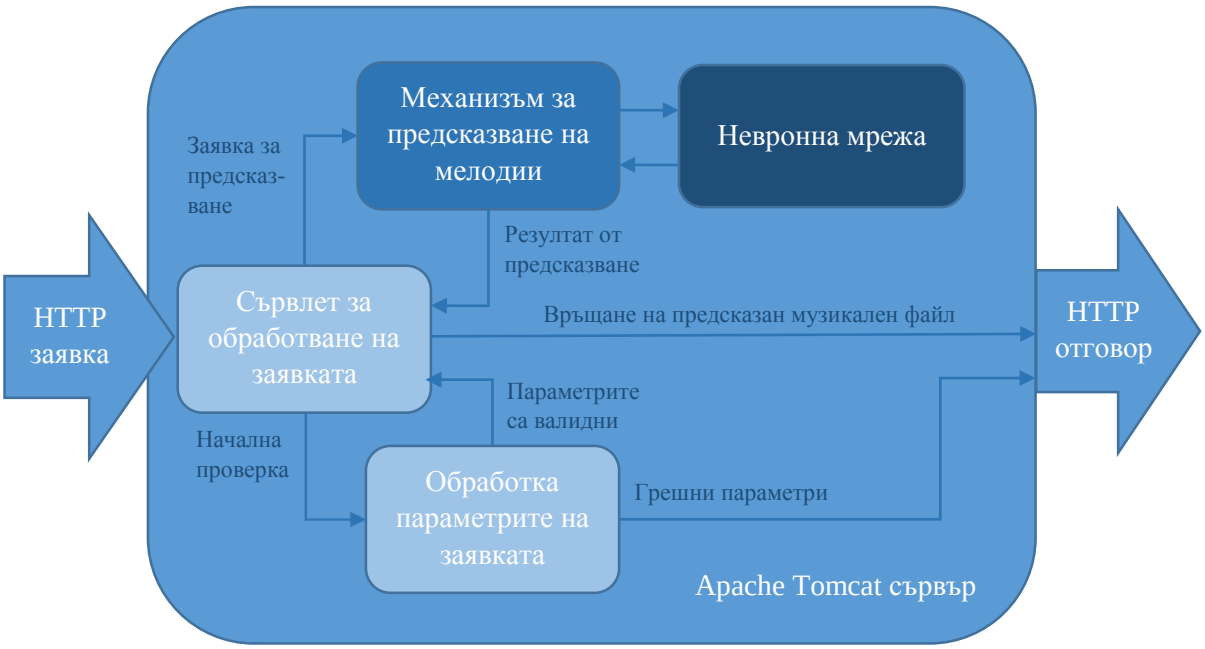
Пакетът за работа с MIDI е съставен от голям брой класове. Разгледаните по-горе са тези, които са съществени за разработването на текущото приложение.

Класът MidiSystem спомага за извличане на MIDI последователности от фалове в локалната файлова система. Благодарение на него програмистът не е необходимо да знае в детайли структурата на MIDI файловете, тъй като MidiSystem връща прочетените от файл данни като структуриран набор от обекти, инстанции на гореописаните класове.

3. Описание на програмното решение

3.1. Цялостна структура и концепция на приложението

Реализираното програмно решение за система за предсказване на мелодии е под формата на сървърно приложение, чиято връзка с външната среда е изградена на база на HTTP (Hypertext Transfer Protocol) протокола.



Фигура 11. *Опростена структура на сървърното приложение*

На **Фигура 11** е показана общата структура на приложението. То е разработено на програмния език **Java** в среда **Eclipse Juno**. За мрежови сървър е използван **Apache Tomcat v7.0**.

Интерфейсът на приложението е предназначен за ползване от програмисти. Необходимо е програмистът да структурира валидна HTTP заявка, съдържаща правилните параметри, изискванията за които ще бъдат разгледани по-късно в тази глава.

Цялото приложение е разделено на три модула (Java пакета). Те са следните:

- **Невронна мрежа** (пакет “neuralnetwork”) – В този пакет се съдържа невронната мрежа с обратно разпространение на грешката, която е ключовият компонент за предсказването на мелодии. Невронната мрежа е написана като обектно-ориентиран модел, който ще бъде разгледан детайлно в следващия раздел.
- **Система за предсказване и обработка на MIDI данни** (пакет “midiparser”) – Тук са включени класовете за работа с MIDI интерфейса, както и интерфейсът за преобразуване на музикалната

информация в информация, удобна за обработка от невронната мрежа. Този модул е разгледан в раздел **3.3.**

- **Система за обработка на HTTP заявки** (пакет “musicprediction”) – Тук е включен основният сървлет за обработване на HTTP заявките, заедно с няколко помощни класа, включващи проверка за валидност и обработка на параметрите на заявката, символни низове и клас за връзка с база от данни. Този пакет е разгледан в раздел **3.4.**

Системата изпълнява следните действия. Клиентът изпраща HTTP заявка към приложението. Заявката съдържа като параметри MIDI файл или избрана примерна песен от база от данни, заедно с настройки за невронната мрежа. Като резултат се получава отговор с прикачен MIDI файл, съдържащ предсказаната мелодия.

Заявката първо се приема от главния сървлет. Той извиква механизъм за валидация и обработка на параметрите на заявката. Ако параметрите не отговарят на изискванията, към клиента се връща отговор с текст на грешката. В противен случай, сървлетът прехвърля обработените данни към системата за предсказване. Там данните се обработват до достигане на формат, удобен за използване от невронната мрежа. Тогава те се прехвърлят към мрежата и тя започва своето обучение. След като приключи с обучението, невронната мрежа влиза в режим на предсказване. Приключвайки, мрежата връща предсказаните данни на системата за предсказване. Там новите данни се преобразуват в MIDI информация и се предават към сървлета. Той от своя страна съзнава нов файл и го изпраща на клиента като отговор. Ако в процеса на обработка някъде възникне грешка или изключение, на клиента се връща отговор, който съдържа обяснителен текст за възникналия проблем.

Важно е да се отбележи, че системата работи с опростени MIDI файлове. Това ще рече, че те трябва да съдържат единствено съобщения от тип Note on и Note off и да имат само една писта (Track). Това е така, защото системата може да предсказва само прости мелодии. Такива са мелодии, изсвирени от един инструмент и то само с единични тонове. Не се приемат акорди и многозвучия, защото тяхната обработка би усложнила процеса на предсказване.

3.2. Реализация на невронната мрежа

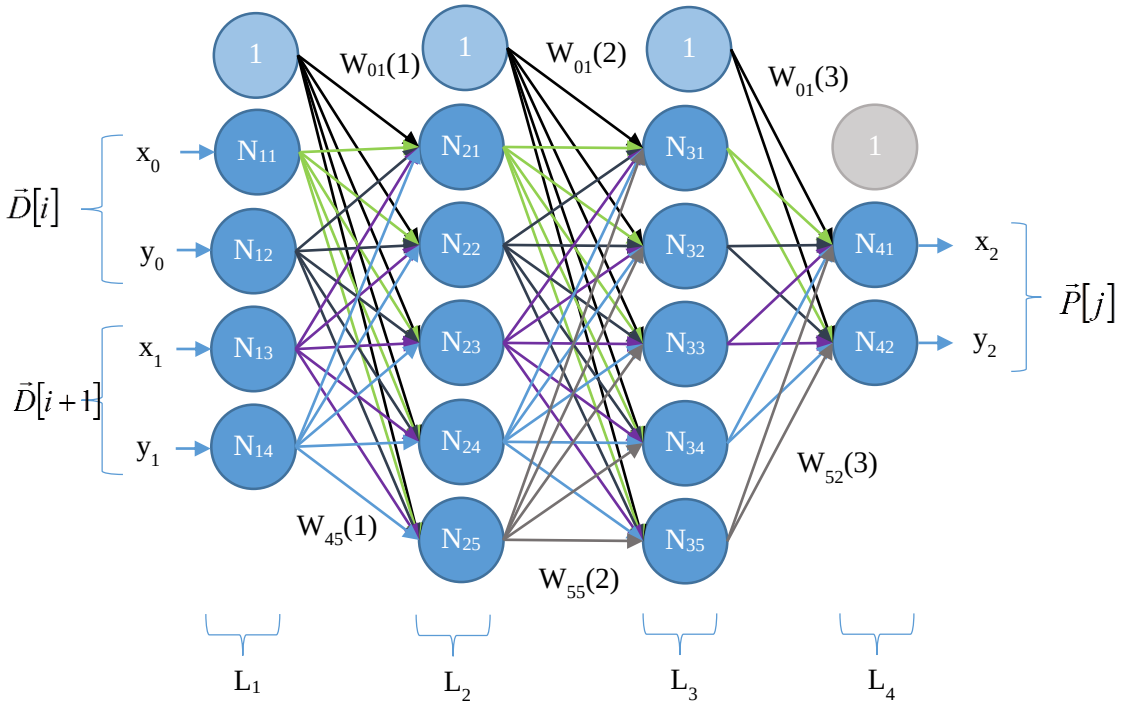
Основната задача при решаването на проблема за предсказване на мелодии е разработването на обучаващия се механизъм, който има способността да изгради модел на входната мелодия. За конкретната реализация такъв механизъм се явява **невронна мрежа.** Разработената невронна мрежа е от типа невронна мрежа с обратно разпространение на грешката (на англ. Backpropagation Neural Network). Тя има променлив брой входове, изходи, слоеве и неврони в тях. Тези променливи

се задават при създаване на мрежата. Програмният код на разработената мрежа можа да бъде разгледан в **Приложение 1**.

Предимството на избрания модел на обектно-ориентирано програмиране е, че невронната мрежа е добре разбираема и лесна за надграждане и промяна. Мрежата има ясно разграничени модулни елементи, които съвпадат с абстрактната представа за невронна мрежа.

Недостатъците на този подход са свързани основно с бързодействието на алгоритъма. Както беше обяснено в **1.4**, обектно-ориентираното програмиране внася допълнително забавяне в изпълнението на невронната мрежа.

Обща структура



Фигура 12. *Схема на разработената невронна мрежа с примерно зададени характеристики.*

На **Фигура 12** е дадена схемата на невронната мрежа, за която са в сила следните настройки:

- На мрежата е зададена **размерност** 2. Това означава, че всеки даннов вектор, който ще бъде подаван на входа на мрежата, или очакван на изхода ѝ ще бъде с дължина 2 елемента - $\vec{D}[i] = [x_i, y_i]$.
- Мрежата има 2 **скрити слоя**. Смисълът на слоевете в невронните мрежи беше описан в раздел **2.1**.
- Всеки от скритите слоеве има по 5 **неврона** и един спомагателен.
- Големината на прозореца е 2. Следователно, при всяка итерация на невронната мрежа, на входа ѝ се подават по два даннови вектора.

Както беше описано в точка **2.1.3**, в началото на всеки слой добавяме по един спомагателен елемент с постоянна стойност 1. Този елемент винаги е 0-вият и за това останалите неврони в слоя започват своята номерация от 1. Нулевият невронна няма връзка с предходния слой, а само с невроните от следващия. Забелязва се, че спомагателен елемент има и в изходния слой. Това се налага, поради изискванията за съвместимост с останалите слоеве. Невронната мрежа не разграничава слоевете на входни, скрити или изходни, а просто следи тяхната позиция.

Невронната мрежа се управлява от класа **PredictionNeuralNetwork**. Той служи за координиране комуникацията между слоевете. Чрез него се управляват настройките на невронната мрежа заедно с данните за вход и изход. Също така, този клас осигурява връзката на мрежата с външния свят.

Всеки слой L_k (където k е поредният номер на слоя) е инстанция на класа **Layer**. Слоят контролира невроните, които са част от него, и тяхната връзка с невроните от предходния и следващия слой. Както беше споменато по-рано, всеки слой има по един спомагателен неврон, който има постоянна стойност 1. Този неврон се обработва по специален начин от слоя.

Невроните N_{ki} (където k е номерът на слоя, в който се намира неврона, а i е поредният номер на този неврон) са инстанции на класа **Neuron**. Всеки неврон има стойност, която се изчислява на всяка итерация (освен тази на спомагателните неврони, тъй като винаги е 1). Невроните също имат стойност на грешката, която има подобно значение като стойността на неврона, но се изчислява при каскадното връщане (обратното разпространение на грешката).

Между два неврона от съседни слоеве съществува връзка. Всяка връзка е инстанция на класа **Link**. Връзката има тегло, входен и изходен неврон. Смисълът на теглото беше описан в раздел **2.1.** Предаването на данните става в посока от входния към изходния неврон. Изчисляването на грешката при каскадното връщане става по посока от изходния към входния неврон. Връзките се отбелязват с $W_{ij}(k)$, където i е поредният номер на входния неврон в слоя му, j е поредният номер на изходния неврон в неговия слой, а k е номерът на слоя, от който е част входният неврон. Всяка връзка има способността да коригира теглата си в зависимост от стойността на входния неврон, грешката на изходния неврон и избраната скаларна променлива (η).

При инициализация на невронната мрежа се задава вектор от данни $\vec{D}$. Елементите на този вектор от своя страна също са вектори. Всеки елемент на $\vec{D}$ има толкова на брой елементи, колкото е зададената размерност на мрежата. В случая на тази, която е дадена на **Фигура 12**, елементите са 2 - $\vec{D}[i] = [x_i, y_i]$. Тъй като зададената размерност там също е 2, то следва, че на входа на мрежата ще имаме вектор, съставен от $\vec{D}[i]$ и $\vec{D}[i+1]$, т. е. $\vec{I} = [x_i, y_i, x_{i+1}, y_{i+1}]$. Та изхода на мрежата ще очакваме вектор, с брой елементи, колкото е размерността на мрежата, т. е. 2. Ще отбележим този вектор с $\vec{O} = [x_{i+2}, y_{i+2}]$.

Когато мрежата се намира в режим на обучение, на входа и се предават последователно елементите на $\vec{D}$. На изхода получаваме вектор $\vec{O}$, който се сравнява със следващия по ред елемент на $\vec{D}$. Принципът на обучение на мрежата е същият, който беше описан в точка **2.1.3**.

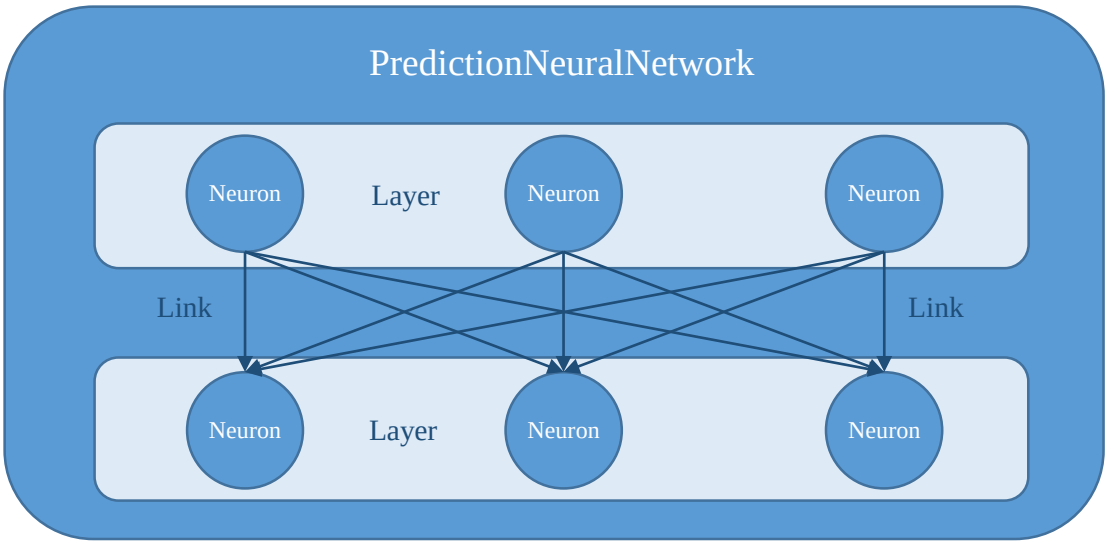
Когато мрежата влезе в режим на предсказване, изходните вектори $\vec{O}$ вече не се сравняват, а се трупат в нов вектор с предсказаните данни $\vec{P}$. В процеса на предсказване невронната мрежа може да достигне до края на $\vec{D}$. Когато това се случи, на входа на мрежата започват последователно да се включват предсказаните вектори от $\vec{P}$. Така входът на невронната мрежа става:

$\vec{I} = \vec{P}[i] \cup \vec{P}[i+1] = [x_i, y_i] \cup [x_{i+1}, y_{i+1}] = [x_i, y_i, x_{i+1}, y_{i+1}]$ и $\vec{P}[i+2] = \vec{O} = [x_{i+2}, y_{i+2}]$

Активационната функция на невроните в мрежата е същата като тази използвана в примерния алгоритъм от раздел **2.1.3**:

$$\varphi = \frac{1}{1 + e^{-net_j}}$$

От гледна точка на класовата структура на програмното решение, схемата на невронната мрежа е показана на **Фигура 13**.



Фигура 13. *Схема на невронната врежа от гледна точка на класовата структура. Елементите на схемата са отбелязани с имената на класовете, към които принадлежат.*

Описание на класовете

Java класовете, описващи невронната мрежа с обратно разпространение на грешката са обединени в Java пакета “neuralnetwork”. Те са:

Клас “PredictionNeuralNetwork”

Това е класът, който обгръща невронната мрежа и е единственият публичен клас в пакета. Той има четири частни полета:

- **Слоеве (layers)** – Това поле е колекция от тип ArrayList, съдържаща обекти от клас Layer. В него се съдържат всички слоеве в мрежата. Това включва входния, скритите и изходния слой.
- **Данни (dataSets)** – Колекция от тип ArrayList, елементите на която са масиви от тип double. Тук се съхранява данновият вектор $\vec{D}$, а когато мрежата влезе в режим на предсказване, в него се трупат и предсказаните елементи. Следователно, в края на процеса на предсказване в това поле се съдържа $\vec{D} \bigcup \vec{P}$.
- **Големина на прозореца (windowSize)** – Това поле е от целочислен тип и съхранява големината на прозореца, описан в точка **2.1.4**.
- **Индекс на началото на предсказаните данни в колекцията от данни (predictedValuesStartIndex)** – Съдържа поредният номер на първия предсказан елемент от колекцията **dataSets**. Обикновено съвпада с броя на елементите в $\vec{D}$.

Спецификация на методите на класа е дадена в **Таблица 1**. Действието на по-значимите методи са описани след нея.

Таблица 1. *Детайли за методите на класа “PredictionNeuralNetwork”*

Име на метод	**Достъп**	**Параметри**	**Тип на върнатата стойност**
PredictionNeuralNetwork (конструктор)	публичен	**dimensionsCount** – цяло число, **windowSize** – цяло число, **dataSets** – колекция ArrayList с елементи - масиви от тип double, **neuronsCountPerHiddenLayer** – масив от цели числа	N/A
getLayers (гетер на layers)	публичен	няма	Колекция ArrayList с елементи от клас Layer,

setLayers (сетер на layers)	частен	**layers** - колекция ArrayList с елементи от клас Layer,	void
getDataSets (гетер на dataSets)	публичен	няма	Колекция ArrayList с елементи - масиви от тип double
setDataSets (сетер на dataSets)	публичен	**dataSets** – колекция ArrayList с елементи - масиви от тип double	void
getWindowSize (гетер на windowSize)	публичен	няма	Цяло число
setWindowSize (сетер на windowSize)	публичен	**windowSize** – цяло число	void
getPredictedValuesStartIndex (гетер на predictedValuesStartIndex)	публичен	няма	Цяло число
setPredictedValuesStartIndex (сетер на predictedValuesStartIndex)	частен	**predictedValuesStartIndex** – цяло число	void
getPredictedValues	публичен	няма	Колекция ArrayList с елементи - масиви от тип double
addLayer	частен	**layer** – обект от клас Layer	void
processLayer	частен	**layer** – обект от клас Layer, **expectedValues** – масив от тип double	void
predict	частен	няма	void
epoch	частен	**dataSetIndex** – цяло число	void
resetNeurons	частен	няма	void
start	публичен	**repetitions** – цяло число, **predictionsCount** – цяло число	void
toString	публичен	няма	Символен низ

- **Конструктор на класа** – Инициализира невронната мрежа. Параметърът **dimensionsCount** определя размерността на мрежата. Параметърът **windowSize** задава големината на прозореца. Чрез параметърът **dataSets** се задават данните за обучение на мрежата. Параметърът **neuronsCountPerHiddenLayer** задава броят на скритите слоеве и броят на невроните в тях. Всеки елемент в този параметър отговаря за единичен слой, а стойността на елемента означава броя на невроните в съответния слой. Така, например, ако зададен на този параметър като аргумент масива [5,6,7], в невронната мрежа ще бъдат създадени три скрити слоя като първият ще има 5 неврона, вторият – 6, а третият 7.
- **getPredictedValues** – Връща предсказаните елементи от мрежата. Ако мрежата е приключила с предсказването, този метод ще върне вектора $\vec{P}$. В противен случай ще върне празна колекция или, ако в dataSets няма елементи, null.
- **addLayer** – Метод за частно ползване, който добавя слой в колекцията layers.
- **processLayer** – Този **рекурсивен** метод се използва само в процеса на обучение на мрежата и служи за изчисляване на стойностите на невроните и грешките в тях. Освен това, по време на неговото изпълнение се извършва корекцията на теглата на връзките. Параметърът **layer** определя слоя, който се обработва в момента. В параметъра **expectedValues** се задават действителните стойности на елементите, които ще очакваме на изхода на мрежата. Методът изпълнява следните действия: Ако слоят, който ще се обработва, не е входния, се изчисляват стойностите на невроните в него. След това, ако слоят не е изходният, се извиква рекурсивно същият метод, но за следващия по ред слой. При рекурсивното връщане, се изчисляват грешките на невроните на база на грешките от предходните слоеве. Ако текущият слой е бил изходния, вместо да се влезе в ново рекурсивно извикване, директно се изчисляват грешките в невроните в слоя, сравнявайки техните стойности със стойностите на реалните данни, които се намират в параметъра expectedValues.
- **predict** – Методът извършва една итерация на предсказване. През тази итерация се изчисляват единствено стойностите на невроните. Няма каскадно връщане назад и изчисляване на грешки. За вход се използват последните елементи от колекцията dataSets, толкова на брой, колкото е зададено в windowSize. Изходните елементи, получени като резултат от итерацията, се добавят към края на dataSets.
- **epoch** – Методът извършва една итерация на обучение на мрежата (т. нар. **епоха**). Параметърът **dataSetIndex** определя индексът на

елемента от dataSets, който бъде в началото на прозореца. Този метод извиква метода processLayer.

- **resetNeurons** – Установява невроните от всички слоеве в начално състояние (0-ва стойност и 0-ва грешка на неврона).
- **start** – Стартира работата на невронната мрежа. Параметърът **repetitions** определя колко пъти ще се повтори обучението на мрежата върху зададените данни в dataSets (например 30000). Параметърът **predictionsCount** определя броя на предсказаните даннови елементи, който се изисква. Методът извършва обучението на мрежата. След завършване на зададения брой повторения, методът задава режим на предсказване, който приключва при достигане на зададения брой итерации за предсказване.

Клас "Layer"

Описва един слой от невронната мрежа. Класът е достъпен само в рамките на пакета. Съдържа следните частни полета:

- **Помощен неврон (auxiliaryNeuron)** – Съдържа обект от клас Neuron. Значението на помощния неврон беше описано в раздел **3.2.1.**
- **Неврони (neurons)** – Колекция от обекти от клас Neuron. Съдържа невроните от слоя.
- **Предходен слой (previousLayer)** – Поле от тип клас Layer. Референция към предишния поред слой.
- **Следващ слой (followingLayer)** – Поле от тип клас Layer. Референция към следващия поред слой.

Спецификация на методите на класа е дадена в **Таблица 2**. Действието на по-значимите методи са описани след нея.

Таблица 2. *Детайли за методите на класа "Layer"*

Име на метод	**Достъп**	**Параметри**	**Тип на върнатата стойност**
Layer (конструктор)	публичен	int **neuronsCount**	N/A
initNeurons	частен	int **neuronsCount**	void
linkAuxiliaryNeuronWithFollowingLayer	частен	няма	void
linkNeuronWithFollowingLayer	частен	int **neuronIndex**	void
linkAllNeuronsWithFollowingLayer	частен	няма	void

getAuxiliaryNeuron (гетер на auxiliaryNeuron)	публичен	няма	Neuron
setAuxiliaryNeuron (сетер на auxiliaryNeuron)	частен	Neuron **auxiliaryNeuron**	void
getNeurons (гетер на neurons)	публичен	няма	ArrayList<Neuron>
getNeuronValues	публичен	няма	double[]
setNeurons (сетер на neurons)	частен	ArrayList<Neuron> **neurons**	void
setNeuronValues	публичен	double[] **neuronValues**	void
getPreviousLayer (гетер на previousLayer)	публичен	няма	Layer
setPreviousLayer (сетер на previousLayer)	частен	Layer **previousLayer**	void
getFollowingLayer (гетер на followingLayer)	публичен	няма	Layer
setFollowingLayer (сетер на followingLayer)	публичен	Layer **followingLayer**	void
addNeuron	публичен	Neuron **neuron**	void
getNeuron	публичен	int **index**	Neuron
size	публичен	няма	int
calculateNeuronsValues	публичен	няма	void
calculateNeuronsErrors	публичен	няма	void
calculateNeuronsErrors	публичен	double[] **expectedValues**	void
resetNeurons	публичен	няма	void
toString	публичен	няма	String

- **Конструктор** – Инициализира слоя и невроните в него. Параметърът **neuronsCount** задава броя на невроните в слоя.
- **linkAuxiliaryNeuronWithFollowingLayer** – Създава връзки (обекти от клас Link) между спомагателния неврон на слоя и невроните от следващия слой.
- **linkNeuronWithFollowingLayer** – Свързва избран неврон от слоя с всички неврони на следващия слой.

- **linkAllNeuronsWithFollowingLayer** – Изпълнява предния метод за всички неврони в текущия слой.
- **calculateNeuronsValues** – Изчислява стойностите на всички неврони в слоя.
- **calculateNeuronsErrors** – Дефинирани са два такива метода. Първият (без параметри) се използва за изчисляване на грешките на невроните и корекция на теглата на връзките, за които невроните са входни, когато текущият слой е входен или скрит. Когато слоят е изходен, се извиква вторият метод, който приема като параметър масив от очакваните правилни стойности за изхода. Грешката се изчислява на базата на разликата между този параметър и получения изход.
- **resetNeurons** – Установява в 0 стойностите и грешките на всички неврони в слоя.

Клас "Neuron"

Описва един неврон от невронната мрежа. Класът е достъпен само в рамките на пакета. Съдържа следните частни полета:

- **Стойност (value)** – Целочислено поле. Съдържа стойността на неврона.
- **Грешка (error)** – Целочислено поле. Съдържа грешката на неврона.
- **Входни връзки (inputs)** – Колекция ArrayList от връзките (обекти от клас Link), които свързват неврона с невроните от предходния слой.
- **Изходни връзки (outputs)** – Колекция ArrayList от връзките (обекти от клас Link), които свързват неврона с невроните от следващия слой.
- **Слой (layer)** – Поле от тип клас Layer. Съдържа референция към слоя, към който принадлежи неврона.

Спецификация на методите на класа е дадена в **Таблица 3.** Действието на по-значимите методи са описани след нея.

Таблица 3. *Детайли за методите на класа "Neuron"*

Име на метод	**Достъп**	**Параметри**	**Тип на върнатата стойност**
Neuron (конструктор)	публичен	няма	N/A
Neuron (конструктор)	публичен	double **value**	N/A
getValue (гетер на value)	публичен	няма	double
setValue (сетер на value)	публичен	double **value**	void

getInputs (гетер на inputs)	публичен	няма	ArrayList<Link>
setInputs (сетер на inputs)	частен	ArrayList<Link> **inputs**	void
getOutputs (гетер на outputs)	публичен	няма	ArrayList<Link>
setOutputs (сетер на outputs)	частен	ArrayList<Link> **outputs**	void
getLayer (гетер на layer)	публичен	няма	Layer
setLayer (сетер на layer)	публичен	Layer **layer**	void
getError (гетер на error)	публичен	няма	double
setError (сетер на error)	публичен	double **error**	void
addInputLink	публичен	Link **link**	void
addOutputLink	публичен	Link **link**	void
linkInputWith	публичен	Neuron **otherNeuron**	void
linkOutputWith	публичен	Neuron **otherNeuron**	void
calculateValue	публичен	няма	void
calculateError	публичен	няма	void
calculateError	публичен	double **expectedValue**	void
correctOutputLinksWeights	частен	няма	void
reset	публичен	няма	void
toString	публичен	няма	String

- **Конструктори** – Този клас има два конструктора. Първият инициализира неврона със стойност равна на 0, а вторият задава стойност на неврона равна на стойността на параметъра **value**.
- **addInputLink** – Добавя обект от клас Link към колекцията inputs.
- **addOutputLink** – Добавя обект от клас Link към колекцията outputs.
- **linkInputWith** – Този метод създава нов обект от клас Link, за който текущият неврон се явява изходен, а входен е невронът, посочен в параметъра **otherNeuron**.
- **linkOutputWith** – Този метод създава нов обект от клас Link, за който текущият неврон се явява входен, а изходен е невронът, посочен в параметъра **otherNeuron**.

- **calculateValue** – Метод, който изчислява стойността на неврона, спрямо стойностите на невроните, с които е свързан текущия чрез връзките в inputs.
- **calculateError** – И в този клас има два метода за изчисляване на грешката. Първият метод (без параметри) изчислява грешката на невроните, които са част от скрит слой. Този метод извиква, преди да приключи, метода correctOutputLinksWeights. Вторият метод изчислява грешката на невроните, които са част от изходния слой, на базата на отклонението между стойността на неврона и очакваната стойност, която е подадена като параметър **expectedValue**.
- **correctOutputLinksWeights** – Задава на връзките от колекцията outputs да настроят своите тегла.
- **reset** – Метод, който установява неврона в изходно състояние. Нулират се стойността и грешката на неврона.

Клас "Link"

Описва една връзка между неврони в невронната мрежа. Класът е достъпен само в рамките на пакета. Съдържа следните частни полета:

- **Тегло на връзката (weight)** – Поле от тип double.
- **Входен неврон (input)** – Референция към обект от клас Neuron. Това е невронът, който се намира в началото на връзката (в лявата страна).
- **Изходен неврон (output)** – Референция към обект от клас Neuron. Това е невронът, който се намира в края на връзката (в дясната страна).
- **Скаларен коефициент (WEIGHT_PROPORTION)** – Това поле е константа от тип Double. То съдържа коефициентът η, описан в раздел **2.1.3**. Зададена му е стойност 0,35.

Спецификация на методите на класа е дадена в **Таблица 4**. Действието на по-значимите методи са описани след нея.

Таблица 4. *Детайли за методите на класа "Link"*

Име на метод	**Достъп**	**Параметри**	**Тип на върнатата стойност**
Link (конструктор)	публичен	Neuron **input**, Neuron **output**	N/A
getWeight (гетер на weight)	публичен	няма	double
setWeight (сетер на weight)	частен	double **weight**	void
getInput (гетер на input)	публичен	няма	Neuron

setInput (сетер на input)	публичен	Neuron **input**	void
getOutput (гетер на output)	публичен	няма	Neuron
setOutput (сетер на output)	публичен	Neuron **output**	void
correctWeight	публичен	няма	void
getForwardValue	публичен	няма	double
getBackwardError	публичен	няма	double

- **Конструктор** – Инициализира връзката. Приема два параметъра – входен (**input**) и изходен (**output**) неврон. Конструкторът задава произволна стойност на теглото на връзката в диапазона [-0,1;0,1].
- **correctWeight** – Изчислява и задава нова, подобрена стойност на теглото на връзката.
- **getForwardValue** – Изчислява и връща продукта от умножението на стойността на входния неврон и теглото на връзката.
- **getBackwardError** – Изчислява и връща продукта от умножението на грешката на изходния неврон и теглото на връзката.

Настройки и взаимодействие с външната среда

Както беше споменато в предния раздел, единственият клас, достъпен извън рамките на пакета "neuralnetwork" е **PredictionNeuralNetwork**. Следователно, всяка връзка на външната среда с невронната мрежа минава през този клас.

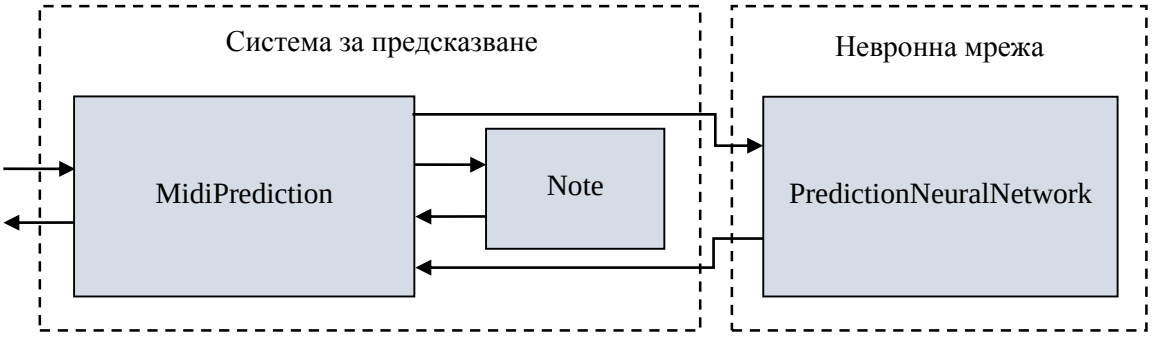
Първата стъпка е да се инициализира мрежата. Това става чрез конструктора на споменатия клас. В него задаваме размерността на мрежата. Дава се възможност също така за определяне броя на скритите слоеве и броя на невроните в тях. Трябва да се подхожда с внимание при задаването на тази настройка. Както беше споменато по-рано, неправилно избраният брой скрити слоеве може да доведе до затруднения в решаването на задачата от невронната мрежа. Големината на прозореца също трябва да се избере внимателно, според насоките, описани в **2.1.4**. Като резултат от работата на мрежата получаваме набор от предсказани данни, които се извличат с помощта на метода **getPredictedValues**.

Трябва да отбележим, че ако броят на зададените елементи в полето dataSets е по-малък от големината на прозореца windowSize, обучението на мрежата е невъзможно и поради това възниква изключение. Също важно уточнение е, че ако зададената размерност на мрежата е по-голяма от броя на елементите в някой от масивите в dataSets, мрежата отново ще върне грешка. Най-добър вариант е, ако всички масиви в dataSets имат по еднакъв брой елементи. Ако това не е спазено, то поне размерността на невронната мрежа да е зададена толкова, колкото е броят на елементите в най-малкия масив в dataSets.

3.3. Реализация на системата за предсказване

Системата за предсказване е модулът, който задава на невронната мрежа (описана в предния раздел) конкретната задача да предсказва мелодии. Тази система обхваща етапите на извличането на музикалните данни от MIDI последователност, предаването им към невронната мрежа, получаването на предсказаните данни и връщането им към външната среда, под формата на MIDI файл.

Тази система е описана в два класа – **MidiPrediction** и **Note**, които са дефинирани в пакета “midiparser”.



Фигура 14. *Опростена схема на системата за предсказване.*

Както се вижда на **Фигура 14**, връзката с външната среда се извършва от класа MidiPrediction. Класът Note служи за преобразуване на музикалните данни в такива, които са удобни за използване от невронната мрежа и обратно. Той е недостъпен за класовете, намиращи се извън пакета.

Принцип на действие

Фигура 15. *Принцип на действие на системата за предсказване на мелодии*

Последователността на действията, които извършва модулът за предсказване на мелодии, е изобразена на **Фигура 15.** Като **външна среда** ще възприемаме Java клас, който се намира извън пакета "midiparser" и който прави заявка за предсказване на мелодия към класа MidiPrediction. Предсказването на мелодия се извършва на 3 етапа, всеки от които започва със заявка от външната среда. Резултат се връща след третия етап. Етапите са изобразени на горната фигура и са както следва:

- **Инициализация** – Този етап започва със създаването на нов обект от клас MidiPrediction, като се извиква неговият **конструктор.** В заявката за инициализация на системата за предсказване се съдържа и MIDI последователността, която ще служи за основа на предсказването. Конструкторът преобразува музикалните данни от оригиналната MIDI мелодия в последователност от масиви от тип double, използвайки за тази цел класа Note. Това преобразуване е проследено на фигурата от стъпки **1** и **2.** Преобразуването е необходимо, за да бъдат представени музикалните данни във формат, удобен за обработване от невронната мрежа. Класът Note служи за преобразуване на една нота от мелодията, както в MIDI съобщение, така и в масив със стойности от тип double, който да бъде използван от невронната мрежа. Можем да кажем, че класът Note действа като свързващо звено между механизма за обработване на MIDI информация и невронната мрежа.
- **Предсказване** – В този етап се извършва същинската работа на системата. Оригиналната мелодия се предава към входа на невронната мрежа (стъпка **3**). Тя от своя страна се обучава на базата на тези данни и връща предсказаната мелодия, отново във вид на колекция от масиви от тип double (стъпка **4**).
- **Заявка за резултат** – В последния етап на действие от външната идва заявка за връщане на резултат от предсказването. В стъпка **5** първичните данни, върнати от мрежата, се преобразуват в MIDI съобщения, отново благодарение на класа Note. Тогава предсказаната мелодия се преобразува в MIDI последователност (стъпка **6**) и се предава към външната среда.

Описание на класовете

Java класовете, включени в пакета "midiparser" са следните (техният код може да бъде разгледан в **Приложение 2**):

<u>Клас "MidiPrediction"</u>

Този клас обгръща цялата система за предсказване на MIDI мелодии. Той съдържа следните полета:

- **Първоначална MIDI последователност (originalSequence)** – Това поле съдържа оригиналната MIDI последователност, подадена от заявката от външната следа при инициализация.
- **Данни от първоначалната мелодия (musicData)** – Това е колекция от масиви от тип double, която съхранява преобразуваната оригиналната мелодия във вид, удобен за обработка от невронната мрежа.
- **Данни за предсказаната мелодия (predictedMusicData)** – Колекция от масиви от тип double, в която се записва предсказаната от невронната мрежа мелодия.
- **Невронна мрежа (neuralNetwork)** – Това поле е от тип PredictionNeuralNetwork (този клас беше разгледан в предния раздел). Предназначен е за съхранение на невронната мрежа, която ще предсказва мелодиите.
- **Скрити слоеве (HIDDEN_LAYERS)** – Това константно поле е масив от целочислени стойности. То се използва при инициализацията на невронната мрежа. Стойността му се подава на конструктора на класа PredictionNeuralNetwork като аргумент на параметъра neuronsCountPerHiddenLayer.

Спецификация на методите на класа е дадена в **Таблица 5**. Действието на по-значимите методи са описани след нея.

Таблица 5. *Детайли за методите на класа "MidiPrediction"*

Име на метод	**Достъп**	**Параметри**	**Тип на върнатата стойност**
MidiPrediction (конструктор)	публичен	Sequence **originalSequence**, int **windowSize**	N/A
setOriginalSequence (сетер на originalSequence)	публичен	Sequence **originalSequence**	void
parseMusicData	частен	няма	void
predict	публичен	int **numberOfTones**, int **numberOfRepetitionsForTraining**	void
getPredictedSecuence	публичен	няма	Sequence
playSequence	публичен	Sequence **sequence**	void
saveSequence	публичен	Sequence **sequence**, String **filePath**	void

toString	публичен	няма	String

- **Конструктор** – Инициализира системата за предсказване на мелодии. Параметърът **originalSequence** съдържа оригиналната MIDI последователност, според която ще се прави предсказването. Параметърът **windowSize** задава големината на прозореца на невронната мрежа. При инициализацията оригиналната мелодия се конвертира в колекция от масиви от тип double. Също така, тук се създава и невронната мрежа.
- **parseMusicData** – Преобразува MIDI съобщенията от оригиналната мелодия в масиви със стойности от тип double, използвайки за целта класа Note.
- **predict** – Задава начало на процеса на предсказване, стартирайки невронната мрежа. Параметърът **numberOfTones** съдържа дължината на желаната предсказана мелодия в брой тонове. Параметърът **numberOfRepetitionsForTraining** задава колко пъти ще повтори обучението си невронната мрежа, върху зададената мелодия.
- **getPredictedSecuence** - Преобразува данните от полето predictedMusicData в MIDI последователност (обект от клас **Sequence**, от пакета javax.sount.midi) и я връща като резултат.
- **playSequence** – Статичен метод, който служи за прослушване на MIDI последователността **sequence**. За изпълняващо устройство на мелодията се използва зададеният по подразбиране MIDI секвенсер в операционната система.
- **saveSequence** – Статичен метод, който служи за запис на избраната MIDI последователност **sequence** във файл, с избрано име и път - **filePath**.

Клас "Note"

Този клас описва една нота от мелодия и се използва за нейното представяне в различни формати. Има следните полета:

- **Канал (channel)** – Целочислено поле. Съхранява канала, на който се изпълнява съответната нота. (Стойност в интервала: [0; 15]).
- **Тон (tone)** – Целочислено поле. Съхранява тона на нотата.
- **Сила (velocity)** – Целочислено поле. Съхранява силата на звука.
- **Продължителност (ticksLength)** – Целочислено поле. Съхранява продължителността на изпълнение на звука в брой отброявания.
- **Разделителна способност на MIDI последователността (sequenceResolution)** – Целочислено поле. Съхранява разделителната

способност, съответстваща на MIDI последователността, от която е част нотата. **Разделителната способност** се прилага върху метронома, който отброява моментите в MIDI. Тя определя темпото на мелодията. Резолюцията определя броя отброявания (ticks) за четвъртина нота.

- **Пропорция на разделителната способност (resolutionProportion)** – Поле от тип double. Съхранява частното на sequenceResolution и INNER_RESOLUTION. Използва се за ограничаване на размера на ticksLength в допустимите граници.
- **Вътрешна резолюция (INNER_RESOLUTION)** – Целочислено статично поле. Това е постоянната разделителна способност, която се използва при конвертирането на данните. Всяка нота вътрешно в класа са представя в тази резолюция, независимо от оригиналната, която се използва в нейната MIDI последователност. Има стойност 100.
- **Максимална стойност за канал (MAX_CHANNEL)** – Целочислено статично поле. Константа, задаваща ограничение за максимална стойност на channel. Има стойност 16.
- **Максимална стойност за байт (MAX_BYTE)** – Целочислено статично поле. Константа, задаваща ограничение за максимална стойност на полетата tone, velocity. Има стойност 128.
- **Максимална продължителност (MAX_TICK_LENGTH)** – Целочислено статично поле. Константа, съдържаща максималната стойност, която може да приеме ticksLength. Има стойност 400.

Спецификация на методите на класа е дадена в **Таблица 6.** Действието на по-значимите методи са описани след нея.

Таблица 6. *Детайли за методите на класа "Note"*

Име на метод	**Достъп**	**Параметри**	**Тип на върнатата стойност**
Note (конструктор)	публичен	ShortMessage **midiMessage**, int **ticksLength**, int **sequenceResolution**	N/A
Note (конструктор)	публичен	double[] **data**, int **sequenceResolution**	N/A
getMessage	публичен	няма	ShortMessage
getTicksLength (гетер на ticksLength)	публичен	няма	int
setTicksLength (сетер на ticksLength)	частен	int **ticksLength**	void

setTicksLength (сетер на ticksLength)	частен	double **value**	void
setChannel (сетер на channel)	частен	int **channel**	void
setChannel (сетер на channel)	частен	double **value**	void
setTone (сетер на tone)	частен	int **tone**	void
setTone (сетер на tone)	частен	double **value**	void
setVelocity (сетер на velocity)	частен	int **velocity**	void
setVelocity (сетер на velocity)	частен	double **value**	void
setSequenceResolution (сетер на sequenceResolution)	частен	int **sequenceResolution**	void
setResolutionProportion	частен	няма	void
checkValue	частен	int **value**, int **max**	boolean
getDoubles	публичен	няма	double[]
doubleToInt	частен	double **value**, int **max**	int
intToDouble	частен	int **value**, int **max**	double

- **Конструктори** – Класът има два конструктора. Първият се използва когато системата за предсказване трябва да преобразува MIDI съобщенията от оригиналната мелодия в масиви от тип double. В този случай, нотата се създава от дадени MIDI съобщение **midiMessage**, продължителност на звучене **ticksLength** и разделителна способност на последователността - **sequenceResolution**. Вторият конструктор се използва, когато трябва да се конвертират данните, предсказани от невронната мрежа, в MIDI събития. В този случай, нотата се създава на базата на новите данни (масив от тип double, **data**) и разделителната способност на MIDI последователността - **sequenceResolution**.
- **getMessage** – Генерира ново MIDI съобщение от тип **Note on**. То описва текущата нота на базата на полетата channel, tone и velocity.
- **Сетери** – В таблицата са описани по два сетера за някои от характеристиките на нотата. Това е така, защото класът се използва в

два режима, можем да ги наречем прав и обратен. В правия режим, когато се преобразуват MIDI данните в масиви от реални числа, се използват сетерите, които приемат целочислени параметри. В обратния режим, когато преобразуваме масиви от реални числа в MIDI данни, използваме сетерите, които приемат като параметри реални числа (от тип double).

- **setResolutionProportion** - Изчислява отношението между sequenceResolution и INNER_RESOLUTION и го задава като стойност на resolutionProportion.
- **checkValue** – Използва се за проверка при задаване на стойности на полетата, за които са определени ограничения на стойностите. Проверява дали **value** е в диапазона [0; **max**). Ако не е, се генерира изключение.
- **getDoubles** – Връща характеристиките на нотата под формата на масив от реални числа (double), който се използва от невронната мрежа.
- **doubleToInt** – Преобразува double стойността на дадена характеристика на нотата (между 0 и 1) в нейната целочислена стойност, с която се представя в MIDI стандарта.
- **intToDouble** – Преобразува целочислената стойност на дадена характеристика на нотата в double стойност, която е удобна за представяне пред невронната мрежа (между 0 и 1).

Взаимодействие с външната среда

За да се стартира музикалното предсказване, е необходимо външната среда да предостави MIDI последователност на системата за предсказване. Тази последователност представлява обект от клас Sequence. Както беше уточнено, тази последователност трябва да бъде опростена. За предсказване се взима само първата писта (track). Наличието на други писти не е желателно и дори може да доведе до възникване на изключение. Това се случва, когато в първата писта се съхранява единствено служебна MIDI информация и никакви музикални данни (т. е. съобщения от тип Note on). От пистата се извличат единствено съобщенията от тип Note on. Останалите се игнорират. Добавянето на функционалност за обработване на последователности, съдържащи повече от една писти и на съобщения, различни от Note on, е перспектива за бъдещо развитие на приложението за предсказване на мелодии.

При инициализиране на системата за предсказване се задава и големината на прозореца на невронната мрежа. Тук трябва да се вземат предвид насоките, описани в точка **3.2.3**.

В етапа на предсказване заявката, изпратена от външната среда, съдържа два параметъра. Първият определя броя на нотите, на който трябва да е равна дължината на предсказаната мелодия. За стойността на този параметър няма горна

граница и той може да се използва за експериментиране от страна на потребителя. Вторият параметър, задаващ броя на повторенията за обучение на невронната мрежа, също е неограничен. За него, обаче, трябва да се има предвид, че прекалено малка стойност (примерно 1000) може да доведе до незадоволителни резултати от предсказването. За контраст, прекалено голяма стойност (например 100000) ще доведе до интересни резултати в предсказанието, но ще отнеме много изчислително време (около 10 минути, в зависимост от производителността на машината).

За да получи резултат от предсказването, външната среда трябва да направи заявка за извличане на резултата, като извика методът **getPredictedSecuence.**

3.4. Внедряване на системата за предсказване в уеб приложение

Системата за предсказване е реализирана под формата на Java класове и може да бъде използвана от всеки вид Java приложение. За демонстрация на възможностите на системата за предсказване е създадено уеб сървърно приложение, което се изпълнява от **Apache Tomcat** сървър. Причините за избора на точно този тип приложение за демонстрация на системата са следните:

- Уеб приложенията могат да бъдат използвани от всякакъв тип клиентски програми, тъй като обработва стандартни HTTP заявки. Най-често използваните клиентски програми са браузърите. Приложението може да бъде достъпено от всяка програма, независимо от езика, на който е написана, стига тя да изпрати валидна HTTP заявка, съдържаща необходимите параметри.
- Този тип приложения са достъпни за потребители, както от локалната машина, така и от потребителите в интернет. Това прави приложението достъпно за по-широка аудитория.
- Не е необходимо потребителите да имат инсталирана Java среда на машините си. Достатъчно е да имат браузър и достъп до интернет.

Недостатъците при използването на този тип приложение са следните:

- На машината, на която ще се изпълнява приложението, трябва да има инсталиран Apache Tomcat сървър. Това усложнява процеса на инсталиране на приложението на компютърната система, която ще играе ролята на сървър.
- Приложението е зависимо от допълнителни фактори (срив на сървъра, невъзможност за достъп до база от данни и др.), което повишава вероятността от поява на откази на системата.

Изпълнение под Apache Tomcat сървър

Съществуват множество уеб сървъри, предназначени за изпълняване на Java уеб приложения. По-известните от тях са Apache Tomcat, Glassfish, JBoss. Предимството на Tomcat пред останалите е, че е по-лек за системата. Това е необходимост в случая, тъй като за момента приложението се изпълнява на персонален компютър. Tomcat поддържа единствено стандартите Servlet и JSP, но това е напълно достатъчно, за да изпълнява приложението за предсказване на мелодии. Друг фактор за избора на този сървър е неговата доказана надеждност.

За изпълнението на системата за предсказване като уеб приложение е необходимо да бъдат изпълнени следните условия:

- Необходимо е да има поне един клас от тип сървлет (наследяващ базовия клас HttpServlet), който да приема HTTP заявката от клиента, да стартира системата за предсказване и да върне като отговор предсказаната мелодия на клиента.
- За да се осъществи връзка на Java приложението с база от данни, е необходимо да се инсталира JDBC (Java DataBase Connectivity[17]) драйвер. В случая на конкретната реализация е инсталиран JDBC драйвер за връзка с MySQL база от данни.

Описание на класовете

Java класовете, отговорни за управлението на уеб приложението са описани по-долу. Кодът на някои от тях може да бъде намерен в **Приложение 3**. Тези класове се намират в Java пакета “musicprediction”.

- **Клас “Predict” (Сървлет)** – Този клас наследява базовия клас HttpServlet и осъществява връзката на приложението с външната среда. Сървлетът отговаря за обработването на HTTP заявките. За валидни се приемат само **POST** заявки. Валидацията и обработването на параметрите на заявката се извършва от класа **ParamsModel**. Този клас стартира системата за предсказване и извлича предсказаната мелодия от нея. Той е отговорен за връщането на отговор към клиента, под формата на генериран MIDI файл. При възникване на грешка или изключение, този клас прихваща текста на възникналото събитие и го връща като символен низ на клиента.
- **Клас “ParamsModel”** – Този клас задава модела на параметрите в заявката. Той определя техните ограничения и следи за валидността на данните. След като завърши с проверката, обектът от този клас съхранява данните, изпратени от клиента, в удобен за работа формат. Той съхранява изпратения MIDI файл, ако има такъв, в буферна директория. След приключване на задачата за предсказване, обектът

от този клас е заставен да изтрие временния файл от файловата система.

- **Клас "ExampleSongs"** – Този клас служи за свързване на приложението с MySQL база от данни. Той изгражда интерфейс за извличане на MIDI файловете, записани в базата, за да може системата за предсказване да ги използва пълноценно. Също така този клас има метод за извличане единствено на имената на съхранените в базата песни, което служи за улеснение на клиента в избора му на мелодия за предсказване.
- **Клас "Texts"** – В този клас се съдържат повечето символни низове, използвани за комуникация с клиента. Низовете, дефинирани в този клас, са на български език. Обособяването им в отделен клас е направено с цел улеснение при необходимост от превеждане на интерфейса на друг език.

Връзка с външната среда и обработка на HTTP заявки

Връзката на приложението с външната среда се извършва от сървлета Predict. Комуникацията с нея се базира на стандартния протокол HTTP. Както беше пояснено по-рано, за валидни се приемат единствено POST заявки. Тези заявки трябва да съдържат следните параметри:

- **MIDI файл (midiFile)** – Този файл трябва да се съдържа под формата на поток от цифрови данни в HTTP заявката. Той трябва да валиден MIDI файл (с разширение ".mid"). Изискванията за съдържанието на този файл бяха описани в точка **3.3.3.** Това поле може да бъде празно. В такъв случай задължително трябва да бъде валиден следващият параметър.
- **Примерна песен от база от данни (exampleSong)** – Този параметър съдържа уникалния ключ на песента, с който тя се представя в базата от данни. Ключът представлява символен низ, който недвусмислено посочва записът от базата от данни, който трябва да бъде използван за предсказване на мелодията. Клиентът може да научи ключовете на наличните в базата песни и техните имена, благодарение на функционалността предоставена от класа ExampleSongs.
- **Дължина на предсказаната мелодия (predictedLength)** – Този параметър има същия смисъл като параметъра **numberOfTones**, предаван на метода **predict** от класа **MidiPrediction**. Поради причини, свързани с производителността и надеждността на системата, на този параметър е сложено ограничение да приема стойност в диапазона [1; 100000].

- **Повторения за обучение (trainingRepetitions)** – Този параметър има същия смисъл като параметъра **numberOfRepetitionsForTraining**, предаван на метода **predict** от класа **MidiPrediction**. Поради причини, свързани с производителността и надеждността на системата, на този параметър е сложено ограничение да приема стойност в диапазона [1; 1000000].
- **Големина на прозореца (windowSize)** - Този параметър има същия смисъл като параметъра **windowSize**, предаван на конструктора на класа **MidiPrediction**. Поради причини, свързани с производителността и надеждността на системата, на този параметър е сложено ограничение да приема стойност в диапазона [1; 100].

Връзка с MySQL база от данни

Тъй като намирането на MIDI файлове, отговарящи на изискванията на системата за предсказване на мелодии, може да затрудни потребителите, приложението е свързано с MySQL база от данни, която съхранява примерни такива мелодии. Базата от данни се състои от една единствена таблица **“songs”**, която има следната структура:

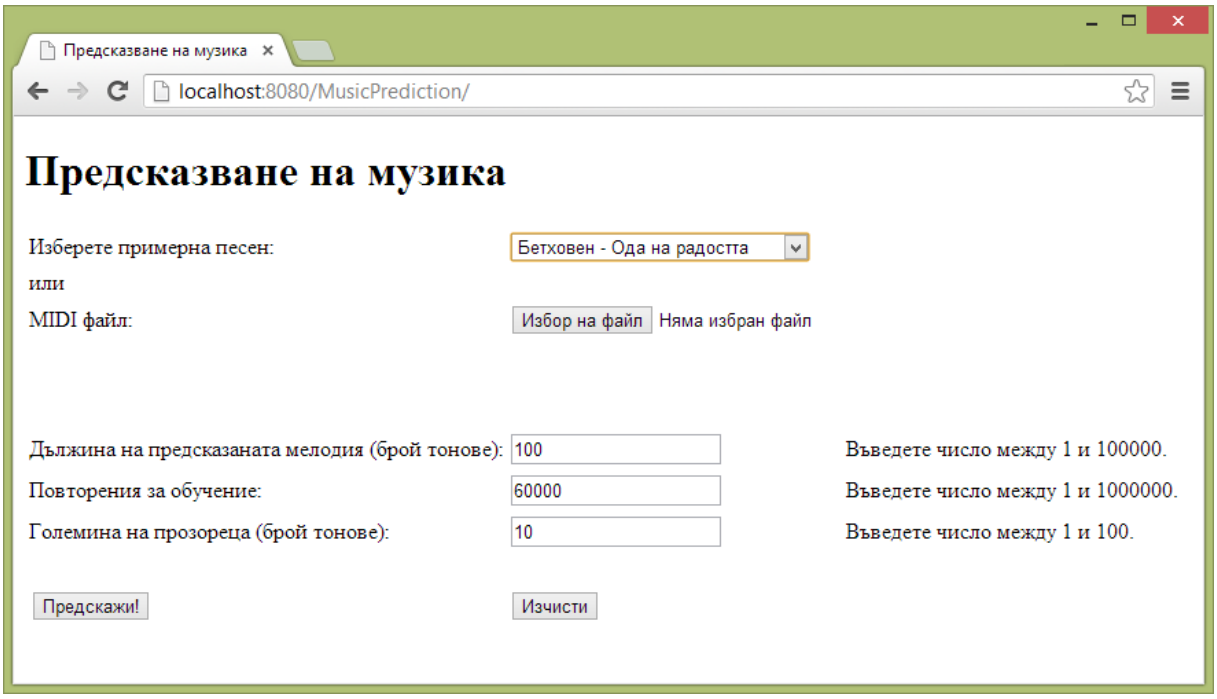
- **ID** – Идентификационен номер на песента.
- **Name** – Име на песента, написано информативно за потребителя.
- **Key** – Уникален ключ на песента под формата на символен низ, който звучи интуитивно.
- **FilePath** – Път до MIDI файла във файловата система на сървърния компютър.

Връзката на приложението с базата се извършва, благодарение на JDBC драйвера за MySQL бази от данни. Класът, който отговаря за свързването на системата с базата данни е ExampleSongs.

4. Ръководство за програмиста

В предишната глава бяха разгледани интерфейсите на отделните модули на приложението за предсказване на мелодии. Ако, обаче, разгледаме цялостно приложението като черна кутия, единственият начин, по който можем да взаимодействаме с него се явяват HTTP заявките. Приемайки, че за съставянето на HTTP заявка е необходима известна квалификация на потребителите на системата, ръководството за нейното използване ще бъде насочено към потребителите - програмисти.

За демонстрация на примерно използване на системата е създадена JSP (Java Server Pages) страницата, изобразена на **Фигура 16**.



Фигура 16. *Примерен потребителски интерфейс на приложението за предсказване на мелодии.*

Тази проста страница предоставя възможност за използване на системата и от потребители, които нямат необходимите знания за съставяне на HTTP заявки. Страницата предлага на потребителя за избере примерна песен или да избере MIDI файл от локалната си файлова система. Изборът на примерна песен става посредством падащо меню, в което са изброени наличните песни в базата от данни. Всеки ред от падащото меню съдържа като стойност уникалния ключ на съответната песен, а пред потребителя се извежда нейното име. Избирането на файл става със стандартно HTML поле от тип “file”. Във формуляра намират място и останалите настройки за предсказването. Изпращането на HTTP заявката към сървъра става посредством стандартно изпращане (Submit) на HTML елемента “form”. Програмният код на тази страница може да бъде намерен в **Приложение 4.**

На фигурата по-горе полетата за настройка на системата за предсказване са зададени с примерни стойности. Ако представим HTTP заявката, която ще бъде изпратена към сървъра, в JSON формат, тя би изглеждала по следния начин:

```
{
 "exampleSong" : "ode_to_joy",
 "midiFile" : "",
 "predictedLength" : "100",
 "trainingRepetitions" : "60000",
 "windowSize" : "10"
}
```

Смисълът на отделните параметри беше разгледан в точка **3.4.3**. Вижда се, че на параметъра exampleSong е присвоен символен низ, който отговаря на ключа на избраната примерна песен. Тъй като има избрана примерна песен, полето за прикачване на файл – midiFile, може да остане празно. Проверката за валидност на параметрите на заявката се извършва от Java приложението на сървъра, което значи, че изпращането на грешни стойности или стойности, които са извън допустимите граници, ще доведе до връщане на грешка към потребителя. До грешка ще се стигне също, ако потребителят нито избере примерна песен, нито прикачи MIDI файл.

Естествено използването на HTML страница, генерирана от JSP, е само един от вариантите за изграждане на интерфейс на системата. Програмистът има свободата да впрегне въображението си в съставянето на интерфейсната среда. Тя може да бъде изградена на всеки програмен език, поддържащ мрежови операции.

5. Заключение

5.1. Оценка

Системата за предсказване на мелодии, разработена за текущата дипломна работа, е създадена с демонстрационна и изследователска цел. Тази система, в настоящия си вид, няма приложение в решаването на реален проблем. В сравнение с други области в компютърното програмиране, разработването на програми за предсказване и генериране на мелодии е сравнително слабо развита област. Въпреки че съществуват много по-сложни реализации в тази сфера, текущото приложение също има своите предимства и недостатъци спрямо тях. Като недостатъци можем да отбележим следните:

- Съществен недостатък е, че системата работи единствено с опростени MIDI файлове. Поради това ограничение, предсказването на акорди, множество инструменти или ефекти в мелодията за момента е невъзможно. Единствените елементи в мелодията, които могат да бъдат предсказани, са простите единични тонове (съобщения от тип Note on). Това ограничение, обаче, може сравнително лесно да бъде премахнато, при евентуално доработване на системата.
- Друг недостатък е, че предсказаните мелодии често представляват хаотична последователност от тонове. Това най-често се случва, ако невронната мрежа не е обучена достатъчно добре върху оригиналната мелодия. Тази хаотичност се дължи на причината, че невронната мрежа с обратно разпространение не съхранява никаква информация за предишното си състояние след коя да е итерация. Тоновете се генерират един по един и не се следи цялостната музикална структура на мелодията. Това поведение на системата може да се поправи, ако се смени типът на невронната мрежа, отговорна за предсказването. Една подходяща такава мрежа би била предложената от Мозер[2] рекурентна автоматично предсказваща конекционистка мрежа. Трябва да се отбележи, обаче, че използването на такъв тип мрежа би усложнило допълнително системата за предсказване на мелодии.
- Недостатък е също, че при предсказване на по-дълги последователности от тонове, колко по-далеч от оригиналната мелодия отива предсказанието, толкова повече се отклонява то от правилното музикално звучене. Обикновено това се отново случва, когато невронната мрежа не е добре обучена. Дори и при добре обучена мрежа, обаче, предсказанието винаги ще има известни отклонения от хармоничното музикално звучене. Този проблем може да се реши отново с подмяна на типа на невронната мрежа.

- Известни проблеми създава и бавното обучение на мрежата. Неговата бързина на изпълнение зависи от множество фактори. Колкото по-разнообразна е мелодията, която искаме да предскажем, толкова по-дълго трябва да се обучава мрежата над нея. Ако мелодията е много разнообразна, едно добро обучение може да изисква 100 000 повторения на обучението. Това обучение може да отнеме около 5 - 10 минути. Това забавяне се дължи основно на това, че невронната мрежа е написана в обектно-ориентиран модел, който, както беше описано в началото на тази документация, забавя допълнително нейната работа.

Разработеното приложение за предсказване на мелодии има следните предимства:

- Колкото по-голям брой повторения за обучение прави невронната мрежа, толкова по-добро ще бъде предсказването на мелодията. При по-прости мелодии, предсказанието може дори да преповтори самата мелодия. При по-сложните мелодии това е малко вероятно. Въпреки това, като резултат от предсказването могат да излязат интересни за слушане последователности от тонове.
- Предимството на това, че системата работи с прости MIDI последователности, е че те лесно могат да бъдат проследени и изследвани. Тъй като последователностите са опростени, лесно може да се види до колко невронната мрежа се е обучила добре и по какъв начин е конструирала мелодията.
- Голямо предимство на приложението е, че е съвместимо за използване с всякакъв вид системи. Като интерфейс на приложението се използват стандартни HTTP заявки и то може да бъде достъпвано от всяка съвременна операционна система.
- Друго голяма предимство на системата за предсказване на мелодии е, че тя е лесна за надграждане. Благодарение на обектно-ориентираната логика, използвана при изграждането на приложението, за надграждането му или добавянето на нови функционалности не се изискват големи промени в построената вече логика. Това предимство прави приложението перспективно за бъдещи подобрения.

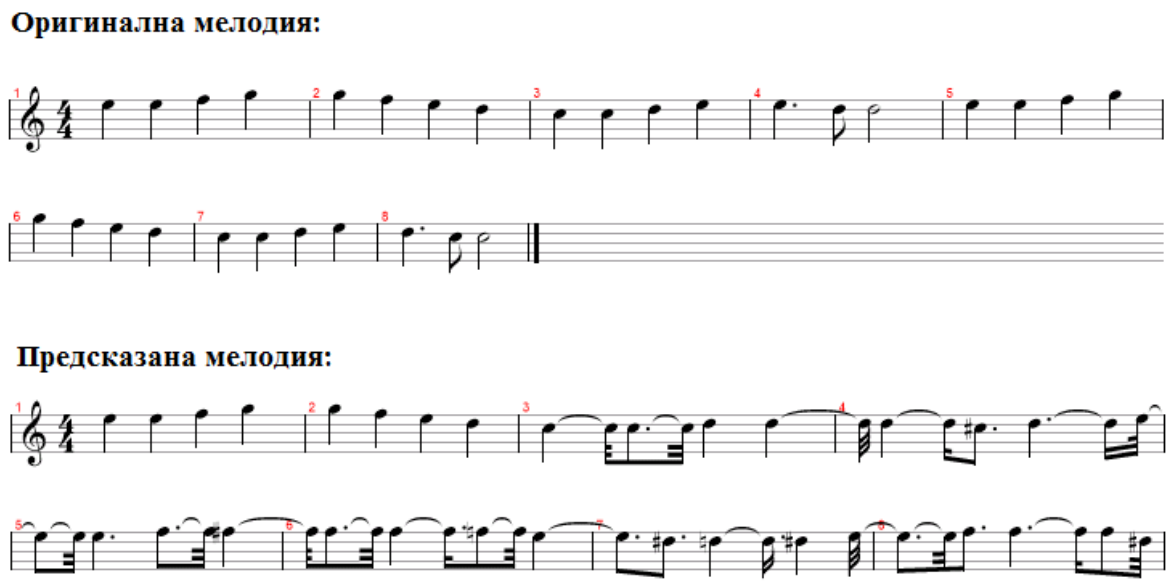
5.2. Тестване

В този раздел ще разгледаме една примерна сесия на използване на системата от потребител. Интерфейсът, който потребителят ще използва, е същият, като този от **Фигура 16.**

Потребителят задава за предсказване примерна мелодия от базата от данни. Тази мелодия е извадка от известен момент от **Одата на радостта** на **Лудвиг ван Бетховен**. Тъй като има избрана примерна мелодия, не се избира MIDI файл за

прикачване. Като дължина на предсказаната мелодия се задава 31. Повторенията за предсказване ще бъдат 60 000, а големината на прозореца – 10.

След двуминутно обучение на мрежата, клиентът получава като резултат предсказаната мелодия под формата на MIDI файл. На **Фигура 17** са показани нотните текстове за сравнение между оригиналната и предсказаната мелодия.



Фигура 17. *Сравнение между оригинална мелодия и предсказана такава. Оригиналната мелодия е от Одата на радостта на Лудвиг ван Бетховен.*

От горната фигура можем да направим следните изводи:

- Системата почти повтаря оригиналната мелодия в първите 8 ноти (приблизително, колкото е прозореца - 10). След отдалечаване от мелодията с 10 ноти, предсказаната мелодия започва все повече да се изкривява от правилното направление.
- Това поведение може да се дължи на недостатъчния брой повторения за обучение на мрежата. За получаване на по-добри резултати, потребителят може да опита да покачи броя на повторенията, примерно на 100 000.
- Потребителят, също така, може да опита да промени големината на прозореца и да проследи поведението на системата при различните стойности.

5.3. Предложения за развитие

Както беше уточнено по-рано, приложението в сегашния си вид не е приложимо за решаване на реални проблеми, а е предназначено за демонстрация и изследване. Въпреки това, то е лесно надградимо и може с малко промени да

бъде приведено в напълно функционално приложение. Основните предпоставки за получаване на пълна функционалност са следните:

- Премахване на ограниченията за MIDI файловете. В бъдещите версии на приложението трябва да могат да се предсказват всякакъв вид MIDI файлове. Това включва файлове с множество инструменти, които свирят едновременно, както и съдържащи музикални ефекти. Проблем, който може да възникне при премахването на ограниченията е, че работата на невронната мрежа ще се увеличи значително и предсказването на мелодии е възможно да се забави още повече.
- Трябва да се подобри системата за предсказване. Тя трябва да успява да структурира мелодията по такъв начин, че да се избегне генерирането на хаотични тонове в предсказаните данни. Това подобрение е по-трудно реализуемо от предното.
- Трябва да се оптимизира обучението на невронната мрежа да работи по-бързо. Това подобрение може да наложи пренаписването на по-голямата част от кода на приложението.

Това приложение може да се развива и в посока предсказване на аналогова музика. Тук, обаче, случаят с извличането на музикалната информация от звуковият сигнал се явява основен проблем. В това направление може да се обособят следните подходи за предсказване:

- Първият подход е да се разглежда аналоговата музика като графика на функция, стойностите на която се представят като времева серия. Тази времева серия може лесно да бъде прекарана през невронната мрежа. Проблемът в този случай е, че информацията, върху която ще обучаваме мрежата, ще бъде в огромно количество и най-вероятно предсказването ще става изключително бавно. Успехът не е гарантиран.
- Другият подход е чрез алгоритми за обработка на звукови вълни да се обособи музикалната информация, съдържана в аналоговите данни. В този случай извличането на последователностите от ноти се явява много трудна, дори невъзможна задача. Успехът отново не е гарантиран.

Необходимо е да се направят опити за създаване на системи за предсказване на мелодии чрез изброените по-горе два подхода и да се определи до каква степен са приложими те. Трудно е да се предвиди какви ще бъдат резултатите от изпълнението на подобни системи.

6. Литература

1. http://en.wikipedia.org/wiki/Computer_music - Статия за компютърна музика в Уикипедия.
2. Mozer M. C., Neural network music composition by prediction: Exploring the benefits of psychoacoustic constraints and multiscale processing, University of Colorado, 1994, 1
3. http://en.wikipedia.org/wiki/International_Computer_Music_Association - Статия за ICMA в Уикипедия.
4. http://en.wikipedia.org/wiki/IRCAM - Статия за IRCAM в Уикипедия.
5. http://bg.wikipedia.org/wiki/Невронна_мрежа - Статия за невронни мрежи в Уикипедия.
6. Нишева М. М., Шишков Д. П., Изкуствен интелект, Интеграл, Добрич, 1995, 142-154
7. Тренчев И., Миланов П., Пенчева Н., Мирчев И., Невронни мрежи, Югозападен университет „Неофит Рилски“
8. http://en.wikipedia.org/wiki/Digital_audio - Статия за цифрово представяне на звук в Уикипедия.
9. http://en.wikipedia.org/wiki/WAV - Статия за Wave файлове в Уикипедия.
10. http://bg.wikipedia.org/wiki/Аудио_файл_формат - Статия за звукови файлови формати в Уикипедия.
11. http://en.wikipedia.org/wiki/MP3 - Статия за MP3 файлове в Уикипедия.
12. http://bg.wikipedia.org/wiki/Windows_Media_Audio - Статия за WMA файлове в Уикипедия.
13. http://en.wikipedia.org/wiki/FLAC - Статия за FLAC файлове в Уикипедия.
14. http://www.blitter.com/~russtopia/MIDI/~jglatt/tech/midispec.htm - Спецификация на MIDI стандарта
15. http://docs.oracle.com/javase/tutorial/sound/overview-MIDI.html - Документация на пакета javax.sound.midi.
16. http://www.electronics.dit.ie/staff/tscarff/Music_technology/midi/midi_note_numbers_for_octaves.htm - Статия за представянето на нотите в MIDI.
17. http://en.wikipedia.org/wiki/Java_Database_Connectivity - Статия за JDBC в Уикипедия.

7. Приложения

7.1. Приложение 1

Код на класовете от пакет “neuralnetwork”.

Клас “PredictionNeuralNetwork”

```
package neuralnetwork;

import java.util.ArrayList;

public class PredictionNeuralNetwork {
    private ArrayList<Layer> layers;
    private ArrayList<double[]> dataSets;
    private int windowSize;
    private int predictedValuesStartIndex;

    public PredictionNeuralNetwork(int dimensionsCount, int windowSize, ArrayList<double[]>
dataSets, int[] neuronsCountPerHiddenLayer) {
        this.setDataSets(dataSets);
        this.setWindowSize(windowSize);
        this.setPredictedValuesStartIndex(dataSets.size());
        this.setLayers(new ArrayList<Layer>());

        // създава се входния слой
        this.addLayer(new Layer(dimensionsCount * windowSize));

        // създават се скритите слоеве
        for(int i = 0; i < neuronsCountPerHiddenLayer.length; i++) {
            this.addLayer(new Layer(neuronsCountPerHiddenLayer[i]));
        }

        // създава се изходния слой
        this.addLayer(new Layer(dimensionsCount));

        // създават се връзки между слоевете
        for(int i = 0; i < this.layers.size() - 1; i++) {
            Layer layer = this.layers.get(i);
            Layer followingLayer = this.layers.get(i + 1);
            layer.setFollowingLayer(followingLayer);
        }
    }

    public ArrayList<Layer> getLayers() {
        return layers;
    }

    private void setLayers(ArrayList<Layer> layers) {
        this.layers = layers;
    }

    public ArrayList<double[]> getDataSets() {
        return dataSets;
    }

    public void setDataSets(ArrayList<double[]> dataSets) {
        this.dataSets = dataSets;
    }

    public int getWindowSize() {
        return windowSize;
    }
```

```
    public void setWindowSize(int windowSize) {
        this.windowSize = windowSize;
    }

    public int getPredictedValuesStartIndex() {
        return predictedValuesStartIndex;
    }

    private void setPredictedValuesStartIndex(int predictedValuesStartIndex) {
        this.predictedValuesStartIndex = predictedValuesStartIndex;
    }

    public ArrayList<double[]> getPredictedValues() {
        if(this.predictedValuesStartIndex < this.dataSets.size() - 1) {
            return new
ArrayList<double[]>(this.dataSets.subList(this.predictedValuesStartIndex,
this.dataSets.size() - 1));
        }
        else {
            return null;
        }
    }

    private void addLayer(Layer layer) {
        this.layers.add(layer);
    }

    private void processLayer(Layer layer, double[] expectedValues) {
        Layer followingLayer = layer.getFollowingLayer();
        if(layer.getPreviousLayer() != null) {
            layer.calculateNeuronsValues();
        }
        if(followingLayer != null) {
            processLayer(followingLayer, expectedValues);
            layer.calculateNeuronsErrors();
        }
        else {
            layer.calculateNeuronsErrors(expectedValues);
        }
    }

    private void predict() {
        double[] inputs = new double[this.layers.get(0).size()];
        int last = 0;
        int i;
        int dataSetIndex = this.dataSets.size() - this.windowSize;

        try {
            if(this.windowSize >= this.dataSets.size()) {
                throw(new Exception("Броят на данните е по-малък от големината на
прозореца!"));
            }
        }
        catch(Exception e) {
            System.err.println(e);
            return;
        }

        // задава се входният вектор на мрежата
        for(i = dataSetIndex; i < this.dataSets.size(); i++) {
            double[] dataSet = this.dataSets.get(i);
            for(int k = 0; k < dataSet.length; k++) {
                inputs[last++] = dataSet[k];
            }
        }
```

```
        this.resetNeurons();
        this.layers.get(0).setNeuronValues(inputs);

        // изчисляват се стойностите на невроните
        for(i = 1; i < this.layers.size(); i++) {
            this.layers.get(i).calculateNeuronsValues();
        }

        // изходният вектор се поставя в края на колекцията с данните
        this.dataSets.add(this.layers.get(i - 1).getNeuronValues());
    }

    private void epoch(int dataSetIndex) {
        double[] inputs = new double[this.layers.get(0).size()];
        int last = 0;

        try {
            if(this.windowSize >= this.dataSets.size()) {
                throw(new Exception("Броят на данните е по-малък от големината на
прозореца!"));
            }
        }
        catch(Exception e) {
            System.err.println(e);
            return;
        }

        // задава се входният вектор на мрежата
        for(int i = dataSetIndex; i < dataSetIndex + this.windowSize; i++) {
            double[] dataSet = this.dataSets.get(i % this.dataSets.size());
            for(int k = 0; k < dataSet.length; k++) {
                inputs[last++] = dataSet[k];
            }
        }
        this.resetNeurons();
        this.layers.get(0).setNeuronValues(inputs);

        // започва рекурсивно обработване на слоевете
        this.processLayer(this.layers.get(0), this.dataSets.get((dataSetIndex +
this.windowSize) % this.dataSets.size()));
    }

    private void resetNeurons() {
        for(int i = 0; i < this.layers.size(); i++) {
            this.layers.get(i).resetNeurons();
        }
    }

    public void start(int repetitions, int predictionsCount) {
        int predictionStart = this.dataSets.size() - this.windowSize;
        int predictionEnd = predictionStart + predictionsCount;
        int iterationsForTraining = this.dataSets.size() * repetitions;

        // обучение на мрежата
        for(int i = 0; i < iterationsForTraining; i++) {
            this.epoch(i % this.dataSets.size());
        }

        // предсказване
        for(int i = predictionStart; i < predictionEnd; i++) {
            this.predict();
        }
    }

    @Override
    public String toString() {
```

```
        StringBuilder strBldr = new StringBuilder();
        strBldr.append("Слоеве:\n");
        for(int i = 0; i < this.layers.size(); i++) {
            strBldr.append(this.layers.get(i));
            strBldr.append("\n");
        }
        return strBldr.toString();
    }
}
```

Клас “Layer”

```
package neuralnetwork;

import java.util.ArrayList;

class Layer {
    private Neuron auxiliaryNeuron;
    private ArrayList<Neuron> neurons;
    private Layer previousLayer;
    private Layer followingLayer;

    public Layer(int neuronsCount) {
        this.setNeurons(new ArrayList<Neuron>());
        this.initNeurons(neuronsCount);
    }

    private void initNeurons(int neuronsCount) {
        this.setAuxiliaryNeuron(new Neuron(1.0));
        for(int i = 0; i < neuronsCount; i++) {
            this.addNeuron(new Neuron());
        }
    }

    private void linkAuxiliaryNeuronWithFollowingLayer() {
        for(int i = 0; i < this.followingLayer.size(); i++) {
            this.auxiliaryNeuron.linkOutputWith(this.followingLayer.getNeuron(i));
        }
    }

    private void linkNeuronWithFollowingLayer(int neuronIndex) {
        Neuron neuron = this.neurons.get(neuronIndex);
        for(int i = 0; i < this.followingLayer.size(); i++) {
            neuron.linkOutputWith(this.followingLayer.getNeuron(i));
        }
    }

    private void linkAllNeuronsWithFollowingLayer() {
        this.linkAuxiliaryNeuronWithFollowingLayer();
        for(int i = 0; i < this.neurons.size(); i++) {
            this.linkNeuronWithFollowingLayer(i);
        }
    }

    public Neuron getAuxiliaryNeuron() {
        return auxiliaryNeuron;
    }

    private void setAuxiliaryNeuron(Neuron auxiliaryNeuron) {
        this.auxiliaryNeuron = auxiliaryNeuron;
    }

    public ArrayList<Neuron> getNeurons() {
        return this.neurons;
    }
```

```
public double[] getNeuronValues() {
    double[] values = new double[this.neurons.size()];
    int last = 0;
    for(Neuron neuron : this.neurons) {
        values[last++] = neuron.getValue();
    }
    return values;
}

private void setNeurons(ArrayList<Neuron> neurons) {
    this.neurons = neurons;
}

// използва се само за входния слой
public void setNeuronValues(double[] neuronValues) {
    for(int i = 0; i < neuronValues.length; i++) {
        this.neurons.get(i).setValue(neuronValues[i]);
    }
}

public Layer getPreviousLayer() {
    return previousLayer;
}

private void setPreviousLayer(Layer previousLayer) {
    this.previousLayer = previousLayer;
}

public Layer getFollowingLayer() {
    return followingLayer;
}

public void setFollowingLayer(Layer followingLayer) {
    followingLayer.setPreviousLayer(this);
    this.followingLayer = followingLayer;
    this.linkAllNeuronsWithFollowingLayer();
}

public void addNeuron(Neuron neuron) {
    neuron.setLayer(this);
    this.neurons.add(neuron);
}

public Neuron getNeuron(int index) {
    return this.neurons.get(index);
}

public int size() {
    return this.neurons.size();
}

public void calculateNeuronsValues() {
    for(int i = 0; i < this.neurons.size(); i++) {
        this.neurons.get(i).calculateValue();
    }
}

// за входния и скритите слоеве
public void calculateNeuronsErrors() {
    this.auxiliaryNeuron.calculateError();
    for(int i = 0; i < this.neurons.size(); i++) {
        this.neurons.get(i).calculateError();
    }
}
```

```
    // за изходния слой
    public void calculateNeuronsErrors(double[] expectedValues) {
        for(int i = 0; i < this.neurons.size(); i++) {
            this.neurons.get(i).calculateError(expectedValues[i]);
        }
    }

    public void resetNeurons() {
        for(int i = 0; i < this.neurons.size(); i++) {
            this.neurons.get(i).reset();
        }
    }

    @Override
    public String toString() {
        StringBuilder strBldr = new StringBuilder();
        strBldr.append("================================\n");
        strBldr.append("Неврони:\n");
        strBldr.append(this.auxiliaryNeuron);
        strBldr.append("\n");
        for(int i = 0; i < this.neurons.size(); i++) {
            strBldr.append(this.neurons.get(i));
            strBldr.append("\n");
        }
        strBldr.append("================================\n");
        return strBldr.toString();
    }
}
```

Клас “Neuron”

```
package neuralnetwork;

import java.util.ArrayList;

class Neuron {
    private double value;
    private double error;
    private ArrayList<Link> inputs;
    private ArrayList<Link> outputs;
    private Layer layer;

    public Neuron() {
        this.setInputs(new ArrayList<Link>());
        this.setOutputs(new ArrayList<Link>());
        this.setLayer(null);
        this.reset();
    }

    public Neuron(double value) {
        this.setValue(value);
        this.setError(0);
        this.setInputs(new ArrayList<Link>());
        this.setOutputs(new ArrayList<Link>());
        this.setLayer(null);
    }

    public double getValue() {
        return value;
    }

    public void setValue(double value) {
        this.value = value;
    }
```

```
public ArrayList<Link> getInputs() {
    return inputs;
}

private void setInputs(ArrayList<Link> inputs) {
    this.inputs = inputs;
}

public ArrayList<Link> getOutputs() {
    return outputs;
}

private void setOutputs(ArrayList<Link> outputs) {
    this.outputs = outputs;
}

public Layer getLayer() {
    return layer;
}

public void setLayer(Layer layer) {
    this.layer = layer;
}

public double getError() {
    return error;
}

public void setError(double error) {
    this.error = error;
}

public void addInputLink(Link link) {
    this.inputs.add(link);
}

public void addOutputLink(Link link) {
    this.outputs.add(link);
}

public void linkInputWith(Neuron otherNeuron) {
    new Link(otherNeuron, this);
}

public void linkOutputWith(Neuron otherNeuron) {
    new Link(this, otherNeuron);
}

public void calculateValue() {
    double sum = 0.0;

    // сумират се изходните стойности на входните връзки на неврона
    for(int i = 0; i < this.inputs.size(); i++) {
        sum += this.inputs.get(i).getForwardValue();
    }

    // активационна функция на неврона
    this.setValue(1 / (1 + Math.exp(-sum)));
}

// за неврони от скритите слоеве
public void calculateError() {
    double sum = 0.0;

    // сумират се грешките от изходните връзки на неврона
    for(int i = 0; i < this.outputs.size(); i++) {
```

```
            sum += this.outputs.get(i).getBackwardError();
        }

        // функция за определяне на грешката на неврона
        this.setError(this.value * (1 - this.value) * sum);

        // актуализиране теглата на връзките
        this.correctOutputLinksWeights();
    }

    // за неврони от изходния слой
    public void calculateError(double expectedValue) {
        this.setError(this.value * (1 - this.value) * (expectedValue - this.value));
    }

    private void correctOutputLinksWeights() {
        for(int i = 0; i < this.outputs.size(); i++) {
            this.outputs.get(i).correctWeight();
        }
    }

    public void reset() {
        this.setValue(0);
        this.setError(0);
    }

    @Override
    public String toString() {
        StringBuilder strBldr = new StringBuilder();
        strBldr.append("Стойност: ");
        strBldr.append(this.value);
        strBldr.append(", Входни връзки: ");
        strBldr.append(this.inputs != null ? this.inputs.size() : "няма");
        strBldr.append(", Изходни връзки: ");
        strBldr.append(this.outputs != null ? this.outputs.size() : "няма");
        return strBldr.toString();
    }
}
```

Клас “Link”

```
package neuralnetwork;

class Link {
    private double weight;
    private Neuron input;
    private Neuron output;
    private static final Double WEIGHT_PROPORTION = 0.35;

    public Link(Neuron input, Neuron output) {
        input.addOutputLink(this);
        output.addInputLink(this);
        this.setInput(input);
        this.setOutput(output);
        // теглото се инициализира с произволна стойност между -0,1 и 0,1
        this.setWeight(-0.2 * Math.random() + 0.1);
    }

    public double getWeight() {
        return weight;
    }

    private void setWeight(double weight) {
        this.weight = weight;
    }
```

```
    public Neuron getInput() {
        return input;
    }

    public void setInput(Neuron input) {
        this.input = input;
    }

    public Neuron getOutput() {
        return output;
    }

    public void setOutput(Neuron output) {
        this.output = output;
    }

    public void correctWeight() {
        // корекция на теглото
        this.weight += this.input.getValue() * this.getOutput().getError() *
Link.WEIGHT_PROPORTION;
    }

    public double getForwardValue() {
        return this.input.getValue() * this.weight;
    }

    public double getBackwardError() {
        return this.output.getError() * this.weight;
    }
}
```

7.2. Приложение 2

Код на класовете от пакет “midiparser”.

Клас “MidiPrediction”

```
package midiparser;

import java.io.File;
import java.io.IOException;
import java.util.ArrayList;

import javax.sound.midi.InvalidMidiDataException;
import javax.sound.midi.MidiEvent;
import javax.sound.midi.MidiMessage;
import javax.sound.midi.MidiSystem;
import javax.sound.midi.MidiUnavailableException;
import javax.sound.midi.Sequence;
import javax.sound.midi.Sequencer;
import javax.sound.midi.ShortMessage;
import javax.sound.midi.Track;

import neuralnetwork.PredictionNeuralNetwork;

public class MidiPrediction {
    private Sequence originalSequence;
    private ArrayList<double[]> musicData;
    private ArrayList<double[]> predictedMusicData;
    private PredictionNeuralNetwork neuralNetwork;
    private static int[] HIDDEN_LAYERS = new int[]{80};
```

```
    public MidiPrediction(Sequence originalSequence, int windowSize) {
        this.setOriginalSequence(originalSequence);
        this.parseMusicData();
        this.neuralNetwork = new PredictionNeuralNetwork(4, windowSize, this.musicData,
MidiPrediction.HIDDEN_LAYERS);
    }

    public void setOriginalSequence(Sequence originalSequence) {
        this.originalSequence = originalSequence;
    }

    private void parseMusicData() {
        Track[] tracks = this.originalSequence.getTracks();
        Track track = tracks.length > 1 ? tracks[1] : tracks[0]; // избира се първата писта
        int resolution = this.originalSequence.getResolution();

        this.musicData = new ArrayList<double[]>();

        for(int i = 0; i < track.size(); i++) {
            MidiEvent midiEvent = track.get(i);
            MidiMessage midiMessage = track.get(i).getMessage();
            Note note;
            int status = midiMessage.getStatus();
            int tick;

            if(status != ShortMessage.NOTE_ON) {
                // игнориране на съобщенията, които не са Note on
                continue;
            }

            if(i < track.size() - 1) {
                tick = (int)(track.get(i + 1).getTick() - midiEvent.getTick());
            }
            else {
                // експериментално се определя дължената на последната нота
                tick = (int)(midiEvent.getTick() - track.get(i - 1).getTick());
            }

            note = new Note((ShortMessage)midiMessage, tick, resolution);

            this.musicData.add(note.getDoubles());
        }
    }

    public void predict(int numberOfTones, int numberOfRepetitionsForTraining) {
        // стартиране на невронната мрежа
        this.neuralNetwork.start(numberOfRepetitionsForTraining, numberOfTones);
        this.predictedMusicData = this.neuralNetwork.getPredictedValues();
    }

    public Sequence getPredictedSecuence() throws InvalidMidiDataException {
        Sequence predictedSequence = new Sequence(this.originalSequence.getDivisionType(),
this.originalSequence.getResolution());
        Track track = predictedSequence.createTrack();
        long absoluteTicks = 0;
        for(int i = 0; i < this.predictedMusicData.size(); i++) {
            Note note = new Note(this.predictedMusicData.get(i),
predictedSequence.getResolution());
            MidiEvent midiEvent = new MidiEvent(note.getMessage(), absoluteTicks);
            absoluteTicks += note.getTicksLength();
            track.add(midiEvent);
        }
        return predictedSequence;
    }
```

```
    public static void playSequence(Sequence sequence) throws InvalidMidiDataException,
MidiUnavailableException, InterruptedException {
        Sequencer sequencer = MidiSystem.getSequencer();

        if(!sequencer.isOpen()) {
            sequencer.open();
        }
        else {
            sequencer.close();
        }

        sequencer.setSequence(sequence);

        System.out.println("Start!");

        sequencer.start();
        while(sequencer.isRunning()) {
            Thread.sleep(10);
        }
        sequencer.close();

        System.out.println("Finish!");
    }

    public static void saveSequence(Sequence sequence, String filePath) throws IOException
{
        MidiSystem.write(sequence, 0, new File(filePath));
    }

    @Override
    public String toString() {
        StringBuilder strBldr = new StringBuilder();
        for(int i = 0; i < this.musicData.size(); i++) {
            double[] datum = this.musicData.get(i);
            strBldr.append("Нота: " + datum[0]);
            strBldr.append(", Сила: " + datum[1]);
            strBldr.append(", Време: " + datum[2] + "\n");
        }
        return strBldr.toString();
    }
}
```

Клас “Note”

```
package midiparser;

import javax.sound.midi.InvalidMidiDataException;
import javax.sound.midi.ShortMessage;

class Note {
    private int channel;
    private int tone;
    private int velocity;
    private int ticksLength;
    private int sequenceResolution;
    private double resolutionProportion;
    private static int INNER_RESOLUTION = 100;
    private static int MAX_CHANNEL = 16;
    private static int MAX_BYTE = 128;
    private static int MAX_TICK_LENGTH = 400;

    public Note(ShortMessage midiMessage, int ticksLength, int sequenceResolution) {
        this.setSequenceResolution(sequenceResolution);
        this.setResolutionProportion();
```

```
        this.setChannel(midiMessage.getChannel());
        this.setTone(midiMessage.getData1());
        this.setVelocity(midiMessage.getData2());
        this.setTicksLength(ticksLength);
    }

    public Note(double[] data, int sequenceResolution) {
        this.setSequenceResolution(sequenceResolution);
        this.setResolutionProportion();

        this.setChannel(data[0]);
        this.setTone(data[1]);
        this.setVelocity(data[2]);
        this.setTicksLength(data[3]);
    }

    public ShortMessage getMessage() throws InvalidMidiDataException {
        return new ShortMessage(ShortMessage.NOTE_ON, this.channel, this.tone,
this.velocity);
    }

    public int getTicksLength() {
        return (int)Math.round((double)this.ticksLength * this.resolutionProportion);
    }

    private void setTicksLength(int ticksLength) {
        ticksLength = (int)Math.round((double)ticksLength / this.resolutionProportion);
        if(this.checkValue(ticksLength, Note.MAX_TICK_LENGTH)) {
            this.ticksLength = ticksLength;
        }
    }

    private void setTicksLength(double value) {
        int ticksLength = doubleToInt(value, Note.MAX_TICK_LENGTH);
        if(this.checkValue(ticksLength, Note.MAX_TICK_LENGTH)) {
            this.ticksLength = ticksLength;
        }
    }

    private void setChannel(int channel) {
        if(this.checkValue(tone, Note.MAX_CHANNEL)) {
            this.channel = channel;
        }
    }

    private void setChannel(double value) {
        int channel = doubleToInt(value, Note.MAX_CHANNEL);
        if(this.checkValue(tone, Note.MAX_CHANNEL)) {
            this.channel = channel;
        }
    }

    private void setTone(int tone) {
        if(this.checkValue(tone, Note.MAX_BYTE)) {
            this.tone = tone;
        }
    }

    private void setTone(double value) {
        int tone = doubleToInt(value, Note.MAX_BYTE);
        if(this.checkValue(tone, Note.MAX_BYTE)) {
            this.tone = tone;
        }
    }

    private void setVelocity(int velocity) {
```

```
        if(this.checkValue(velocity, Note.MAX_BYTE)) {
            this.velocity = velocity;
        }
    }

    private void setVelocity(double value) {
        int velocity = doubleToInt(value, Note.MAX_BYTE);
        if(this.checkValue(velocity, Note.MAX_BYTE)) {
            this.velocity = velocity;
        }
    }

    private void setSequenceResolution(int sequenceResolution) {
        this.sequenceResolution = sequenceResolution;
    }

    private void setResolutionProportion() {
        this.resolutionProportion = (double)this.sequenceResolution /
(double)Note.INNER_RESOLUTION;
    }

    private boolean checkValue(int value, int max) {
        try {
            if(value < 0 || value > max) {
                throw new Exception("Стойността е извън границите! Граници: [0; " + max +
"], Стойност: " + value);
            }
            return true;
        }
        catch(Exception e) {
            System.err.println(e);
        }
        return false;
    }

    public double[] getDoubles() {
        double[] data = new double[4];
        data[0] = this.intToDouble(this.channel, Note.MAX_CHANNEL);
        data[1] = this.intToDouble(this.tone, Note.MAX_BYTE);
        data[2] = this.intToDouble(this.velocity, Note.MAX_BYTE);
        data[3] = this.intToDouble(this.ticksLength, Note.MAX_TICK_LENGTH);
        return data;
    }

    private int doubleToInt(double value, int max) {
        return (int)Math.round(value * (double)max);
    }

    private double intToDouble(int value, int max) {
        return (double)value / (double)max;
    }
}
```

7.3. Приложение 3

Код на класовете от пакет “musicprediction”.

Клас “Predict” (Сървлет)

```
package musicprediction;
import java.io.File;
import java.io.IOException;
```

```java
import java.io.OutputStream;
import java.io.PrintWriter;
import java.util.List;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.sound.midi.InvalidMidiDataException;
import javax.sound.midi.MidiSystem;
import javax.sound.midi.Sequence;

import midiparser.MidiPrediction;

import org.apache.tomcat.util.http.fileupload.disk.DiskFileItemFactory;
import org.apache.tomcat.util.http.fileupload.servlet.ServletFileUpload;
import org.apache.tomcat.util.http.fileupload.FileItem;
import org.apache.tomcat.util.http.fileupload.FileItemFactory;

/**
 * Servlet implementation class Predict
 */
@WebServlet("/Predict")
public class Predict extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public Predict() {
        super();
    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        try {
            throw new Exception(Texts.ERR_INVALID_REQUEST);
        } catch (Exception e) {
            this.exceptionsHandler(e, response);
            return;
        }
    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        Sequence predictedSequence;
        String predictedFileName;
        ParamsModel params;

        try {
            // обработване параметрите на заявката
            params = this.handleParams(request);
            // извличане на предсказаните данни
            predictedSequence = this.getPredicted(params);
            predictedFileName = "predicted_" + (params.hasAttachedFile() ?
params.getMidiFile().getName() : params.getExampleSong().getName());
            // връщане на предсказаната мелодия
            this.returnPredicted(predictedSequence, predictedFileName, response);
```

```
            // изчистване на временните ресурси
            params.destroy();
        } catch (Exception e) {
            this.exceptionsHandler(e, response);
            return;
        }
    }

    private ParamsModel handleParams(HttpServletRequest request) throws Exception {
        boolean isMultipart = ServletFileUpload.isMultipartContent(request);
        if(!isMultipart) {
            throw new Exception(Texts.ERR_INVALID_REQUEST);
        }
        // извличане на данните от параметрите на заявката
        FileItemFactory factory = new DiskFileItemFactory();
        ServletContext servletContext = this.getServletConfig().getServletContext();
        File repository = (File)
servletContext.getAttribute("javax.servlet.context.tempdir");
        ((DiskFileItemFactory) factory).setRepository(repository);
        ServletFileUpload upload = new ServletFileUpload(factory);
        List<FileItem> items = upload.parseRequest(request);
        return new ParamsModel(items);
    }

    private Sequence getPredicted(ParamsModel params) throws InvalidMidiDataException,
IOException {
        Sequence sequence, predictedSequence;
        if (params.hasAttachedFile()) {
            sequence = MidiSystem.getSequence(params.getMidiFile());
        }
        else {
            sequence = MidiSystem.getSequence(params.getExampleSong());
        }
        MidiPrediction midiPrediction = new MidiPrediction(sequence,
params.getWindowSize());
        // заявка за предсказване
        midiPrediction.predict(params.getPredictedLength(),
params.getTrainingRepetitions());
        predictedSequence = midiPrediction.getPredictedSecuence();
        return predictedSequence;
    }

    private void returnPredicted(Sequence predictedSequence, String name,
HttpServletResponse response) throws IOException {
        OutputStream os;
        response.setContentType("audio/midi");
        response.setHeader("Content-Disposition", "attachment; filename=" + name + ";");
        os = response.getOutputStream();
        MidiSystem.write(predictedSequence, 0, os);
        os.close();
    }

    // обработване на изключенията
    private void exceptionsHandler(Exception e, HttpServletResponse response) throws
IOException {
        response.setContentType("text/html");
        response.setCharacterEncoding("UTF-8");
        PrintWriter pw = response.getWriter();
        pw.println(e.getMessage());
        pw.close();
    }

}
```

Клас “ParamsModel”

```
package musicprediction;
import java.io.File;
import java.util.List;

import javax.sound.midi.InvalidMidiDataException;
import javax.sound.midi.MidiSystem;

import org.apache.tomcat.util.http.fileupload.FileItem;

public class ParamsModel {
    private boolean hasAttachedFile;

    private File exampleSong;
    private File midiFile;

    public static int MIN_predictedLength = 1;
    public static int MAX_predictedLength = 100000;
    private int predictedLength;

    public static int MIN_trainingRepetitions = 1;
    public static int MAX_trainingRepetitions = 1000000;
    private int trainingRepetitions;

    public static int MIN_windowSize = 1;
    public static int MAX_windowSize = 100;
    private int windowSize;

    private static String TEMP_DIR = "E:/Projects/MusicPrediction/Temp/";

    public ParamsModel(List<FileItem> items) throws Exception {
        this.hasAttachedFile = false;
        try {
            for(FileItem item : items) {
                String fieldName = item.getFieldName();
                // разпределяне на параметрите на заявката
                if(item.isFormField()) {
                    switch(fieldName.toLowerCase()) {
                    case "examplesong":
                        this.setExampleSong(item.getString());
                        break;
                    case "predictedlength":
                        this.setPredictedLength(new Integer(item.getString()));
                        break;
                    case "trainingrepetitions":
                        this.setTrainingRepetitions(new Integer(item.getString()));
                        break;
                    case "windowsize":
                        this.setWindowSize(new Integer(item.getString()));
                        break;
                    }
                }
                else {
                    if(fieldName.equalsIgnoreCase("midifile")) {
                        this.setMidiFile(item);
                    }
                }
            }
            if (this.midiFile == null && this.exampleSong == null) {
                throw new Exception(Texts.ERR_NO_SONG);
            }
        }
        catch(Exception e) {
            throw e;
        }
```

```
    }

    public boolean hasAttachedFile() {
        return hasAttachedFile;
    }

    public File getExampleSong() {
        return exampleSong;
    }

    public void setExampleSong(String exampleSong) throws Exception {
        if (!exampleSong.isEmpty()) {
            this.exampleSong = ExampleSongs.getSong(exampleSong);
        }
    }

    private void setMidiFile(FileItem item) throws Exception {
        if (item.get().length > 0) {
            File file = new File(ParamsModel.TEMP_DIR + item.getName());
            item.write(file);
            this.midiFile = file;
            this.hasAttachedFile = true;
            try {
                // проверяваме дали прикаченият файл е действително MIDI
                MidiSystem.getSequence(this.midiFile);
            }
            catch(InvalidMidiDataException e) {
                this.destroy();
                throw new Exception(Texts.ERR_INVALID_MIDI_DATA);
            }
        }
    }

    public File getMidiFile() {
        return midiFile;
    }

    public int getPredictedLength() {
        return predictedLength;
    }

    public void setPredictedLength(int predictedLength) throws Exception {
        Texts.validateNumber(Texts.predictedLength, predictedLength,
ParamsModel.MIN_predictedLength, ParamsModel.MAX_predictedLength);
        this.predictedLength = predictedLength;
    }

    public int getTrainingRepetitions() {
        return trainingRepetitions;
    }

    public void setTrainingRepetitions(int trainingRepetitions) throws Exception {
        Texts.validateNumber(Texts.trainingRepetitions, trainingRepetitions,
ParamsModel.MIN_trainingRepetitions, ParamsModel.MAX_trainingRepetitions);
        this.trainingRepetitions = trainingRepetitions;
    }

    public int getWindowSize() {
        return windowSize;
    }

    public void setWindowSize(int windowSize) throws Exception {
        Texts.validateNumber(Texts.windowSize, windowSize, ParamsModel.MIN_windowSize,
ParamsModel.MAX_windowSize);
        this.windowSize = windowSize;
    }
```

```
    public void destroy() throws Exception {
        if (this.midiFile.exists()) {
            this.midiFile.delete();
        }
    }
}
```

Клас “ExampleSongs”

```
package musicprediction;

import java.io.File;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.Hashtable;
import java.util.Iterator;
import java.util.Map;

public abstract class ExampleSongs {
    // Конфигурации за базата данни
    private static final String dbUrl = "jdbc:mysql://localhost:3306/music_prediction";
    private static final String dbUsername = "root";
    private static final String dbPassword = "";

    // метод за изпращане на SELECT заявка със зададени критерии
    private static ResultSet selectQuery(Map<String, Object> criteria) throws Exception {
        Connection dbConnect = DriverManager.getConnection(dbUrl, dbUsername, dbPassword);
        PreparedStatement dbStatement;
        String where = "";
        String query = "SELECT * FROM `songs` WHERE %s";
        Iterator<Map.Entry<String, Object>> iterator = criteria.entrySet().iterator();
        ArrayList<Object> values = new ArrayList<Object>();
        boolean hasNext = iterator.hasNext();

        while (hasNext) {
            Map.Entry<String, Object> criterium = iterator.next();
            values.add(criterium.getValue());
            where += "`" + criterium.getKey() + "` = ?";
            hasNext = iterator.hasNext();
            if (hasNext) {
                where += " AND ";
            }
        }
        dbStatement = dbConnect.prepareStatement(query = String.format(query, where));
        for (int i = 1, ii = values.size(); i <= ii; i++) {
            dbStatement.setString(i, values.get(i - 1).toString());
        }
        return dbStatement.executeQuery();
    }

    // метод за изпращане на SELECT заявка за извличане на всички записи
    private static ResultSet selectQuery() throws Exception {
        Class.forName("com.mysql.jdbc.Driver");
        Connection dbConnect = DriverManager.getConnection(dbUrl, dbUsername, dbPassword);
        Statement dbStatement = dbConnect.createStatement();
        String query = "SELECT * FROM `songs`";

        return dbStatement.executeQuery(query);
    }
```

```
    // извличане на песен, посочена от базата данни
    public static File getSong(String key) throws Exception {
        Hashtable<String, Object> criteria = new Hashtable<String, Object>();
        ResultSet result;
        String file;

        criteria.put("Key", key);
        result = selectQuery(criteria);
        result.first();
        file = result.getString("FilePath");
        return new File(file);
    }

    // извличане имената и ключовете на всички песни в базата от данни
    public static Hashtable<String, String> getAllSongsMap() throws Exception {
        Hashtable<String, String> songs = new Hashtable<String, String>();
        ResultSet result;

        result = selectQuery();
        if(result.first()) {
            do {
                songs.put(result.getString("Key"), result.getString("Name"));
            } while (result.next());
        }
        return songs;
    }

    // извличане имената и ключовете на всички песни
    // и представянето им в удобен за преглеждане HTML формат
    public static String getAllSongsOptionsHTML() throws Exception {
        Map<String, String> songs = ExampleSongs.getAllSongsMap();
        Iterator<Map.Entry<String, String>> iterator = songs.entrySet().iterator();
        StringBuilder html = new StringBuilder();

        html.append("<option value=\"\"></option>");

        while (iterator.hasNext()) {
            Map.Entry<String, String> song = iterator.next();
            html
                .append("<option value=\"")
                .append(song.getKey())
                .append("\">")
                .append(song.getValue())
                .append("</option>");
        }

        return html.toString();
    }
}
```

Клас “Texts”

```
package musicprediction;

public class Texts {
    // Съобщения
    public static String ERR_INVALID_REQUEST = "Невалидна заявка. Достъпът е отказан";
    public static String ERR_INVALID_MIDI_DATA = "Моля, въведете валиден MIDI файл.";
    public static String ERR_NUMBER_OUT_OF_RANGE = "Стойността на \"%s\" трябва да бъде
между %d и %d. Вие сте въвели %d.";
    public static String ERR_NO_SONG = "Не сте избрели песен.";

    public static String INFO_NUMBER_RANGE = "Въведете число между %d и %d.";

    // Други низове
```

```
    public static String OR = "или";

    public static String midiFile = "MIDI файл";
    public static String predictedLength = "Дължина на предсказаната мелодия (брой
тонове)";
    public static String trainingRepetitions = "Повторения за обучение";
    public static String windowSize = "Големина на прозореца (брой тонове)";
    public static String musicPrediction = "Предсказване на музика";
    public static String exampleSong = "Изберете примерна песен";

    // Метод за валидация на числа
    public static void validateNumber(String propertyName, int value, int min, int max)
throws Exception {
        if(value < min || value > max) {
            throw new Exception(String.format(
                    Texts.ERR_NUMBER_OUT_OF_RANGE,
                    propertyName, min, max, value
            ));
        }
    }
}
```

7.4. Приложение 4

JSP и HTML код на страницата, разработена като примерен интерфейс на приложението:

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="musicprediction.*" %>
<!DOCTYPE html>
<html>
<head>
<title><%= Texts.musicPrediction %></title>
<meta charset="utf-8"/>
</head>
<body>
<div>
<h1><%= Texts.musicPrediction %></h1>
<form method="post" action="Predict" enctype="multipart/form-data">
    <table>
    <tr>
        <td><label for="exampleSong"><%= Texts.exampleSong %>: </label></td>
        <td>
            <select name="exampleSong" id="exampleSong">
            <%= ExampleSongs.getAllSongsOptionsHTML() %>
            </select>
        </td>
    </tr>
    <tr>
        <td><%= Texts.OR %></td>
        <td></td>
    </tr>
    <tr>
        <td><label for="midiFile"><%= Texts.midiFile %>: </label></td>
        <td><input type="file" accept="audio/midi" name="midiFile" id="midiFile"/></td>
    </tr>
    <tr>
        <td><br/><br/><br/></td>
    </tr>
    <tr>
        <td><label for="predictedLength"><%= Texts.predictedLength %>: </label></td>
        <td><input name="predictedLength" id="predictedLength" value="100"/></td>
```

```
        <td><%= String.format(Texts.INFO_NUMBER_RANGE, ParamsModel.MIN_predictedLength,
ParamsModel.MAX_predictedLength) %></td>
    </tr>
    <tr>
        <td><label for="trainingRepetitions"><%= Texts.trainingRepetitions %>:
</label></td>
        <td><input name="trainingRepetitions" id="trainingRepetitions" value="60000"/></td>
        <td><%= String.format(Texts.INFO_NUMBER_RANGE, ParamsModel.MIN_trainingRepetitions,
ParamsModel.MAX_trainingRepetitions) %></td>
    </tr>
    <tr>
        <td><label for="windowSize"><%= Texts.windowSize %>: </label></td>
        <td><input name="windowSize" id="windowSize" value="10"/></td>
        <td><%= String.format(Texts.INFO_NUMBER_RANGE, ParamsModel.MIN_windowSize,
ParamsModel.MAX_windowSize) %></td>
    </tr>
    <tr>
        <td><br/></td>
    </tr>
    <tr>
        <td><input type="submit" value="Предскажи!"/></td>
        <td><input type="reset" value="Изчисти"/></td>
    </tr>
    </table>
</form>
</div>
</body>
</html>
```