Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Hálózati Rendszerek és Szolgáltatások Tanszék

Popovics Bence

POLIFONIKUS ZONGORAJÁTÉK AUTOMATIKUS LEJEGYZÉSE

KONZULENS

Dr. Rucz Péter

BUDAPEST, 2024

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott **Popovics Bence**, szigorló hallgató kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2024. 12. 15.

...…….

Popovics Bence

Összefoglaló

A zenei lejegyzés tonális zenei hangok hangmagasságának és idejének hallás után történő azonosítását jelenti. Többszólamú zenei környezetben, a szólamok számának ismerete nélkül összetett feladat lehet az egyszerre megszólaló hangok felismerése. Egy, a hangok automatikus azonosítására képes, polifonikus környezetben is hatékonyan működő eszköz segítséget nyújthat különböző zenei feladatokban.

Jelen dolgozat tárgyalja az egyszerre megszólaló tonális zenei hangok azonosításában rejlő kihívásokat, bemutat néhány determinisztikus megközelítést és betekintést nyújt egy eljárásba, ami klasszikus jelfeldolgozási lépéseket és gépi tanulást használva ad megoldást a problémára. Az emberi hallás működését utánzó módszer gammatone szűrőket, adaptív oszcillátorokat és egy Transformer modellt alkalmaz. A dolgozat végén bemutatott Python alkalmazás hozzáadott információ nélkül, jó hibaaránnyal képes azonosítani a megszólaló hangokat zongorajátékról készült hangfelvételeken. Az alkalmazás a hangfelvételeket MIDI fájlokká alakítja, amely széles körben használt, zenei információk hordozására szolgáló formátum, kotta előállítására vagy szintetizátorok megszólaltatására is alkalmas.

A dolgozat az eljárás hatékonyságának mérésével és továbbfejlesztési lehetőségek tárgyalásával zárul.

Abstract

In music, transcription is the practice of recognizing the pitch and time of musical notes through hearing. In polyphonic music, it could be challenging to identify all the simultaneously played notes without knowing the number of the voices. A tool, which can automatically recognize the notes and which works efficiently in polyphonic music, could be helpful in different musical tasks.

This thesis discusses the challenges of the identification of simultaneously played notes, shows a few deterministic approaches and provides an insight of a method, which gives a solution for the problem using classical signal processing and machine learning. The process imitates the operation of the human hearing system with gammatone filters, adaptive oscillators and a Transformer model. The Python application presented in this work can efficiently recognize the notes on a piano recording without further given information. The application converts the recordings into MIDI, a broadly used file format for carrying musical information. This format is convenient for creating musical sheets or playing on different synthesizers.

The evaluation of the method and ideas for further development are presented in the end of this thesis.

1 Bevezetés

1.1 Motiváció

A zenei tanulmányok, de különösen a jazz-oktatás szerves részét képezik megírt és improvizált zenei részletek hallás után történő lejegyzése (*transzkripció* készítés) és megtanulása, ami nagyban fejleszti többek közt a hallást, a zenei gondolkodást, a stílus- és ritmusérzéket. Zongoristaként régóta foglalkoztat a harmóniák világa és a különböző módon megszólaltatott akkordok színeivel való játék, melyek esetében a használt hangok hallás után történő pontos beazonosítása az egyszólamú dallam leírásánál összetettebb feladat lehet. A témaválasztásomat egy olyan alkalmazás létrehozása motiválta, amely praktikus segítséget nyújthat polifonikus zongorajáték pontos lejegyzésére.

A megvalósítandó feladat egy olyan szoftver létrehozása volt, amely szóló zongora játékáról készült hangfelvételeken képes automatikusan beazonosítani, hogy a hangszer mely billentyűit mikor ütötte le a játékos. Az alkalmazás felé támasztott fontos elvárás, hogy többszólamú esetben is jól használható eredményeket adjon, a szólamok számának előzetes ismerete nélkül. Szempont volt továbbá a felhasználóbarát megközelítés, ezért a szoftver grafikus felülettel rendelkezik, a lejegyzés eredményét pedig egy, a zenészek számára könnyedén és sokoldalúan használható fájlformátumban adja vissza.

1.2 A dolgozat felépítése

Munkámban először a feladatban rejlő kihívásokat és az automatikus lejegyzés néhány lehetséges módját ismertetem. Az általam választott megoldás két, jól elkülöníthető szakaszra bomlik. Az elsőben hagyományos jelfeldolgozási lépésekkel és adaptív oszcillátorok használatával feldolgozom a hangjeleket, a kapott formátumú adatot pedig a második, nemdeterminisztikus részben egy neurális hálózat segítségével értékelem ki. A dolgozatban részletezem a használt metódusokat és a feladat megoldásának menetét, végül értékelem és összegzem az eredményeket, valamint kitérek a továbbfejlesztési lehetőségekre is.

2 Zenei lejegyzés

2.1 A kotta

A zenei lejegyzés klasszikus formája a hangfelvételen rögzített vagy élőben előadott zenei részlet hallás után történő lekottázása. A technikát zeneszerzők, hangszerelők, jazz muzsikusok és népzenei gyűjtők is sűrűn használják. A készség gyakorlása nagyban fejleszti a zenei hallást, ezért része a zenei képzéseknek is.

A kotta egy viszonylag standard, mégis organikusan fejlődő jelölésrendszer, amely optimális eszköz zenei instrukciók széles skálájának – úgy mint forma, hangnem, tempó, metrum, szólamok, hangmagasság, megszólaltatás hossza, ritmus, dinamika, artikuláció, hangsúly, kötések, előadási mód, szöveg stb. – tömörített, a zenészek számára világos közlésére.[1]

Ahhoz, hogy egy hangfelvételből szoftver által automatikus módon létrehozott, könnyen olvasható kottát készítsünk, számtalan hozzáadott információra van szükség, vagy ezek hiányában nagy méreteket ölthet a megoldandó feladat komplexitása. Ha például nem ismert a tempó és a metrum, lehetetlen a hangjegyeket időben értelmes módon leírni. A megoldásomban automatikus kottagenerálás helyett a hangfelvételeket MIDI fájlokká alakítom.

2.2 MIDI

A szintetizátorok, dobgépek és digitális zenei szerkesztőprogramok (*Digital Audio Workstation*, DAW) világában a zenei információk tömörített rögzítésére és továbbítására nyújt hatékony megoldást a *Musical Instrument Digital Interface* (MIDI) protokoll. Az üzenet alapú szabvány a ’80-as évek elején jelent meg és mára ipari sztenderddé vált a digitális hangszerek valós idejű vezérlése, több hangszer szinkronizálása, a hangszerelés vagy akár a színpadtechnika területén, de

[1] Fontos megjegyezni, hogy a kotta kizárólag az európai klasszikus zenéhez visszavezethető zenei hagyományokban számít hasznos eszköznek, eltérő hangrendszerekben működő, komplex díszítéseket alkalmazó vagy más természetű jelöléseket igénylő zene precíz leírására nem alkalmas.

videojátékoknál és a robotikában is használják, megtalálható minden mobiltelefonban és személyi számítógépben.

A protokoll nem hangot továbbít, hanem a hang megszólaltatásához szükséges információt a céleszköz számára, amihez különböző típusú parancsokat definiáltak. A két legfontosabb és legsűrűbben használt üzenettípus a hang megszólalását és elhallgatását indikáló *Note On* és *Note Off* parancsok. Léteznek egyéb típusú parancsok is (pl. *Key Pressure*, *Program Change*, *Pitch Bend*), amelyek színes palettát biztosítanak szintetizátorok vezérlésére, a zongorajáték lejegyzése szempontjából viszont csak az első két típus lesz releváns.

Minden MIDI üzenetre igaz, hogy egy parancsbájtból és a hozzátartozó adatbájtokból áll. A *Note On* és *Note Off* parancsok után a szabvány szerint két-két adatbájt következik, melyek közül az első a hangot azonosítja (*note*), a második pedig a megszólalás hangosságát (*velocity*). A *Note Off* esetében a *velocity* érték nem minden esetben releváns, de néhány hangszínnél (pl. csembaló pengetőzörej hangosságának jelzése) hasznos lehet.

Az egyszerű architektúra érdekében a protokoll a parancs- és adatbájtokat kezdő bitjük szerint különbözteti meg: a parancsok 1-gyel, az adatok pedig 0-val kezdődnek. A parancsok esetében az első 4 bit felel a parancs típusának azonosítására, az utolsó 4 pedig a cél MIDI csatornát jelöli – ez utóbbi következményeként egy forrás egyszerre 16 különböző hangszín megszólaltatására képes, akár különböző eszközökön. A *Note Off* a 8-as, a *Note On* pedig a 9-es indexet kapta. A *note* és a *velocity* esetében 7-7 bit marad az adat tárolására, tehát 128 különböző érték. Ez mindkét esetben elegendőnek bizonyul: félhangokkal számolva ennyi érték több, mint 10 oktávnyi hang azonosítását teszi lehetővé a C-1 és a G9 billentyűk között (az ember által hallható tartomány kevesebb, mint 10 oktáv), a hangerősség esetében pedig megfelelő felbontást enged meg. Vannak MIDI eszközök, amik nem használnak *Note Off* parancsot, ezeknél olyan, ekvivalens jelentéstartalmú *Note On* üzenet jelzi a hang elhallgatását, amiben 0 a *velocity* értéke. Az első csatornán, egy elég hangosan (101-es (65H) *velocity*) megszólaló A4 hang (69-es (45H) *note*) *Note On* üzenete az alábbi lesz:

90H 45H 65H

A szabvány nem csak valós idejű vezérlést tesz lehetővé, a MIDI fájlformátum segítségével mód nyílik MIDI üzenetek rögzítésére, szerkesztésére és többszöri lejátszására is. [1]

A MID vagy MIDI kiterjesztésű fájlformátumban az üzenetek egy vagy több csatornára bontva (*track*), végrehajtási sorrendben helyezkednek el. A fájlban lévő üzenetekhez egy-egy relatív Δt időbélyeg társul, ami minden esetben az előző üzenet óta eltelt időmennyiséget jelöli. $\Delta t = 0$ esetén az üzenet késleltetés nélkül, az előzővel azonos időben kerül végrehajtásra, lehetővé téve egyszerre több hang megszólaltatását. Δt -nek *tick* a mértékegysége, ami egy, a rögzített zeneszám tempójához igazodó érték. A dolgozatom 4.4. részében részletesen ismertetem a *tick* – másodperc konverziót.

Egy MIDI fájl lehetőséget biztosít az üzenetekből álló adat mellett dalszöveg, tempó információ és más típusú meta-adatok tárolására is. Szinte minden ma is használt kottagrafikai szerkesztőszoftver (*Sibelius, MuseScore, Finale, Dorico* stb.) képes MIDI üzeneteket kottajelekké alakítani, ezzel nagyban felgyorsítva egy-egy zenei produkció létrejöttét.

Dolgozatomban egy olyan szoftver létrehozását mutatom be, ami zongorajátékról készült WAV formátumú audio fájlból automatikusan állítja elő a hangok leütésének MIDI reprezentációját. Ezt követően számos lehetőség nyílik a felhasználó számára a kimenet szerkesztésére, felhasználására vagy lejátszására.

2.3 A polifonikus környezet kihívásai

A többszólamú zenei jelek feldolgozása sok szempontból kihívást jelent. Azokat a zenei hangokat, amelyekhez érzett hangmagasságot tudunk rendelni, tonális hangoknak nevezzük, a dallam, illetve a harmóniák lejegyzésénél ilyen hangok detektálása a feladat. A tonális zenei hang egyik jellemzője, hogy az alapfrekvenciáján és ennek egész számú többszörösein megszólaló szinuszjelekből áll.[2] Ezek amplitúdóinak aránya határozza meg a megszólaló hang hangszínét, ami alapján például a hangszerek hangját is meg tudjuk különböztetni egymástól. A zongorahang esetében jellemzően az alaphang és az első felharmonikus frekvenciáján jelennek meg a

[2] A valóságban a tiszta szinuszjelhez képest a harmonikus komponensek amplitúdója és frekvenciája a periódusuknál jóval nagyobb időskálán nézve lassan változhat (pl.: lecsengés, tremoló vagy vibrato effektusok).

legnagyobb amplitúdók, ezek pontos aránya hangszertípustól és a billentés erejétől is függhet. Az első felharmonikus frekvenciája egybeesik a megszólalt hanghoz képest egy oktávval magasabb hang alapfrekvenciájával, ami miatt jelentkezhetnek úgynevezett oktávhibák a detektálás során.[3] Egy lejegyző alkalmazásban ez a tulajdonság nehezítheti a pontos kiértékelést, hiszen pusztán az alapfrekvenciák amplitúdóját vizsgálva a ténylegesen megszólalt hang mellett fals eredményként az egy oktávval magasabb vagy mélyebb hangot is detektálhatjuk, vagy épp, hogy figyelmen kívül hagyhatjuk valamelyiket, ha egyszerre szólalnak meg.

Ezen túl mind idő-, mind pedig frekvenciatartományban megfelelő felbontásra van szükség, hogy a megszólaló harmonikus komponensek frekvenciái pontosan mérhetők és elkülöníthetőek legyenek, valamint hogy a hangok megszólalásának idejét is precízen követni tudjuk.

A következőkben bemutatok néhány lehetséges eljárást többszólamú zenei jelek automatikus lejegyzésére.

2.4 Automatikus eljárások

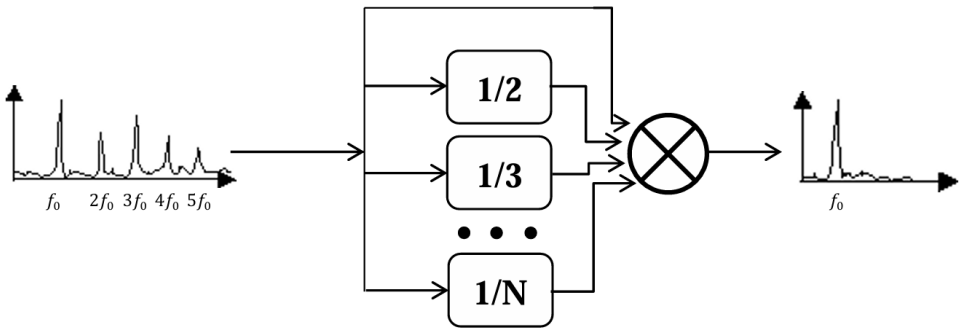
2.4.1 Akkordtípus felismerése HPS módszerrel

Várfi László egy olyan alkalmazásról ír a szakdolgozatában [2], ami az egyszerre megszólaló hangok számának ismeretében képes valós hangfelvételeken felismerni a megszólaló akkordok alaphangját és típusát. A rendszer működését különböző gitárokon, zongorán és énekszólamokon is kipróbálta.

Az eljárás rögzített méretű, csúszó időablakokon FFT (*Fast Fourier Transform*) alkalmazásával kiértékeli a jel spektrumát, amin a HPS (*Harmonic Product Spectrum*) módszer, valamint frekvenciapontosítási és szűrési lépések többszöri alkalmazásával meghatározza az akkordban megszólaló alaphangok osztályát. A rendszer a hangok osztályának ismeretében végül közli az egymást követő akkordokat a felhasználóval.

[3] A valós húrok ütéssel meggerjesztett részhangjai az inharmonicitás jelensége miatt nincsenek tökéletesen harmonikus viszonyban egymással. A zongora is ezzel a hangkeltési mechanizmussal működik, viszont az inharmonicitásból adódó eltérés jelen alkalmazás szempontjából elhanyagolható mértékű.

Általában nemtriviális feladat tonális zenei hangok alapfrekvenciájának automatikus detektálása, mert előfordulhat, hogy valamelyik felharmonikus amplitúdója dominál a hangban, emiatt a spektrumon futtatott egyszerű maximumkeresés nem elegendő. A HPS módszer a mintavételi frekvencia 2 és N közötti egész számú többszörösein újramintavételezi a jelet, és az eredeti spektrumot skálázza a túlmintavételezett jelek spektrumaival. A felhangrendszer miatt ez azt eredményezi, hogy az alapfrekvenciát az első $N-1$ felharmonikus amplitúdójával súlyozzuk, a többi harmonikus komponens pedig erős csillapodást szenved a magasabb frekvenciájú és alacsonyabb amplitúdójú komponensek miatt. N megfelelő megválasztásával a 2.1. ábrán látható módon átalakított spektrum maximuma az alapfrekvencia környékére kerül.



2.1. ábra: A HPS módszer működésének vázlatos rajza. A szorzás előtti művelet a spektrum újramintavételezés miatti „összenyomását" jelöli [2]

A diszkrét spektrum felbontása az időablak méretétől függ (jelen megvalósításban ~10 Hz), ebből fakadóan a detektált alapfrekvencia hibája alacsonyabb frekvenciákon akár hangtévesztést is okozhat. A jelet a HPS módszerrel detektált $\hat{f}_0$ mértékében egyoldali frekvenciamodulációval nulla környékére tolva, a kapott jelet pedig egy aluláteresztő szűrővel szűrve egy komplex harmonikus jelet kapunk, melynek frekvenciája a valós és a detektált alapfrekvencia közötti Δf különbség lesz. Diszkrét időben a kapott jel minden szomszédos mintájára teljesül az

$$x_n = Ae^{j2\pi\Delta f\Delta t}x_{n-1}$$

egyenlőség, ahol A a jel amplitúdója és Δt az időfelbontás. Ezt a tulajdonságot kihasználva a jel mintáira felírt,

$$\begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{bmatrix} Ae^{j2\pi\Delta f\Delta t} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

túlhatározott egyenletrendszer megoldásából kiszámítható Δf és így a hangjel valódi, $f_0 = \hat{f}_0 + \Delta f$ alapfrekvenciája.

Ahhoz, hogy több alaphangot is detektálhassunk, ki kell szűrnünk a megtalált alapfrekvenciát a pozitív és a negatív frekvenciatartományokban. Ezt követően ismét elvégezhető a fenti műveletsor, minek után az akkord egy másik hangjának alapfrekvenciáját találjuk meg. Az eljárást annyiszor ismételjük, amennyi az akkordokban keresendő hangok előre meghatározott száma.

Az alapfrekvenciák ismeretében megállapítható a hangok osztálya, ami alapján feltölthető egy 12 elemű vektor (*kromavektor*) a detektált alaphangok intenzitásával a hangosztálynak megfelelő indexeken. A kromavektor indexei az egyenletes hangolás 12 félhangját jelölik C-től H-ig. A kiszámított kromavektorokat az időablakok szerint egymás után rendezve és az értékek változását időtartománybeli átlagolással simítva jól követhetők a detektált hangok az akkordváltások közben is. Az akkordok felismerése a detektált értékek és előre rögzített kromavektor sablonok összehasonlításával történik.

Az eljárás jó hibaaránnyal ismeri fel a rögzített szólamszámú akkordokat, viszont a valódi hangmagasságok meghatározására, vagy egyazon felvételen előforduló, a hangok számában különböző akkordok pontos felismerésére nem alkalmas.

2.4.2 NMF algoritmus dallam lejegyzésére

Dallamok automatikus lejegyzésének egy lehetséges módja a hangjel nemnegatív mátrixfaktorizálást (NMF) használó algoritmus segítségével történő feldolgozása. A módszerről Szemerey Helén ír TDK dolgozatában [3], akinek sikerült egy- és kétszólamú zenei jelek esetében is jól működő feldolgozási rendszert fejlesztenie.

Az eljárás a vizsgált zenei részlet spektrogramjának abszolút értékéből indul ki, melyet egy nemnegatív $\mathbf{V} \in \mathbb{R}_+^{m\times n}$ mátrix ír le. Keressük a $\mathbf{W} \in \mathbb{R}_+^{m\times r}$ és $\mathbf{H} \in \mathbb{R}_+^{r\times n}$ mátrixokat, melyek szorzata jól közelíti **V**-t:

$$\mathbf{V} \approx \mathbf{WH}.$$

W oszlopait r darab spektrum alkotja. Ezek az algoritmus bázisai, melyek – r értékének jó megválasztásával – megfeleltethetők a zenei részletben megszólaló hangoknak. **H** sorai az egyes spektrumok időbeli aktivitását jelölik az eredeti jelben. Mindkét mátrixra teljesül, hogy nemnegatív értékek szerepelnek bennük, mely tulajdonságot kihasználva jól megközelíthető az eredeti jel a két mátrix kombinációjaként úgy, hogy az adott pillanatban megszólaló hangok egyértelműen megfeleltethetők a bázis elemeivel. Az algoritmus az eredeti **V** és a reprodukált **WH** mátrixok közötti euklideszi távolság négyzeteként definiált költségfüggvény minimalizálásával állapítja meg a bázisok aktivitását minden időpillanatban, figyelembe véve a zenei hangok lassú változását és az eredő spektrum előállításához szükséges lehető legkevesebb báziselem használatának igényét.

A bázis egyedi hangok spektrumait tartalmazza, melyekből könnyedén megállapítható, hogy mi a hozzájuk rendelt hang magassága. Az algoritmus hatékonyságában kulcsszerepet játszik a bázis – és ez által a **W** és **H** mátrixok – méretének helyes megválasztása. Kisméretűnek választott bázis esetén az algoritmus képtelen minden megszólaló hangot detektálni, nagy elemszám mellett pedig redundancia lép fel az egyes hangok reprezentációi között, ami a további feldolgozást bonyolítja. Ha az NMF algoritmust használva jó közelítéssel meg szeretnénk állapítani, hogy az egyes pillanatokban mely zenei hangok szólaltak meg, előre ismernünk kell az egyedi hangok számát a vizsgált zenei részletben, ami erős korlátozás.

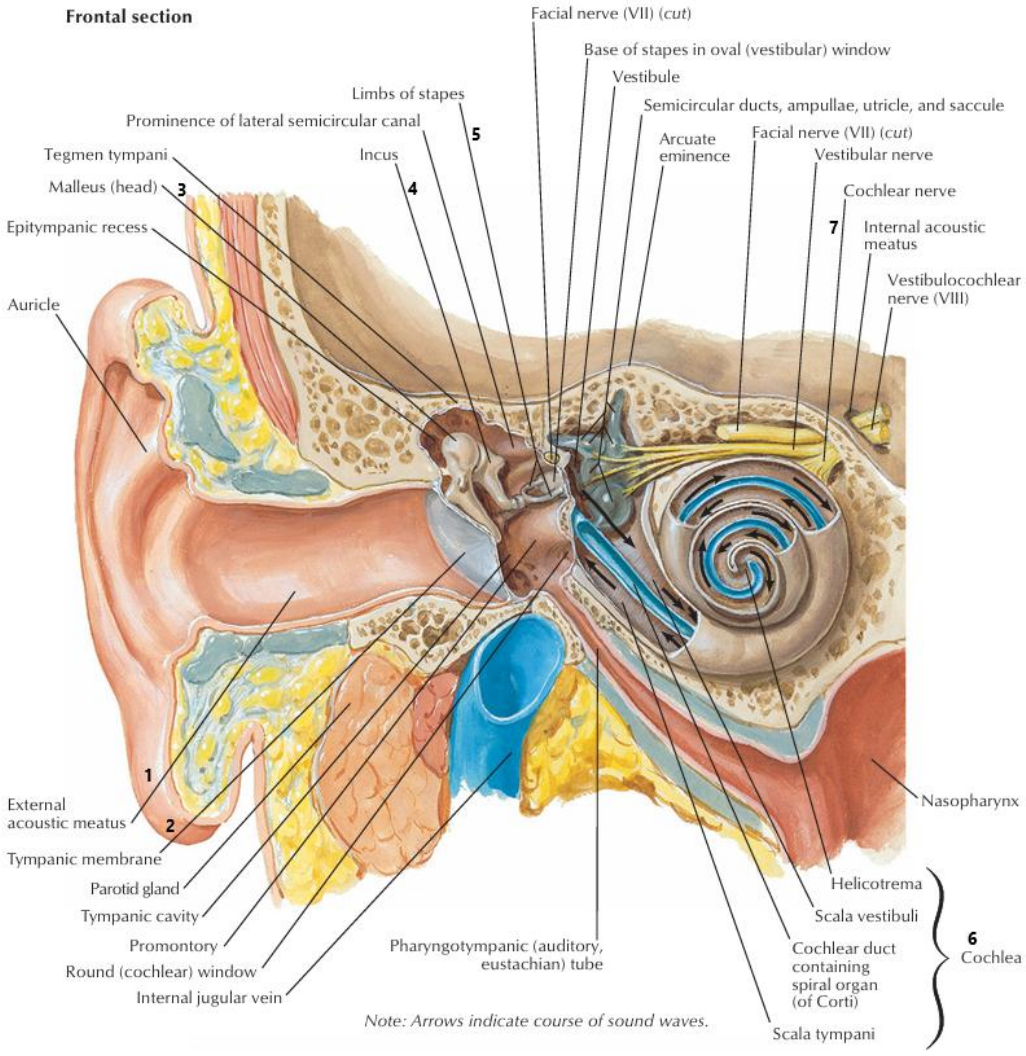
2.4.3 Nemdeterminisztikus megközelítés

A diplomamunkám keretében olyan alkalmazás elkészítését tűztem ki célul, amely képes a zongorajátékról készült hangfelvételen adott időpillanatban egyszerre megszólaló hangok számának ismerete vagy egyéb hozzáadott információ nélkül, jó hibaaránnyal megállapítani, hogy mely billentyűk mikor lettek leütve.

Klapuri és Davy könyvében [4] találkoztam Marolt [5] munkájával, amelyben a témámhoz hasonló követelményeknek tudott sikeresen megfelelni. Ő a hangjel feldolgozását az emberi hallás működését modellező lépésekkel végezte el, a hangok kiértékeléséhez neurális hálózatokat használt, kimenetként pedig egy MIDI fájlt állított elő, mely alapján értékelhető volt a rendszer működése. Az általa leírt jelfeldolgozási metódust korszerű gépi tanulási technológiákkal vegyítve álltam neki a munkának. Az eljáráshoz szükséges lépéseket a következő részben ismertetem.

3 A választott eljárás bemutatása

A választott eljárást mélyen inspirálta az emberi hallás működése, ezért a feldolgozási lépések ismertetése előtt érdemes röviden bemutatni az emberi hallószerv felépítését, melynek vázlatos rajza a 3.1. ábrán látható. A hallójáraton (1) keresztül a dobhártyán (2) és a középfül csontjain (3, 4, 5) át jutnak el a belső fülig a rugalmas közegben (általában levegőben) haladó hullámok. A belső fülben találjuk a csigát (6), amely kiterítve ~35 mm hosszú és több ezer belső- és külső szőrsejt található benne. A csigában lévő folyadék a beérkező hullám hatására rezgésbe jön, ami így gerjesztést gyakorol a kialakult csomópontokhoz közel található szőrsejtekre. A gerjesztett sejtek a hallóidegeken (7) keresztül továbbítják az információt az agy felé, ahol létrejön a hangélmény.



3.1. ábra: Az emberi hallószerv felépítése [6]

3.1 Gammatone szűrőbank

A csiga belső struktúrája miatt a szőrsejtekre úgy is tekinthetünk, mint keskenysávú sávateresztő szűrők összességére, hiszen a fülbe jutó hullámok közül azok gyakorolnak hatást egy szőrsejtre, amelyek egyik csomópontja a sejt közelében jelenik meg. Erre a tulajdonságra épít a hallás működését leíró Patterson-Holdsworth-féle modell [7], amely nyolcadfokú gammatone szűrők csoportja segítségével bontja a komplex hangot szűkebb csatornákra.

A szűrők megvalósításához Slaney eljárását [8] alkalmaztam, aki négy darab másodfokú digitális szűrő összekapcsolásával hozta létre a nyolcadfokú gammatone szűrőket. A négy szűrő együtthatóit az alábbiak szerint határozta meg:

$$B_0 = -2, B_2 = 0, A_0 = -2, A_1 = \frac{4C}{E}, A_2 = -\frac{2}{E^2},$$

$$B_{11} = \frac{2T_sC}{E} + \frac{T_sS\sqrt{3+2^{\frac{2}{3}}}}{E},$$

$$B_{21} = \frac{2T_sC}{E} + \frac{T_sS\sqrt{3-2^{\frac{2}{3}}}}{E},$$

$$B_{31} = \frac{2T_sC}{E} - \frac{T_sS\sqrt{3+2^{\frac{2}{3}}}}{E},$$

$$B_{41} = \frac{2T_sC}{E} - \frac{T_sS\sqrt{3-2^{\frac{2}{3}}}}{E},$$

ahol

$$T_s = \frac{1}{f_s},$$

$$C = \cos(2\pi f_c T_s),$$

$$S = \sin(2\pi f_c T_s),$$

$$E = \mathrm{e}^{bT_s}.$$

B_{i1} az i. szűrőhöz tartozó B_1 együttható, f_s a hangfelvétel mintavételi frekvenciája, amin a szűrést végezni szeretnénk, f_c a szűrő középfrekvenciája és b a sávszélessége. Ezeket felhasználva így áll elő az i. szűrő:

$$H_i(z) = \frac{B_0 z^2 + B_{i1} z + B_2}{A_0 z^2 + A_1 z + A_2}.$$

A négy szűrő egymás utáni alkalmazásával valósul meg egy gammatone szűrés.

A hallást modellező szűrőbankok esetében a szomszédos szűrők középfrekvenciája közötti távolság közel exponenciálisan nő, és a csillapításuk -3 dB-es értékei között meghatározott sávszélességük függ a középfrekvenciától. Az általam használt szűrőbanknak nem kell a teljes hallható tartományt lefednie és a szőrsejtek számánál jóval kevesebb szűrő is elegendő, hiszen nem a hallás működésének pontos reprodukálása a cél. Ahhoz, hogy bármely $f_{\text{low}} < f_{\text{high}}$ közötti frekvenciatartományt képesek legyünk tetszőleges $n > 1$ darab szűrővel felosztani, miközben a modell elé támasztott elvárások sem sérülnek, a Slaney-féle eljárást ki kell egészíteni egy SF skálázási faktorral. A szűrők középfrekvenciái és sávszélességei így a következő módon állapíthatók meg:

$$SF = Q_{\text{Ear}} \frac{\log\left(f_{\text{high}} + Q_{\text{Ear}} BW_{\min}\right) - \log(f_{\text{low}} + Q_{\text{Ear}} BW_{\min})}{n - 1},$$

$$ERB(f) = \frac{f}{Q_{\text{Ear}}} + BW_{\min},$$

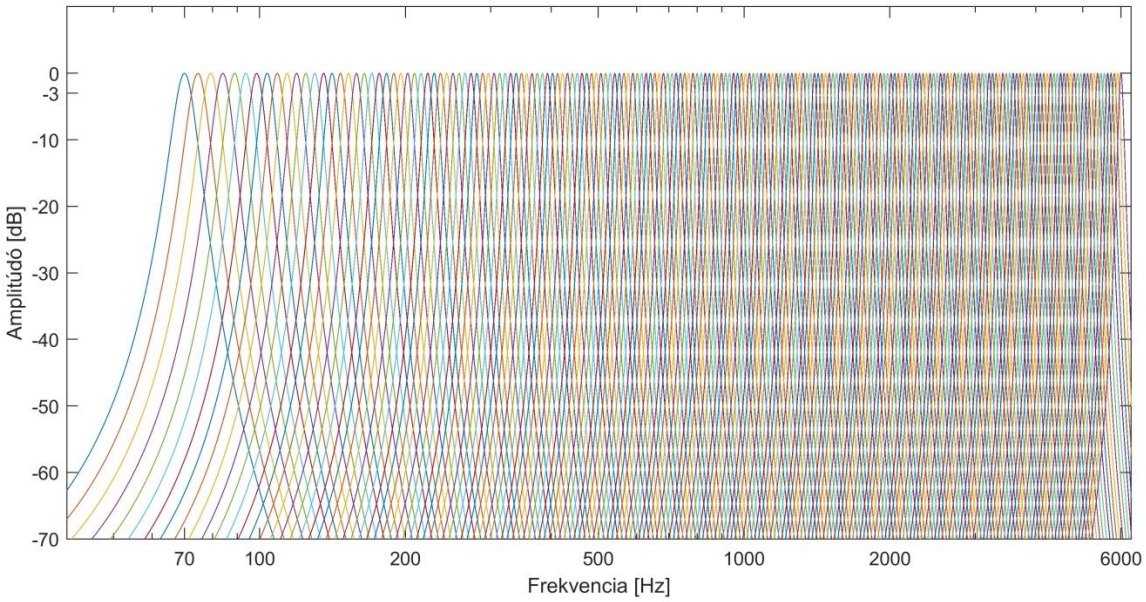
$$f_{ci} = -Q_{\text{Ear}} BW_{\min} + \frac{f_{\text{high}} + Q_{\text{Ear}} BW_{\min}}{\exp\left(\frac{SF(n - i - 1)}{Q_{\text{Ear}}}\right)},$$

$$b_i = 1.019 ERB(f_{ci}) 2\pi SF \frac{n + 1}{n},$$

ahol $i \in [1, n]$ a szűrő indexe és $ERB(f)$ az *Equivalent Rectangular Bandwidth* függvény, ami a középfrekvencia-sávszélesség viszonyt írja le. Az emberi halláshoz igazodó együtthatói: $Q_{\text{Ear}} = 9.26449$, $BW_{\min} = 24.7$ Hz.

A zongorajáték lejegyzésének Marolt-féle megoldása 200 gammatone szűrőt használt, melyek középfrekvenciái 70 és 6000 Hz között helyezkednek el. Ez a kiosztás a 88 billentyűs hangszer legmélyebb alapfrekvenciái kivételével az összes fontosabb harmonikus komponenst kezeli, a szűrők száma miatt pedig úgy bontja fel a vizsgált frekvenciatartományt, hogy legalább félhangonként szeparálja a komponenseket. Mivel a zongorán nincs lehetőség két félhang közötti hangok megszólaltatására, ez a szűrőszám elegendőnek tekinthető. A 70 Hz alatti komponenseket azért hagyták ki a későbbi feldolgozásból, mert alacsony frekvenciájuk miatt túl nagy késleltetéssel

lehetett csak ellenőrizni a szűrés utáni jel periodicitását. Ezeket a döntéseket megalapozottnak tartottam, ezért én is ezekkel a beállításokkal végeztem el a hangfelvételek szűrését, így 200 csatornára bontva a jelet. A 3.2. ábrán látható a használt gammatone szűrők átviteli karakterisztikája. A kritikus sávok középfrekvenciájának kiosztása nem teljesen logaritmikus a kisfrekvenciás tartományban, ez a tulajdonság a szűrőknél is megfigyelhető.



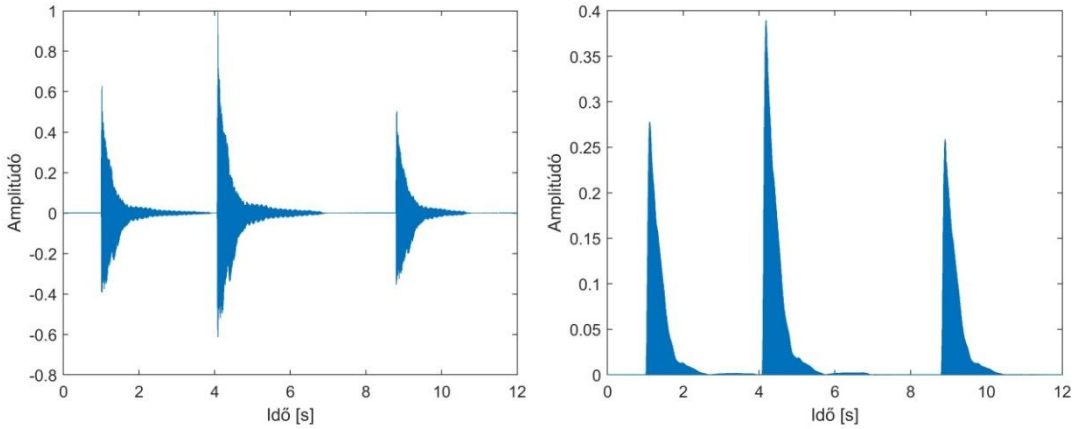
3.2. ábra: A 200 gammatone szűrő átviteli karakterisztikája

3.2 Szőrsejt modellje

A szűrőbank méretezése miatt a szűrést követően minden csatornán vagy kicsiny amplitúdójú sávszűrt zajt kapunk, vagy jó közelítéssel periodikus jelet. Tonális zenei hangok felismerése a cél, így ha képesek lennénk minden időpillanatban dönteni az adott csatornán jelentkező jel periodicitásáról, az tovább tisztítaná a hasznos jelet. Egy szűrő kimenete egy szőrsejt gerjesztőjelének is tekinthető, amelyből előáll a hallóideg felé továbbított jel. A szőrsejtek működését a Meddis-féle modell [9] írja le nagy pontossággal, melynek lényege, hogy a mozgásba hozott szőrsejt bizonyos mértékben csökkenti a jel dinamikatartományát (szaturál) és a beérkező hullám periódusához igazodóan tüzel, aminek szintje az amplitúdótól is függ. A tüzelések tüskéi jól használhatóak a periodicitás mérésére.

A Meddis-féle modellnél egyszerűbb eljárást alkalmaztam. A gammatone szűrés előtt 1 és -1 közé normalizáltam a hangfelvételt, a szűrést követően egyenirányítottam a

jelet, majd az így kapott félhullámokon megkerestem a lokális maximumpontokat. Az előállt gerjesztőjel 1 értékű tüskéket tartalmaz a lokális maximumok helyén, máshol 0. Minden tüske között annyi idő telik el, amekkora aktuálisan a csatornán lévő hullám periódusa, ezért az így kapott gerjesztőjel alkalmas a periodicitás ellenőrzésére. A 3.2. ábrán egy hangjel hullámformája látható normalizálást és egyenirányítást követően.



3.2. ábra: Egy billentyű háromszori leütéséről készült felvétel normalizált hullámformája (bal) és a szűrést, valamint egyenirányítást követően az egyik csatorna félhullámai (jobb)

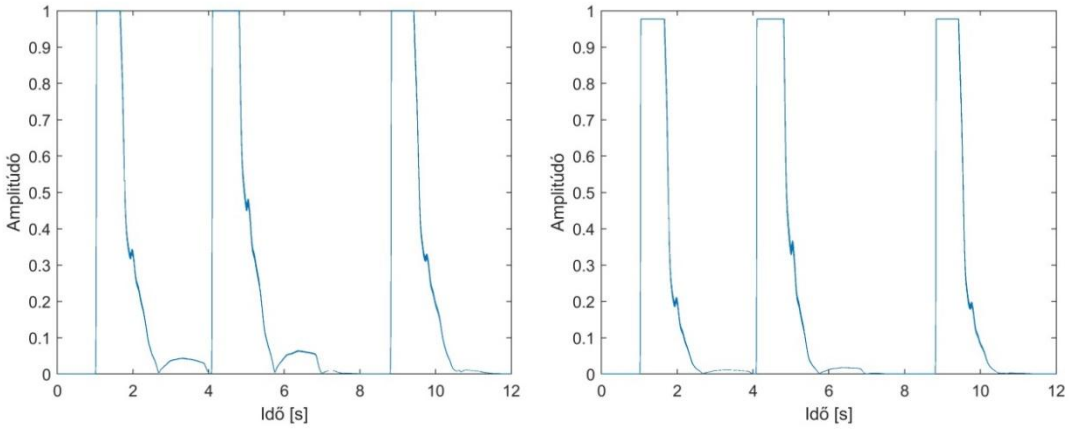
Hogy a szőrsejt amplitúdóérzékenységét is használjam, exponenciális átlagolással kiszámítottam az egyenirányított jel amplitúdóburkolóját. Időátlagolási konstansnak a $\tau = \frac{1}{f_c}$ értéket választottam, ahol f_c az aktuális csatorna szűrőjének középfrekvenciája. A burkoló előállítása a gyakorlatban egy elsőfokú szűrő kétszeri alkalmazásával történik, melynek együtthatói:

$$B_0 = 1 - e^{-\frac{\Delta t}{\tau}}, B_1 = 0, A_0 = 1, A_1 = e^{-\frac{\Delta t}{\tau}},$$

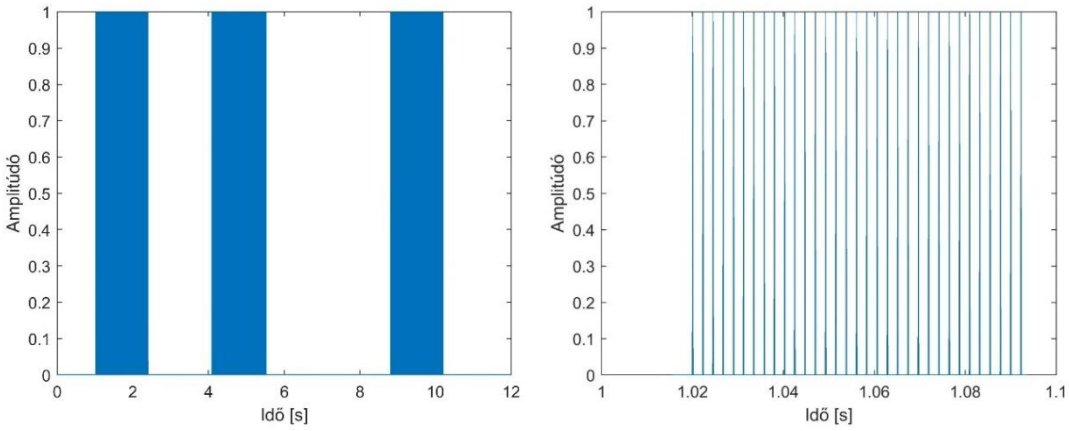
ahol $\Delta t = \frac{1}{f_s}$ a felvétel időbeli felbontása.

A szaturálást úgy valósítottam meg, hogy az előállt burkolót vágtam egy konstans érték fölött és normalizáltam az így limitált jelet, ezzel csökkentve a dinamikai különbségeket. Hogy az alacsonyabb amplitúdójú hasznos jelek erősítése mellett csillapítani tudjam a zaj szintjét, a $sgm(x) = \frac{1}{1+\exp(-5(x-0.25))}$ szigmoiddal is súlyoztam a burkolót, aminek hatására az alacsonyabb amplitúdójú komponensek jobban csillapodtak. A gerjesztőjelek kimenetét nullának vettem azokon a helyeken, ahol az így kapott burkoló szintje nem ér el egy küszöbértéket. Vágási konstansnak és

küszöbértéknek is 0.02-t választottam a megoldásomban. A 3.3. ábrán egy csatorna amplitúdóburkolója látható a szaturálás valamint a szigmoid alkalmazása után, a 3.4. ábrán pedig a csatornán létrehozott gerjesztőjel figyelhető meg.



3.3 ábra: Egy csatorna amplitúdóburkolója a szigmoid alkalmazása előtt (bal) és után (jobb)



3.4. ábra: A 3.3. ábrán látható csatornán előállított periodikus gerjesztőjel (bal) és nagyított változata a gerjesztés tüskéivel (jobb)

3.3 Periodicitás mérése adaptív oszcillátorokkal

A csatornákon létrejött gerjesztőjelek periodikusságának ellenőrzésére Large-Kolen oszcillátorokat [10] használtam. Ezeket eredetileg zenei metrum és ritmikai egységek azonosítása céljából hozták létre, melyek jelentősen nagyobb periódussal rendelkeznek, mint a tonális zenei hangok komponensei, viszont jelen problémára is jó megoldást nyújtanak.

Mindegyik csatornához társítható egy oszcillátor, melynek kezdő fázisa nulla, kezdeti periódusa $\frac{1}{f_c}$, a periódus változásának korlátai pedig a szűrő sávhatárai reciprokának megfelelő értékek. Az oszcillátorok bemenetként megkapják a hozzájuk tartozó csatorna gerjesztőjelét és a tüskék érkezésétől függően képesek változtatni a periódusukat és a fázisukat, hogy minél pontosabban szinkronba kerüljenek a gerjesztőjellel – ott legyen maximum környéki az oszcillátor kimenete, ahol a gerjesztés érkezik. A periódusok változásának korlátozása miatt nincs megengedve a távolabbi komponensekre való szinkronizálás.

Egy Large-Kolen oszcillátor működése az alábbi függvényekkel írható le, melyek sorrendje a szinkronizálásnál használt számítási lépések sorrendjének is megfelel:

$$a(t) = \cos\big(\varphi(t)\big) - 1, \tag{1}$$

ahol $a(t)$ az úgynevezett aktivációs függvény, $\varphi(t)$ pedig az oszcillátor fázisa.

$$o(t) = 1 + \tanh\left(\gamma a(t)\right), \tag{2}$$

ahol $o(t)$ az oszcillátor kimenete, γ pedig egy pozitív konstans, melynek értékével egyenesen arányos az oszcillátor kimenetének időérzékenysége. A szinkronizáció közben egy Large-Kolen oszcillátor belső állapotának változásait a következő egyenletek adják:

$$\Delta t_0 \;=\; \eta_1 s(t) p \,\mathrm{sech}^2(\gamma a(t)) \sin(\varphi), \tag{3}$$

$$\Delta\Omega \;=\; \eta_2 s(t) \,\mathrm{sech}^2(\gamma a(t)) \sin(\varphi) \frac{\partial p}{\partial \Omega}, \tag{4}$$

$$\Omega \;=\; \Omega + \;\Delta\Omega, \tag{5}$$

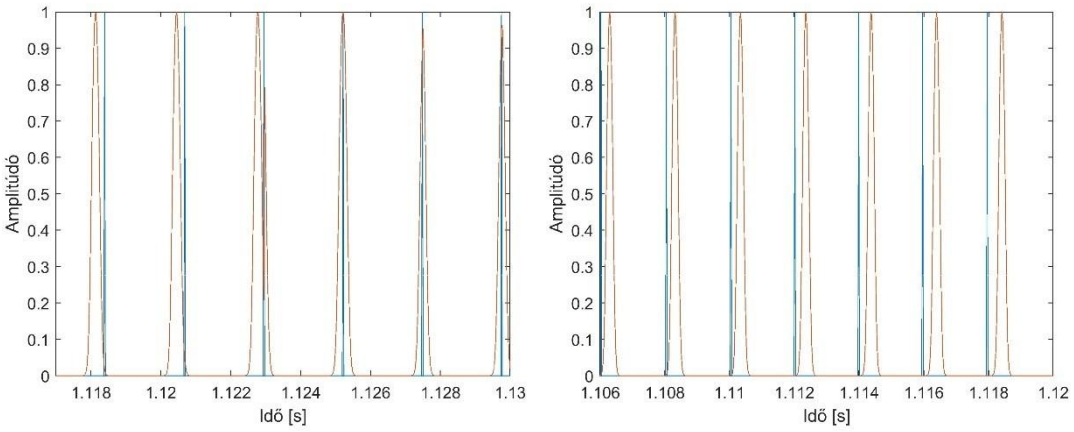
$$p = p_{\min} + 0.5(p_{\max} - p_{\min})(1 - \;\tanh(\Omega)), \tag{6}$$

$$\Delta\varphi = \;\frac{2\pi}{p}\left(\frac{1}{f_s} - \;\Delta t_0\right), \tag{7}$$

melyeknél $s(t)$ a gerjesztőjel, p és φ a periódus és a fázis, p_{max} és $p_{\min}$ a periódus felső és alsó korlátai, f_s a hangfelvétel mintavételi frekvenciája, η_1 és η_2 konstansok a fázis-frekvencia összehatás erősségét határozzák meg, a Δt_0 tag a fázis-, $\Delta\Omega$ és Ω pedig a periódus változtatásáért felelős belső paraméterek. Ω kezdeti értéke 0. Jól látható, hogy a gerjesztőjel a belső paramétereket állítja közvetlenül a (3) és (4) sorokban, az oszcillátor periódusa és fázisa pedig (6) és (7) szerint változik, ezek felhasználásával.

Az egyszerűbb ábrázolás kedvéért folytonos időben írtam le az oszcillátorok működését, a digitális jelfeldolgozás viszont diszkrét idővel dolgozik. Ennek megfelelően a továbbiakban a gerjesztőjelre és az oszcillátor kimenetére tekintsünk minták sorozataként.

A konstrukció működését jól szemlélteti a 3.5. ábra, ahol egy periodikus és egy nemperiodikus gerjesztőjel, valamint a rájuk szinkronizáló oszcillátorok kimenete látható. Minél sikeresebben megy végbe a szinkronizálás, annál inkább fut együtt az oszcillátor kimenete a gerjesztés tüskéivel.



3.5. ábra: Egy sikeres (bal) és egy sikertelen (jobb) szinkronizálás részletei. Kékkel az s(t), narancssárgával pedig az o(t) függvények.

Az oszcillátorok használatával nő a számítás komplexitása, viszont ha van periodikus jel a vizsgált csatornán, arra az oszcillátor nagy pontossággal rá tud szinkronizálni. A műveletek végig időtartományban történnek, a hangfelvétel mintavételi frekvenciája által meghatározott felbontással. Így a szűrőkkel a frekvencia-, az oszcillátorokkal pedig az időtartományban egyszerre nyílik lehetőség elegendően nagy pontosságú számításra.

A keskenysávú szűrők miatt az oszcillátorok egyértelműen megfeleltethetők a tonális zenei hangok komponenseinek, ezért nincs szükség a pontosan bemért frekvenciára, csak azt kell megvizsgálni, hogy mennyire volt sikeres a szinkronizálás. Ehhez egy folytonosan változó értéket praktikus használni.

Egy jól szinkronizált oszcillátor a kimenetén 1-hez közeli értéket vesz fel a gerjesztés érkezésének pillanatában. Ha periodikus gerjesztés érkezik, ideális esetben az oszcillátor p periódusa a bemenet periódusával lesz egyenlő. Az oszcillátorok működése

miatt az ideális szinkronizálás melletti kimenet a gerjesztés pillanatában legalább akkora lesz, mint amit $p_{\min}$ periódussal kapnánk. Ezt az értéket o_{ideal}-nak neveztem, (1), (2) és (7) szerint így kaphatjuk meg:

$$o_{\text{ideal}} = 1 + \tanh\left(\gamma \cos\left(\frac{2\pi}{p_{\min} f_s}\right) - 1\right).$$

o_{ideal} jól használható annak a vizsgálatára, hogy aktuálisan mennyire találja el az oszcillátor a gerjesztést. Ha a gerjesztés pillanatában az oszcillátor kimenetének értéke legalább o_{ideal}, a szinkronizáció sikeresnek tekinthető az adott periódusban. Nagyobb $p_{\min}$ mellett o_{ideal} értéke is nő, ami bizonyos oszcillátorok esetén önmagában túl szigorú feltételt szabna a sikeresség ellenőrzésére. Emiatt küszöbnek a $\min(o_{\text{ideal}}, 0.95)$ értéket választom egy adott oszcillátor esetében.

A szinkronizációs sikeresség értékeléséhez minden oszcillátor kap egy $flag$ változót. Amennyiben a gerjesztés pillanatában az oszcillátor kimenete megüti a fent definiált küszöböt, $flag$ értékét 1-be állítom, ezzel jelezve, hogy az adott periódusban az oszcillátor kimenete elfogadható mértékben találta el a gerjesztőjelet.

Az oszcillátor a gerjesztésre $\varphi = 0$ környékén a legérzékenyebb, ekkor történik meg a paraméterei értékének frissítése. Ettől a ponttól távolodva az érzékenység meredeken csökken $\varphi = \pi$-ig, ahol a legérzéketlenebb. A szinkronizáció sikerességi értékét ezért minden periódusban közvetlenül a $\varphi = \pi$ érték átlépése után frissítem.

A sikeresség pillanatnyi értékét a következő módon számítom ki:

$$c_i = (1 - Q)c_{i-1} + Qflag, \tag{8}$$

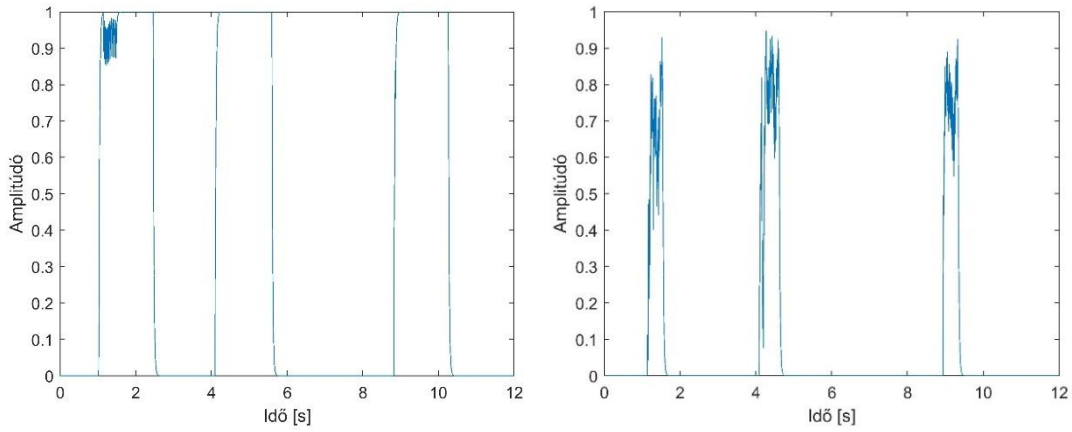
ahol c_i a sikeresség értéke az i. időpillanatban, Q pedig az exponenciális időátlagolást megvalósító súly, ami lehetővé teszi a sikeresség értékének folytonos változását. Q értékét így határoztam meg:

$$Q = 1 - e^{-\frac{p}{\tau}}, \tag{9}$$

ahol p az oszcillátor kezdő periódusa, τ pedig az exponenciális időátlagolási konstans, melyet 0.02 másodpercnek választottam.

A sikeresség kiértékelését követően $flag$ értéke nullába vált. Az eljárás lehetővé teszi, hogy c értéke nőjön, ha jól-, és csökkenjen, ha rosszul szinkronizált periódusok követik egymást, két kiértékelés között pedig ne változzon. c maximumát 1-nek,

minimumát pedig 0-nak választottam. Szemléltetésként a 3.6. ábrán látható egy hangfelvételen megjelenő tonális hang adott komponensére és zajra szinkronizáló oszcillátorok sikerességi kimenete.



3.6. ábra: Egy jól (bal) és egy rosszul szinkronizált oszcillátor (jobb) szinkronizációs sikerességének változása

3.4 Oszcillátorcsoportok

Mivel egy zenei hang komponensei harmonikus viszonyban vannak egymással (frekvenciáik egyszerű arányosságokat mutatnak), lehetőség nyílik arra, hogy egy sikeresen szinkronizált oszcillátor hozzásegítse a hanghoz tartozó többi oszcillátort is a nagyobb sikeresség eléréséhez. Ennek érdekében – a Marolt-féle [5] eljáráshoz hasonlóan – csoportokba rendeztem az oszcillátorokat a zongorabillentyűk harmonikus komponenseinek frekvenciái szerint úgy, hogy egy csoportba maximum tíz darab oszcillátor kerüljön (az alapfrekvencia mellett az első kilenc felharmonikushoz tartozó). Egy csoport legalább egy oszcillátorból áll és egy oszcillátor több csoportban is szerepelhet. Ez utóbbi engedmény a legtöbb esetben elvárás is, hiszen az egymással harmonikus viszonyban álló hangok komponenseinek halmazai nagyban fedik egymást. Ha adott például két hang, amelyek oktáv távolságra vannak egymástól – alapfrekvenciáik 1:2 viszonyban állnak –, a magasabb hang komponensei a mélyebb hang felhangjaiként is megjelennek, kétszer nagyobb sorszámokkal. Így egy sikeresen szinkronizált komponens legfeljebb tíz különböző csoport oszcillátorainak sikerességét is befolyásolhatja, eltérő szerepekben.

Adott pillanatban sikeresen szinkronizált oszcillátornak azt tekintem, amelyik aktuális c értéke legalább 0.9, sikertelen szinkronizációnak pedig azt az állapotot,

amikor ez a mennyiség kisebb, mint 0.5. Az oszcillátorcsoporton belüli egymásra hatás lépéseként a csoport egyik (*forrás*) tagja hat egy másikra (*cél*). Annak érdekében, hogy az egymásra hatás javítsa, és ne rontsa a szinkronizáció minőségét, a forrás csak adott pillanatban sikeresen szinkronizált oszcillátor lehet, míg a célnak ezzel egy időben sikertelennek kell lennie.

Az egymásra hatás során a forrás közvetlenül megváltoztathatja a cél periódusát és szinkronizációs sikerességét a következő szabályok szerint:

$$r_p = \exp\left(-1000\left(\frac{dp_d}{sp_s} - 1\right)^2\right), \tag{10}$$

$$r_c = \exp\left(-2.3\frac{c_d^2}{c_s^2}\right), \tag{11}$$

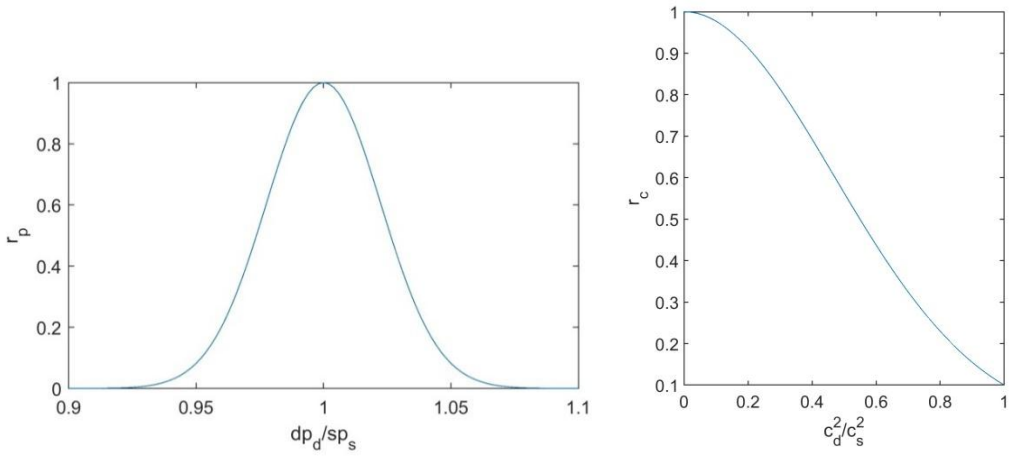
$$p_d = p_d + \frac{sp_s - dp_d}{d} r_p r_c w_{sd}, \tag{12}$$

$$c_d = c_d + c_d r_p r_c w_{sd}, \tag{13}$$

ahol s és d a forrás és a cél oszcillátoroknak a csoporton belüli indexei (adott csoporton belül a harmonikus komponensek sorrendjének megfelelően, 1 és 10 közötti érték), p_s, p_d, c_s és c_d rendre az oszcillátorok periódusai és sikeresség-értékei.

Az r_p függvény (10) egy haranggörbét definiál, ami a maximumát abban az esetben éri el, ha a $\frac{dp_d}{sp_s} = 1$ egyenlőség fennáll, ez ugyanis azt jelenti, hogy a két oszcillátor periódusa – a természetes felhangok azon tulajdonsága miatt, hogy a felharmonikusok az alapfrekvencia egész számú többszöröseinek megfelelő rezgésszámokon jelennek meg – tökéletes harmonikus viszonyban van egymással. Ekkor a függvény értéke 1, ettől a ponttól távolodva pedig jelentősen csökken. Ennek megfelelően (12) és (13) azokban az esetekben változtat nagyobb súllyal a cél oszcillátor paraméterein, amelyekben erősebb a harmonikus viszony a két periódus között.

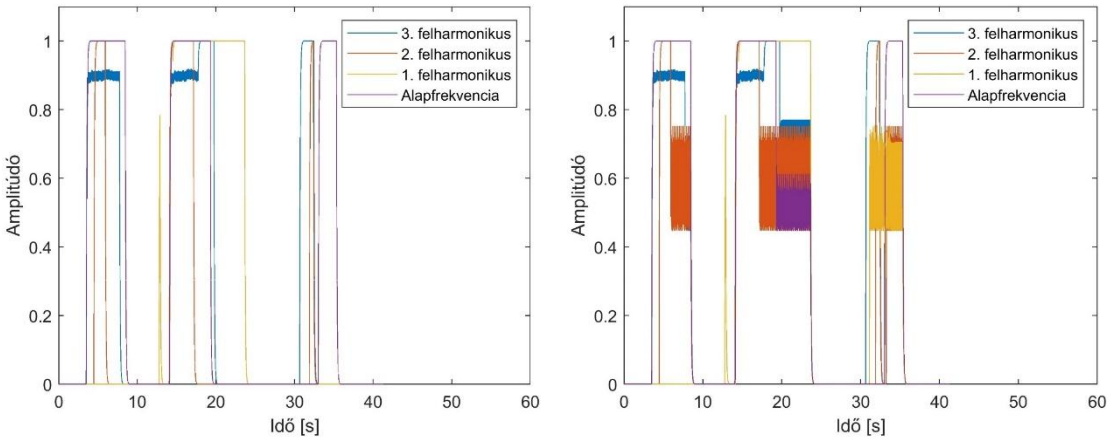
r_c a sikerességi viszonyt vizsgálja a két oszcillátor között (11). A két oszcillátor sikerességi értéke közötti különbség növekedése mellett r_c is magasabb értéket vesz fel. Így (12) és (13) akkor módosít többet a cél oszcillátor paraméterein, ha a cél sikeressége alacsonyabb, tehát nagyobb segítséget is elbír. r_p és r_c görbéi a 3.7. ábrán láthatók.



3.7. ábra: Az r_p (bal) és r_c (jobb) görbék

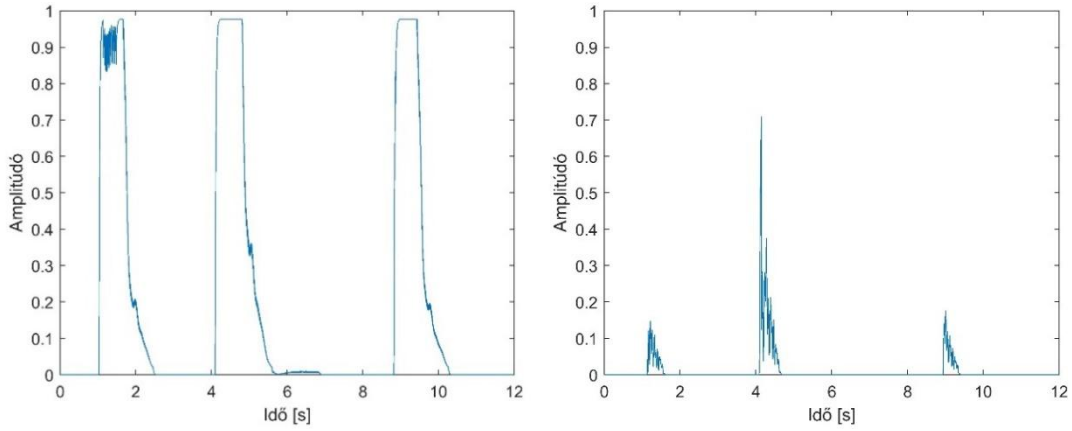
Az alacsonyabb indexű harmonikus komponensek oszcillátorai keskenyebb frekvenciatartományon belül mozognak, emiatt érzékenyebbek a periódus nagy változására. Ezt szem előtt tartva célszerű bevezetni azt a megkötést, hogy egy csoporton belül a magasabb sorszámú oszcillátorok hatása az alacsonyabb indexűekre kisebb mértékű legyen, mint fordított helyzetben. Ezt a működést valósítja meg a w_{sd} súly, amely a hatás oszcillátorcsoporton belüli irányáról hordoz információt. w_{sd} az összes oszcillátor-párra definiálható mindkét irányban olyan módon, hogy $s < d$ esetén $w_{sd} > w_{ds}$ teljesüljön. Megoldásomban a mindenkori w_{sd} értékét $0.1d$-nek választottam, mely egyszerűsítés eleget tesz a fenti feltételeknek és megfelelően működik.

A 3.8. ábrán összehasonlítható egy hang első négy harmonikus komponenséhez tartozó oszcillátorok saját szinkronizációs sikeressége a csoportba kapcsolás hatásával. Jól látható – főleg a második megszólaló hang esetében –, hogyan tartják magasabb értéken egymás sikerességét a csoport oszcillátorai, amíg a fent ismertetett feltételek teljesülnek.



3.8. ábra: Egy oszcillátorcsoport első négy oszcillátorának sikerességi görbéje az oszcillátorok egymásra hatása nélkül (bal) és a csoporton beüli egymásra hatás alkalmazásával (jobb)

Az alkalmazásomban a csatornák jeleinek amplitúdóját és a periodicitást mérő szinkronizációs sikerességet egyaránt használom, ezért a hangjel feldolgozási szakaszának zárásaként az oszcillátorok sikerességi görbéjét súlyozom a hozzá tartozó csatorna amplitúdóburkolójával. A 3.9. ábrán a jelfeldolgozási műveletsor kimenete látható egy periodikus és egy nemperiodikus jelet tartalmazó csatornán.



3.9. ábra: Kimenetek periodikus (bal) és nemperiodikus (jobb) jelet tartalmazó csatornákon

3.5 Neurális hálózatok

A hallás mechanizmusával analóg módon működő eljárás következő lépéseként, a szőrsejteken és hallóidegeken keresztül eljutunk az agyig, ahol a beérkező jelekből létrejön a hangélmény. A zenei képzésekben hallásfejlesztési gyakorlatok segítségével elsajátítható különböző zenei elemek – például hangszín, hangközök, dallamok, harmóniák, ritmusok – hallás útján történő felismerése a megszólaló hangképben

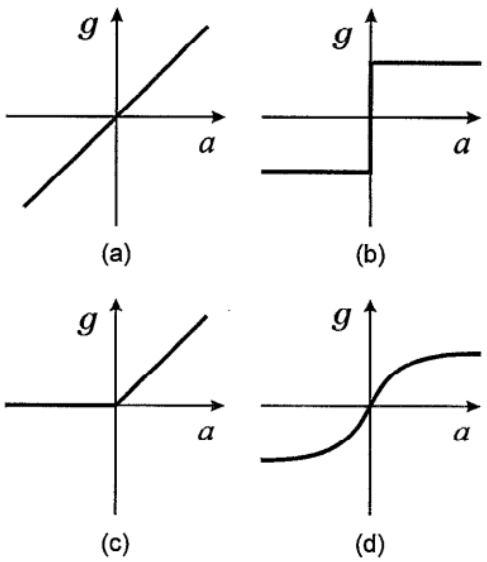
megjelenő mintázatok azonosításával. A szinkronizálási eljárás kimenetén létrejövő komplex struktúrák azonosításához, így a megszólalt hangokról való döntés rétegének megvalósításához, kézenfekvő lehet valamilyen neurális hálózati architektúra használata. [11]

A neurális hálózat olyan számítógépes modell, amelyet egymással kapcsolatban álló, mesterséges neuronok alkotnak. A mesterséges neuronok a biológiai idegsejtekhez hasonlóan működnek, bemeneti impulzusokra adnak valamilyen erősségű választ, amit aktivációnak nevezünk. A gépi tanulásban a mesterséges neuronok McChulloch-Pitts-féle modelljét használják, ami egy neuron z aktivációját az alábbiak szerint írja le:

$$a = \sum_{i=1}^{d} w_i x_i + w_0,$$

$$z = g(a),$$

ahol x_i a neuron i. bemenete, w_i a hozzá tartozó-, w_0 pedig a torzításnak (*bias*, b) is nevezett súly. A súlyok tetszőleges valós értéket felvehetnek, a torzítás szabályozza, hogy a bemenetek súlyozott összegének milyen küszöbértéket kell elérniük a neuron aktivációjához. A neuron z aktivációját a $g(a)$ aktivációs függvény alkalmazásával kapjuk. A 3.10. ábrán néhány tipikus, a gyakorlatban használt aktivációs függvény grafikonja látható.



3.10. ábra: A linear (a), threshold (b), ReLU (c) és sigmoid (d) aktivációs függvények görbéi [11]

Mesterséges neuronok összekapcsolásával hozhatók létre a neurális hálózatok, melyek egyik legegyszerűbb típusa az előre csatolt hálózat (*Feed Forward Network*). Az előre csatolt hálózatokra úgy is tekinthetünk, mint egy általános célú nemlineáris függvényre, amely bemeneti változók halmazát a kimeneti változók halmazába transzformálja. Az előre csatolt hálózatok neuronjait egymást követő rétegekbe rendezhetjük oly módon, hogy egy adott réteg egyik neuronjának x_i bemenetére az előző réteg i. neuronjának aktivációja kerül. Az első réteg aktivációit az **x** bemeneti vektor értékei adják, az **y** kimeneti vektor pedig az utolsó réteg aktivációiból ered. A bemeneti- és kimeneti rétegek **x**, illetve **y** dimenzióival megegyező számú neuronból állnak. A két réteg közé tetszőleges számú, úgynevezett rejtett réteg helyezhető, melyek különböző mennyiségű neuront foglalhatnak magukba.

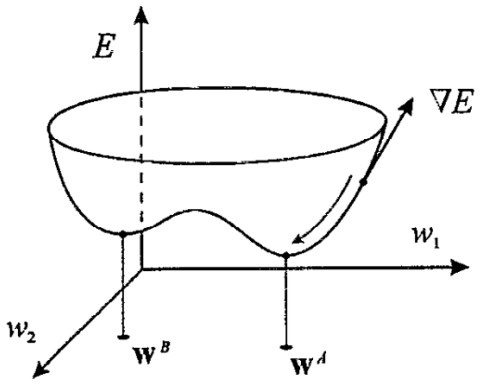
Egy előre csatolt hálózat kimenete az aktuális bemenet mellett a neuronok súlyaitól is függ, ezért az adott alkalmazásnak eleget tevő transzformációt a súlyok értékeinek megfelelő beállításával érhetjük el. A súlyok értékeinek jó beállítását a tanításnak nevezett optimalizációs eljárás során kaphatjuk meg, amelyhez nagy mennyiségű adatra van szükség bemeneti mintákkal és – felügyelt tanulás esetén – a hozzájuk tartozó valós kimenetekkel.

A tanítás a hálózatnak a bemeneti adatpontokra adott kimenetei és a valós, elvárt kimenetek közötti átlagos hibát csökkenti.[4] Az átlagos hiba függvényére úgy is tekinthetünk, mint a súlyok tere feletti felületre, aminek a minimumát keressük a tanítás során. A láncszabály miatt az $E(\mathbf{w})$ hibafüggvény (vagy *loss függvény*) parciálisan deriválható a hálózat összes súlya szerint, amivel megkaphatjuk a függvény $\nabla E(\mathbf{w})$ gradiensét a súlyértékek adott pontjában, ahol $\mathbf{w}$ a hálózat összes súlya.[5] Mivel a gradiens a függvény meredekségét jelöli, az alacsonyabb függvényértékek felé a $-\nabla E(\mathbf{w})$ irányban jutunk el. A tanítást ütemező algoritmus a tanulási ráta (*learning rate*) szerinti mértékben és $-\nabla E(\mathbf{w})$ irányban megváltoztatja a súlyvektor értékeit, ezzel csökkentve az átlagos hibát (*gradient descent*). Az eljárást a teljes tanító adaton több iterációban (*epoch*) megismételve eljuthatunk a hibafüggvény egyik lokális

[4] A hibát a feladat természetétől függően definiálhatjuk.

[5] A gyakorlatban a gradiens kiszámítása a hiba visszaterjesztésnek (*error backpropagation*) nevezett eljárással történik.

minimumpontjába a 3.11. ábrán szemléltetett módon.[6] Memóriakezelési okokból a gyakorlatban sokszor nem egyszerre használják a teljes tanító adatot a súlyok változtatására, hanem adatpontok egy rögzített méretű részhalmazának (*batch*) átlagos hibáját veszik. Ilyenkor egy epoch-on belül többször változik a modell állapota.



3.11. ábra: Az *E*(w) hibafüggvény a súlyok tere feletti görbéjének sematikus ábrája. $\mathbf{w^A}$ a hiba globális-, $\mathbf{w^B}$ pedig az egyik lokális minimumához tartozó súlyvektorok [11]

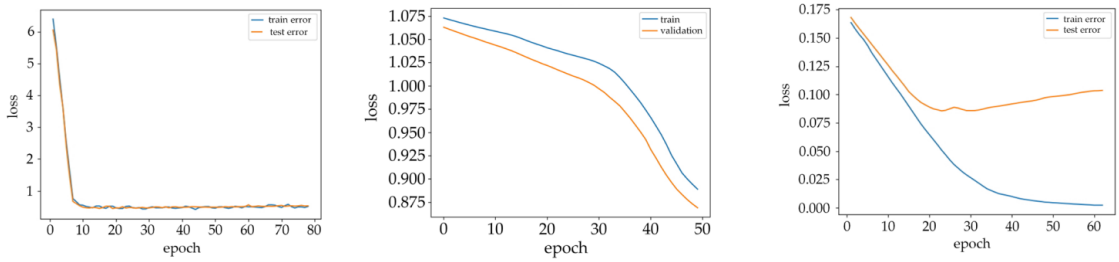
A sikeres tanítás sok tényezőtől függ. Mindenekelőtt elegendő, a feladat szempontjából releváns és megfelelő minőségű adatra van szükség, ami nem tartalmaz redundanciát, kellő mértékben zajcsökkentett és a megoldandó problémát tekintve reprezentatív. A létrehozott hálózat mérete és architektúrája az adat és a feladat komplexitásához kell, hogy igazodjon – egy túlságosan kis méretű modell nem képes megfelelő mértékben leképezni a kívánt transzformációt (alultanulás, *underfitting*), egy túl nagy hálózat pedig túlilleszthet a tanításhoz használt adatra, aminek következtében nem generalizál jól, új adatpontra hibás kimenetet adhat (túltanulás, *overfitting*). Számít továbbá az is, hogy melyik ponton állítjuk le a tanítást.

Az adatot érdemes tanító-, kiértékelő- és teszthalmazra osztani, 80/10/10 vagy ehhez hasonló arányban. A tanítást kizárólag a tanító halmazon (*training set*) végezzük, számon tartjuk az átlagos hibát és minden epoch után kiszámítjuk a modell aktuális állapotán a kiértékelő halmaz (*validation set*) átlagos hibáját is. Ez utóbbi folyamán nem változik a modell állapota, ez a halmaz annak ellenőrzésére szolgál, hogy a modell a

[6] A tanítás nemdeterminisztikus eljárás. A súlyok kezdetben véletlenszerű értékeket vesznek fel, emiatt nem garantált a hiba globális minimumának megtalálása. Általában egy kellően alacsony értékű lokális minimumhoz tartozó súlyvektor megtalálása is elegendő lehet.

tanító halmazon kívüli adatpontokon mennyire működik megfelelően. A teszthalmazt (*test set*) kizárólag a tanítást követően használjuk, az eredményül kapott modell pontosságának mérésére korábban nem vizsgált adatpontokon. A tanítási folyamat követéséhez és a modell működésének ellenőrzéséhez érdemes egy grafikonon ábrázolni a tanító- és kiértékelő halmaz hibájának változását a tanítás iterációi során. Ez alapján ki lehet szűrni, ha esetleg alul-, vagy túltanul a modell, illetve döntés hozható a tanítás leállításáról (manuálisan, vagy a célra kifejlesztett algoritmusok egyikének használatával [12]).

Az optimális tanítás során a tanító- és a kiértékelő halmaz hibái elérnek egy állandósultnak tekinthető állapotot, ahol az értékek nem változnak jelentősen és mindkét hiba elegendően alacsony mértékű. A két görbe közötti távolság mutatja a modell generalizáló képességét. Alultanulás esetén a modell a tanító halmaz elemeire sem ad elfogadható kimenetet, magasak a hibaértékek. A túltanulás jele az, ha a tanító halmaz hibájának csökkenésével együtt csökken a kiértékelő halmaz hibája, egy pont után viszont utóbbi növekedni kezd. Ekkor a korai leállás (*early stopping*) technikáját szokás alkalmazni, azaz a modellnek azon iteráció előtti állapotát használjuk, ahol a kiértékelő halmaz hibája növekedni kezd. A 3.12. ábrán a tanító- és kiértékelő halmazok hibájának változása látható optimális tanítás, valamint alul- és túltanulás esetén.[7]



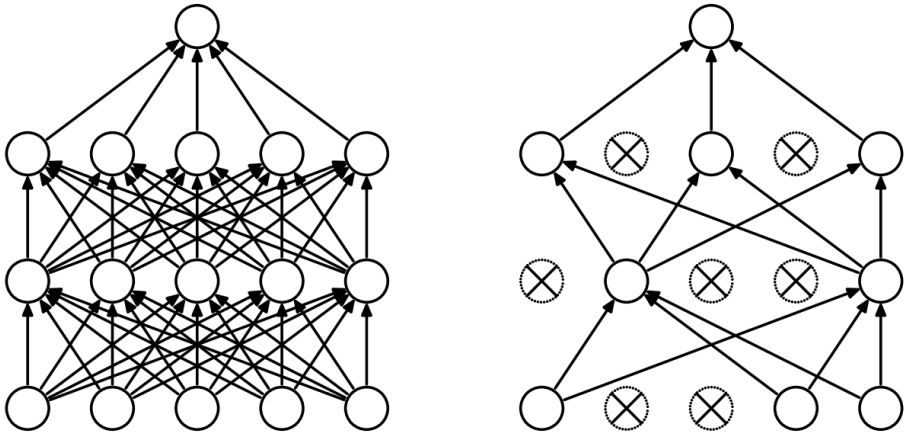
(a) (b) (c)

3.12. ábra: Optimális tanulási görbe (a), alultanulás (b) és túltanulás (c) [12]

Nagyméretű modelleknél érdemes a *dropout* technikát használni a túltanulás csökkentése érdekében. [13] Ez annyit tesz, hogy a neuronok a hozzájuk tartozó súlyokkal együtt p_{dropout} valószínűséggel véletlenszerűen „kiesnek” a tanítás során, nem vesznek részt az aktuális kimenet előállításában, rábírva a hálózatot a neuronok

[7] Az ábrák jelmagyarázata nem konzisztens. A „test error” és „validation” szavak egyaránt a kiértékelő halmaz hibájára utalnak.

kiegyenlített használatára, ami növeli a modell generalizáló képességét. A 3.13. ábrán látható a dropout hatásának sematikus rajza.



3.13. ábra: Egy neurális hálózat sematikus rajza 2 rejtett réteggel (bal) és a hálózat dropout hatására „zsugorított" változata (jobb) [13]

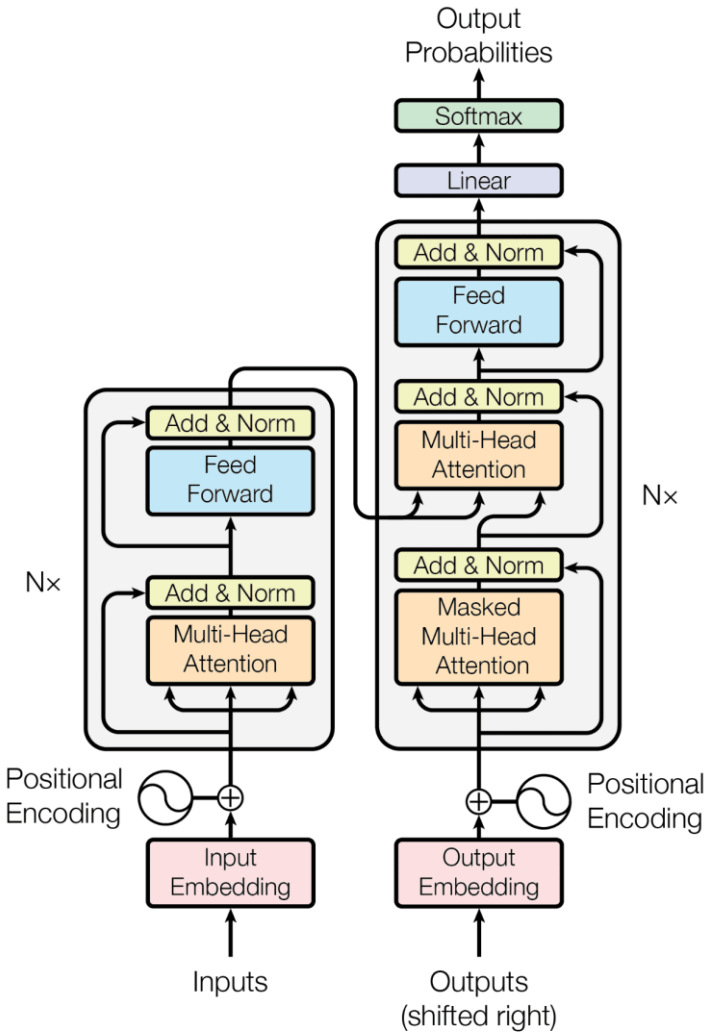
Különböző természetű feladatokhoz különböző típusú neurális hálózati architektúrák használata javasolt. A zongorahang esetében az adatra minták szekvenciájaként tekintek, ahol az időbeliség miatt fontos a sorrend és a korábbi minták hatása is. A megoldásomban Hawthorne-hoz hasonlóan [14] a szekvenciák kezelésére létrehozott Transformer architektúrát használtam, ami hatékonyabb a korábbi, hasonló feladatok megoldására fejlesztett rekurrens modelleknél (pl. LSTM [15]).

3.6 A Transformer architektúra

A Transformer architektúrát [16] 2017-ben hozták létre, elsősorban természetes nyelvi feladatok megoldása céljából (fordítás, szöveggenerálás, összefoglaló írása stb.), viszont más típusú problémák esetében (pl. képgenerálás [17] vagy beszédfelismerés [18]) is hatékonynak bizonyult. Ez a megközelítés képezi napjaink nagy nyelvi modelljeinek (BERT [19], GPT [20] stb.) alapját. Ebben a fejezetben a természetes nyelvek közötti fordítás példájával mutatom be az architektúra működését.

Természetes nyelvek esetében szükség van egy-egy szótárra a forrás- és a cél nyelvhez. A szótár elemeit *tokeneknek* nevezzük, amik lehetnek a vizsgált nyelv szavai, de akár szótagok vagy karakterek is. Az architektúra nem képes tetszőleges hosszúságú adatot kezelni, ezért választani kell egy $n_{\max}$ maximális szekvenciahosszt a bemeneti oldalon és az összes, ennél rövidebb szekvenciát feltölti egy speciális kitöltő (*padding*) tokennel. Ezt a tokent a hálózat figyelmen kívül hagyja a tanítás és a fordítás során.

A Transformer a forrásnyelv szótárának ismeretében először a bemeneti szekvenciát egy beágyazási eljárással (*Input Embedding*) tokenenként leképezi d_{model} dimenziójú vektorok sorozatára. Az így kapott $n_{\max} \times d_{\text{model}}$ dimenziójú mátrix a bemeneti szekvencia tokeneinek a d_{model} dimenziójú vektortérben vett reprezentációja. A beágyazás hasonlóan megy végbe a kimeneti oldalon is (*Output Embedding*), ugyanazzal a d_{model} beágyazási dimenzióval, a célnyelv szótárával és a kimeneti $m_{\max}$ szekvenciahossz alkalmazásával. A maximális szekvenciahosszok és a beágyazási dimenzió rögzítésével, valamint a rövid szekvenciák feltöltésével tetszőleges bemenet és kimenet mellett az egyes lépésekben mindig azonos méretű mátrixokon végez számításokat a modell, ami nagyban egyszerűsíti a működését.



3.14. ábra: A Transformer architektúra felépítése [16]

A Transformer makró szinten a kódoló-dekódoló (*Encoder-Decoder*) struktúrát követi. A beágyazott bemenetet először a kódoló átalakítja, értelmezi, majd a dekódoló

tokenenként előállítja a kimeneti szekvenciát. A kimenet előállítása auto-regresszív módon történik: a dekódoló az aktuális kimeneti tokent hozzáfűzi az előzőekhez és az így előállított hiányos szekvencián elvégzi a beágyazást. A dekódoló minden lépésben az addigi predikciók és a kódoló kimenete alapján dönt a következő kimeneti tokenről. A kimenet végét a speciális végjel (*end*) token jelzi: amikor megjelenik a kimeneten, véget ér a fordítás.

A kódoló és a dekódoló különböző *Multi-Head Attention* és előre csatolt neurális hálózat (*Feed Forward Network*) alrétegekre bomlik. Az összes alrétegre teljesül, hogy a kimeneti mátrix dimenziója megegyezik a bemenetével. Minden alréteg művelete után az architektúra összeadja a két mátrixot és normalizálja az eredményt.

Egy kódoló réteg egy Multi-Head Attention és egy Feed Forward alrétegből áll, egy dekódoló rétegben pedig egy Masked Multi-Head Attention, egy Multi-Head Attention és egy Feed Forward alréteg követi egymást, ezek felépítését és működését alább ismertetem. Egymás után akár több kódoló és dekódoló is kapcsolható, ilyenkor az első kivételével minden kódoló bemenete az előző réteg kimenetével egyezik meg és ez analóg módon teljesül az egymás után kapcsolt dekódoló rétegekre is. Az utolsó kódoló réteg kimenete csatlakozik az összes dekódoló Multi-Head Attention alrétegéhez, ahol találkozik egymással a forrás- és a célnyelv szekvenciáinak beágyazása.

Az utolsó dekódoló kimenetét egy lineáris neuron réteg egy, a kimeneti szótár méretével megegyező dimenziójú vektorrá transzformálja, ami a *softmax* művelet segítségével valószínűségi eloszlássá alakítható. A legmagasabb valószínűség pozícióján lévő token lesz a modell jóslata a kimeneti szekvencia aktuális eleméről.

A 3.14. ábra bal oldalán egy kódoló, jobb oldalán pedig egy dekódoló réteg látható. Az alrétegek felépítésének ismertetése előtt az Attention mechanizmust mutatom be.

3.6.1 Scaled Dot-Product Attention

Az Attention mechanizmus szerepe a modellen belül az, hogy a tokenek sorozatát reprezentáló beágyazott mátrix sorai között összefüggéseket találjon, az egymással kapcsolatban lévő sorokat pedig úgy változtassa meg, hogy a beágyazásban az elkódolt szöveg kontextusa, a tokeneknek a szöveg jelentésére gyakorolt hatása is megjelenjen.

Az eljárás alapelemét a [16] szerzői által *Scaled Dot-Product Attention*-nek nevezett műveletsor képezi, amelyet a következők szerint definiáltak:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^{\mathrm{T}}}{\sqrt{d_k}})\mathbf{V}.$$

A **Q** (*query*), **K** (*key*) és **V** (*value*) mátrixok a bemeneti mátrixból és a megfelelő $\mathbf{W}^{\mathrm{Q}} \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $\mathbf{W}^{\mathrm{K}} \in \mathbb{R}^{d_{\text{model}} \times d_k}$ és $\mathbf{W}^{\mathrm{Q}} \in \mathbb{R}^{d_{\text{model}} \times d_v}$ súlymátrixok szorzatából áll elő. Ezek a beágyazásokat a d_{model} értékénél jellemzően alacsonyabb d_k illetve d_v dimenziójú vektortérbe transzformálják. A súlymátrixok elemeit a tanítás során állítja be a modell.

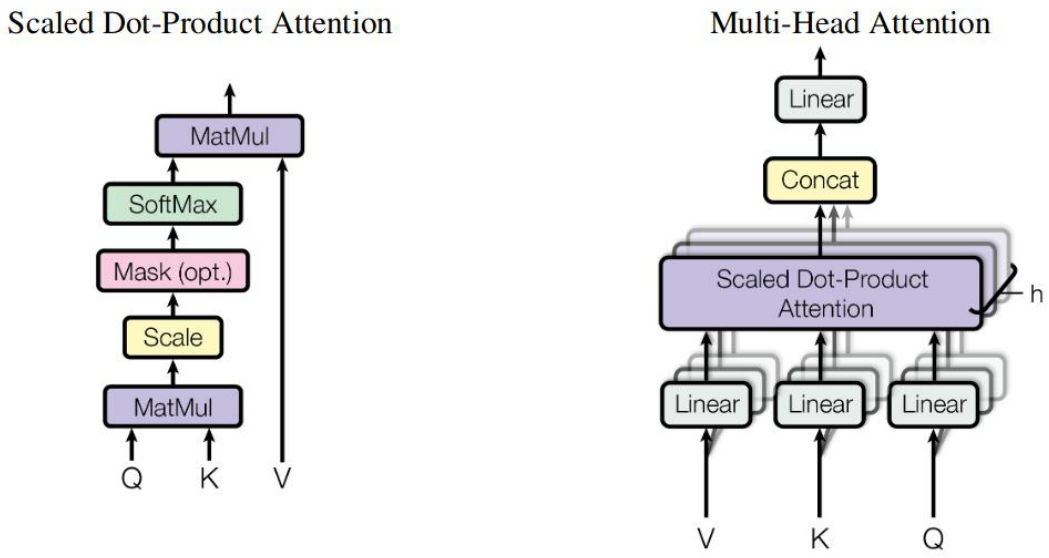
Mivel a mátrixszorzásra úgy is tekinthetünk, mint a bal oldali mátrix sorait és a jobb oldali mátrix oszlopait alkotó vektorok skaláris szorzatainak összessége, a $\mathbf{Q}\mathbf{K}^{\mathrm{T}}$ elemei a tokenek d_k dimenziójú reprezentációi közötti korrelációkat fejezik ki. Minél közelebb van a d_k dimenziós térben az egyik tokent reprezentáló vektor egy másikhoz, annál nagyobb lesz a vektoriális szorzatuk. A hálózat feladata a tanulás során, hogy a súlymátrixok beállításával olyan leképezést találjon, ami a szöveg kontextusában összetartozó tokeneket egymáshoz közel helyezi el a d_k dimenziójú vektortérben. A kapott $\mathbf{Q}\mathbf{K}^{\mathrm{T}}$ mátrix *i*. sorának *j*. oszlopában lévő érték azt mutatja, hogy a szekvencia i. tokenének értelmezésében mekkora szerepet játszik a *j*. token, általánosságban a szöveg mely részeire kell nagyobb figyelmet fordítani.

Ha elvégezzük a $\mathbf{Q}\mathbf{K}^{\mathrm{T}}$ és **V** közötti mátrixszorzást, a szekvenciának a d_v dimenziójú reprezentációját kapjuk, ahol minden token beágyazása a szekvencia többi pozícióján lévő **V** szerinti vektor súlyozott összegével lesz egyenlő. Ezzel a szekvencia összes elemében elkódoljuk a kontextust is.

$\mathbf{Q}\mathbf{K}^{\mathrm{T}}$ elemei tetszőleges valós értéket felvehetnek. Annak érdekében, hogy a vektorok súlyozott összege jól írja le a köztük lévő kapcsolatokat, a $\mathbf{Q}\mathbf{K}^{\mathrm{T}}$ sorait először diszkrét valószínűségi eloszlássá alakítjuk a softmax művelet segítségével:

$$\text{softmax}(\mathbf{z})_i = \frac{e^{\mathbf{z}_i}}{\sum_{j=1}^{K} e^{\mathbf{z}_j}},$$

ahol $\mathbf{z_i}$ a $\mathbf{z} \in \mathbb{R}^K$ vektor *i*. eleme. Az így kapott vektor elemei pozitív valós számok, összegük 1 és mivel nagyobb $\mathbf{z_i}$ mellett $\text{softmax}(\mathbf{z})_i$ értéke is magasabb, a kapott eloszlás követi a $\mathbf{Q}\mathbf{K}^{\mathrm{T}}$ mátrix soraiban leírt korrelációs mintázatot. A Scaled Dot-Product Attention-ben a $\mathbf{Q}\mathbf{K}^{\mathrm{T}}$ mátrix hatékonysági okokból skálázva van $\frac{1}{\sqrt{d_k}}$-val.



3.15. ábra: A Scaled Dot-Product Attention (bal) és a Multi-Head Attention (jobb) eljárások blokkdiagramja [16]

3.6.2 Multi-Head Attention

A Transformer architektúra párhuzamosan több Scaled Dot-Product Attention műveletet hajt végre, különböző súlymátrixokkal. Ezek az Attention fejek, amik együtt alkotják a Multi-Head Attention-t:

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \dots \text{head}_h)\mathbf{W}^O,$$

ahol $\text{head}_i = \text{Attention}(\mathbf{Q}\mathbf{W}_i^{\text{Q}}, \mathbf{K}\mathbf{W}_i^{\text{K}}, \mathbf{V}\mathbf{W}_i^{\text{V}})$ az i. Attention fej, $\mathbf{Q}$, $\mathbf{K}$ és $\mathbf{V}$ itt a MultiHead művelet transzformáció előtti, d_{model} beágyazási dimenziójú bemeneteit jelölik, h a fejek száma, $\mathbf{W}_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $\mathbf{W}_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$ és $\mathbf{W}_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ az i. Attention fej megfelelő súlymátrixai, $\mathbf{W}^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$ pedig a kimeneti súlymátrix.

Egy Attention fej kimenetén a $\mathbf{Q}$ szerinti szekvencia d_v dimenziójú, kontextussal ellátott reprezentációja jelenik meg. Az összes Attention fej kimenetét a beágyazási dimenzió mentén összefűzve és az eredményt jobbról megszorozva a tanítás során előálló $\mathbf{W}^O$ súlymátrixszal megkapjuk a bemeneti szekvenciának a $\mathbf{Q}$-val megegyező méretű beágyazását. Az egynél több Attention fej lehetővé teszi, hogy a szekvencián belül egyszerre többfajta kapcsolatrendszert alakítson ki a modell. Ha a d_k és d_v értékeket $d_k = d_v = d_{\text{model}}/h$ szerint állítjuk be, a Multi-Head Attention művelet ugyanannyi paramétert tartalmaz, mintha végig a d_{model} beágyazási dimenzióval és

egyetlen Attention fejjel dolgoznánk. A 3.15. ábrán látható a Scaled Dot-Product Attention és a Multi-Head Attention eljárások blokkdiagramja.

A modellben három különböző Multi-Head Attention blokk található:

1) A kódolóban lévő a **Q**, **K** és **V** mátrixokat a bemeneti szekvencia beágyazásából állítja elő a modell, célja a bemeneti tokenek közötti kapcsolatok felépítése, a bemenet kontextus-dúsított kódolása. Egyetlen szekvencia elemeit használja, ezért ezt a lépést Self-Attention-nek is nevezik.

2) A dekódoló Masked Multi-Head Attention blokkja a kódolóban hasonlóan Self-Attention elven működik. A modell tokenenként állítja elő a kimenetet, a korábbi kimeneteket felhasználva. Hogy ezt a kimenet tetszőleges állapotában elérje, a dekódolóban található Masked Multi-Head Attention blokk fejei a softmax művelet előtt $-\infty$ értékűre állítják a $\mathbf{QK}^{\mathrm{T}}$ mátrix minden *i.* sorának azon *j.* elemét, ahol a $j > i$ reláció fennáll (a főátló feletti háromszögben). A softmax után ezeken a pozíciókon 0 értéket kapunk, így az Attention réteg nem tekint előre, az összes tokenre csak önmagának és a korábbi pozíciók tokeneinek beágyazása hat.

3) A dekódoló Multi-Head Attention blokkja kapcsolja össze a kódolóból érkező információt a dekódolóval. Itt **Q** a dekódoló előző alrétegéből származik, **K** és **V** viszont az utolsó kódoló kimenetéből erednek. Ezen a ponton valósul meg a két nyelv közötti fordítás, ami miatt ezt a lépést Cross-Attention-nek is nevezik.

3.6.3 Feed Forward

A Feed Forward alréteg egy-egy előre csatolt neurális hálózatot tartalmaz a szekvencia összes pozíciójára. Mindegyiknek egyetlen, d_{ff} dimenziójú rejtett rétege van, *ReLU* aktivációval, a kimeneti réteg pedig visszatranszformálja a beágyazást a d_{model} dimenzióba. Egy pozíció Feed Forward rétegének kompakt leírása **x** bemenettel, a hálózat $\mathbf{W}_i$ súlymátrixaival és $\mathbf{b}_i$ torzításaival:

$$\mathrm{FFN}(\mathbf{x}) = \max(0, \mathbf{x}\mathbf{W}_1 + \mathbf{b}_1)\,\mathbf{W}_2 + \mathbf{b}_2.$$

3.6.4 Positional Encoding

Annak érdekében, hogy a modell követni és használni tudja a szekvenciák elemeinek egymásutániságát, a bemenet és kimenet első beágyazását követően a kapott

mátrixokhoz célszerű hozzáadni egy mintázatot, amire rá tud tanulni a hálózat. A mintázat alkalmazását *Positional Encoding*-nak nevezik, ami valamilyen rögzített, vagy a tanítás során alakuló eljárás is lehet. A Positional Encoding kimeneti mátrixának mérete megegyezik a beágyazási mátrixéval.

A kódolást [16] szerzői rögzített módon, különböző frekvenciájú szinusz és koszinusz függvényekkel valósították meg:

$$\mathbf{PE}_{(pos,2i)} = \sin\left(pos/10000^{2i/d_{model}}\right),$$

$$\mathbf{PE}_{(pos,2i+1)} = \cos\left(pos/10000^{2i/d_{model}}\right),$$

ahol *pos* a szekvencia-, *i* pedig a beágyazási dimenzió adott eleme. A kapott beágyazás dimenziói periodikus függvényeknek felelnek meg, melyek hullámhosszai mértani sorozatot alkotnak 2π és $10000 \cdot 2\pi$ között. A kapott mintázat lehetőséget ad a pozíciók közötti relatív távolságok egyszerű követésére, mivel tetszőleges k eltolás mellett $\mathbf{PE}_{pos+k}$ előállítható $\mathbf{PE}_{pos}$ valamilyen lineáris transzformációjával.

4 A feladat megoldása

4.1 Áttekintés

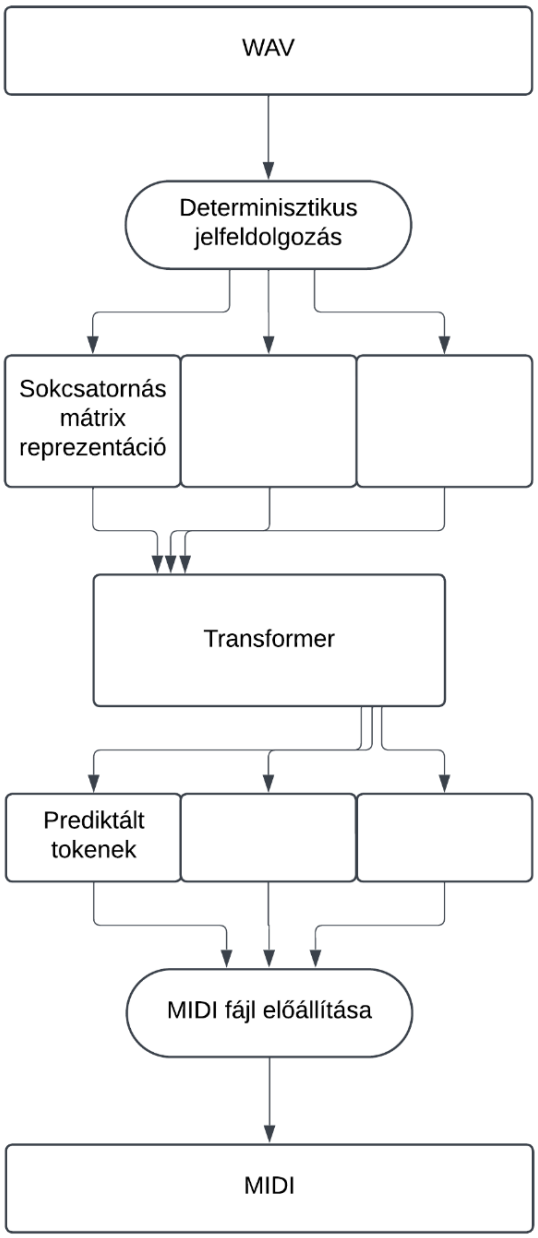
A választott feladat megoldására létrehozott alkalmazásom zongorajátékról készített hangfelvételeken valamekkora bizonyossággal felismeri a billentyűleütéseket és MIDI üzenetek formájában lejegyzi az eredményeket. Mindezt automatikusan, egyéb hozzáadott információ vagy az egyszerre megszólaló hangok számának megkötése nélkül teszi.

Az eljárásom WAV típusú hangfelvételt vár a bemeneten, ami egy rögzített bitmélységű, tömörítetlen audioformátum. Ezek a fájlok a felvétel hullámformáját tartalmazzák valamilyen f_s gyakorisággal mintavételezve, ami általában 44.1 vagy 48 kHz. A hangfelvételeket a 3. fejezetben bemutatott, determinisztikus jelfeldolgozási lépéseket használva bontom 200 csatornára és készítem elő a további számításokhoz.

A hangfelvételek értelmezésére egy Transformer modellt használok. Az architektúra működése miatt a bemenetet és a kimenetet egyaránt korlátozott hosszúságú sorozatokként kell kezelnem, ezért a hangfelvételeket 2 másodperces szeletekben dolgozom fel,[8] 10 milliszekundumonként mintavételezve.[9] Ezzel a lépésközzel a modell bemenetére a 200 csatornára bontott hangfelvételnek maximum 200 időpillanatát tartalmazó mátrix reprezentációk kerülnek. A Transformer a bemeneti mátrixok alapján az adott időszeletben megszólaltatott billentyűk hangmagasságát és a leütések idejét leíró tokenek sorozatát hozza létre. Az eljárás végén a kimeneti szekvenciákból jön létre a MIDI fájl. A WAV-MIDI átalakítás teljes folyamatát a 4.1. ábra mutatja be.

[8] A 2 másodperces választás az időszeletek méretére teljesen önkényesen született. A rövid időablak csökkentheti a modell hibázási lehetőségét, miközben már ennyi idő alatt is sok zenei esemény történhet. A megoldásomban a rögzített hosszúságú időszeletek határa éles, nincs köztük átfedés, a vágás nincs tekintettel a hangesemények időpontjára vagy az egy időablakban megjelenő hangok számára. Ez a megközelítés robosztus és könnyen megvalósítható, de az 5.2. fejezetben kitérek további ötletekre is.

[9] Az emberi fül tehetetlensége miatt a 10 milliszekundumos különbséggel megszólaló hangeseményeket időben képtelenek vagyunk szétválasztani. Emiatt ezzel a lépésközzel lényegi információ vesztése nélkül csökkenthettem a vizsgált adat felbontását.

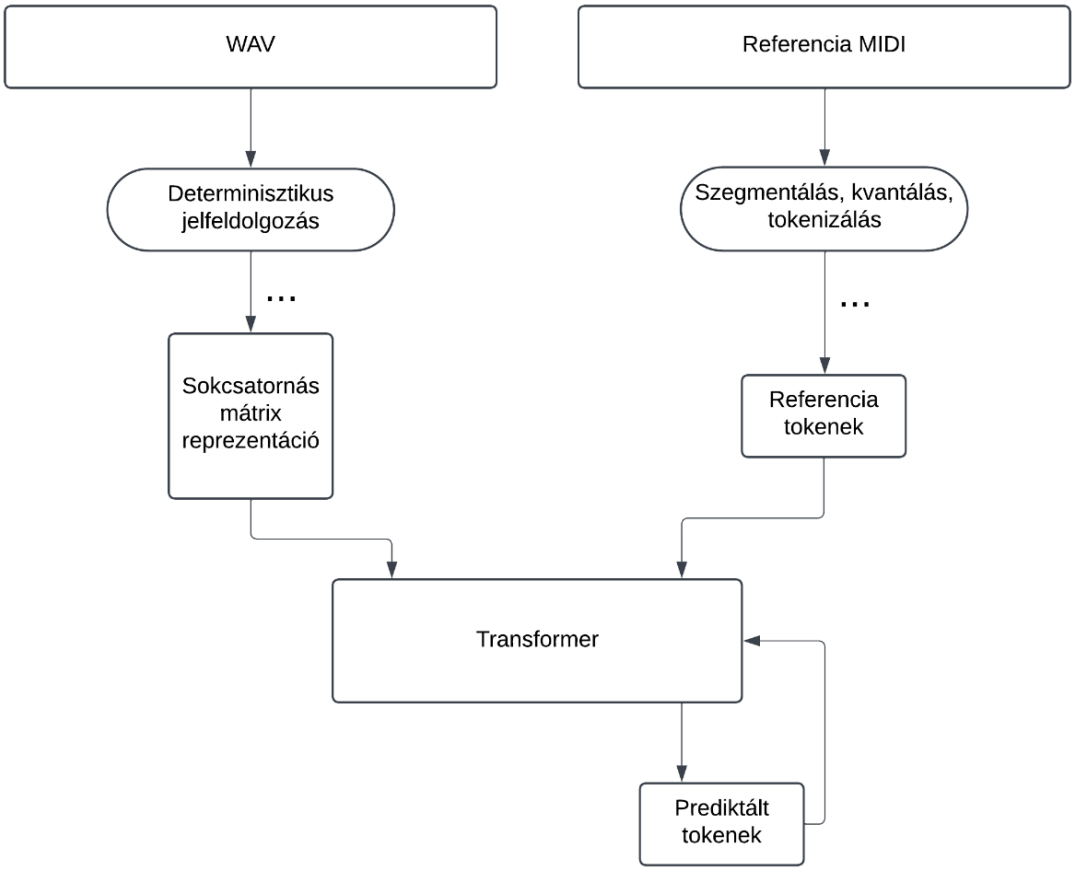


4.1. ábra: Az eljárás folyamata a hangfelvétel beolvasásától a MDI fájl előállításáig

Fontos megjegyezni, hogy a megoldásom kizárólag a billentyűleütések pontos detektálására fókuszál, a leütés hangosságát, a billentyűk felengedését és a pedálok használatát figyelmen kívül hagyja. A nagyon halkan megszólaló vagy sokáig nyomva tartott (emiatt elhalkuló) billentyűk felengedésének vizsgálata nagyban növelné a rendszer bizonytalanságát, a pedálok tompító vagy zengető hatásának egyértelmű azonosításában pedig az adott zongora hangszíne és a terem akusztikai tulajdonságai is hatással vannak, amelyikben a hangszer található. Olyan megoldás létrehozását tűztem ki célul, ami robosztus a hangfelvételek minősége, készítésük körülményei és a felvett

zongorák tekintetében, ezért korlátoztam a feladatot a hangindítások felismerésére. Tekintve, hogy a hallgató számára az egyszerre megszólaló billentyűk azonosításához képest sokkal kevésbé okoz nehézséget a hangok hosszának és hangosságának követése, az alkalmazásom a fenti megkötések mellett is hasznos eszköznek bizonyulhat. A kimeneti MIDI fájlban minden hang megszólalása azonos ideig tart.

A Transformer modell jó működéséhez nagy mennyiségű bemeneti-kimeneti szekvenciapárokon megfelelő beállítások mellett, kellő ideig végzett tanítás szükséges. Az elvárt kimeneteket a bemeneti hangfelvételekhez tartozó referencia MIDI fájlokból állítottam elő. A MIDI üzeneteket szintén 2 másodperces szakaszokra bontottam, a billentyűleütés-események idejét a 10 milliszekundumos lépésközhöz igazodva kvantáltam, majd a Transformer modell számára is értelmezhető tokenek sorozatává alakítottam.



4.2. ábra: A tanításhoz szükséges bemenet és kimenet előállításának lépései és a Transformer modell egy tanítási iterációja egyetlen szekvenciapáron bemutatva

Tanulás közben a modell a bemeneti szekvenciákból előállítja a prediktált kimeneteket, amiknek a referenciáktól való eltérése határozza meg a Transformer súlyainak változását. Egy sikeres tanítási folyamat eredményeként a modell súlyai olyan értékeket vesznek fel, amik mellett képes lesz elfogadhatóan alacsony hibaaránnyal létrehozni a kimeneti tokeneket. A 4.2. ábrán a tanításhoz szükséges adatok előfeldolgozásának és egy tanítási iterációnak a vázlata látható egyetlen bemeneti-kimeneti szekvenciapárra.

A következő alfejezetekben részletesen bemutatom a hangfelvételek és MIDI üzenetek megfelelő előfeldolgozásának implementálását, a modell megalkotását és tanítását, valamint az elkészült alkalmazás grafikus felületét.

4.2 Determinisztikus jelfeldolgozás

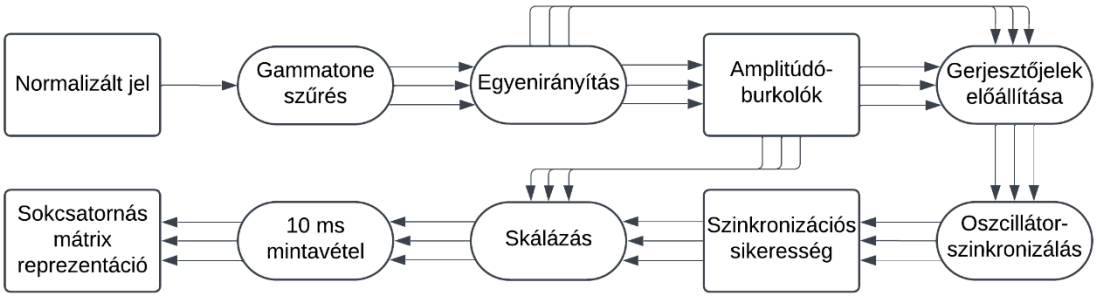
A hangfelvételek előfeldolgozását Matlab környezetben valósítottam meg. Az átalakítás eredményeként az oszcillátorok szinkronizációs sikerességének a csatornák amplitúdóburkolóival súlyozott értékeit várom 10 milliszekundumos lépésközzel, 2 másodperc hosszúságú szeleteken.

A beolvasott WAV fájl jelét első lépésként normalizáltam a jobb kivezérlés érdekében. A normalizált jelen minden további feldolgozási műveletet blokkonként hajtottam végre, a 4.3. ábra szerinti lépésekben.[10]

A hangfelvétel csatornákra bontásához Marolt [5] után 200 darab gammatone szűrőt használtam, melyek középfrekvenciái 70 és 6000 Hz között található. A 3.1. fejezetben ismertetett szűrőket egy *gammatoneFilters* függvény hozza létre, ami a mintavételi frekvencia, a frekvenciahatárok és a szűrők számának ismeretében visszaadja az A és B együtthatókat, a szűrők középfrekvenciáit és a sávszélességüket. A blokkok jelének szűréséhez egy *GammatoneFilterbank* osztályt használok, ami a

[10] A blokk alapú feldolgozás memóriakímélő és a kimeneti formátum szempontjából is indokolt. A blokkok méretét a jelfeldolgozási lépésekben 24 másodpercesre választottam, mert a műveletsor összesített ideje így volt minimális. A kimenetet a jelfeldolgozási műveletsor végén osztom tovább a 2 másodperces szeletekre. Miközben blokkméretenként haladok végig a hangfelvételen, alkalmazok egy puffert is, ami figyelembe veszi a blokkot követő 100 milliszekundum mintáit. Ezzel biztosítom, hogy a blokk alapú feldolgozás ne okozzon hibát például a lokális maximum keresésnél, az eljárás eredménye így ekvivalens a teljes hangfelvételen történő számításokkal.

konstruktorában létrehozza a szűrőket és az egyetlen, *filterAudio* metódusában visszaadja a paraméterként kapott blokk szűrt csatornáit.[11] Az osztály nyomon követi a blokkos feldolgozás menetét és a feldolgozás végén elmenti a szűrők állapotát, hogy a következő blokk szűrése is helyesen történjen. A gammatone szűrést követően sokcsatornás jelet kapunk.



4.3. ábra: A hangfelvétel egy blokkjának determinisztikus jelfeldolgozási lépései a normalizált jel szűrésétől a kimeneti mátrix létrehozásáig

A csatornák hullámformáinak negatív értékeit elhagyva kaptam meg az egyenirányított félhullámokat. Ezek amplitúdóburkolóinak előállításához létrehoztam egy *ChannelLevel* osztályt, aminek egyetlen, *filterChannels* névre hallgató metódusa valósítja meg a 3.2. fejezetben leírt exponenciális átlagolást, szaturálást és a szigmoid műveletét. A *ChannelLevel* a *GammatoneFilterbank* osztályhoz hasonlóan blokkonként kezeli és menti a szűrés állapotának változását.[12] Az egyenirányított jelből lokális maximum kereséssel létrehozom a periodicitás mérésére szolgáló gerjesztőjeleket is. Az amplitúdóburkoló segítségével maszkoltam a gerjesztőjeleket: ahol a burkoló értéke alacsonyabb a 0.02-nek választott küszöbnél, ott 0-val szorzom a gerjesztést.

A gerjesztőjelek az adaptív oszcillátorok bemenetére kerülnek, amelyek kiszámítják minden csatornán a szinkronizációs sikeresség változását. Az oszcillátorok és oszcillátorcsoportok megvalósítását a 4.3. fejezetben tárgyalom részleteiben.

---

[11] A számítással töltött idő csökkentése érdekében a négy másodfokú szűrő használata helyett két eredő szűrőt hoztam létre, melyek együtthatóit az 1-2 és 3-4 sorszámú szűrőpárok együtthatóinak konvolúciójával kaphatjuk meg. Az eredő szűrők stabilnak bizonyultak, így a számítási iterációk csökkentése mellett a négy szűrő egymást követő használatával megegyező eredményt kaptam.

[12] A gammatone szűrőkhöz hasonlóan, gyorsítási célból összevontam a burkoló két szűrőjét is.

A szinkronizáció alatt egy oszcillátorcsoport tagjai a zongora egy adott billentyűjéhez tartozó harmonikus komponensek periodicitását mérik. A megfelelő oszcillátorok csoportosításához az egyenletes hangolás és a felhangrendszer szabályait használtam. Először a referenciául szolgáló A4 hang[13] alapfrekvenciájához képest kiszámítottam a hangszer többi billentyűjének alapfrekvenciáját, majd ezek alapján megadtam a 88 alaphang első 10 harmonikus komponensét. Az alapfrekvenciák értékeinek kiszámításához az egyenletes hangolás következő szabályát használtam, ami két, egymástól félhang távolságra lévő alapfrekvencia relatív távolságát írja le:

$$f_{i+1} = \sqrt[12]{2} f_i,$$

ahol f_i és f_{i+1} a zongora i. és $i+1$. hangjának alapfrekvenciái. A felharmonikusok frekvenciái az alapfrekvenciák egész számú többszöröseinek felelnek meg. A kiszámított frekvenciaértékeket a **P** mátrixba rendeztem úgy, hogy a sorok a hangokat, az oszlopok pedig a harmonikus komponenseket jelölik. A **P** mátrixból létrehoztam a vele azonos méretű **F** mátrixot, ami minden harmonikus komponenshez azt az indexet rendeli hozzá, amely sorszámú gammatone szűrő középfrekvenciája a legkisebb távolságra található az adott komponens frekvenciájától.[14] Minden olyan pozícióban, ahol a harmonikus komponens rezgésszáma kívül esik a gammatone szűrőbank által lefedett tartományo, **F** eleme 0 értéket vesz fel.[15]

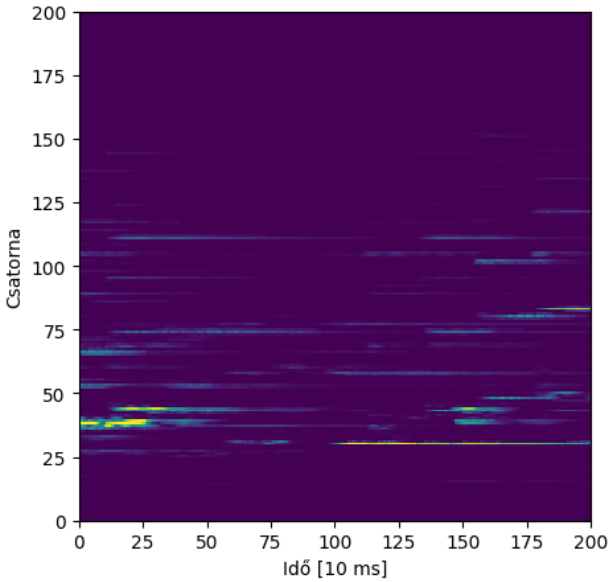
A szinkronizációs sikerességet minden csatornán skálázom az amplitúdóburkolókkal. Ez a lépés tisztítja a hasznos jelet, hiszen a nagyobb amplitúdójú, de nem periodikus szakaszokat és a halkabb részhangokat egyaránt csillapítja. Az így létrehozott kimenetet 10 milliszekundumos lépésközzel mintavételezve és a számítási blokkot 2 másodperces időszeletekre darabolva kapom meg a sokcsatornás mátrix reprezentációkat, amelyek a Transformer modell bemenetére kerülnek. A szűrők száma és az időszelet választása miatt a megoldásomban ezek 200×200 méretű négyzetes mátrixok, melyekre a 4.4. ábrán látható egy példa.

[13] Az A4 vagy „egyvonalas A” hangot használják a legtöbb hangszer hangolására, leggyakrabban 440 Hz alapfrekvenciával. A megoldásomban is ezt választottam alapértelmezettnek.

[14] Az indexelés 1-gyel kezdődik.

[15] A szűrőbank nem fedi le az 55.3 Hz alatti és 6305.8 Hz feletti frekvenciákat.

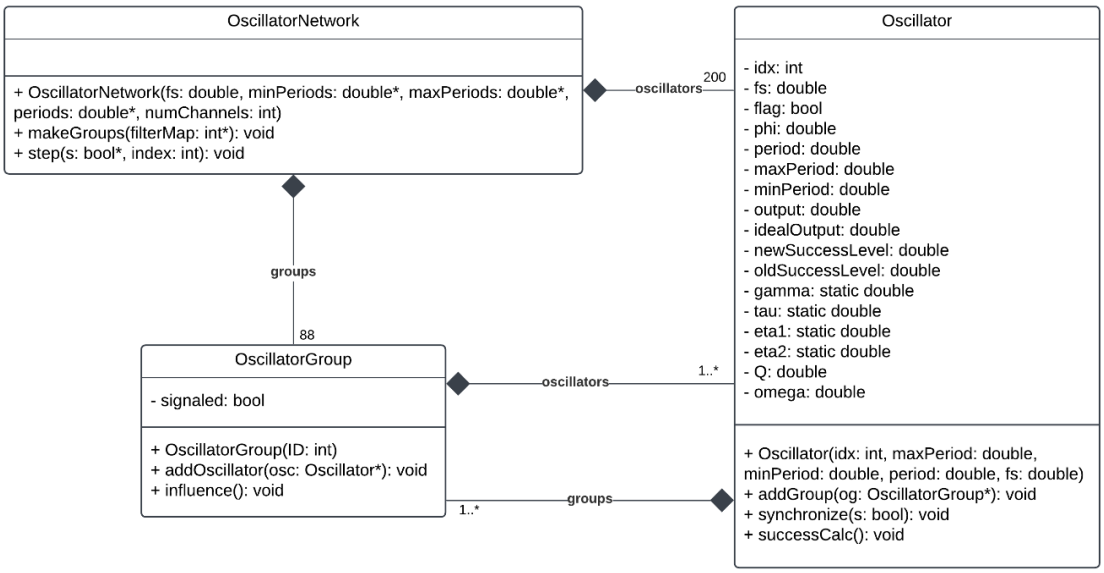
Az általam használt legmagasabb gammatone szűrő felső sávhatára 6305.8 Hz, a szűrést követően az ennél magasabb frekvenciákat nem veszem figyelembe. Ezt kihasználva a feldolgozási folyamatot a hangjel alul-mintavételezésével kezdtem, $f_{\text{resampling}} = 12800\ \text{Hz}$ mintavételi frekvenciával. Mivel ez az érték több, mint kétszerese a felső sávhatárnak, $f_{\text{resampling}}$ használatával veszteségmentesen visszaállíthatók a vizsgált tartomány hangjai, a mintavételi tételnek megfelelően. Az újra-mintavételezést követően jelentősen (kb. negyedére) csökken a hangfelvétel mintáinak száma, ami gyorsítja és egyszerűsíti az összes számítást, hiszen azok mindegyikét időtartományban, a mintákon haladva végzem. A megoldásomban a hangfelvétel beolvasásától a kimeneti mátrixok előállításáig tartó végrehajtási idő nagyjából a vizsgált hangfelvétel hosszának felével egyenlő.



4.4. ábra: A hangjel egyik 2 másodperces szeletének többcsatornás mátrix reprezentációja

4.3 Oszcillátor-szinkronizálás

Az úgynevezett MEX fájlok (*MATLAB Executable*) lehetőséget biztosítanak C, C++ vagy Fortran nyelven írt szubrutinok meghívására Matlab környezetből egy függvényen keresztül, a típusok közötti konverzió támogatásával. Az oszcillátorok szinkronizálását, a csoporton belüli egymásra hatásokat és a szinkronizációs sikeresség mérését C++ osztályokkal valósítottam meg a számítások gyorsabb végrehajtása érdekében. A Matlab kód egy MEX függvényen keresztül kommunikál a C++ osztályokkal, melyek UML diagramja a 4.5. ábrán látható.



4.5. ábra: Az OscillatorNetwork, OscillatorGroup és Oscillator osztályok UML diagramja

4.3.1 OscillatorNetwork

Az *OscillatorNetwork* osztály fogja össze az oszcillátorokat és oszcillátorcsoportokat, kezeli a működésüket.

Az osztály a konstruktorában létrehoz *numChannels* darab *Oscillator* példányt, amelyek kezdő periódusait (*periods*), a periódusaik alsó és felső határait (*minPeriods, maxPeriods*) és az *fs* mintavételi frekvenciát paraméterként kapja meg. A periódusértékeket a gammatone szűrők középfrekvenciájának, illetve sávhatárainak reciproka adja. Az *Oscillator* osztályokra mutató pointereket az *oscillators* tömbben tárolja.

Az oszcillátorcsoportok létrehozását és az oszcillátorok összekapcsolását a *makeGroups* metódus végzi. A paraméterként kapott *filterMap* tömb a 4.2. alfejezetben leírt **F** mátrixnak felel meg. A függvény minden olyan billentyűhöz, aminek az alaphangjához oszcillátort tudtunk rendelni, létrehoz egy *OscillatorGroup* osztályt a billentyűt azonosító *ID*-val. Ezt követően felveszi a csoportba a *filterMap* adott sora pozitív értékeinek megfelelő sorszámú oszcillátorokat és minden felvett oszcillátorban

rögzíti az aktuális csoport pointerét.[16] Az oszcillátorcsoportokra mutató pointereket a saját *groups* tömbjében is tárolja az *OscillatorNetwork*.

A *step* metódus az oszcillátorok szinkronizációjának egy lépését valósítja meg. A paraméterként kapott *s* tömb a 200 csatorna gerjesztése a *j*. időpillanatban. A függvény meghívja az összes oszcillátor *synchronize* metódusát a csatornához tartozó gerjesztéssel. A *step* minden 100. időpillanatban kezdeményezi a csoporton belüli egymásra hatást az *influence* függvény meghívásával minden olyan oszcillátorcsoportban, amelynek *signaled* attribútuma igaz (a feltétel azoknál az oszcillátorcsoportoknál teljesül, amelyek adott pillanatban tartalmaznak forrás oszcillátort).

4.3.2 Oscillator

Az *Oscillator* egy elemi oszcillátornak a 3.3. alfejezetben leírt működését valósítja meg, belső paramétereinek változtatásával.

Az osztály a konstruktorában beállítja a futásához szükséges attribútumok kezdő értékét:

- *idx*, *period*, *minPeriod*, *maxPeriod* és *fs* a paraméterként kapott értékeknek megfelelően,
- *gamma* = 10, *tau* = 0.02, *eta1* = 0.2, *eta2* = 1500 minden oszcillátor esetében,
- *idealOutput* és *Q* a 3.3. alfejezet szerint,
- *flag* = false,
- a többi attribútum értéke 0.

A *synchronize* metódusban az oszcillátor frissíti az *output* és a belső változók értékét a paraméterként kapott *s* gerjesztés függvényében, a 3.3. alfejezetben leírt egyenleteknek megfelelően. Ha az adott időpillanatban jött gerjesztés és *output* értéke legalább akkora, mint az *idealOutput* szerint meghatározott küszöbérték, *flag* true lesz.

[16] Az oszcillátorok számozása itt 0-val kezdődik, ezért a *filterMap*-ben található értékből 1-et kivonva kapjuk a megfelelő oszcillátor sorszámát.

Ha az adott időpillanatban phi (φ) átlépte π-t, a függvény meghívja az osztály *successCalc* metódusát és *flag* értékét false-ba állítja.

Az *addGroup* metódus hozzáadja az oszcillátor *groups* tömbjéhez a paraméterként kapott *OscillatorGroup* pointert.

A *successCalc* hajtja végre a sikerességi szint számítását és frissítését az *oldSuccessLevel*, *newSuccessLevel* és *Q* attribútumok segítségével. Az *oldSuccessLevel* először felveszi *newSuccessLevel* értékét, majd az aktuális szinkronizációs sikerességi szintet a *newSuccessLevel* változó tárolja a 3.3. alfejezetben leírt számítást követően. Ha a friss sikerességi szint értéke 0.9-nél magasabb – tehát az oszcillátor szerepelhet forrásként a csoportokon belül – az oszcillátor az összes csoportjának *signaled* attribútumát igaz értékűre állítja.

4.3.3 OscillatorGroup

Az *OscillatorGroup* a konstruktorában létrehozza az *ID* azonosítójú oszcillátorcsoportot és a *signaled* attribútumát, false kezdőértékkel. Ez az attribútum jelzi, ha a csoport bármely oszcillátora a 3.4. alfejezetben definiált kritériumok szerint aktuálisan forrás oszcillátor.

Az *addOscillator* függvény hozzáadja a csoport *oscillators* tömbjéhez a paraméterként kapott *Oscillator* pointert.

Az *influence* metódus az oszcillátorcsoporthoz tartozó oszcillátorokon végrehajtja a 3.4. alfejezet szerinti egymásra hatást az oszcillátorok csoporton belül betöltött szerepének megfelelően. Az egymásra hatást követően *signaled* értéke false lesz.

Az *Oscillator* és *OscillatorGroup* osztályok rendelkeznek getter és setter függvényekkel is, amelyekkel képesek változtatni egymás attribútumait, de ezekre nem térek ki részletesebben.

A MEX függvény rögzíti és a számítást követően visszaadja a csatornák szinkronizációs sikerességének értékét az aktuális feldolgozási blokk minden időpillanatára. Ezt a lépést követi az amplitúdóburkolókkal való skálázás és a 10 milliszekundumos lépésközzel történő mintavételezés a 4.2. alfejezetben tárgyalt és a 4.3. ábrán látható módon.

4.4 MIDI tokenizálása

A Transformer kimenetén a MIDI eseményeknek szöveges reprezentációját várom, tokenek formájában. A cél az, hogy a hangfelvétel feldolgozása után kapott mátrixokkal tartalmilag megegyező, a billentyűeseményeket jelző kompakt leírást adjunk. Egy billentyűleütés eseményének azonosításához kétféle információra van szükség: a leütött billentyű sorszámára és a leütés idejére.

A zongorabillentyűket 0-tól 87-ig indexelve definiáltam a billentyűt azonosító <*n=0*>, <*n=1*>, …, <*n=87*> tokeneket. A MIDI fájlformátum az események közötti relatív időt tartja számon, a megoldásomban viszont a leütés eseményének az időszeleten elfoglalt abszolút idejét vizsgálom. A hangfelvétel mátrix reprezentációinak felbontásához igazodva egy időszeleten 200 lehetséges pillanatban érkezhet hangesemény, amiket a <*t=0*>, <*t=1*>, …, <*t=199*> tokenekkel kezelek. Egy hangeseményt így egy <*n=x*><*t=y*> tokenpár jelöl, ahol $x \in [0, 87]$ és $y \in [0, 199]$. A Transformer kimenetén ilyen tokenpárok sorozatát várom.[17] A MIDI üzenetek tokenizált leírásának ötletét Hawthorne [14] munkája adta.

A modell tanításához tokenizáltam a referencia MIDI fájlokat, hogy az elvárt kimenetet a megfelelő formátumban adjam meg. A MIDI fájlok feldolgozását Python környezetben, a Mido (*MIDI Objects for Python*) könyvtár segítségével oldottam meg. A könyvtár hasznos eszközöket nyújt MIDI fájlok olvasására, írására és értelmezésére.

Egy MIDI esemény abszolút idejét az azt megelőző összes parancs Δt időbélyegének összege adja (lásd: 2.2. alfejezet). A MIDI protokoll az időt *tick* értékekben méri, amit egy zenei negyedhang felbontásának legkisebb egységeként definiál. Egy negyed hang mikroszekundumban mért hosszát a fájl *tempo* paramétere adja meg. Ennek alapértelmezett értéke 500000 µs, ami 120 BMP-nek (*Beats Per Minute*) felel meg és a *Set Tempo* meta-üzenettel változtatható. A MIDI fájl TPB (*Ticks Per Beat*) paramétere adja meg *tick*-ben egy negyedhang felbontását. Egy Δt időbélyeg milliszekundumban kifejezett Δt_{ms} értékét így kapjuk meg:

$$\Delta t_{\text{ms}} = \frac{tempo}{1000 \cdot TPB} \Delta t.$$

[17] A fordítás során nem teszek különbséget a két tokentípus között, ezért a minél pontosabb tokenértékek megtalálása mellett a modell feladata a kimenet érvényes formátumának megtanulása is.

Az i. MIDI üzenet milliszekundumban megadott abszolút t_i értéke formálisan a

$$t_i = \frac{tempo}{1000 \cdot TPB} \sum_{j=0}^{i} \Delta t_j$$

kifejezéssel írható le, ahol Δt_j a MIDI fájl j. üzenetének időbélyege. Az abszolút időpillanatok értékét 10-zel való osztás és kerekítés után a hangfelvétel mátrixos reprezentációival megegyező időfelbontást kaptam, 2 másodperces ablakonként 200 különböző lehetséges értékkel.

A billentyűleütések hangmagassága egyértelműen kiolvasható a *Note On* üzenetekből[18], ezeket és a fenti átalakítás szerint kvantált abszolút időt használva egyszerűen megadható minden 2 másodperces időszelet hangindítás eseményeinek <*n*=*x*><*t*=*y*> tokenpárok sorozatából álló leírása.[19]

4.5 Adat

A modell tanításához a MAESTRO (***M**IDI and **A**udio **E**dited for **S**ynchronous **Tr**acks and **O**rganization*) adathalmaz v3.0.0 verzióját használtam [21]. Az adat virtuóz zongoraművek WAV formátumú hangfelvételeit és a hozzájuk tartozó MIDI fájlokat tartalmazza. A MIDI fájlokban a billentyűk és pedálok lenyomásával és felengedésével kapcsolatos üzenetek találhatók, ügyelve rá, hogy a hangszert vezérlő események a két fájltípusban ~3 milliszekundumos pontossággal együtt érkezzenek.

Az adathalmaz nagyjából 200 órányi zongorajátékból áll egy online zenei verseny résztvevőinek előadásában. Az 1276 darab hangfelvételen XVII. és XX. század közötti szerzők szóló zongorára írt művei hallhatók.

Az adatbázist gépi tanulási feladatokhoz hozták létre, ezért ajánlott tanító-, kiértékelő- és teszt halmazokra osztották a következő szempontok szerint:

---

[18] A MIDI protokoll a zongora hangterjedelménél mélyebb hangokat is kezel, a hangszer legmélyebb billentyűje a 21-es indexet kapta. A tokenizálásnál figyelembe veszem az eltolást.

[19] Észrevettem, hogy a keskenysávú gammatone szűrők használata miatt a hangfelvételek mátrixos reprezentációban a hangindulás események a szűrők sávszélességétől függően 40-50 milliszekundummal lemaradnak a tokenekhez képest. Annak érdekében, hogy a két reprezentáció szinkronban legyen, minden t_i abszolút időponthoz hozzáadtam 45 milliszekundumot a kvantálás előtt.

- Egy zenemű különböző előadásai nem szerepelhetnek több halmazban
- A tanító/kiértékelő/teszt halmazok az összes bennük szereplő hangfelvétel hosszát tekintve nagyjából 80/10/10 arányban legyenek. Ez az arányosság legyen érvényes az egyes szerzők műveinek szétosztására is.
- A kiértékelő- és teszthalmaz változatos legyen a művek szempontjából, a nagy számban előadott zongoradarabok a tanító halmazba kerüljenek. [22]

A halmazok dimenzióit a 4.1. táblázat foglalja össze.

A WAV és MIDI fájlokat a 4.1., 4.2. és 4.3. fejezetekben ismertetett módon készítettem elő a tanításhoz és kiértékeléshez, követve az adat ajánlott felosztását. A több, mint 8 napnyi hangfelvétel előfeldolgozása nagyjából 4 napig tartott. Az átalakítást követően a bemeneti mátrixokat NumPy (*Numerical Python*), a kimeneti tokensorozatokat pedig TXT formátumban mentettem el, minden időszeletet külön fájlban. A beolvasás megkönnyítése érdekében az összetartozó bemenet-kimenet párokat azonos névvel láttam el.

Adathalmaz	**Felvételek (db)**	**Hossz (óra)**	**Méret (GB)**	**Hangok (millió)**
Tanító	962	159.2	96.3	5.66
Kiértékelő	137	19.4	11.8	0.64
Teszt	177	20.0	12.1	0.74
Összesen	**1276**	**198.7**	**120.2**	**7.04**

4.1. táblázat: A MAESTRO adatbázis tanító, kiértékelő és teszt halmazainak méretei

4.6 Transformer modell

A hangok felismerésére szolgáló Transformer modellt Python környezetben, a Pytorch könyvtár használatával hoztam létre. A modellem a [16] szerinti architektúrát követi. A Halthor-féle [23] forrás egy Pytorch alapú Transformer modellt használ természetes nyelvek közötti fordításra. A saját kódom elkészítéséhez ebből az implementációból merítettem ötleteket.

A Transformer modellem nem szövegek közötti fordítást hajt végre, hanem a 4.1. és 4.2. ábrák szerinti adatfolyamoknak megfelelően a hangfelvétel szeleteinek mátrixos reprezentációiból állítja elő tokenek sorozatát. A megoldásomban a d_{model} beágyazási dimenziót 512-nek választottam. Ehhez igazodva a bemenet beágyazását egy lineáris neurális réteggel valósítottam meg, ami a bemeneti szekvencia elemeit a 200 dimenziós vektortérből a d_{model} dimenzióba vetíti. A kimeneti beágyazáshoz szükség van egy szótárra, ami a hangesemények leírására szolgáló *<n=0>, ..., <n=87>, <t=0>, ..., <t=199>* 288 tokent, valamint az *<end>* és *<pad>* speciális tokeneket tartalmazza. A szótárt használva egy token beágyazását úgy kaphatjuk meg, hogy először létrehozunk egy, a szótár méretével megegyező hosszúságú (itt 290 elemű) nullvektort, amin 1-be állítjuk a token szótár szerinti indexének megfelelő elemet. A kapott vektort a bemenethez hasonlóan egy lineáris réteggel képeztem le a d_{model} dimenzióba.

A létrehozott mátrixokon a modell a 3.6. fejezetben ismertetett működést valósítja meg. Több tanítási próbálkozás tapasztalata alapján végül egy viszonylag kis méretű, 2 kódoló-dekódoló rétegből álló modellt választottam $d_{\text{ff}} = 2048$ és $p_{\text{dropout}} = 0.1$ beállításokkal a Feed Forward alrétegekben és $h = 8$ Attention fejjel. A kimeneti szekvencia maximális hosszának 290-et választottam, ami 145 billentyűleütés leírására alkalmas minden 2 másodperces szakaszon.[20] Az így létrehozott modell 15 114 019 változtatható súllyal rendelkezik, melyek értékét a tanítási folyamat állítja be. A kiválasztott modell teljes mérete 57.66 MB, ami az architektúrát leíró állandókat és a tanításkor meghatározott súlyokat egyaránt tartalmazza.

4.7 Tanítás

A Transformer architektúra és általában a neurális hálózatok struktúrája lehetőséget ad a nagyméretű számítások párhuzamos futtatására, ezért a tanításukat GPU-n végzik. A választott modellt 4 napig tanítottam a HUN-REN Számítástechnikai és Automatizálási Kutatóintézet Mesterséges Intelligencia Nemzeti Laboratóriumának egyik NVIDIA A100-SXM4-40GB grafikus kártyáján.[21] A batch mérete 32 volt.

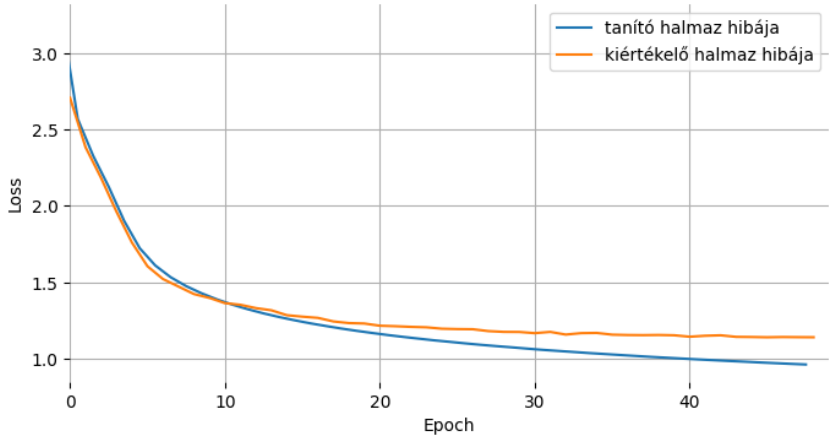
[20] Az összes, tanításhoz, kiértékeléshez és teszteléshez használt tokensorozat maximális hossza 284 volt. A maximális szekvencia hossza nem szükségszerűen azonos a szótár méretével.

[21] A folyamatot némileg lassította a külön fájlokban tárolt adatpontok többszöri beolvasása.

A tanítás során a predikciós hiba számítására a Pytorch saját, nyelvi feladatoknál is használatos *CrossEntropyLoss* függvényét alkalmaztam. A függvény a prediktált kimeneti szekvencia eltérését elemenként hasonlítja össze az elvárt kimenettel minden olyan pozícióban, ahol a kimeneti szekvenciában nem a <*pad*> token található.

A gradiens számításához és a súlyok értékének frissítéséhez a Pytorch saját *Adam* optimalizálóját használtam, 10^{-4} tanulási rátával. A 4.6. ábrán megfigyelhető az első 49 epoch átlagos hibájának (*loss*) változása a tanító- és kiértékelő halmazokon. Jól látható, hogy a tanító halmaz hibájának lassuló, de folyamatos csökkenése mellett a kiértékelő halmaz hibája egy idő után megállapodott az 1.14 körüli értékeken.

A hangfelvételek lejegyzését megvalósító alkalmazásomban a billentyűk felismerésére a 45. epoch utáni modellt választottam.[22]



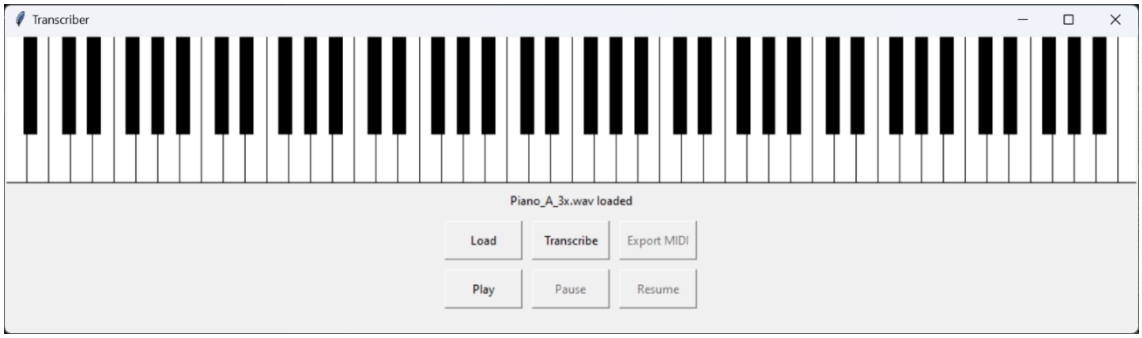
4.6. ábra: A tanító- és kiértékelő halmaz hibájának változása a tanítás során

4.8 Grafikus felület

Tetszőleges zongorajátékról készített WAV hangfelvételen a 4.1. ábrán bemutatott folyamat megvalósítására és az eljárás működésének szemléltetésére készítettem egy alkalmazást, ami a 4.7. ábrán látható, felhasználóbarát grafikus felülettel rendelkezik. A felhasználó a gombok segítségével lejátszhatja és

[22] A választás önkényes, mivel a modell ezen a ponton nem mutatja jelét a túltanulásnak (a kiértékelő halmaz hibája nem kezdett el emelkedni) és a tanító halmaz hibája is ereszkedik még. A tanítás ez alapján folytatható, amire kitérek az 5.2. alfejezetben. Az 5.1. részben részletesen kiértékelem a választott modellt és bemutatom, hogy már a viszonylag rövid tanítást követően is meggyőző eredményeket hozott.

feldolgozhatja a kiválasztott hangfelvételt, a kirajzolt billentyűzeten megfigyelheti az eljárás eredményeit és elmentheti a létrehozott MIDI fájlt.



4.7. ábra: Az alkalmazás grafikus felülete

Az alkalmazást Python környezetben készítettem el, a grafikus elemekhez a Tkinter, a felvételek lejátszásához pedig a Pygame könyvtárakat használva. A determinisztikus jelfeldolgozás Matlab környezetben megírt kódját a MATLAB Engine API for Python csomag segítségével hívom meg az alkalmazásból. A Transformer modell a Pytorch-, a MIDI fájlt létrehozó függvények pedig a Mido könyvtárakat használják.

A felhasználó a *Load* gomb hatására a fájlendszerben navigálva kiválaszthatja a feldolgozni kívánt WAV fájlt. A betöltött hangfelvétel lejátszása a *Play*, *Pause* és *Resume* gombokkal vezérelhető.

A *Transcribe* gomb megnyomását követően az alkalmazás a Matlab kódot használva létrehozza a felvétel 2 másodperces szakaszainak sokcsatornás mátrix reprezentációit, melyeket sorszámozott fájlok formájában egy dedikált munkakönyvtárba helyez. Beolvasást követően innen kerülnek a mátrixok a Transformer bemenetére, amely sorban leképezi a hangok tokeneit minden időszelethez. Mivel a számítások ideje függ a hangfelvétel hosszától, az alkalmazás folyamatjelző sávokkal jelzi a feldolgozás állapotát a felhasználó számára. Az alkalmazás ellenőrzi és javítja a Transformer kimenetét, ezzel garantálva, hogy a szekvenciák a MIDI formátumhoz illeszkedve a billentyűk leütését egyértelműen azonosító <*n=x*><*t=y*> tokenpárok sorozatai legyenek. A szűrt, egymást követő szekvenciák alapján megállapítom a billentyűk megszólalásának valódi idejét és mindegyikhez hozzárendelek egy felengedési eseményt is úgy, hogy minden billentyű 1 másodpercig szóljon. Ezt követően a leütéseket és felengedéseket a valódi idejük szerint sorba rendezem és mindegyik időbélyegét 45 milliszekundummal csökkentem a gammatone

szűrők késleltetésének kompenzálása miatt. Ha a felhasználó a feldolgozást követően játssza le a hangfelvételt, a kirajzolt billentyűzet az események listája alapján, frissülő színezéssel követi a billentyűk állapotának változását, ezzel adva lehetőséget a kapott eredmények megfigyelésére.

Az *Export MIDI* gomb hatására az alkalmazás az események listájából létrehozza és a felhasználó által kiválasztott helyre menti a lejegyzett MIDI fájlt. A kimeneti MIDI fájlt egy *track*-en, $\frac{4}{4}$ ütemmutatóval, a $tempo = 500000$ és $TPB = 480$ beállításokkal készítem el. A billentyűk leütésének *velocity* értéke minden esetben 64, ami közepes erősségű megszólaltatást jelent. Az üzenetek létrehozásánál figyelek az abszolút időpillanatok relatív, *tick* mértékegység szerinti átalakítására. A felhasználó a létrehozott MIDI fájlt célszoftverek segítségével szabadon megszólaltathatja, szerkesztheti, szólamokra bonthatja vagy akár kottát is készíthet belőle.

5 Végeredmény

5.1 Értékelés

Csupán a 4.6. ábrán látható hibaértékek alapján nehéz kiértékelni a modell lejegyző képességét. A megfelelő billentyűk eltalálása például fontosabb, mint a leütés pontos idejének detektálása, a két tokentípus között mégsem tesz különbséget a modell. Előbbi esetében már fél hang tévesztés is hibának számít, míg utóbbinál a tempó függvényében néhány 10 milliszekundumos hiba is elfogadható. Probléma továbbá az is, hogy ha a modell a viszonylag egy időben megszólaló hangok magasságát helyesen találja el, megszólalásuk idejét pedig elfogadható hibával, a hangokat jelző tokenpárokat viszont nem az elvárt kimenet szerinti sorrendben adja meg, átrendezés nélkül magasabb hibát kapunk, annak ellenére, hogy a két leírás információtartalma azonos. A modell releváns teljesítőképességének értékelése céljából az alábbi definíciókat vezettem be:

- *megtalált hang* – a prediktált billentyűleütés-eseménytől maximum 50 milliszekundumos eltéréssel megszólal ugyanaz a billentyű a referenciakimenetben.
- *hibás hang* – a prediktált eseményhez nem tartozik a *megtalált hang* feltételét kielégítő esemény a referencia kimenetben. Ezt további kategóriákra bontottam:
  - *oktávhiba* – a prediktált esemény egy oktávval magasabb vagy mélyebb hangra kielégítené a *megtalált hang* feltételét.
  - *kromatikus hiba* – a prediktált esemény egy fél hanggal magasabb vagy mélyebb hangra kielégítené a *megtalált hang* feltételét.
  - *hang kései felismerése* – prediktált esemény hangmagassághelyes és olyan billentyűre vonatkozik, ami szól az adott időpillanatban, de a valódi megszólaláshoz képest több, mint 50 milliszekundumos késéssel érkezik.
  - *egyéb hiba* – a fenti kategóriák egyikébe sem sorolható esemény.

A modell teljesítőképességét a fenti definíciók szerint mértem. A 88 billentyűs hangszert 9 oktávra szokás osztani, melyek határait a C hangok képezik. Ezek közül a legmélyebb („szubkontra”) összesen három hangot tartalmaz, a legmagasabb („ötvonalas”) pedig egyetlen C billentyűből áll. A megoldásomban használt gammatone szűrők sávszélessége miatt a 55.3 Hz alatti harmonikus komponensekhez nem tartozik a mátrix reprezentációban szereplő csatorna, ami miatt a második legmélyebb oktáv („kontra”) A# hangja alatt nagyobb bizonytalanságra számíthatunk. Zongorán mind kíséret-, mind pedig szóló játék közben leginkább az ennél magasabb tartomány a használatos, ami a MAESTRO adatbázis felvételein is tetten érhető. Nem meglepő módon a modellem a hangszer középső oktávjain a legpontosabb, hiszen a tanító halmazban szereplő legtöbb hangesemény is itt található. A modell pontosságának mérését oktávonként végeztem el, a tanító halmazon összesen négyszer előforduló 9. oktáv C hangját nem vettem bele az elemzésbe. A 4.7. ábrán láthatók a modell statisztikai eredményei a tanító-, kiértékelő- és teszthalmazokon.

Hasonló célú, automatikus lejegyző megoldások esetében Kashino [24] *note recognition rate* metrikáját szokás használni a modell hangfelismerési képességének mérésére. Ez egy százalékos érték, definíciója a következő:

$$R = 100\left(\frac{right - wrong}{total}\frac{1}{2} + \frac{1}{2}\right) [\%],$$

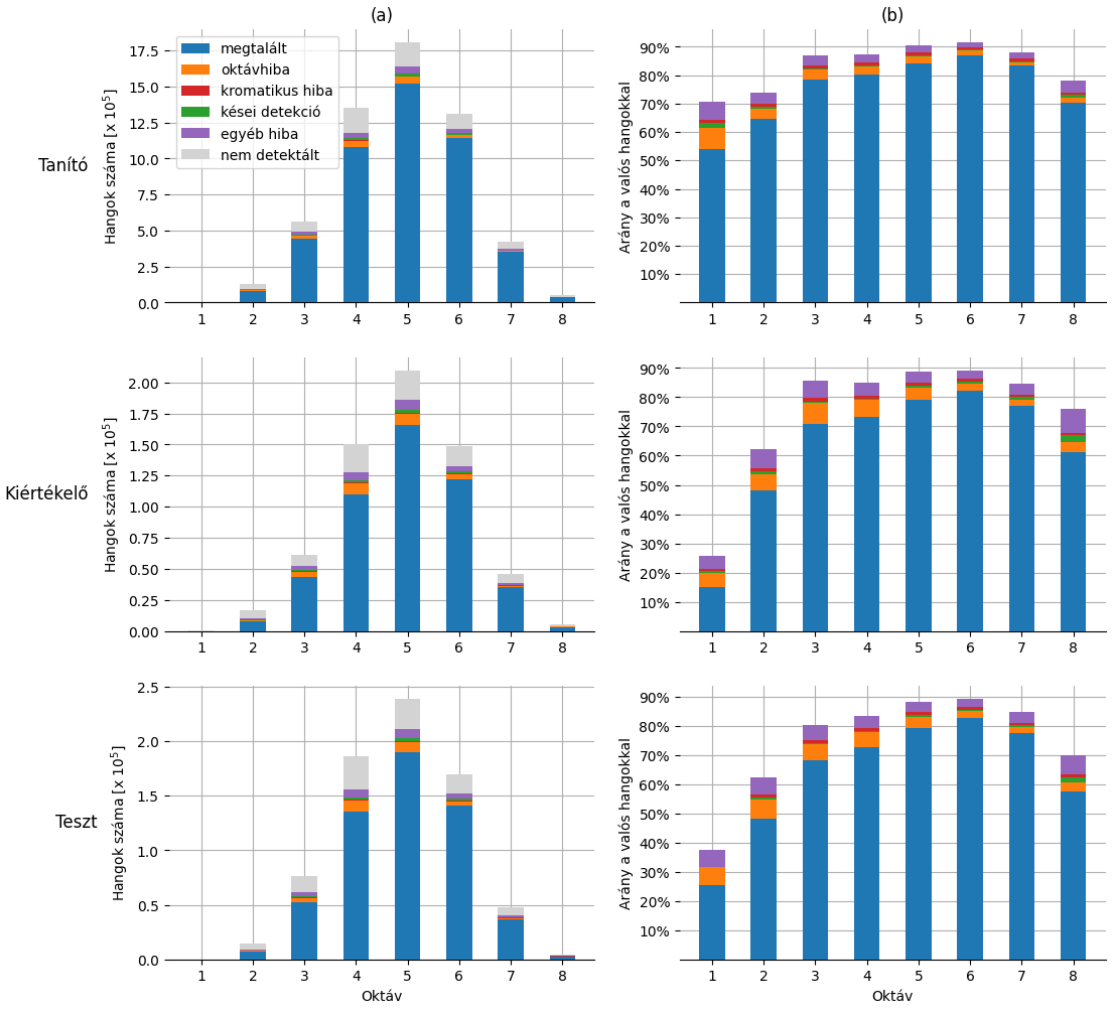
ahol *right* az összes *megtalált hang-*, *wrong* az összes *hibás hang* száma, *total* pedig a valódi billentyűleütések összesített száma a vizsgált halmazon. A választott modellem oktávonként mért és a teljes zongorára vonatkozó *R* értékeit a 4.2. táblázatban foglalom össze. Releváns tartománynak nevezem és külön vizsgálom a 3-8 oktávokat is.

Adathalmaz	**1.**	**2.**	**3.**	**4.**	**5.**	**6.**	**7.**	**8.**	**Teljes**	**Releváns**
Tanító	68.5	77.8	85	86.5	88.9	91.3	89.4	81	**88.2**	**88.4**
Kiértékelő	52.2	67	78	80.6	84.8	87.5	84.6	73	**83.2**	**83.7**
Teszt	56.7	67	78	81	85	88	85	72.6	**83.6**	**83.9**

4.2. táblázat: A modell *R* értékei az egyes oktávokat, a teljes zongorát és a releváns tartományt vizsgálva

A modell értékelésére a teszt halmaz eredményét használom, mely alapján sikerült olyan alkalmazást készítenem, amely további hozzáadott információ használata

nélkül, 83.6%-os bizonyossággal helyesen ismeri fel a hangokat szóló zongorajátékról készült hangfelvételeken.



4.7. ábra: A modell statisztikai eredményei a három adathalmazon a modell kimenete és a valós billentyűesemények összevetéséből. Az (a) oszlop diagramjain a definiált csoportokba eső, prediktált hangesemények száma látható oktávonként. A *nem detektált* osztály az összes valós hang és a modell kimenetén kapott hangok száma közti különbséget jelöli. Kitűnik a 4-6 oktávok jelentősége. A (b) oszlopban az oktávok megtalált és hibásan detektált hangjainak arányai figyelhetők meg az adathalmaz felvételein az adott oktávban megszólaló összes valós hang számához viszonyítva. Jól látható, hogy az első két oktáv pontossága elmarad a többihez képest.

A modell pontosságát kipróbáltam különböző nehézségű művek és gyakorlatok hangfelvételein, amelyek nem szerepeltek a tanító- és kiértékelő halmazokban:

- Kétszólamú diatonikus és kromatikus skálagyakorlatok: 93%
- Szűk- és tágfekvésű jazz harmóniák: 81.4%
- Kétkezes jazz harmóniák: 79.4%

- Chick Corea – *Children's Song No. 1*: 92.8%
- Bartók Béla – *Mikrokozmosz V, No. 130, „Falusi tréfa"*: 89.6%
- Claude Debussy – *Préludes I, „Des pas sur la neige"***:** 84.8%
- J. S. Bach – *WTC I, Ab-dúr Prelűdium és Fúga*: 94.4%
- Ludwig van Beethoven – *9. E-dúr zongoraszonáta*: 84.2%
- Liszt Ferenc – *IX. magyar rapszódia*: 80.4%

5.2 Továbbfejlesztési lehetőségek

A projekt továbbfejlesztésére számos lehetőség nyílik. Egyrészt a determinisztikus jelfeldolgozási modul Matlab kódját érdemes volna Python nyelven is elkészíteni. Így nagyban egyszerűsödne az architektúra és nem lenne szükséges Matlab környezet telepítése az alkalmazás futtatásához.

A hangjel előfeldolgozásában van tere a csatornák számával, az időszeletek méretével vagy a szűrők és oszcillátorok paramétereinek értékével való további kísérletezésnek. Az időszeletek közötti éles határok helyett az eredményül kapott mátrixok átfedésben is lehetnek egymással, ami dúsítaná az adatot. Az adat mennyiségét növelhetné további felvételek hozzáadása is, illetve az összes hangfájlt hangszínszabályozókkal, zengetőkkel vagy különböző intenzitású szélessávú zaj felkeverésével módosítva további adatpontok hozhatók létre. Utóbbiakkal többszörösére skálázható az adatbázis mérete, ráadásul a modell robusztussága is növekedhet az eltérő minőségű felvételekre.

A választott Transformer modell tanítása a görbék alapján folytatható, de akár másképp paraméterezett és tanított hálózatokkal is kísérletezhetünk. A billentyűket és időpontokat jelző tokenek eltérő szerepét a hibafüggvény definiálásánál is ki lehet használni, a hangmagasságban való eltérésre nagyobb hibát adva mint az időbeli pontatlanságra. A tokenpárok sorrendjéből eredő hibák kiküszöbölése is megfontolandó volna.

A rendszer elé támasztott követelmények kiterjeszthetők a leütés erősségének és a billentyűk felengedési idejének detektálására is. Ezekkel nőne az alkalmazás komplexitása, ami a feldolgozási lépések módosítását vagy új modulok integrálását vonhatja maga után.

Az alkalmazás mobil eszközökre készített vagy web alapú változatának létrehozása is lehetséges fejlesztési utak lehetnek. Akár az eljárás jelenlegi képességeire építve a detektált hangok további elemzésével a hangnemet, az akkordok típusát, fordítását felismerő, vagy a megszólaló hangokat megnevező, megjelenítő oktatási célú alkalmazás fejleszthető, esetleg valós idejű kiértékeléssel.

Ambiciózus célnak számítana, ha MIDI helyett automatikusan létrehozott, precíz kottát várnánk az eljárás kimenetén, például MusicXML fájl formájában. Ehhez a tempó, a lüktetés és a hangértékek hosszának automatikus, pontos detektálása is szükséges. Fontos továbbá, hogy az alkalmazás különbséget tudjon tenni a különböző ritmusértékek, tempóváltások, gyorsítások, lassítások, koronák és a játékos pontatlansága között, hiszen ezek mindegyike hatással van a hangok időbeliségére, de egészen különböző jelentéssel és jelölésekkel rendelkeznek. A detektált hangok időbeliségéhez társított jelentéstartalom azonosításával már a használható kotta készítéséhez elegendő információ állna rendelkezésre, a feladatrész megoldása viszont komoly kihívást jelenthet. A hangnem, metrum, hangsúlyok és dinamikai elemek felismerése is hasznos lehet, ahogy a két kéz által játszott hangok értelmes elrendezésének megvalósítása a kottasorok között.

6 Összegzés

A diplomamunkám keretében olyan alkalmazás készítése volt a célom, amely akár többszólamú zongorajátékról készült WAV hangfelvételeken képes azonosítani, hogy mely billentyűket mikor ütötte le a játékos, a lejegyzés eredményét pedig MIDI fájlformátumban adja vissza.

A polifonikus zenei környezetben rejlő főbb kihívások és néhány automatikus megoldási mód bemutatása után az általam választott eljárást ismertettem, amely az emberi hallás működését utánozza és klasszikus jelfeldolgozási lépéseket vegyít gépi tanulási módszerekkel. Részleteztem a gammatone szűrők, adaptív oszcillátorok, neurális hálózatok és a Transformer architektúra működését.

A feladat megoldását tárgyaló részben bemutattam az alkalmazás elemeinek elkészítését, a használt adatbázist, a neurális hálózat tanítását és a létrehozott grafikus felület felépítését, funkcióit.

Végül szemléltettem a választott modell pontosságát, valamint kitértem a továbbfejlesztési lehetőségekre is.

A téma szakmailag izgalmas kihívásnak bizonyult, sok ötletet igényelt és számos területen adott lehetőséget a fejlődésre, kísérletezésre. Az eredményül kapott alkalmazás hasznos segítséget nyújthat különböző zenei feladatokban.

Köszönetnyilvánítás

Először is konzulensemnek, Dr. Rucz Péternek szeretnék köszönetet mondani a témám felkarolásáért és a rengeteg segítségért amit az elmúlt években a munkámhoz kaptam. Köszönöm Dr. Benczúr Andrásnak és a Mesterséges Intelligencia Nemzeti Laboratóriumnak a lehetőséget a modell tanításához szükséges erőforrás használatára. Köszönöm továbbá családom, barátaim, tanáraim, munkahelyi-, konzervatóriumi- és kollégiumi társaim minden segítségét és támogatását a mesterképzést felölelő időszakban.

A kutatás egyes részei az Európai Unió támogatásával valósultak meg, az RRF-2.3.1-21-2022-00004 azonosítójú, Mesterséges Intelligencia Nemzeti Laboratórium projekt keretében.

Irodalomjegyzék

[1] McGill University: *Standard MIDI-File Format Spec. 1.1* https://www.music.mcgill.ca/~ich/classes/mumt306/StandardMIDIfileformat.html

[2] Várfi L.: *Hangmagasság-detektálás többszólamú hangmintában*, Bsc szakdolgozat, BME Villamosmérnöki és Informatikai Kar, 2012

[3] Szemery H.: *Automatikus kottázás NMF-algoritmussal*, TDK dolgozat, BME Villamosmérnöki és Informatikai Kar, 2020, https://tdk.bme.hu/ConferenceFiles/VIK/2020/Paper/Automatikus-kottazas-NMFalgoritmussal.pdf?paperId=10015

[4] Klapuri, A., *Auditory Model-Based Methods for Multiple Fundamental Frequency Estimation*, Klapuri, A., Davy, M.: *Signal Processing Methods for Musical Transcription*, Springer, New York, 2006, pp. 247-248

[5] Marolt, M.: *A connectionist approach to automatic transcription of polyphonic piano music*, in *IEEE Transactions on Multimedia*, 6 (3), 2004, https://doi.org/10.1109/TMM.2004.827507

[6] Netter, F. H.: *Atlas of Human Anatomy*, Saunders/Elsevier, Philadelphia, 2014, 6, pp. 94-100

[7] Patterson, R. D., Robinson, K., Holdsworth, J., McKeown, D., Zhang, C., Allerhand., M.: *Complex Sounds and Audatory Images*, in *Auditory physiology and perception, Proc. 9th International Symposium on Hearing*, 1992, https://www.pdn.cam.ac.uk/system/files/documents/Petal92ish.pdf

[8] Slaney, M.: *An Efficient Implementation of the Patterson-Holdsworth Auditory Filter Bank*, 1993, https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=07eab74512fa7d1b9554274a61a0d077d4a6ecc4

[9] Meddis, R.: *Stimulation of mechanical to neural transduction in the auditory receptor*, in *The Journal of the Acoustical Society of America*, 79, pp. 702–711, 1985, https://doi.org/10.1121/1.393460

[10] Large, E. W., Kolen, J. F.: *Resonance and the Perception of Musical Meter*, in *Connection Science*, 6 (2-3), 1994, pp. 177-208, https://doi.org/10.1080/09540099408915723

[11] Bishop, C. M.: *Neural networks and their applications*, in *Review of Scientific Instruments*, 65, 1994, pp. 1803–1832, https://doi.org/10.1063/1.1144830

[12] Barinov, R., Gai, V., Kuznetsov, G., Golubenko, V.: *Automatic Evaluation of Neural Network Training Results*, in *Computers*, 12 (2), 2023, https://doi.org/10.3390/computers12020026

[13] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*, in *The Journal of Machine Learning Research*, 15 (1), 2014, pp. 1929-1958, https://dl.acm.org/doi/10.5555/2627435.2670313

[14] Hawthorne, C., Simon, I., Swavely, R., Manilow, E., Engel, J.: *Sequence-to-Sequence Piano Transcription with Transformers*, in *Proc. of the 22nd Int. Society for Music Information Retrieval Conf.*, 2021, https://archives.ismir.net/ismir2021/paper/000030.pdf

[15] Hochreiter, S., Schmidhuber, J.: *Long Short-term Memory* in *Neural computation*, 9 (8), 1997, pp. 1735–1780, https://doi.org/10.1162/neco.1997.9.8.1735

[16] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A., Kaiser, L., Polosukhin, I.: *Attention Is All You Need*, 2017, https://doi.org/10.48550/arXiv.1706.03762

[17] Ramesh, A., Dhariwal, P., Nichol, A., Chu, C., Chen, M.: *Hierarchical Text-Conditional Image Generation with CLIP Latents*, 2022, https://doi.org/10.48550/arXiv.2204.06125

[18] Radford, A., Kim, J., Xu, T., Brockman, G., McLeavey, C., Sutskever, I.: *Robust Speech Recognition via Large-Scale Weak Supervision*, 2022, https://doi.org/10.48550/arXiv.2212.04356

[19] Devlin, J., Chang, M., Lee, K., Toutanova, K.: *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 1, 2019, https://doi.org/10.18653/v1/N19-1423

[20] Yenduri, G., Ramalingam, M., Chemmalar, G., Supriya, Y., Srivastava, G., Maddikunta, P., Deepti, G., Jhaveri, R., Prabadevi, B., Wang, W., Vasilakos, A., Gadekallu, T.: *Generative Pre-trained Transformer: A Comprehensive Review on Enabling Technologies, Potential Applications, Emerging Challenges, and Future Directions*, 2023, https://doi.org/10.48550/arXiv.2305.10435

[21] The MAESTRO Dataset, https://magenta.tensorflow.org/datasets/maestro

[22] Hawthorne, C., Stasyuk, A., Roberts, A., Simon, I., Huang A., Dieleman, S., Elsen, E., Engel, J., Eck, D.: *Enabling Factorized Piano Music Modeling and Generation with the MAESTRO Dataset*, in *International Conference on Learning Representations*, 2019, https://doi.org/10.48550/arXiv.1810.12247

[23] Halthor, A.: *Transformer-Neural-Network*, 2023, https://github.com/ajhalthor/Transformer-Neural-Network

[24] Kashino, K., Nakadai, K., Kinoshita, T., Tanaka, H.: *Organization of hierarchical perceptual sounds: music scene analysis with autonomous processing modules and a quantitative information integration mechanism*, in *IJCAI'95: Proceedings of the 14th international joint conference on Artificial intelligence*, 1, 1995, pp. 158-164, https://www.ijcai.org/Proceedings/95-1/Papers/021.pdf