Kiss Andor

Deep Learning alapú zeneszerzés

Önálló laboratórium beszámoló

Konzulens
Gulyás Gábor György

1. Bevezetés

A zeneszerzés az őskor óta jelen van az emberiség életében. Ahogy múlik az idő, úgy jelennek meg ott is egyre modernebb technológiák. Az 1980-as években ahogy berobban a popzenébe a szintetizátor, ott kezdődött el a “gépzene” kifejezés használata. Viszont ott még mindig arról volt szó, hogy emberek által létrehozott zenéket szabtak testre különféle hardveres és szoftveres effektekkel.

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 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 a szoftver adja. Egy gépi tanuló algoritmus adja azt, majd ezután különféle szoftverek és emberi beavatkozás segítségével módosítható a zene, például egy adott effektet tesznek rá.

Önálló laboratórium feladatomként azt tűztem ki magamnak, hogy körüljárjam, hogy milyen lehetőségek vannak ebben a témában, megnézzem, hogy milyen korábbi megoldások voltak, hogyan alkalmazhatók ezek különféle zenei formátumoknál. Emellett azt is egy érdekes feladatnak tartottam, hogy szinte minden zeneszerzéssel kapcsolatos Deep Learning projekt klasszikus zenével foglalkozik, viszont én amellett, hogy klasszikus zenei adathalmazokat is megtekintek, az egyik személyes kedvenc előadóm, az Iron Maiden MIDI sávjait is megnézem, hogy azokat hogyan tudják tanulni a különféle algoritmusok. Amellett, hogy egyik személyes kedvencem, azért is választottam azt az előadót, mert jellegzetes, kicsit repetitív stílusa van, ami szerintem egy neurális hálózatnak egy megérthető, tanulható dolog.

2. Korábbi megoldások

Wavenet [1]: 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. Bővebben itt fejtem ki a működését.

SampleRNN [2]: 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 [3]: Ez a megoldás újra a Wavenethez nyúl vissza, viszont azt egy még nagyobb, autoencoder struktúrára cseréli le.

Performance RNN [4]: Ez a megoldás MIDI-ken dolgozik folytonos zene helyett. Lényegében csak egy stacked LSTM hálózat, amit a MAESTRO dataseten tanítottak.

Magenta MusicVAE [5]: Ez egy MIDI-n tanuló Variational Autoencoder alapú megoldás, amit én is megvalósítottam a paper alapján. Bővebben itt írok a megvalósításról.

MuseGAN [6]: 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 [7]: 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 [8]: 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 [9]: Ez a modell egy konvolúciós GAN segítségével szintetizál folytonos hullámformájú zenét.

MuseNet [10]: 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 [11]: Ez talán a mai legmodernebb, legjobb teljesítményre képes zenegenerátor a deep learning világában. Hatalmas modell, kombinálja a Wavenet 1D konvolúcióit a transformerek attention mechanizmusával.

3. Adatok beszerzése, fájlformátumok

Kétféle adattal dolgoztam a projektem során, amik más-más reprezentációi a zenének. Dolgozok folytonos hullámformájú zenékkel, amiket például mp3 formátumban szerzek be. 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. Az adatbeszerzéshez két megoldást használtam, 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 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:

Note	Octave										
	-1	0	1	2	3	4	5	6	7	8	9
C	0	12	24	36	48	**60**	72	84	96	108	120
C#	1	13	25	37	49	**61**	73	85	97	109	121
D	2	14	26	38	50	**62**	74	86	98	110	122
D#	3	15	27	39	51	**63**	75	87	99	111	123
E	4	16	28	40	52	**64**	76	88	100	112	124
F	5	17	29	41	53	**65**	77	89	101	113	125
F#	6	18	30	42	54	**66**	78	90	102	114	126
G	7	19	31	43	55	**67**	79	91	103	115	127
G#	8	20	32	44	56	**68**	80	92	104	116	
A	9	21	33	45	57	**69**	81	93	105	117	
A#	10	22	34	46	58	**70**	82	94	106	118	
B	11	23	35	47	59	**71**	83	95	107	119	

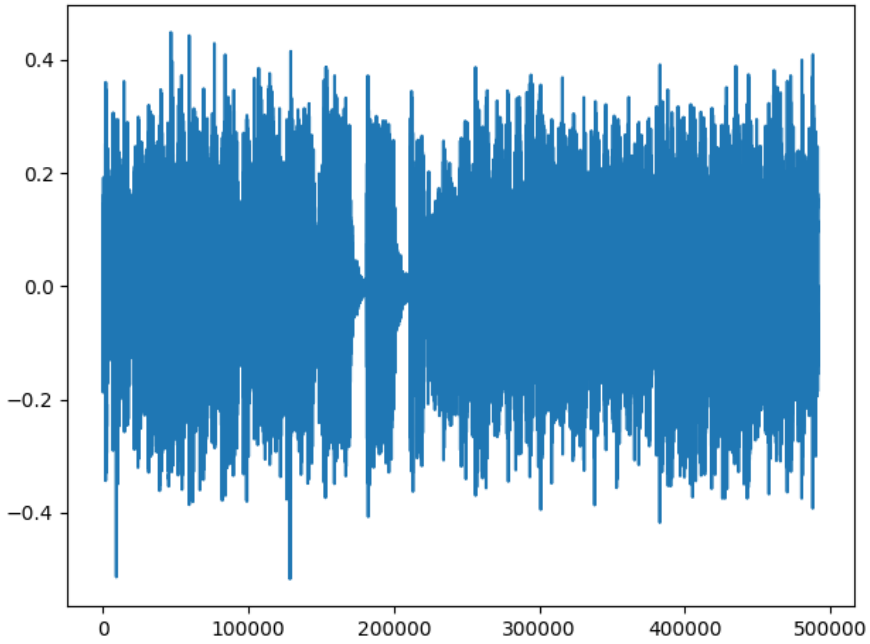
Emellett minden hanghoz a MIDI tárolja azt is, hogy az adott hang mennyi ideig szól, és azt is, hogy milyen erősséggel ütötték meg a hangszert.

A MAESTRO adathalmazon kívül egy másik MIDI beszerzési módot is találtam. Az Ultimate Guitar oldaláról előfizetők legálisan tölthetnek le Guitar Pro tabokat. A gitártabok egyszerűsített kották, amiket gitárjátékosok szoktak használni, mivel a gitár húrjain mutatja, hogy miket kell lefogni, amik 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.

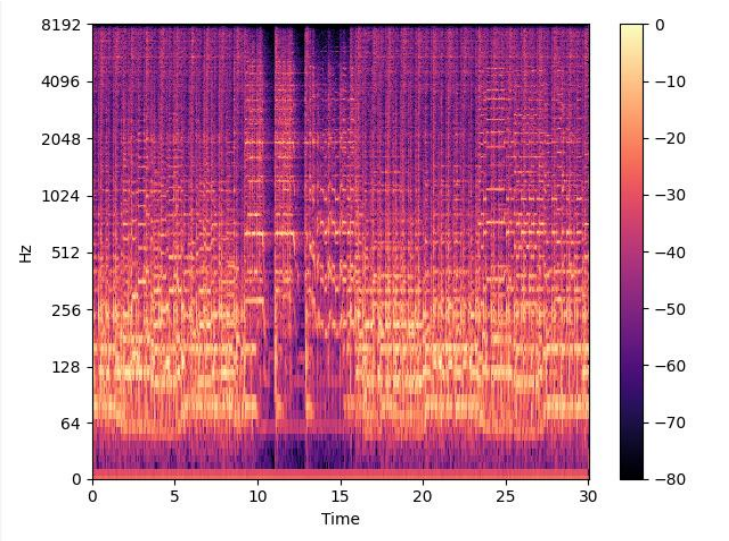
4. Adatok feldolgozása

4.1. Folytonos 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 kapom.



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. 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.

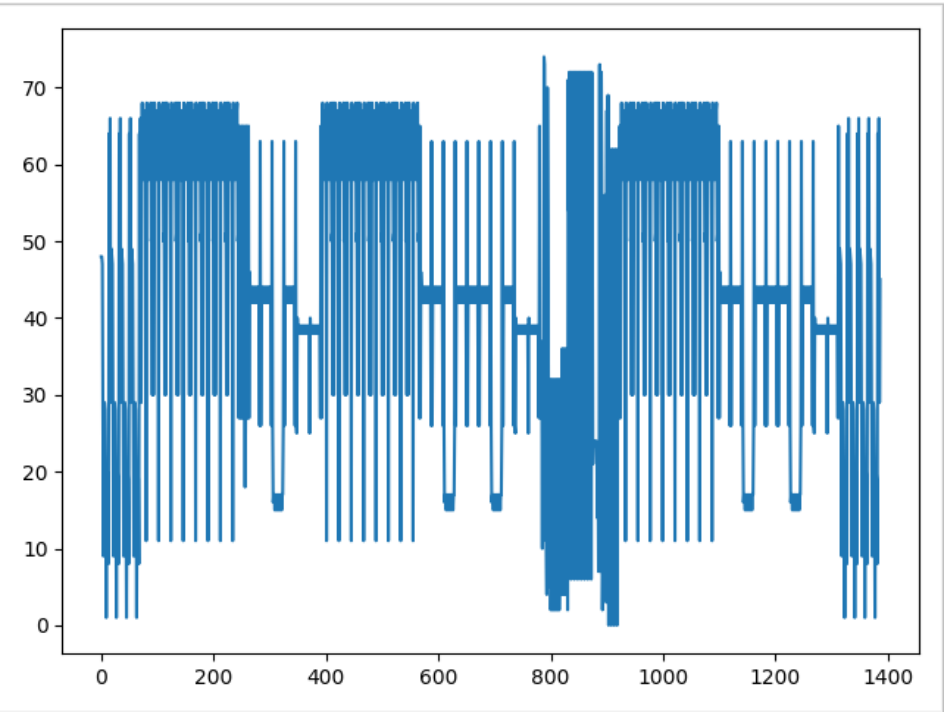


Végül ezeket az folytonos értékeket a [-1; 1] intervallumba normalizáltam, mivel a preprocessing pipelineom következő algoritmusa ezen a tartományon várja az értékeket. Ezaz algoritmus pedig a 8 bites μ-law kódolás. Ennek segítségével a végtelen lehetséges valós értéket véges számú értékké alakítom (a 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.

4.2. MIDI adatok

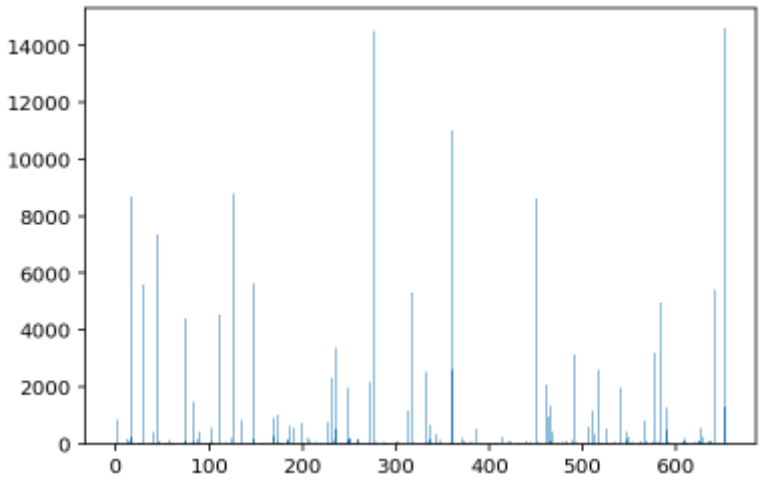
A MIDI reprezentációjú zenéket a music21 Python csomaggal olvastam be. Különbontottam a különböző hangszerek MIDI sávjait, és úgy mentem végig rajtuk, beolvasva az elemeiket. A beolvasás során Pythonos objektumokat kaptam, amik a music21 csomagban definiált osztályok. Ilyenek például a Chord, Note, Rest, 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.



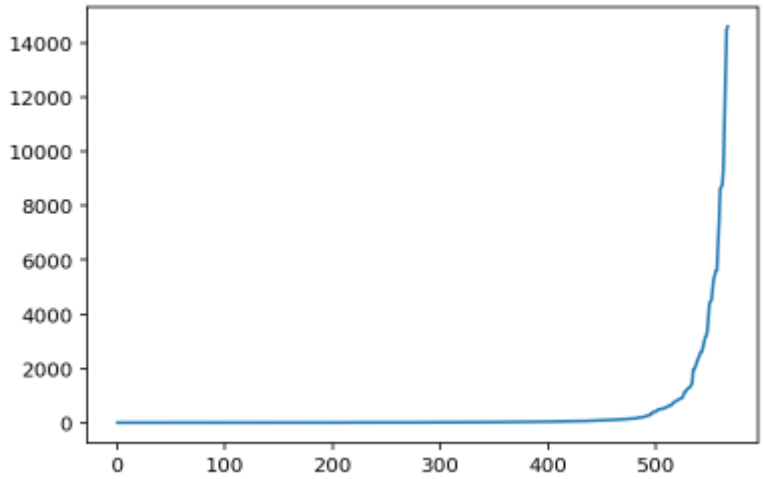
Így nézett ki egy beolvasott, és számokká alakított MIDI sáv. Ezután több különböző számmá kódolási megoldást is számításba vettem. Ezek között vannak olyan megoldások, amik azért vannak, hogy könnyítsenek a gépi tanuló modell feladatán, csökkentve a kimeneti osztályok számát.

- 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 azok tekinthetők irrelevánsnak, kivehetők a feldolgozott adatok halmazából, így csökkentve a kimeneti lehetőségek számát.

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

5. Baseline megoldás

5.1. Elsőrendű Markov-lánc

Mielőtt nekiesnék a neurális hálózatok alkotásának, és a deep learningnek, megnéztem mire képes egy egyszerűbb, valószínűség alapú gépi tanuló modell. Legyen S egy állapottér, hogy:

$$X_1, X_2, X_3, \ldots, X_n, X_{n+1} \epsilon S$$

$$P(X_{n+1} = x | X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n) = P(X_{n+1} = x | X_n = x_n)$$

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.

Ezt zenei alkalmazásra úgy tudnám lefordítani, hogy minden hang egy állapot, és minden i-edik hang csak az őt megelőző, i-1-edik hangtól függ. Ezekhez a modellekhez az mchmm Python csomagot használtam, amit kipróbáltam MIDI-re és folytonos zenére is.

5.2. Markov-lánc a MIDI adatokon

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. Ezalapján a 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.

5.3. Markov-lánc a folytonos zenéken

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.

5.4. A 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.

5.5. Ötletek a Markov-láncos megoldás fejlesztésére

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 $N \times N$-es volt a mátrix, akkor felülről becsülhető $N^2 \times N$-nes mátrixmérettel, ami akkor jönnne elő, ha minden hang után mindegyik előfordulna a feldolgozott MIDI fileokban.

Ezért inkább a deep learning irányába mentem tovább.

6. Deep Learning megoldásaim MIDI adatokon

6.1. LSTM alapú megoldásaim

6.1.1. Tanítást segítő utilityk

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.

Az outputhoz is definiáltam egy ilyen utilityt, ami 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.

$$softmax(x)_i = \frac{exp(x_i)}{\sum_j exp(x_j))}$$

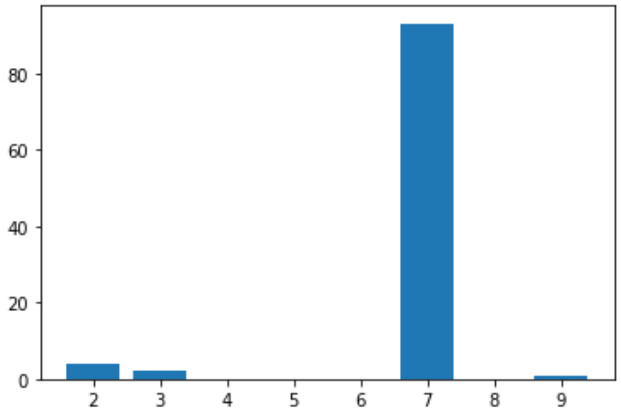
A softmax egy exponenciális alapú valószínűségi eloszlás. Amikor a neurális hálózat egy softmax aktivációval ellátott rétege outputot ad ki magából, akkor egy ilyesmi tömböt jelent ez:

```
[2.82518557e-30 0.00000000e+00 3.41315882e-07 1.00777954e-11
 1.34826385e-20 1.40432097e-22 0.00000000e+00 9.99999642e-01
 0.00000000e+00 1.88994518e-13]
```

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 randomabb 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:

```
[1.16095838e-06 0.00000000e+00 4.79997034e-02 5.96046801e-03
 1.00129543e-04 4.01883828e-05 0.00000000e+00 9.43207453e-01
 0.00000000e+00 2.69089713e-03]
```

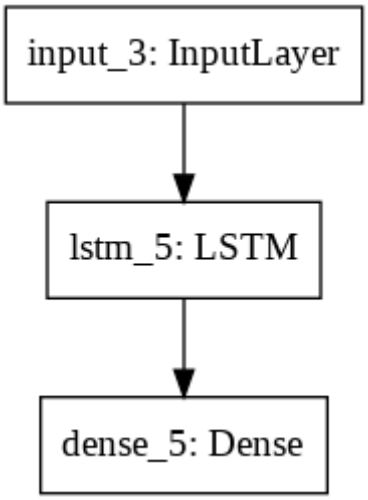
Itt is megfigyelhető az, ami az előbb, hogy az argmax() hívás értéke 7 lenne, viszont a hozzá tartozó valószínűség jelentősen csökkent. A valószínűségekből történő véletlenszerű mintavételt pedig 100-szor lefuttattam, és eredményüket egy oszlopdiagramon ábrázoltam:



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.

6.1.2. Egyrétegű LSTM alapú neurális hálózat

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



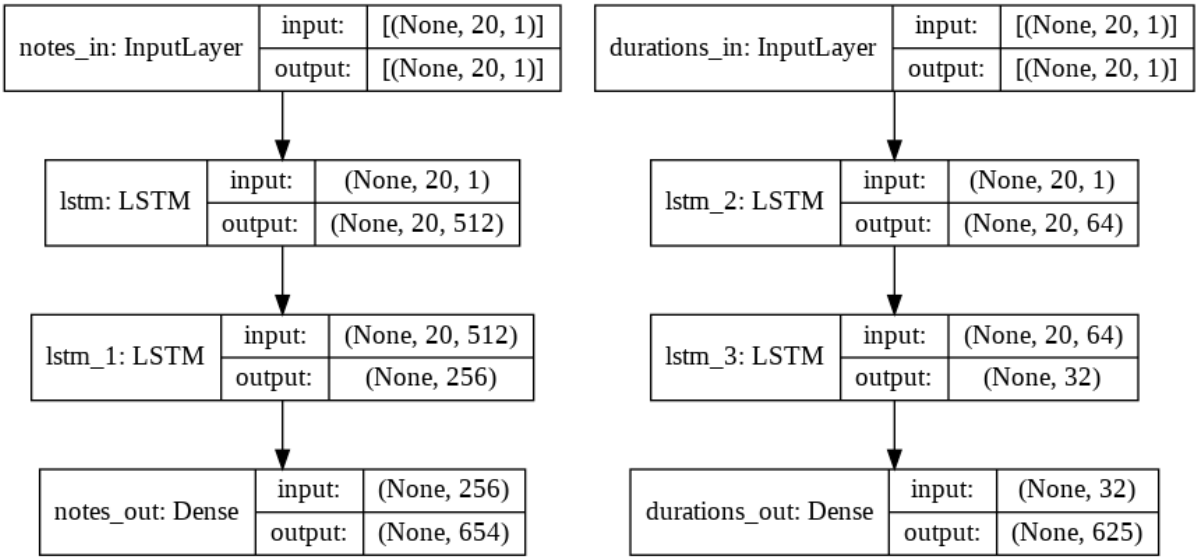
Az input rétegbe belemegy egy szeletnyi számmá kódolt MIDI objektum, az LSTM réteg feldolgozza ezt a szekvenciát, majd a Dense réteg ad egy softmax outputot, ami a következő hangra jósolt érték. Egyszerűbb szekvenciákon, például basszusgitár MIDI sávokon ez is képes volt kellemes eredményt elérni, viszont komplexebb zenékhez nem tudott elég jól tanulni.

6.1.3. Többrétegű, stacked LSTM

A stacked LSTM lényegében ugyanaz, mint az előző megoldásom, csak több LSTM réteg épül egymásra, így a rétegek egymás feldolgozott szekvenciáin is tudnak tanulni. Ennek már sikerült komplexebb szekvenciákat, például gitársávokat is tanulni.

6.1.4. Kétoldalú stacked LSTM

Ez egy olyan megoldásom 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 predikálta a következő hang magasságát, a másik pedig annak a hosszát. Az architektúra így nézett ki:



Ez a megoldás jónak bizonyult, az előző, többrétegű stacked 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 paraméterekl. 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.

6.1.5. Többhangszeres stacked LSTM

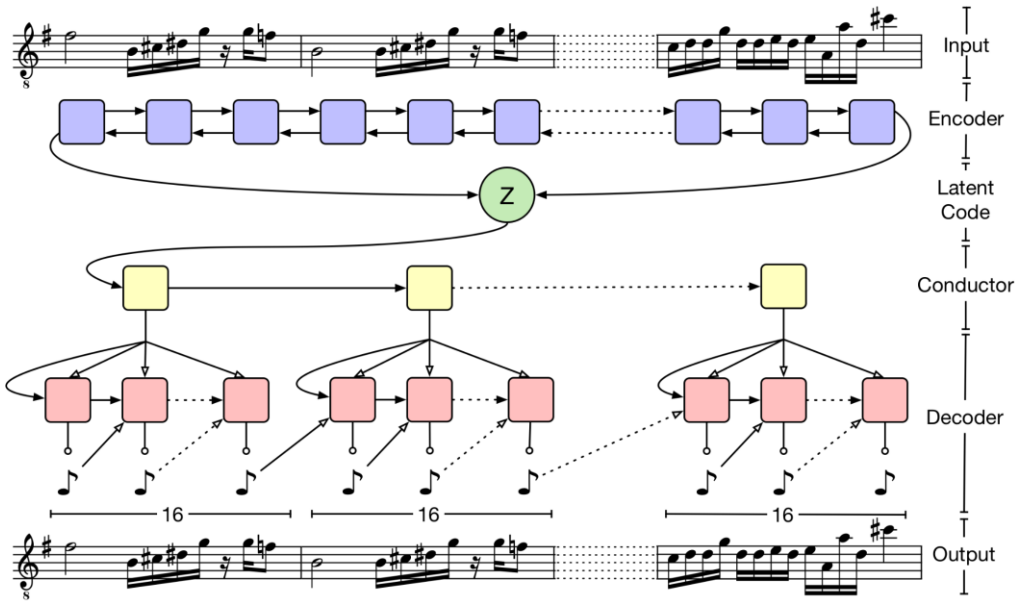
Eddig csak egyféle zenei sávon tanítottam a modellemet. Így külön-külön tudtam gitár, basszusgitár zenéket generálni, viszont együtt a kettőt nem. Az előző modellemből kiindulva oldottam meg ezt a problémát. Most nem csak két összekapcsolt modellt csináltam, hanem négyet. Az egyik diktálja a tempót, 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.

6.1.6. 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:



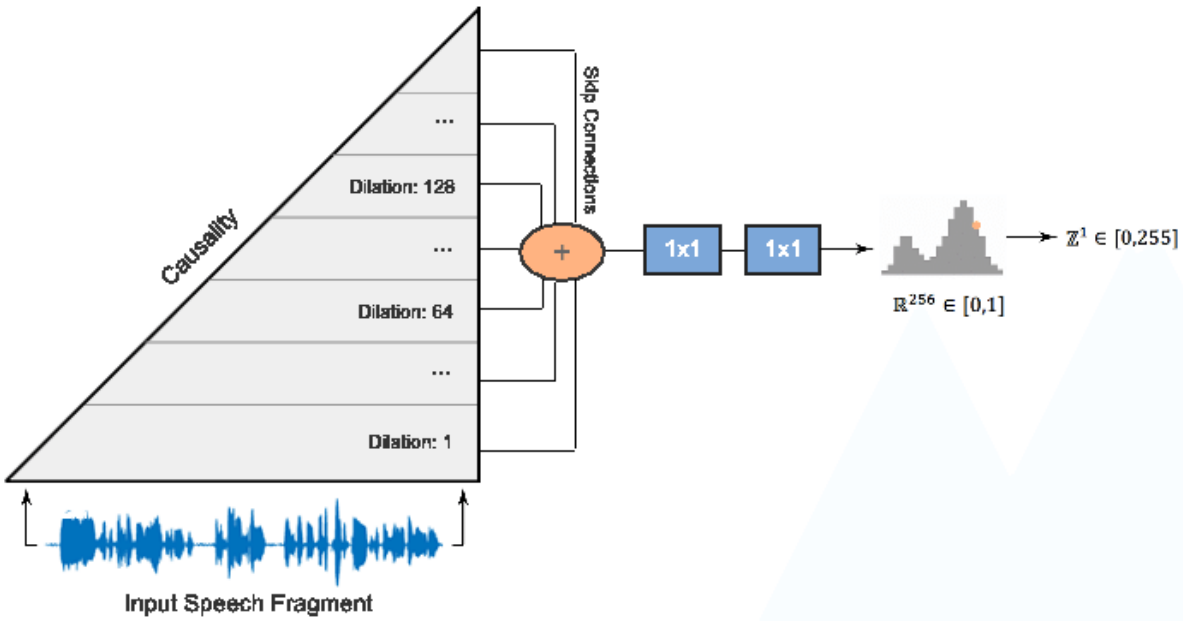
Az autoencoder encoder része Bidirectional LSTM-ekből áll, ezek tanítás során értelmezik az input zenét, és egy látens kódot hoznak belőle létre. Ezt a látens kódot dolgozza fel a decoder, ami először egy Conductornak nevezett LSTM rétegen küldi át a zajt. Ezek a rétegek egy kis feldolgozást végeznek 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.

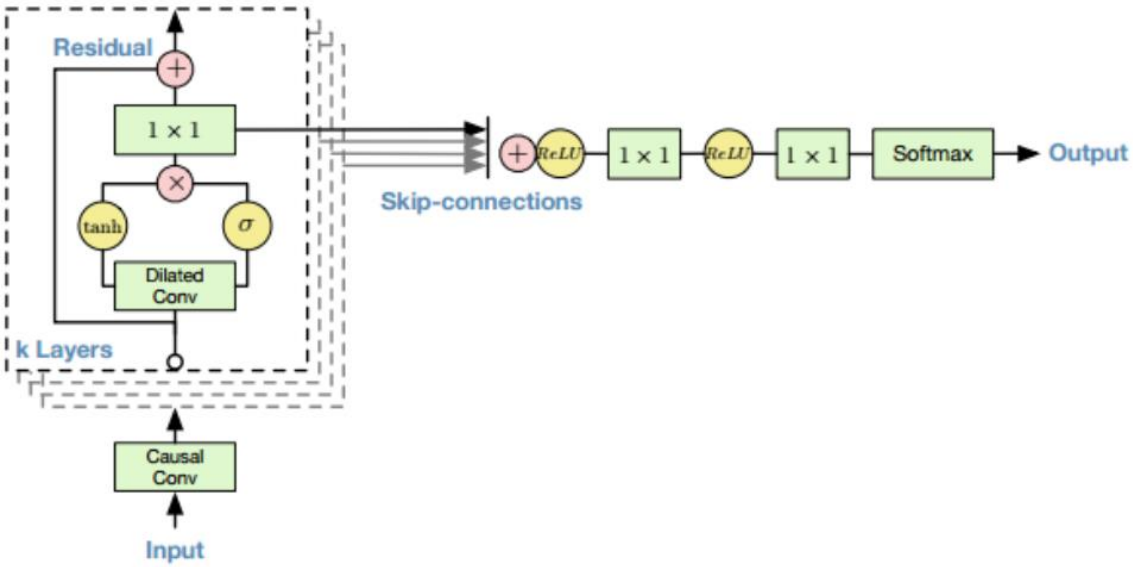
Tanítás során az történik, hogy az input és output szelet ugyanaz, az encoder által tanult zajból az eredeti szeletet próbálja visszaállítani a modell, így módosulnak eközben a decoder súlyai. Generáláskor viszont a decoder kap random zajt inputként, és a behangolt súlyai segítségével abból egy teljesen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

7. Deep Learning megoldásaim folytonos hullámformájú adatokon

7.1. Wavenet

7.1.1. A wavenet architektúra





A Wavenet bemenetként egy szeletet vár a folytonos beszédjelből, vagy zenéből, és azt egy 1D konvolúciós rétegekből álló blokkon vezeti át, amik különböző dilation értékekkel rendelkeznek, így más-más részletességgel néznek rá az inputukra. Itt a konvolúciós rétegek reziduális összeköttetéseket tartalmaznak, azaz nem a kiementüket, 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 Dense réteg adja a kimenetet. A generált hangokat autoregresszíven a következő inputhoz hozzárakjuk, és így generálja sorban egymás után a hangokat a Wavenet.

7.2. A Wavenet tanítás problémái

Az architektúrát sikeresen meg tudtam valósítani, viszont a tanításom már nem volt sikeres, erőforrás limitációk miatt. 12 darab, körülbelül egyenként fél órás, folytonos hullámformájú dalt parseoltam 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 mintavételezésű 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 félre kellett tennem, remélem később egy erőforrásgazdagabb felhőplatformon sikerül végigvinnem egy tanítást.

8. Webes bemutató oldal

Projektem végén, amikor már a deep learning architektúrákkal a kitűzött célomnak megfelelően eleget foglalkoztam, készítettem még egy bemutató weboldalt, ahol egyszerűen meg lehet hallgatni a generált zenéimet. A weboldal csak egy statikus oldal, ahol általam kiválasztott néhány, előre generált zene hallgatható meg, nem lehet dinamikusan, futásidőben generáltatni a deep learning modelleimmel zenét, és adatbázisból sem tölt be nagy mennyiségű adatot. A weblap viszont adaptív, ezen működés eléréséhez bootstrapet használtam. Böngészőben történő megnyitásakor az adatokat tartalmazó HyperText Markup Language (html) file letöltődése után betöltődnek a bootstrap stíluselemei, és a weblaphoz csatolt zenei erőforrás fileok, amik ezután meghallgathatók.

9. Jövőbeli tervek

A téma nagyon élvezetes volt számomra, ezért mindenképp szeretném szakdolgozatként folytatni. A Google Colab korlátozott erőforrásai miatt szeretnék áttérni valami erősebb cloudra, például a Google Cloud Platformra. Az így megszerzett, megnövekedett erőforrásmennyiséggel remélhetőleg képes leszek folytonos hullámformájú zenéket is generálni. Emellett mindenképp szeretnék kipróbálni új architektúrákat is, például a MIDI generáláshoz a modern, attention alapúakat. Végül szeretnék egy dinamikus weboldalt is csinálni, ahol vagy egy adatbázisból lesz nagyon sok zene betölthető, vagy futási időben lehet majd a deep learning modellem segítségével zenéket generáltatni a felhasználónak.

Irodalomjegyzék

[1] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior és K. Kavukcuoglu, „WAVENET: A GENERATIVE MODEL FOR RAW AUDIO," 2016.

[2] S. Mehri, K. Kumar, I. Gulrajani, R. Kumar, S. Jain, J. Sotelo, A. Courville és Y. Bengio, „SAMPLERNN: AN UNCONDITIONAL END-TO-END NEURAL AUDIO GENERATION MODEL," 2017.

[3] J. Engel, C. Resnick, A. Roberts, S. Dieleman, D. Eck, K. Simonyan és M. Norouzi, „Neural Audio Synthesis of Musical Notes with WaveNet Autoencoders," 2017.

[4] S. Oore, I. Simon, S. Dieleman és D. Eck, „Learning to Create Piano Performances," 2017.

[5] A. Roberts, J. Engel, C. Raffel, C. Hawthorne és D. Eck, „A Hierarchical Latent Vector Model for Learning Long-Term Structure in Music," 2018.

[6] H.-W. Dong, W.-Y. Hsiao, L.-C. Yang és Y.-H. Yang, „MuseGAN: Multi-track Sequential Generative Adversarial Networks for Symbolic Music Generation and Accompaniment," 2017.

[7] C. Hawthorne, A. Stasyuk, A. Roberts, I. Simon, C.-Z. A. Huang, S. Dieleman, E. Elsen, J. Engel és D. Eck, „ENABLING FACTORIZED PIANO MUSIC MODELING AND GENERATION WITH THE MAESTRO DATASET," 2019.

[8] C.-Z. A. Huang, A. Vaswani, J. Uszkoreit, N. Shazeer, I. Simon, C. Hawthorne, A. M. Dai, M. D. Hoffman, M. Dinculescu és D. Eck, „MUSIC TRANSFORMER: GENERATING MUSIC WITH LONG-TERM STRUCTURE," 2018.

[9] J. Engel, K. K. Agrawal, S. Chen, I. Gulrajani, C. Donahue és A. Roberts, „GANSYNTH: ADVERSARIAL NEURAL AUDIO SYNTHESIS," 2019.

[10] C. M. Payne, „MuseNet," 2019.

[11] P. Dhariwal, J. Heewoo, C. Payne, J. W. Kim, A. Radford és I. Sutskever, „Jukebox: A Generative Model for Music," 2020.