FELADATKIÍRÁS

Az elektronikusan beadott változatban ez az oldal törlendő. A nyomtatott változatban ennek az oldalnak a helyére a diplomaterv portálról letöltött, jóváhagyott feladatkiírást kell befűzni.



Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Kiss Andor

Zeneszerzés gépi tanulás használatával

Konzulens

Dr. Szegletes Luca

BUDAPEST, 2021

Tartalomjegyzék

[Összefoglaló 7](#_Toc83751673)

[Abstract 8](#_Toc83751674)

[1 Bevezetés 9](#_Toc83751675)

[1.1 Frissítsd a dokumentumot 9](#_Toc83751676)

[1.2 Szakdolgozat, vagy diplomaterv 10](#_Toc83751677)

[1.3 Témaválasztás 10](#_Toc83751678)

[2 Elméleti háttér 11](#_Toc83751679)

[2.1 Korábbi megoldások 11](#_Toc83751680)

[2.1.1 Wavenet 11](#_Toc83751681)

[2.1.2 SampleRNN 11](#_Toc83751682)

[2.1.3 Magenta NSynth 11](#_Toc83751683)

[2.1.4 Performance RNN 11](#_Toc83751684)

[2.1.5 MusicVAE 11](#_Toc83751685)

[2.1.6 MuseGAN 12](#_Toc83751686)

[2.1.7 Wave2MIDI2Wave 12](#_Toc83751687)

[2.1.8 Music Transformer 12](#_Toc83751688)

[2.1.9 GANSynth 12](#_Toc83751689)

[2.1.10 MuseNet 12](#_Toc83751690)

[2.1.11 JukeBox 12](#_Toc83751691)

[3 Technológiai háttér 13](#_Toc83751692)

[4 Megvalósítás 14](#_Toc83751693)

[5 Eredmények 15](#_Toc83751694)

[6 Továbbfejlesztési lehetőségek 16](#_Toc83751695)

[7 Összefoglalás 17](#_Toc83751696)

[7.1 Fejezetek 17](#_Toc83751697)

[7.1.1 Feladatkiírás 17](#_Toc83751698)

[7.1.2 Címoldal 17](#_Toc83751699)

[7.1.3 Tartalomjegyzék 17](#_Toc83751700)

[7.1.4 Nyilatkozat 18](#_Toc83751701)

[7.1.5 Tartalmi összefoglaló 18](#_Toc83751702)

[7.1.6 Bevezetés 18](#_Toc83751703)

[7.1.7 Irodalomkutatás, technológiák, hasonló alkotások bemutatása 19](#_Toc83751704)

[7.1.8 A feladatkiírás pontosítása és részletes elemzése 19](#_Toc83751705)

[7.1.9 Önálló munka bemutatása 19](#_Toc83751706)

[7.1.10 Önálló munka értékelése, mérések, eredmények bemutatása 20](#_Toc83751707)

[7.1.11 Összefoglaló 21](#_Toc83751708)

[7.1.12 Köszönetnyilvánítások 21](#_Toc83751709)

[7.1.13 Részletes és pontos irodalomjegyzék 21](#_Toc83751710)

[7.1.14 Ábrajegyzék, táblázatjegyzék 21](#_Toc83751711)

[7.1.15 Függelék 21](#_Toc83751712)

[7.2 Egyéb tartalmi elemek 22](#_Toc83751713)

[7.2.1 Stílus 22](#_Toc83751714)

[7.2.2 E/1 22](#_Toc83751715)

[7.2.3 Rövidítések 22](#_Toc83751716)

[7.2.4 Technológia megválasztása 23](#_Toc83751717)

[8 Technológiai háttér 24](#_Toc83751718)

[8.1 Általános tudnivalók 24](#_Toc83751719)

[8.2 Stílusok 24](#_Toc83751720)

[8.3 Címsorok 25](#_Toc83751721)

[8.4 Másolás, beillesztés 25](#_Toc83751722)

[8.5 Mezőfrissítés 26](#_Toc83751723)

[8.6 Helyesírás 26](#_Toc83751724)

[8.6.1 Elgépelések 26](#_Toc83751725)

[8.6.2 Egyeztetés hiánya 26](#_Toc83751726)

[8.6.3 Külföldi szavak, kifejezések 26](#_Toc83751727)

[8.6.4 Stb 27](#_Toc83751728)

[8.6.5 Helyesírás ellenőrző 27](#_Toc83751729)

[8.7 Képek 28](#_Toc83751730)

[8.7.1 Beszúrás, formázás 28](#_Toc83751731)

[8.7.2 Képminőség 28](#_Toc83751732)

[8.8 Kereszthivatkozások 29](#_Toc83751733)

[8.9 Irodalomhivatkozások 30](#_Toc83751734)

[8.9.1 Pozícionálás 30](#_Toc83751735)

[8.9.2 Mikor kell hivatkoznom? 31](#_Toc83751736)

[8.10 Word tippek és trükkök 31](#_Toc83751737)

[8.10.1 Navigációs ablak 31](#_Toc83751738)

[8.10.2 Megjegyzések 31](#_Toc83751739)

[8.10.3 Korrektúra 32](#_Toc83751740)

[8.10.4 Gyorsbillentyűk 32](#_Toc83751741)

[8.11 Kódrészletek 32](#_Toc83751742)

[8.11.1 Formázás 33](#_Toc83751743)

[8.11.2 Irodalomjegyzék 34](#_Toc83751744)

[8.12 Utolsó simítások 34](#_Toc83751745)

[9 Irodalomjegyzék 35](#_Toc83751746)

[Függelék 36](#_Toc83751747)

Hallgatói nyilatkozat

Alulírott **Kiss Andor**, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot 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, 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, 2021. 09. 28

...…………………………………………….

Kiss Andor

Összefoglaló

abstract magyarul

Abstract

Ide jön a ½-1 oldalas angol nyelvű összefoglaló, amelynek szövege a Diplomaterv Portálra külön is feltöltésre kerül. Ez a magyar nyelvű összefoglaló angolra fordított változata.

# Bevezetés

A zeneszerzés az őskor óta jelen van az emberiség életében. Ahogy múlik az idő, úgy jelennek meg ott is egyre modernebb technológiák. Kezdetben megjelentek akusztikus hangszerek, például a hárfa, később a lant. Az elektronika megjelenését nem sokkal később követték az elektromos hangszerek is. Egy ilyen híres, még ma is használt elektromos hangszer a szintetizátor volt, ami alapból zenei hullámformákat volt képes generálni különböző célhardverek, áramkörök segítségével. Később a szintetizátorok már egymással, és a számítógépekkel is képesek voltak kommunikálni, a Musical Instrument Digital Interface (MIDI) protokoll segítségével. Ahogy észrevehető ebből a rövid történelmi áttekintőből, a technológia fejlődése, a gépesítés a hangszerekre is nagy hatással volt. Ma már számos olyan számítógépes programot lehet használni, ami zeneszerzésre alkalmas, különféle hangszerek hangjait képes lejátszani, és a felhasználó kezébe adja a kontrollt, aki kreativitása segítségével alkothat, viszont itt még mindig a felhasználó, az ember a gép mögött az igazi készítő, ő választja ki a megfelelő beállításokat, a hangokat és időtartamokat, amikből összeáll a zene.

A gépi tanulás már az 1950-es években is létezett, viszont zeneszerzésre akkor még nem használták. Akkor kezdtek el ezzel foglalkozni, amikor a 2010-es években a mély tanulás (deep learning) technológia robbanásszerű fejlődésen ment keresztül. Ma már lehetséges zenét létrehozni úgy, hogy nem egy ember által szerkesztett zenét módosítanak szoftverek segítségével, hanem az alapot is teljes mértékben a szoftver adja. Természetesen egyéb szoftverek vagy az emberi tényező is megjelenik a procedúrában, például a zene utólagos feldolgozásánál, effektezésénél. Emellett azt sem szabad elfelejteni, hogy a tanító adathalmazt is emberek hozták létre, hiszen a gépi tanuló algoritmus se a semmiből szerzi ismereteit.

A zene, a hangszerek és a gépi tanulás fejlődésének érdekes párhuzama mellett személyes motivációm is van, ami miatt ezt a szakdolgozat témát választottam. Több éve gitározom, elektromos gitáron is, foglalkoztam már zenei szoftverekkel is. Mind a zene, mind a gépi tanulás világát nagyon érdekesnek tartom, így azt tűztem ki magamnak feladatként, hogy körüljárjam a témában rejlő lehetőségeket, feltérképezzem a zene számítógépes formátumait, és természetesen azt, hogy hogyan lehet generatív modellekkel létrehozni azokat.

Egy másik személyes kötődésem teszi érdekessé még ezt a projektet. A legtöbb korábbi gépi tanulás alapú zeneszerzési megoldás klasszikus zenékkel foglalkozik, azon belül is zongoraművekkel. Én nem ezzel szeretnék, hanem az egyik kedvenc előadómmal, az Iron Maidennel. Az ő zenéjük főleg gitáron és basszusgitáron van játszva, természetesen a ritmust adó dobokkal, és ritkán egyéb hangszerekkel, például szintetizátorral kiegészítve. Vokál is van szinte minden zeneszámuknál, viszont én azzal ebben a projektben nem foglalkozok, inkább a hangszerekre helyezem a fókuszt. Egy másik dolog, ami az Iron Maiden mellett szól, hogy jellegzetes, kicsit repetitív stílusuk van, ami szerintem egy gépi tanuló algoritmusnak egy könnyedén megérthető, tanulható dolog.

## Frissítsd a dokumentumot

## Szakdolgozat, vagy diplomaterv

## Témaválasztás

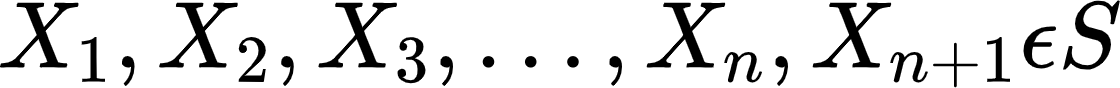
# Elméleti háttér

## Korábbi megoldások

### Markov-lánc

A modern deep learning technológiába való betekintés előtt megnéztem, hogyan lehetne egyszerűbb, valószínűség alapú gépi tanulás segítségével zenét szerezni. Ehhez az elsőrendű Markov-láncokat használtam, aminek definíciója így hangzik:

Legyen S egy állapottér, amire igaz, hogy:

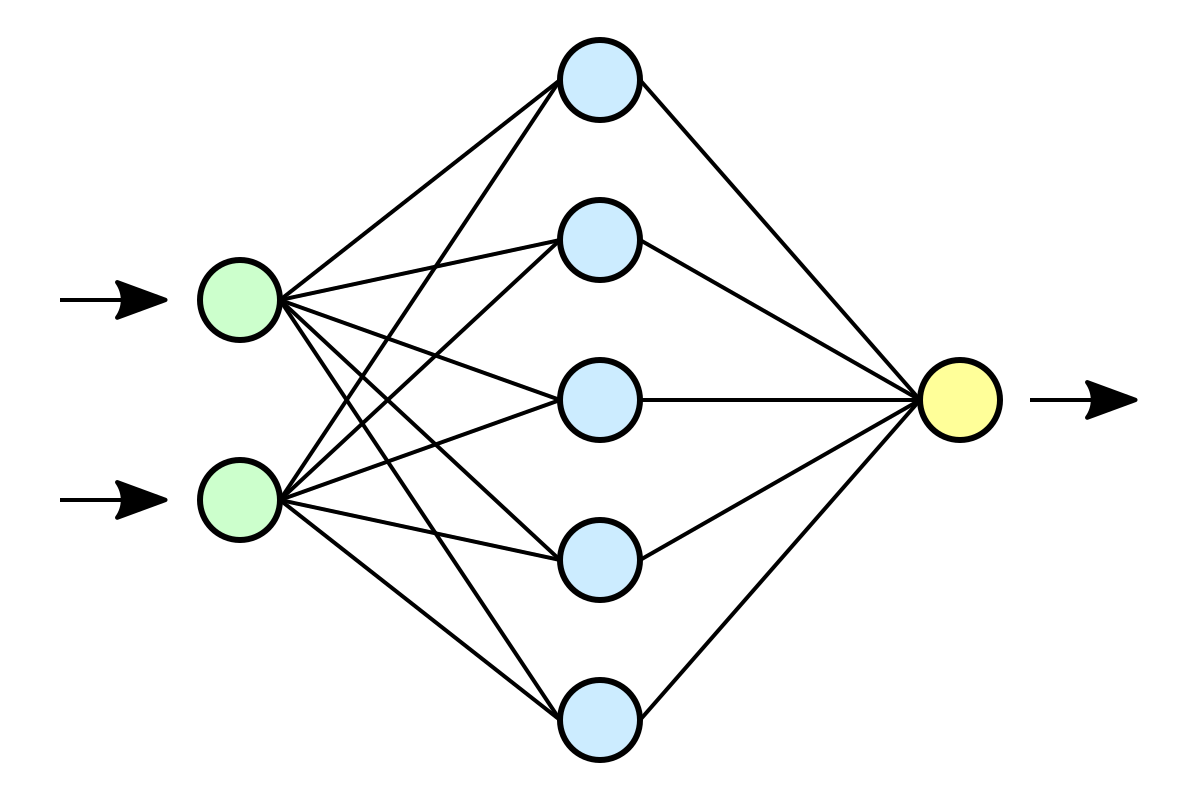
{"backgroundColor":"#ffffff","font":{"size":12,"color":"#000000","family":"Times New Roman"},"code":"$P(X_{n+1} = x | X_{1} = x_{1}, X_{2} = x_{2}, ... , X_{n} = x_{n}) = P(X_{n+1} = x | X_{n} = x_{n} )$","id":"3","backgroundColorModified":false,"type":"$","aid":null,"ts":1619815253609,"cs":"H0u+DXLe7oqhiZ7anOGtEw==","size":{"width":476,"height":16}}

Ekkor egy elsőrendű Markov-láncot definiáltunk. Az elsőrendű Markov lánc egy olyan generatív modell, ahol minden állapot csak az azt megelőzőtől függ. Az átmenetek valószínűségét az úgynevezett átmenetvalószínűség mátrix határozza meg. Ez a mátrix meghatározza, hogy egy adott állapotból egy másikba mekkora valószínűséggel léphetünk át. Ha eszerint a mátrix szerint az állapotokon sorban lépkedünk, akkor létre tudunk hozni egy állapotsorozatot.

Ezt zenei alkalmazásra úgy tudnám lefordítani, hogy minden hang egy állapot jelképez, és minden i-edik hang csak az őt megelőző, i-1-edik hangtól függ. Amikor állapotsorozatot generálok, az a zene hangjait fogja tartalmazni.

### Az elemi neuron és a neurális hálózat

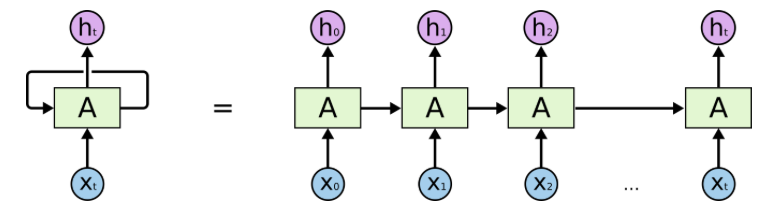
A gépi tanulás mai legnépszerűbb algoritmusának, a deep learningnek az alapvető építőeleme az elemi neuron. A neuron egy n hosszú vektort vár bemenetként, emellett egy n hosszú súlyvektorral, egy aktivációs függvénnyel és egy skalár kimenettel rendelkezik. A bemenetén kapott n hosszú vektort a súlyaival skalárisan összeszorozza, majd kimeneteként kiadja ezen skalárszorzat aktivációs függvényén átvezetett, „aktivált” értékét. Ezeket a neuronokat ritkán használják önmagukban, neurális hálózatokat szoktak belőlük alkotni, amik több, rétegbe szedett, összekapcsolt neuronból állnak.

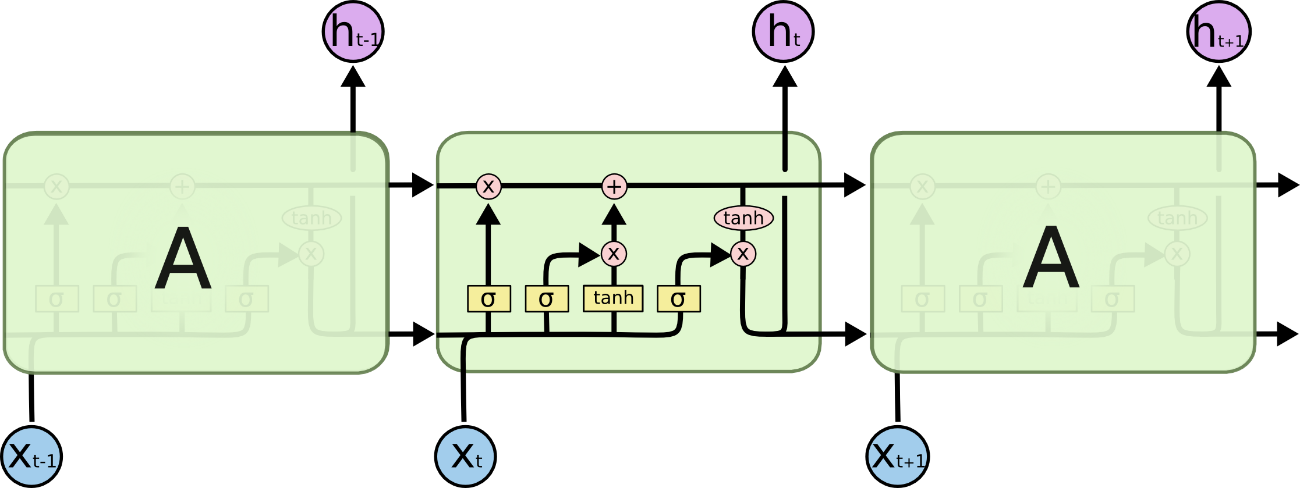


A neurális hálózatokban a bemeneti rétegbe kerül a bemenet, majd továbbításra kerül a következő rétegbe, a réteg súlyaival összeszorozva. Mivel itt már nem csak 1 neuron kapja a bemenetet, hanem több, ezért a súlyvektor helyett már súlymátrixról beszélhetünk, jelen esetben egy -ös mátrixról van szó, mivel a bemeneti rétegben 2 neuronból mennek az adatok a következő réteg 5 neuronjába. A bemeneti adatok összeszorzódnak az első súlymátrixszal, majd az ennek eredményeként kialakult súlymátrix értékei a réteg aktivációs függvényével aktiválva lesznek, és ez az érték megy tovább a következő rétegekbe. A háló ezután úgy tanul, hogy a kimeneti rétegén lévő értéket tanítás során összehasonlítják az elvárt értékkel, majd az ezek között lévő különbség függvényében a hibavisszaterjesztés (backpropagation) algoritmus módosítja a rétegek súlyait, keresve az optimális értékeket. Ahogy ez az érték egyre optimálisabb lesz, úgy fog egyre jobb, az elvárthoz közelebbi értékeket adni a háló.

### Rekurrens neurális hálók

A sima neurális hálók kimenetei csak az adott pillanatbeli bemenettől függnek, így időbeli szekvenciák, például zenék modellezéséhez nem valami jók. Ezt a problémát orvosolják a rekurrens neurális hálók (RNN), amik olyan konstrukciók, hogy kimenetük a korábbi bementeiktől is függ.

 Ha kibontunk egy ilyen rekurrens hálót, akkor egy Markov-lánchoz hasonló objektumot láthatunk. xt időbeli kiterjedéssel rendelkező bemeneti vektor feldolgozásakor az i időpillanatbeli kimenet nem csak az i időpillanatbeli bemenettől függ, hanem a korábbi kimenetektől is. A probléma ezekkel, hogy nem tudnak hoszútávú függőségeket fedolgozni, mivel a Markov-lánchoz hasonlóan csak az eggyel korábbi időpillanatből kapnak plusz információt, így hosszú távon elenyészik a jóval korábbi információ. Ezt a problémát hivatott orvosolni a Long Short Term Memory (LSTM) hálózat.



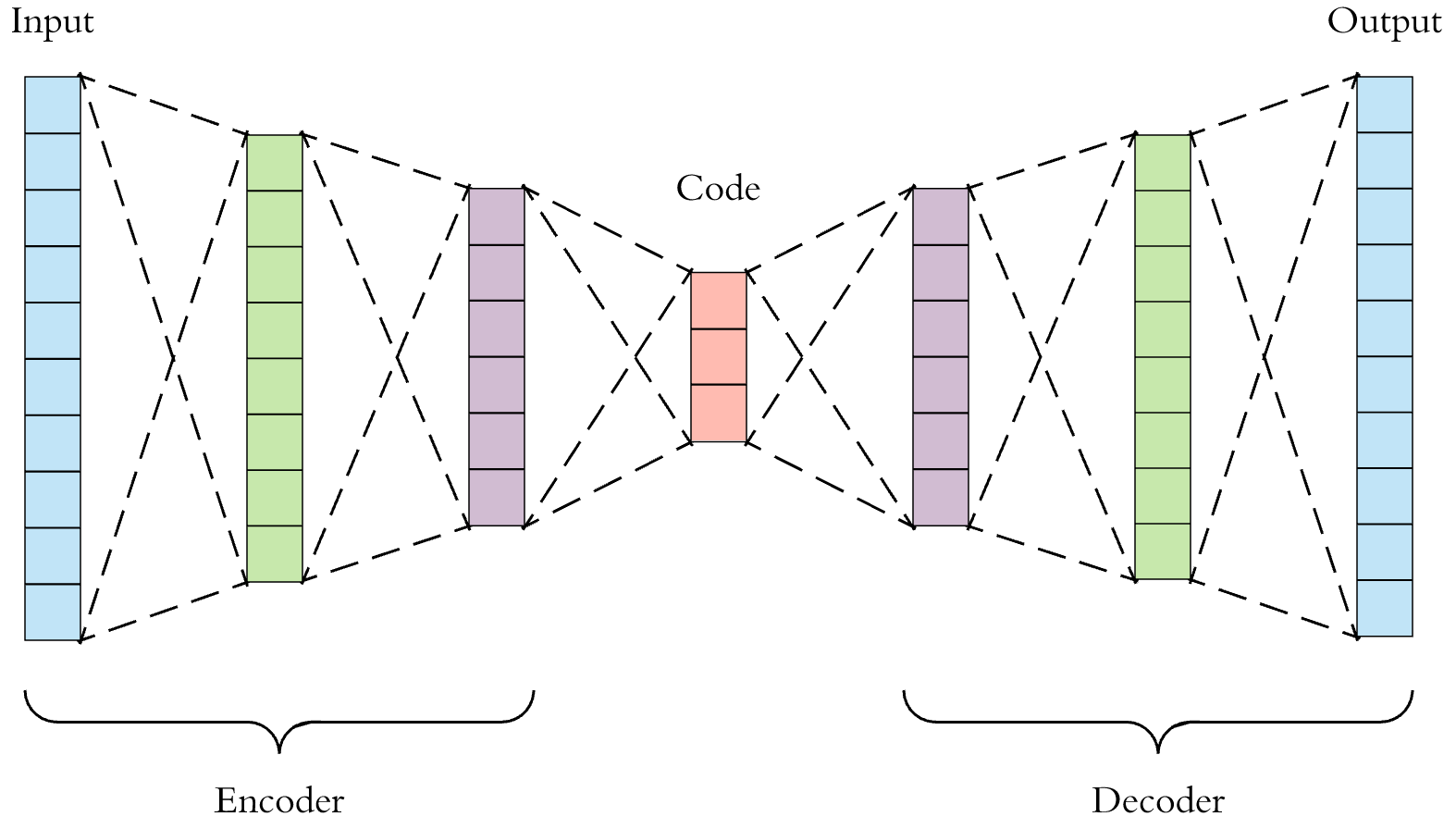
Az LSTM fölső vonalán a sima RNN-hez hasonlóan továbbmennek az adatok, viszont történnek rajtuk változások, amik elősegítik a hosszútávú függőségek feldolgozását. Az LSTM több kapuból áll, amik ezekért a változásokért felelősek. Az első kapu, az input gate egy szigmoid függvénnyel aktivált neuron, ami azt határozza meg, hogy a korábbi LSTM cellából érkező információ mennyire maradjon meg. A szigmoid 0 és 1 közötti kimenettel rendelkezik, amivel összeszorozva az információt, dől el, hogy mennyi maradjon abból meg. A 0 jelenti, hogy teljesen dobjuk el azt, az 1 pedig, hogy teljesen tartsuk meg. A cella belső állapotában lévő információt a súlyozott bemenet szigmoiddal és tangens hiperbolikusszal aktivált szorzata adja, ami aztán a fölső vonalon lévő adathoz adódik hozzá. A szigmoid előtti súlyok a forget gate súlyai, a tanh előtti súlyok pedig az LSTM cella saját súlyai. Végül a jelenlegi kimenetet kell eldöntenünk. Ehhez szintén egy kaput használunk, az output gatet, ami egy újabb szigmoid segítségével dönti el, hogy a fölső vonalon lévő adatból mennyi maradjon meg. Természetesen ennek a kapunak is vannak súlyai, így az LSTM-ről elmondható, hogy 4 súlymátrixszal rendelkezik, amik mind más-más cél érdekében tanulnak, így nehezebben taníthatók, több számítási kapacitást igényelnek, mint a sima RNN-ek, viszont jobban tudják modellezni a hosszú távú függőségeket.

### Az attention mechanizmus

### GPT-2

### Autoencoder hálózatok

Az autoencoder egy olyan neurális hálózat, amely három fontos részből, egy encoderből egy bottleneckből és egy decoderből áll, és a tanítása során a bemenet és a kimenet megegyezik.



Az alap autencoder háló úgy működik, hogy tanulása során a bemenetre és a kimenetre adott adat megegyezik. A lényeg itt a bottleneck rétegen van, ami a kódot hozza létre. A háló egy olyan kódot próbál generálni, és ehhez úgy optimalizálja súlyait, hogy aztán a kódból vissza tudja állítani az inputján kapott eredeti adatokat. Ezzel lehet például klasszifikációt végezni. Ha a bottleneck rétegre annyi neuront helyezünk el, amennyi osztályba szeretnénk rakni a klasszifikálandó adatokat, akkor a betanult hálót tudjuk úgy használni, hogy az osztályozandó adatot az inputra kötjük, akkor a generált kódból már el is dönthető az adat osztálya.

Az autoencodereknek létezik egy generatív változata is, a Variational AutoEncoder (VAE). Ez annyiban különbözik a sima változattól, hogy a kód itt egy normáleloszlású zaj. Az inputból az enkóder létrehozza ezt a zajt, majd abból a dekóder megpróbálja visszaállítani tanulás során az eredeti adatokat. Generáláskor pedig csak a dekóder részt használjuk, amibe inputként zajt helyezünk el, és az abból kiadott outputja lesz a generált adatunk.

### Wavenet

A Google DeepMind csapata által 2016-ban alkotott generatív modell, ami folytonos hullámformájú hangot képes generálni. Ez volt talán az első nagy áttörés a deep learning alapú zenélésben. Text to speechre használták nagyrészt, viszont zene generálására is alkalmas.

### SampleRNN

Ez a megoldás a Wavenethez hasonlóan diszkrét softmax kimenetet használ, viszont lecseréli a konvolúciós rétegeket RNN-ekre, amik jobban tudnak szekvenciákat modellezni, viszont túl sokáig tanulnának, mivel lassabban dolgozzák fel a bemenetet, mint a konvolúciók. Ezt úgy oldják meg, hogy hierarchikus RNN-eket használnak, amik más-más hosszúságú szekvenciákra tekintenek rá, kicsit a Wavenet különböző dilation értékkel rendelkező konvolúcióihoz hasonlóan. Méréseik szerint ez sokkal jobb zenék generálására alkalmas.

### Magenta NSynth

Ez a megoldás újra a Wavenethez nyúl vissza, viszont azt egy még nagyobb, autoencoder struktúrára cseréli le.

### Performance RNN

Ez a megoldás MIDI-ken dolgozik folytonos zene helyett. Lényegében csak egy stacked LSTM hálózat, amit a MAESTRO dataseten tanítottak.

### MusicVAE

Ez egy MIDI-n tanuló Variational Autoencoder alapú megoldás, amit én is megvalósítottam a paper alapján.

### MuseGAN

Ez egy GAN alapú, MIDI generátor megoldás. Egy érdekes gondolat itt az, hogy többhangszeres zenén tanul, és mindegyik hangszer saját generátor modellt kap, így egymástól függetlenül módosíthatóak a paramétereik.

### Wave2MIDI2Wave

Ez egy komplex modell, ami folytonos hullámformájú zene létrehozására képes, viszont a generálást MIDI-n végzi. Először a folytonos hullámformájú inputból MIDI-t csinál, majd generál új hangot a MIDI sávhoz, amit visszaalakít folytonos zenévé, és azt adja outputként.

### Music Transformer

Ez a modell a modernebb nyelvi modellekhez hasonlóan transformer alapú, azaz attention mechanizmust használ a generáláshoz. Ahogy a nyelvi modelleknél, úgy a MIDI generálásban is a transformer alapú megoldások jelentik a jövőt.

### GANSynth

Ez a modell egy konvolúciós GAN segítségével szintetizál folytonos hullámformájú zenét.

### MuseNet

Ahogy a Music Transformer esetében is lehetett látni, a MIDI generálásban átvették az uralmat a transformer alapú megoldások. Ez a modell az OpenAI híres GPT-2 nevű, hatalmas szöveggeneráló transformere, csak MIDI hangok predikciójára alkalmazva.

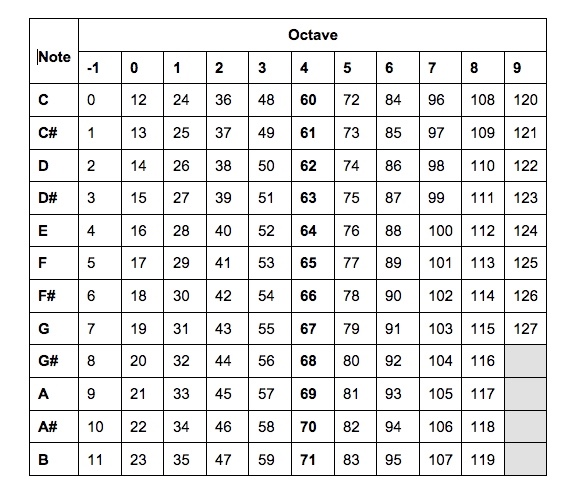
### JukeBox

Ez talán a mai legmodernebb, legjobb teljesítményre képes zenegenerátor a deep learning világában. Hatalmas modell, kombinálja a Wavenet 1D konvolúcióit a transformerek attention mechanizmusával.

# Technológiai háttér

## MIDI

A MIDI fájlok binárisan tárolják a zenét, viszont nem egy bájtfolyamként, folytonosan, az amplitúdókat tárolják, hanem a zenei hangokat. Minden hangot egy 0 és 127 közötti szám jelöl, az alábbi képen látható módon:



Emellett minden hanghoz a MIDI tárolja azt is, hogy az adott hang mennyi ideig szól, és azt is, hogy milyen erősséggel ütötték meg a hangszert.

## Zenei hullámforma

A folytonos hullámformájú fájlok úgy működnek, hogy minden időpillanatban egy valós értékű számmal írják le a zene hullámának amplitúdóját. Ezeket az amplitúdó értékeket olvasom ki belőlük.

## Python

Szakdolgozatom során implementációhoz a Python nyelvet használtam. A Python egy egyszerűen használható, magas szintű, interpretált szkriptnyelv. Dinamikusan típusos, ami azt jelenti, hogy egy adott típusú, például integer változó bármikor kaphat új, más típusú értéket, például stringet, és akkor hiba nélkül fog futni tovább a program, a változónk pedig már string típusú. Ez nagyon kedvező rövidebb, egyszerűbb scriptek megírásához, viszont egy nagyobb kiterjedésű programnál nagyon figyelmesnek kell lenni, hogy ne legyen követhetetlen a változók típusának megváltozása. A nyelv szintaxisa nagyon egyszerű, sok helyen inkább hasonlít az angol nyelvre, mint egy komplex programozási nyelvre. Nem ezek miatt választottam viszont, hanem azért, mert számos remek csomag (package) található hozzá, amiket egy egyszerű csomagtelepító szoftver, a pip segítségével lehet letölteni.

Projektem során az adatok feldolgozásához a folytonos zenék beolvasására a librosa nevű

## Keras/TensorFlow

## Google Colab

## GitHub

## Django

## Zenei hullámforma

# Megvalósítás

## Adatbeszerzés

Kétféle adattal dolgoztam a projektem során, amik más-más reprezentációi a zenének. Dolgozok folytonos hullámformájú zenékkel, amiket például mp3 formátumban szerzek be. Az adatbeszerzéshez két megoldást használtam, amelyből az egyik az, hogy a Spotify for developers Application Programming Interface (API) segítségével Representational State Transfer (REST) kéréseken keresztül letölthetek 30 másodperces előnézeteket a Spotifyon meghallgatható különféle zenékből. Ezeket a kéréseket Python kódból tudom küldeni, majd válaszként kapok egy mp3 fájlt, amit majd feldolgozhatok később. Így tetszőleges stílusú zenét letölthetek a Spotify hatalmas adatbázisából.

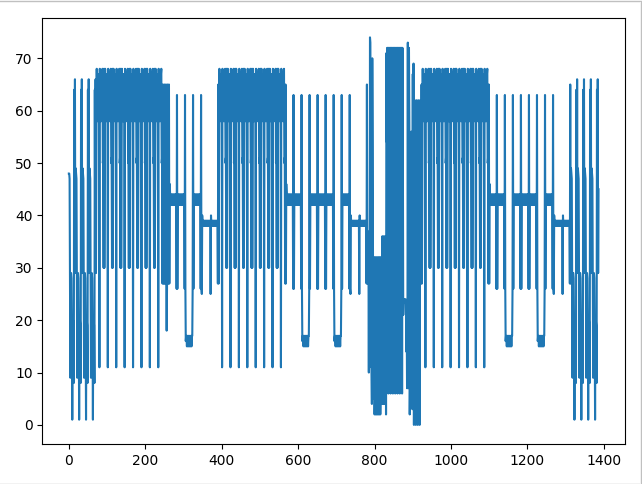
Egy másik megoldás a MAESTRO dataset használata. Ez a Google Magenta csapatának egy adatbázisa, ami ingyenesen letölthető és több, mint 100 GB-nyi klasszikus zenét tartalmaz, zongorán eljátszva. Ennek az adathalmaznak szépsége, hogy nem csak folytonos hullámformaként tartalmazza ezeket a zenéket, hanem MIDI formátumban is, így fel tudom használni projektem másik felén is, ahol MIDI formátumú zenéket használok.

A MAESTRO adathalmazon kívül egy másik MIDI beszerzési módot is találtam, amit viszont az Iron Maiden zenékhez tudok használni. Az Ultimate Guitar oldaláról előfizetők legálisan tölthetnek le Guitar Pro tabokat. A gitártabok egyszerűsített kották, amiket gitárjátékosok szoktak használni, mivel a gitár húrjain mutatják, hogy miket kell lefogni, így egyszerűbben olvashatók, mint a mindenki által ismert zenei kották. A Guitar Pro tabok ezek elektronikus változatai, amik interaktívan lejátszhatók, így még jobban segítik a gitárost az olvasásban és a játékban. Ezeket a fájlokat nem a gitártudásom növelése érdekében töltöttem le, hanem azért, mert ezek egyszerűen konvertálhatók MIDI-vé. A TuxGuitar nevű open-source programot használtam erre a konverzióra.

## Adatfeldolgozás

### MIDI adatok

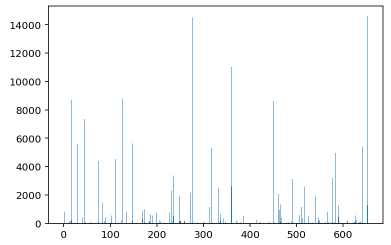
A MIDI reprezentációjú zenéket a music21 Python csomaggal olvastam be. Különbontottam a különböző hangszerek MIDI sávjait, és úgy mentem végig rajtuk, beolvasva az elemeiket. A beolvasás során Pythonos objektumokat kaptam, amik a music21 csomagban definiált osztályok. Ilyenek például a Chord, Note, Rest, Duration osztályok, amik egy akkordot, egy hangot, egy szünetet, és egy hanghosszt reprezentálnak. A MIDI-k hangerősségével nem foglalkoztam, egyrészt azért, mert az már túl sokféle adat lenne, másrészt azért, mert nem is minden MIDI zenénél kapnak a hangok külön hangerősséget, mivel annyira nem releváns ez az adat. Beolvasás után ezeket az objektumokat számokká alakítottam, mivel a gépi tanuló algoritmusok számokon működnek, nem objektumokon. Pythonos dictionaryk segítségével hoztam létre a beolvasott hangok alapján mappereket, amik átalakítják az objektumokat számokká. Itt eljutottam arra a pontra, hogy egész számokkal leírt MIDI sávjaim vannak. Ezeket már ki tudtam plottolni.



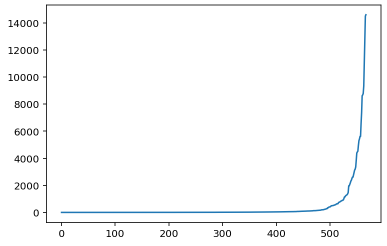
Így nézett ki egy beolvasott, és számokká alakított MIDI sáv. Ezután több különböző számmá kódolási megoldást is számításba vettem. Ezek között vannak olyan megoldások, amik azért vannak, hogy könnyítsenek a gépi tanuló modell feladatán, csökkentve a kimeneti osztályok számát.

* Egyszerűsítés kedvéért az akkordoknak csak az alaphangját (root note) veszem figyelembe, mivel azok többi hangja általában csak annak szinesítéseként van.
* Foglalkozok akkordokkal és különálló hangokkal is.
* Nem foglalkozok a hangok hosszával, feltételezek egy konstans tempót. Popzenében általában 4/4-es ütem van, viszont az Iron Maiden sajátossága, hogy másfajta ritmikákat (például tripletek) használ.
* Külön kódolom a hangmagasságot és a hosszt, így minden MIDI sávból két egész számokból álló tömböt kapok, egyik a hangok magasságát tartalmazza, másik a hangok hosszát.
* Egyben kódolom a kettőt, így lényegében embeddingeket képzek. Ezáltal egy tömböm lesz, amiben sokféle szám lesz, mivel más számértéket kap például egy negyedes hosszal rendelkező E3 hang, mint egy nyolcados hosszal rendelkező E3.

A kódolás után azt vizsgáltam, hogy az adott számértékek hányszor fordulnak elő. Akkordokkal együtt beolvasott MIDI gitársávoknál például így ilyen oszlopdiagramon tudtam ábrázolni ezt.



Látható, hogy vannak nagyon kiemelkedő vonalak, akár 14000 előfordulási számmal, viszont a 600-nál több adatból soknak szinte nem is látható az előfordulási számát jelző oszlop, olyan alacsony az. Növekvő sorrendbe rendezve ezeket az értékeket, meg tudtam tekinteni, hogy milyen függvényhez hasonló ezen értékek eloszlása.



A képen egy exponenciálishoz hasonló függvény képe jelenik meg. Ez azt jelenti, hogy az értékek nagy részének előfordulási száma a zeneszámokban elenyésző. Azt állapítottam meg, hogy ezek kiugró értékek, outlierek, ezeket el lehet dobni a tanítás egyszerűsítése érdekében. Természetesen a zenét minden hang érdekesen tudja színesíteni, viszont, ha egy adott akkord a 93 feldolgozott dal során egy adott értéknél kevesebbszer (például 5-nél kevesebbszer) fordul elő, akkor szerintem tekinthető irrelevánsnak, kivehető a feldolgozott adatok halmazából, így csökkentve a kimeneti lehetőségek számát, segítve a modell tanítását, a lehető legkisebb áldozat árán.

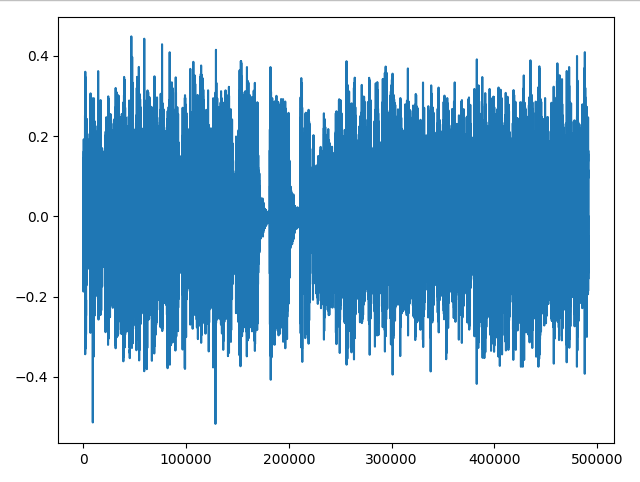
Végül a tanítás további segítése érdekében az adatokat itt is a [0; 1] intervallumba normalizáltam.

A Long-short term memory (LSTM) alapú neurális hálózataim olyanok, hogy egy megadott hosszúságú input zeneszeletből próbálja a háló megjósolni a következő hangot. Például 20 hangból mondja meg a 21.-et. Ehhez létre kellett hoznom egy függvényt, ami felszeleteli az adathalmazomat ilyen párokra. Ez a függvény bemenetként az adathalmazt várja, és a szeletek hosszát, kimenetként pedig olyan input-output párokat ad, ahol az input egy “szelethossz” hosszúságú tömb, ami számokká kódolt MIDI objektumokat tárol, az output pedig az adott szelet után következő, számmá kódolt objektum.

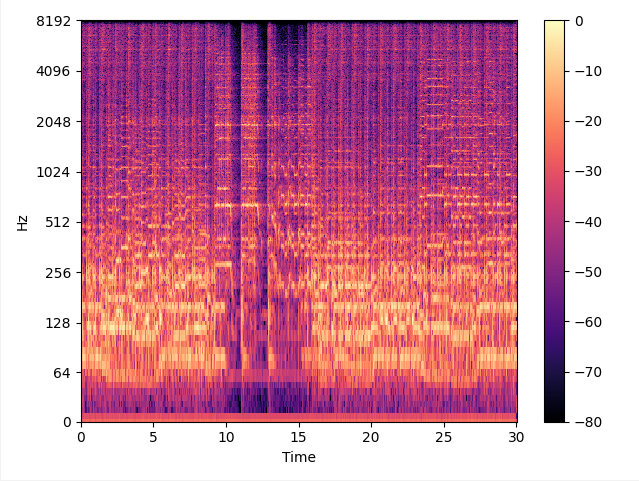
Mivel az LSTM architektúráimban softmax kimenetet használok, az output értékeket one-hot kódolom.

### Zenei hullámformák

A folytonos zenéket tároló mp3 fájlokat a librosa nevű Python csomaggal olvastam be. A beolvasásért felelős függvény egy numpy tömbbel tért vissza, ami float típusú valós számokként tárolta az adott időpillanatbeli amplitúdóját a zenének. Egy ilyen tömb elemeit matplotlib segítségével kiplottolva az alábbi időtartománybeli reprezentációt kaptam:



Ahogy az ábra x tengelyén látható, egy ilyen zeneszám nagyon sok float értékből áll. Ezt megpróbáltam a lehető legalacsonyabb értéken tartani úgy, hogy ne is veszítsek túl sokat a dal minőségéből, alulmintavételezés miatt, viszont ne kelljen a tanításhoz túl sok adattal dolgoznom. Ezért ábrázoltam frekvenciatartományon is a folytonos zenéket, hogy a spektrális tulajdonságaik alapján találhassak egy megfelelő mintavételezési frekvenciát. Az alábbi képen a kiválasztott mintavételezési frekvenciám ábrája található, 16 kHz-t választottam ezen értéknek.



Végül ezeket az folytonos értékeket a [-1; 1] intervallumba normalizáltam, mivel a preprocessing pipelineom következő algoritmusa ezen a tartományon várja az értékeket. Ezaz algoritmus pedig a 8 bites μ-law kódolás. Ennek segítségével a végtelen lehetséges valós értéket véges számú értékké alakítom (a 8 bit miatt 255-té), ezáltal kis veszteség árán tudom diszkrét értékekkel reprezentálni a folytonos zenét. Mivel a gépi tanuló algoritmusok a folytonos számokat, és a normalizált értékeket jobban szeretik, mint az egész értékeket, ezért a kódolás után leosztom a maximum értékkel a tömb minden értékét, így az adatok a [0; 1] intervallumba normalizálódnak.

## Markov-lánc

Jelen esetben nem mentem el odáig, hogy valós értékekké alakítsam, a [0; 1] intervallumba a számokat, meghagytam azokat egész számoknak, amiket stringgé alakítok, és azokat használom az állapotok neveinek. Ezután az mchmm csomag MarkovChain osztálya elkészíti nekem az átmenetmátrixot. Ezen mátrix alapján már ki tudtam indulni egy véletlenszerűen felvett állapotból (első hangja a zenének), és a valószínűségek segítségével tudtam lépkedni az újabb állapotokba. Ezekből a bejárt állapotokból egy tömböt csinálok, aminek elemeit a korábban készített mapperem, és egy kis string manipuláció segítségével visszaalakítok music21 MIDI objektumokká. Ezekből az objektumokból már könnyedén tudtam MIDI fájlokat létrehozni.

A Markov-lánc segítségével Iron Maiden zenéket próbáltam generálni, és először egyszerűbb, majd egyre komplexebb problémákat adtam a gépi tanuló algoritmusnak.

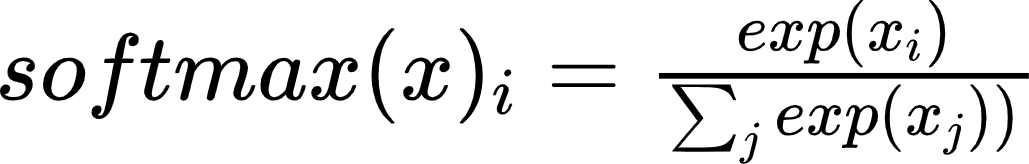
* Csak basszusgitár MIDI részeket dolgoztam fel, mivel az általában egyszerűbb, és kevesebb hangból áll, mint a többi hangszer. Emellett feltételeztem egy konstans tempót és egyszerűsítés végett az akkordoknak csak a fő hangját (root note) vettem figyelembe.
* Ezután maradtam a basszusgitárnál, viszont a teljes akkordokkal dolgoztam.
* Ezután bekapcsoltam a változó hanghosszokat is, külön kódolva, egy másik Markov-lánc segítségével generálva azokat.
* A basszusgitárt lecseréltem rendes gitárra, de az akkordok helyett maradtak csak a root noteok.
* Bekapcsoltam az akkordokat is, viszont itt falba ütköztem, mivel az állapotvalószínűség mátrix létrehozásának ideje elszállt, mivel túl sok fajta hang lehetőség volt, így az nem futott le csak percek alatt, ezért nem mentem tovább, mivel ennél többféle hangból álló zene esetén még sokkal hosszabb lenne a mátrix létrehozásának ideje, így az egybekódolt hossz+hang kombóval nem próbálkoztam.

A μ-law algoritmussal diszkrét értékekké alakított folytonos zenéimen próbáltam a Markov-láncot alkalmazni, viszont a 8 bit túl sok volt, mivel a 256 diszkrét értékre kódolt hangok is túl sok kombinációt eredményeztek, hogy a mátrix belátható időn belül létrejöjjön. Emiatt csökkentettem a kódolás bitszámát 7-re, 128 diszkrét értékem lett. Így néhány perc alatt létrejött a mátrix, viszont a szimulációval akadtak gondok. MIDI esetben egy generált hang sokkal hosszabb időt jelent, mint folytonos hang esetén, mivel folytonos hangnál magas a sampling rate, így ahhoz, hogy akár néhány másodpercnyi audiót is létre lehessen hozni, több tízezer értéket kéne létrehoznom az állapotátmenetek segítségével.

## Deep learning megoldások

### A tanítás elősegítése

Mielőtt elkezdeném az alkalamzott neurális hálóimat részletezni, egy fontos segítő (utility) megoldásra ki kell térnem. Ez a megoldás az LSTM alapú hálózatoknál megjelenő, “beakadás” jelenségét próbálja ellensúlyozni. A “beakadás” azt jelenti, hogy egy input zeneszelet például kizárólag E3 hangokból áll, 20 darabból, ezért a neurális hálózat a következő hangnak is egy E3-mat fog jósolni. Ezután viszont, amikor a következő 20 hangot kapja inputként, ami szintén csak E3-makból áll, mivel a legutolsó output is az volt, ezért szintén E3-mat fog outputként adni a hálózat. Ez mehetne így a végtelenségig, viszont ebből élvezhető zene nem lenne, mivel azért 1 vég nélkül ismételt hangot nem neveznék annak. Ennek elkerülése végett egy kicsit meg kell ismerni a softmax output aktivációs függvényt.



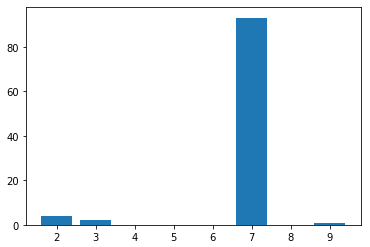
A softmax egy exponenciális alapú valószínűségi eloszlás. Amikor a neurális hálózat egy softmax aktivációval ellátott rétege outputot ad ki magából, akkor egy ilyesmi tömböt jelent ez:



Ha összeadjuk ezeket az értékeket, kijön az 1, tehát ez tényleg egy teljes eseménytér. Amit egy ilyen outputtal gyakran csinálni szoktak, az az, hogy veszik az argmax() értékét, ami jelen esetben a 7, mivel a tömböket 0-tól indexeljük, és a 8. értéke majdnem 1. Ez okozza a beakadás jelenségét, mivel fixen mindig a maximum értéket vesszük outputnak ebből. Ehelyett van erre egy megoldás, ami egy úgy nevezett hőmérsékleti tényezőt alkalmaz, aminek segítségével újraszámolja ezt az eloszlást. Ez a tényező minél nagyobb, annál véletlenszerűbb kimenete lesz, minél kisebb, annál jobban igazodik az eredetihez. Ezután az újraszámlálás után pedig a valószínűség értékekből nem az argmax() értéket vesszük, hanem a valószínűségeik alapján választunk egyet. Így az érték nagy eséllyel az argmax lesz, de nem mindig, és ez az, ami miatt nem lesz megfigyelhető a beakadás jelensége. Az előző példa kimenethez visszatérve, ha lefuttatom ezt a hőmérséklet alapú újraszámolást(nagy hőmérséklet értékkel, hogy látványos legyen), a kimenet így néz ki:



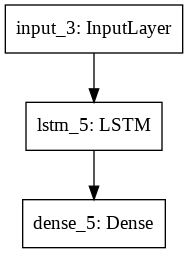
Itt is megfigyelhető az, ami az előbb, hogy az argmax() hívás értéke 7 lenne, viszont a hozzá tartozó valószínűség jelentősen csökkent. A valószínűségekből történő véletlenszerű mintavételt pedig 100-szor lefuttattam, és eredményüket egy oszlopdiagramon ábrázoltam:



Látható, hogy így is a 7-es értéket sorsolja legtöbbször a gép, viszont a 2-es, a 3-mas és a 9-es is kap esélyt. Ez pont elég arra, hogy a beakadás problémáját elkerülje a zeneszerző program. A hőmérséklet érték pedig zeneszerzésnél jelentheti azt, hogy “mennyire engedje szabadon a fantáziáját” a zeneszerző algoritmus. Így például repetitívebb ritmushangszeres részekhez elég alacsonyabb hőmérséklet értékkel foglalkozni, a vadabb szólókhoz pedig mehet a magasabb érték.

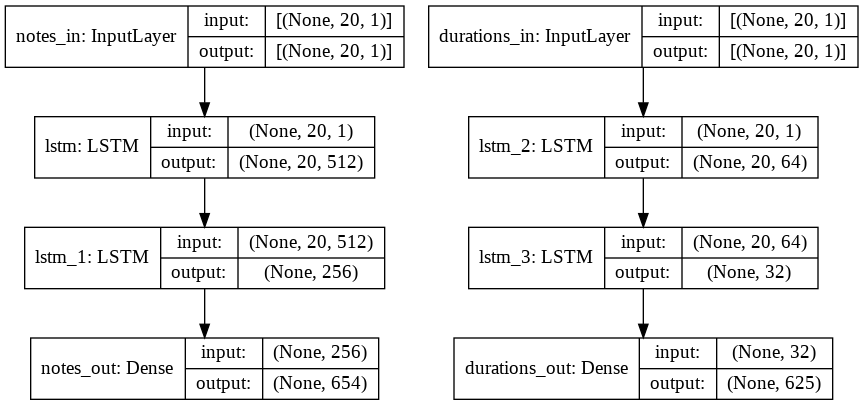
### LSTM alapú neurális hálózat

A deep learninges megoldásaimat egy egyszerű LSTM alapú neurális hálózattal kezdtem. Az architektúra így nézett ki:



Az input rétegbe belemegy egy szeletnyi számmá kódolt MIDI objektum, az LSTM réteg feldolgozza ezt a szekvenciát, majd a Dense réteg ad egy softmax kimenetet, ami a következő hangra jósolt érték. Egyszerűbb szekvenciákon, például basszusgitár MIDI sávokon ez is képes volt kellemes eredményt elérni, viszont komplexebb zenékhez nem tudott elég jól tanulni. Ezt a problémát a háló mélyítésével orvosoltam, beleraktam még egy LSTM réteget, nagyobb neuronszámmal. Így már képes volt komplexebb zenék struktúráit is megtanulni, basszusgitárzenéket már gondok nélkül tudott generálni, és már a gitársávokkal is megbírkózott. Ebben a megoldásomban egyben kellett kódolnom a hangokat a hanghosszaikkal, és a modell együtt adott a kettőre predikciót.

Egy másik LSTM megoldásom is volt, ami külön kódolt hangmagasság és hanghossz értékeken dolgozott. Két bemeneti és két kimeneti rétege volt. Az egyik oldalán inputként beletettem a hanghosszok tömbjét, másik oldalán a hangmagasságokat. Az egyik output réteg jósolta a következő hang magasságát, a másik pedig annak a hosszát. Az architektúra így nézett ki:



Ez a megoldás jónak bizonyult, az előző LSTM architektúrához hasonló eredményt tudott produkálni. Ennek a megoldásnak az az előnye viszont, hogy szét van választva a két generátor, így megadhatók neki más-más paraméterek. Így, ha mondjuk a hangok magasságainak feldolgozásához optimálisabb egy nagyobb neuronszámot választani, mint a hanghosszoknak, akkor itt megtehetem azt, az előző megoldásnál pedig nem.

Próbáltam úgy továbbfejleszteni a modellt, hogy a két, LSTM rétegek által feldolgozott szekvencia egy Merging layer segítségével összefut egy közös részbe, és onnan megy kifele a két output irányába a predikció. Ez viszont nem tudta hozni az eddigi eredményeket, rosszabbul hangzó outputokat tudtam vele generálni, és a tanulás során a loss sem ment le kellően jó értékig.

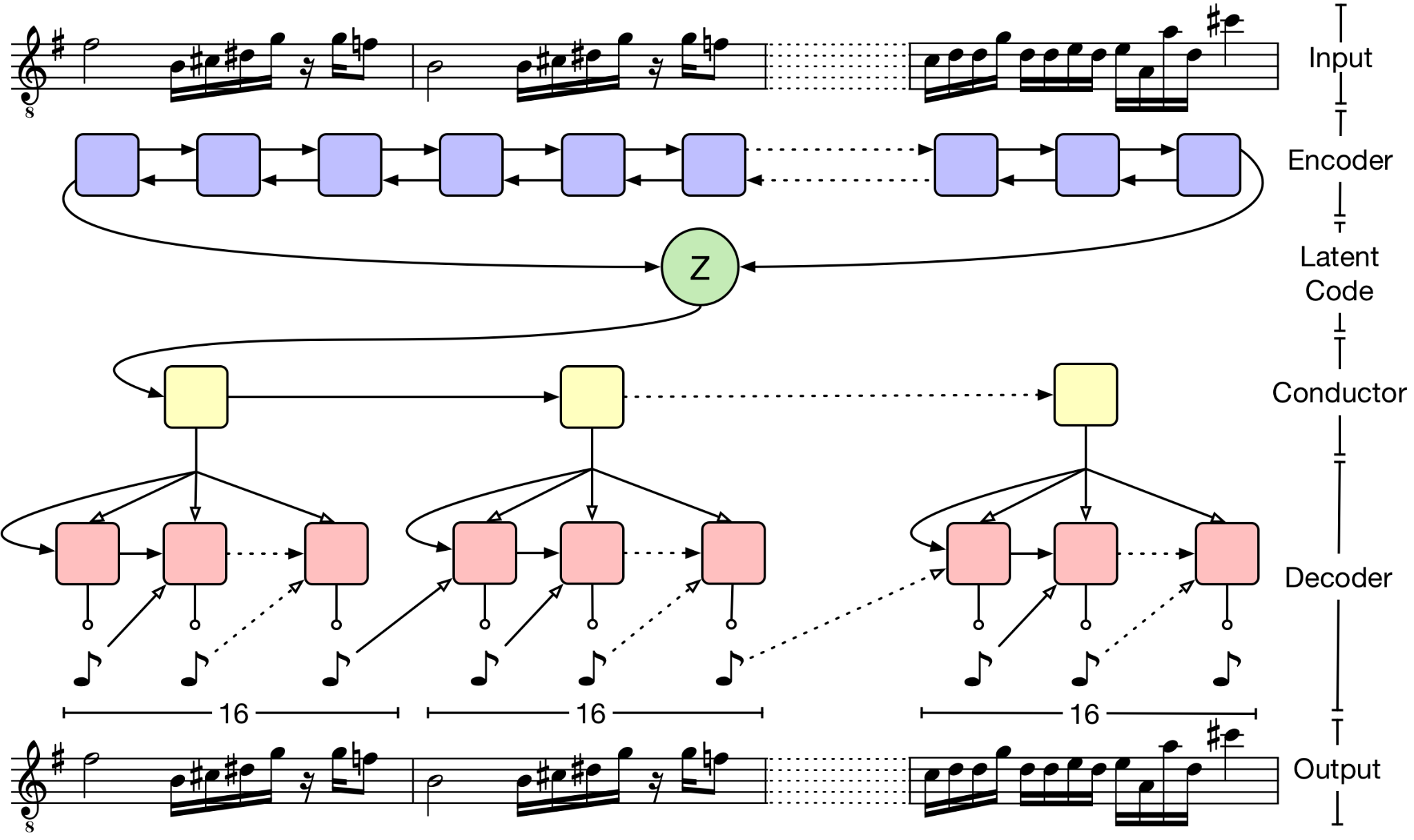
### Többhangszeres stacked LSTM

Eddig csak egyféle zenei sávon tanítottam a modellem, így külön-külön tudtam gitár, basszusgitár zenéket generálni, viszont együtt a kettőt nem. Az előző modellemből kiindulva oldottam meg ezt a problémát. Most nem csak két összekapcsolt modellt csináltam, hanem négyet. Az egyik diktálja a tempót, az predikálja a következő hangnak a hosszát, így garantálva, hogy nem esnek ki a ritmusból egymáshoz képest a “zenészek”. A másik három ága a modellnek gitár, basszusgitár és dob hangokat predikált, így szimulálva egy három tagú rockzenekar működését. Itt például nagyon jól jött, hogy szabadon választhatom meg a neuronszámokat egyes ágakban, így ki tudtam használni, hogy a gitársávokon történő tanuláshoz komplexebb, több cellából álló LSTM-ek szükségesek, a basszusgitárhoz képest. Ez is jó eredményeket produkált, kellemesen hangzó, együttműködő többhangszeres zenéket sikerült létrehoznom.

Egy problélmája volt ennek a megoldásnak, hogy a dob nem működött. A dobok egy más MIDI sávban vannak megoldva, és ugyan bájtszinten sikerült jó predikciókat létrehozni a modellemnek, azt nem tudtam megoldani, hogy a dobos MIDI sáv rendesen működjön, és ütőhangszeres hangja legyen. Ezt viszont nem tekintettem prioritásnak, ezért ezekből a generált zenékből végül kivettem a dobos MIDI sávot.

### MusicVAE

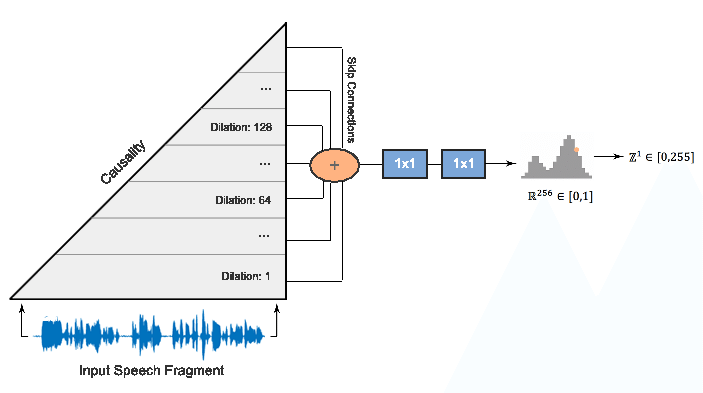
Eddig LSTM alapú, autoregresszív generatív modellekkel foglalkoztam. Ez a megoldás viszont attól teljesen eltér. Itt a generátor egy teljes szeletnyi zenét ad ki magából outputként, nem csak egy hangot. A MusicVAE egy Variational Autoencoder alapú generatív megoldás. Az architektúra így néz ki:

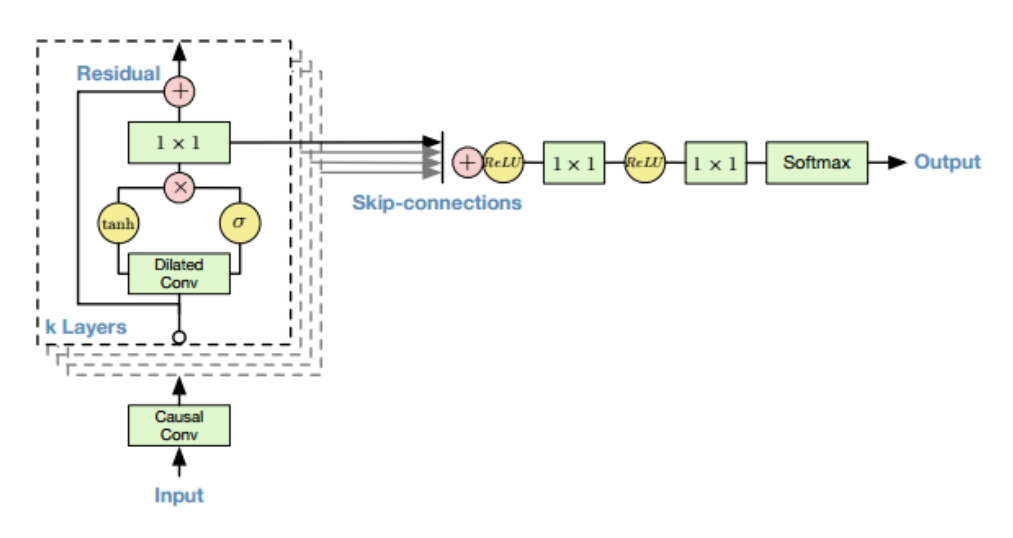


Az autoencoder encoder része Bidirectional LSTM-ekből áll, ezek tanítás során értelmezik az input zenét, és egy látens kódot hoznak belőle létre. Ezt a látens kódot dolgozza fel a decoder, ami először egy Conductornak nevezett LSTM rétegen küldi át a zajt. Ezek a rétegek egy kis feldolgozást végeznek az inputjukon, majd továbbküldik a feldolgozott inputjukat a Decoder LSTM blokkoknak, amik ezekből a szekvenciákból 1-1 output hangot állítanak elő. Ezeket az output hangokat egymás után konkatenálom, így kapom meg az outputot.

Tanítás során az történik, hogy az input és output szelet ugyanaz, az encoder által tanult zajból az eredeti szeletet próbálja visszaállítani a modell, így módosulnak eközben a decoder súlyai. Generáláskor viszont a decoder kap random zajt inputként, és a behangolt súlyai segítségével abból egy teljesen új zenei szekvenciát állít elő. Ezeken a szekvenciákon is végrehajtom a softmax kimenet hőmérséklet alapú újraskálázását, azt figyeltem meg, hogy úgy jobb eredményeket kapok.

### Wavenet





A Wavenet bemenetként egy szeletet vár a folytonos beszédjelből, vagy zenéből, és azt egy 1D konvolúciós rétegekből álló blokkon vezeti át, amik különböző eltolás értékekkel rendelkeznek, így más-más részletességgel néznek rá az bemenetre. Itt a konvolúciós rétegek reziduális összeköttetéseket tartalmaznak, azaz nem a kimenetüket, hanem a kimenetük és bemenetük összegét kötik hozzá a következő réteghez. Ez azért fontos, mert nagy modelleknél képes előjönni a vanishing gradient probléma, ami miatt a mélyebb rétegek nem tanulnak, ez az összeköttetés viszont ezt képes megoldani. Minden ilyen konvolúciós réteg kimenetét kikötik ebből a nagy blokkból egy összeadó rétegbe. Ezen a szummázott, konvolúciókkal feldolgozott hangjelen még néhány 1D konvolúciós szűrő dolgozik, ReLU aktivációkkal. Végül egy softmax aktivációval rendelkező réteg adja a kimenetet. A generált hangokat autoregresszíven a következő inputhoz hozzárakjuk, és így generálja sorban egymás után a hangokat a Wavenet.

### Wavenet

## Webalkalmazás Django frameworkkel

A deep learning modellek által generált zene könnyebb elérhetőségének céljából terveztem egy webalkalmazást, aminek segítségével meghallgathatók a létrehozott zenék. Az alkalmazást Python nyelven készítettem el, mivel a neurális hálóim elő és utófeldolgozó algoritmusait is abban írtam meg, így sok kód átemelhető volt onnan. A Python egyik legnépszerűbb webes frameworkjét, a Djangot használtam.

# Eredmények

## A Wavenet, és a folytonos hullámformák problémái

Ugyan az architektúrát sikeresen meg tudtam valósítani, viszont a tanításom már nem volt sikeres, erőforrás korlátok miatt. 12 darab, körülbelül egyenként félórás, folytonos hullámformájú dalt olvastam be a librosa csomaggal, ami olyan hatalmas adatamennyiséget eredményezett, hogy a Wavenetet a teljes adathalmazon tanítva, egy NVIDIA Tesla T4-es GPU-n 85 óra lett volna egy epoch. Ezért le kellett csökkentenem a tanító adathalmaz méretét a századára, hogy végig tudjam követni a tanítást. Sajnos nem tudott rendesen tanulni a modell, fluktuáltak az eredmények az epochok során, valamikor jobb lett a loss, valamikor rosszabb. Néhány epoch után abbahagytam, és megnéztem, mit tud generálás során a modell. A generálás is hosszú ideig tartott, 2 másodpercnyi 16kHz mintavételezésú hangot 20 perc alatt sikerült generálnia. A probléma az volt, hogy a generált hang minden értéke ugyanaz volt, nem sikerült semmi érdemlegeset tanulnia a modellnek, valószínűleg a kevés tanítási idő, és a kevésre csökkentett adatmenyniség miatt. Így a Wavenetet és a folytonos hullámformák alkalmazását sajnos nem tudtam rendesen körüljárni ebben a projektben.

## Markov-lánc értékelése

Előnyök:

* Gyorsan implementálható egy ilyen megoldás.
* Amennyiben az állapotvalószínűség mátrix nem túl nagy, a generálás ideje is nagyon gyors.
* Még nagy mátrixnál is gyorsabb tud lenni, mint egy neuronháló tanulási ideje.
* Egyszerűbb zenei struktúrák, például basszugitárjáték esetén néha meglepően jól tud teljesíteni.
* Nem jelentkezik nála az LSTM-alapú neuronhálókra jellemző "beakadás".

Hátrányok:

* Folytonos zenei hullámforma generálására nem alkalmas.
* Hosszútávú koherencia egyáltalán nincs a generált zenéiben, mivel csak 1 hangtól függ a következő.
* Komplexebb struktúráknál nagyon randomnak érződik a kimenet.
* Mivel a hanghossz generálása is random történik, a ritmikái teljesen rosszak.

# Összefoglalás

## Továbbfejlesztési lehetőségek

### Markov-lánc továbbfejlesztése

Beletekinteni az mchmm Python csomag implementációjába, mivel nem egy nagyon híres csomag (a fejlesztés idejében 9000 körüli letöltésszámmal rendelkezett), lehet nincs jól optimalizálva nagy állapottérhez. Ezt esetleg lehetne fejleszteni, vagy lehetne teljesen saját implementációt készíteni a Markov-lánc alapú zeneszerzésre, ezek viszont csak a lassúságot oldanák meg, a többi problémát nem.

A hosszútávú koherencia problémán esetleg segítene egy másodrendű Markov-lánc, ami két hangból hozná létre a következőt.

Ehhez nem találtam létező Python csomagot, így magamnak kéne implementálni, viszont nem érzem úgy, hogy megérné ebbe az irányba menni a fejlesztésben. Lassabb lenne, mint az elsőrendű megoldás, mivel a mátrix is nagyobb lenne. A különböző egymás utáni hangkettesek számától függ ez a szám, viszont ha eddig N hanggal -es volt a mátrix, akkor felülről becsülhető -nes mátrixmérettel, ami akkor jönnne elő, ha minden hang után mindegyik előfordulna a feldolgozott MIDI fileokban.

Ezért inkább a deep learning irányába mentem tovább.

# Irodalomjegyzék

|  |  |
| --- | --- |
| [1] | P. Koopman, „How to Write an Abstract,” október 1997. [Online]. Available: https://users.ece.cmu.edu/~koopman/essays/abstract.html. [Hozzáférés dátuma: 20 október 2015]. |
| [2] | W3C, „HTML, The Web’s Core Language,” [Online]. Available: http://www.w3.org/html/. [Hozzáférés dátuma: 20 október 2015]. |
| [3] | K. Nahtkasztlija, „Az idegen szavak toldalékolása,” június 2009. [Online]. Available: http://www.pcguru.hu/blog/kredenc/az-idegen-szavak-toldalekolasa/5062. |

Függelék

Források:

wiki neural network kép

<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

LSTM paper <https://www.bioinf.jku.at/publications/older/2604.pdf>

Autoencoder: <https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798>