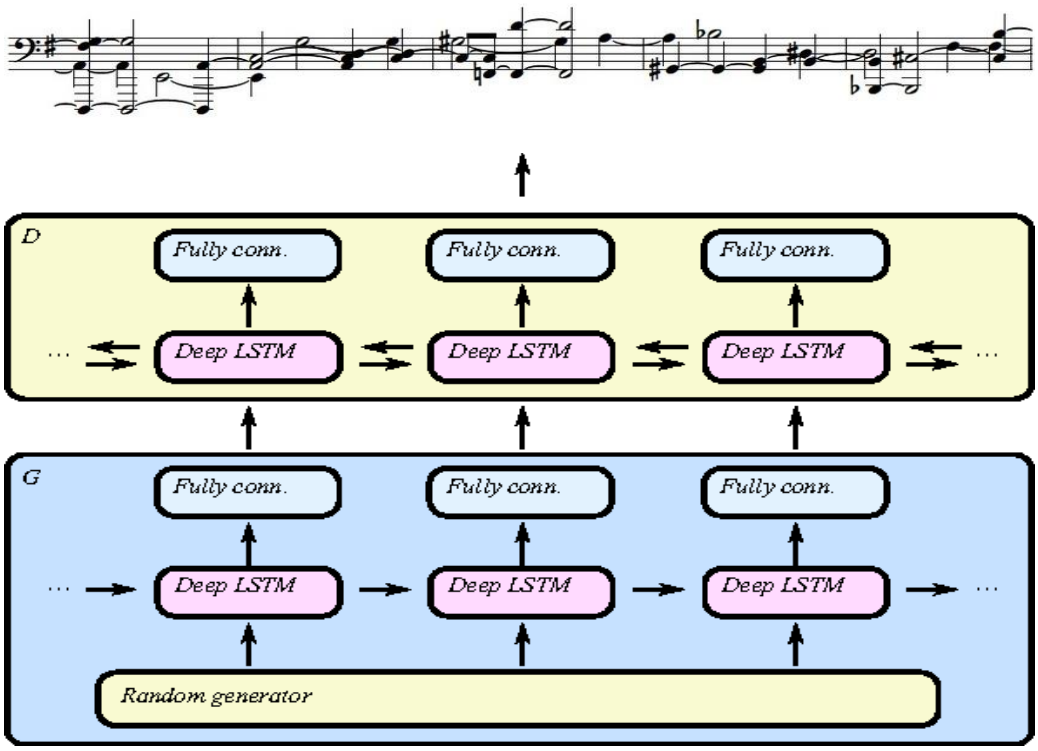
دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

پروژه ی تولید موسیقی به وسیله شبکه های عصبی (C-RNN-GAN)



پروژه ی نهایی درس هوش محاسباتی

پاییز ۱۳۹۹

افراد گروه : آقایان سیدمحمدصالح میرزاطباطبایی، سلمان عامی مطلق، آریا اسپهبدی

زیر نظر استاد دکتر عبداللّهی (دانشکده ی مهندسی برق دانشگاه صنعتی امیرکبیر)


بسم الله الرحمن الرحیم

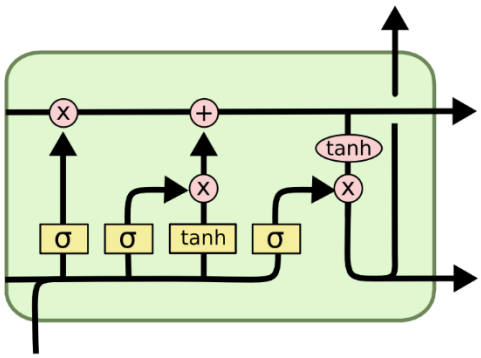
مقدمه

مفهوم موسیقی از قرن های دور برای بشر معنا پیدا کرده است و تا بدین روز همگام با رشد ذهنی انسان در حال پیشرفت بوده است. حال در قرن ۲۱ و با ظهور چشمگیر هوش مصنوعی، درک، یادگیری و ساخت موسیقی، همچون بسیاری از قابلیت های ذهنیِ دیگرِ انسان، توسط شبکه های هوشمند مصنوعی نیز قابل اجرا است.

هدف ما از انجام این پروژه دستیابی به یک مدل شبکه ی عصبی مصنوعی با قابلیت تولید موسیقی می باشد. به وضوح یک شبکه ی عصبی ابتدا نیازمند آموزش دیدن است تا بتواند به کمک آموزه هایش اهداف مورد نظر ما را دنبال کند. نتیجتاً و همانطور که در صورت پروژه نیز تعریف شده است، ما باید داده های آماده ی موسیقیِ را در اختیار این شبکه بگذاریم تا ساختارشان را یاد گرفته و سپس بتواند به تنهایی نوعی از آن را برای ما تولید کند.

راهکارهای موجود

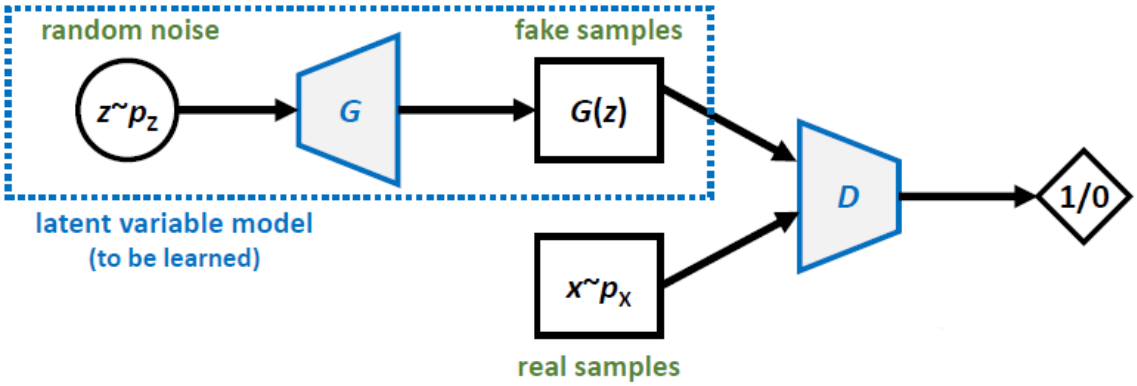
از آنجاییکه شبکه های عصبی مصنوعی انواع مختلفی دارند که هر کدام دارای ویژگی های منحصر به فرد و مناسب برای کاربرد های مخصوصی هستند، تولید موسیقی نیز تنها توسط انواع معدودی از این شبکه ها به درستی قابل اجرا می باشد. از جمله دو نوع مهم این شبکه ها، شبکه های LSTM و GAN می باشند. شبکه های GAN یا مولدِ تخاصمی که نوظهور می باشند اولین بار در سال ۲۰۱۴ توسط Ian J. Goodfellow و طی یک مقاله معرفی شدند. اگرچه شبکه های GAN تا بدین روز پیشرفت های چشم گیری داشته اند و در ساختار های مختلف و پیچیده ای ارایه شده اند، اما معرف آن در سال ۲۰۱۴ به ساده ترین شکل و تنها با دو شبکه ی MLP این ساختار را پیشنهاد داد. این شبکه ها در حال حاضر به شدت مورد توجه محققین این زمینه قرار داشته و روز به روز نیز در حال پیشرفت هستند. در ادامه به طور مفصل به ساختار این شبکه ها و شیوه ی عملکردشان می پردازیم. شبکه های LSTM قدمت بیشتری نسبت به شبکه های GAN دارند و جزو شبکه های RNN می باشند. این شبکه ها به تنهایی می توانند برای تولید موسیقی بکار روند.



ساختار شبکه LSTM

شبکه های GAN (Generative Adversarial Networks)

از آنجاییکه ما برای این پروژه از یکی از انواع شبکه های GAN استفاده برده ایم، در ابتدا نیاز است که ساختار کلی این شبکه ها و شیوه ی عملکرد آن ها را بیان کنیم. همانطور که از اسم اصلی این شبکه ها واضح است، در آنها قرار است که چیزی به صورت تخاصمی یا رقابتی تولید گردد. یک شبکه ی GAN در واقع از دو شبکه ی متفاوت تشکیل شده است؛ یک شبکه به نام discriminator و دیگری به نام generator. در واقع generator قصد دارد که چیزی شبیه داده های واقعی ما تولید کند به گونه ای که شبکه ی discriminator که خود وظیفه ی تشخیص داده های درست (واقعی) و غلط (غیر واقعی) را دارد، آن را یک داده ی واقعی بشناسد. از آنجاییکه discriminator در تلاش است که گول نخورد و همواره داده های غیر واقعی (داده هایی خارج از داده های ما) را از واقعی متمایز کند و برعکس generator هم می خواهد آن را با داده هایش گول بزند، این رابطه ی رقابتی شکل میگیرد که خود باعث بهتر شدن و ارتقای دو شبکه میشود.



بخش generator با گرفتن یکسری عدد تصادفی به صورت نویز به ما خروجی می دهد. ساختار داخلی شبکه های discriminator و generator می تواند به مدل های مختلفی باشد. به طور مثال در آنها از LSTM،MLP ، CNN و ... استفاده کرد. آموزش این شبکه ها بدین صورت است که ابتدا داده ی خود (real samples) و خروجی شبکه ی generator (fake samples) را به ترتیب با تارگت ۱ و ۰ به شبکه ی discriminator میدهیم. تابع هزینه ی این دو شبکه نیز به صورت زیر است:

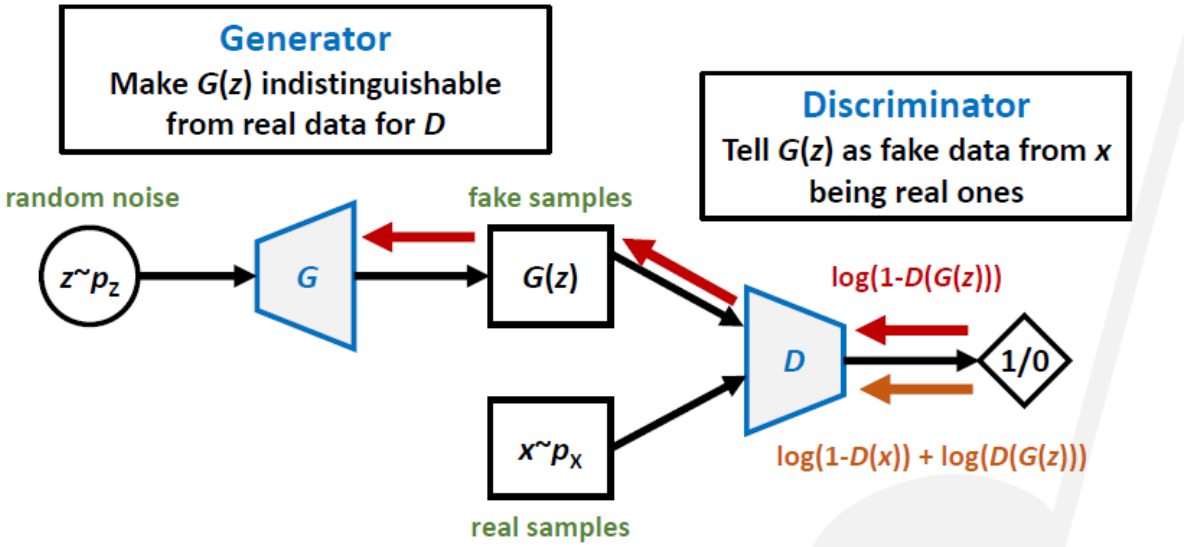
- Discriminator Loss:

$$l^{(D)}(x, z) = -\log\big(D(x)\big) - \log(1 - D(G(z))$$

- Generator Loss:

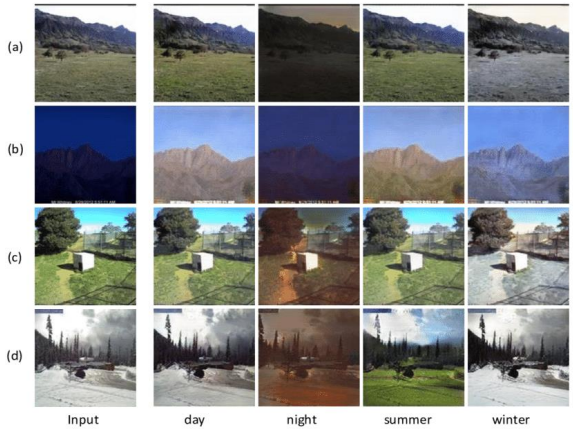
$$l^{(G)}(z) = \log(1 - D(G(z))$$

بدین ترتیب شبکه آموزش داده می شود. نکته ی حایز اهمیت در مورد این شبکه ها این است که بر خلاف خیلی از شبکه های دیگر، اگر از یک حدی بیشتر آموزش دیده شوند، مقدار تابع هزینه ی generator دیگر شروع به افزایش پیدا می کند. علت این امر اینست که با آموزش بیش از حد، شبکه ی discriminator دچار overfitting شده و کوچکترین تغییر در داده های واقعی را تشخیص می دهد که باعث می شود generator دیگر نتواند آن را گول بزند. در تصویر زیر هم رویه ی کلی آموزش این دو شبکه را مشاهده می کنید.

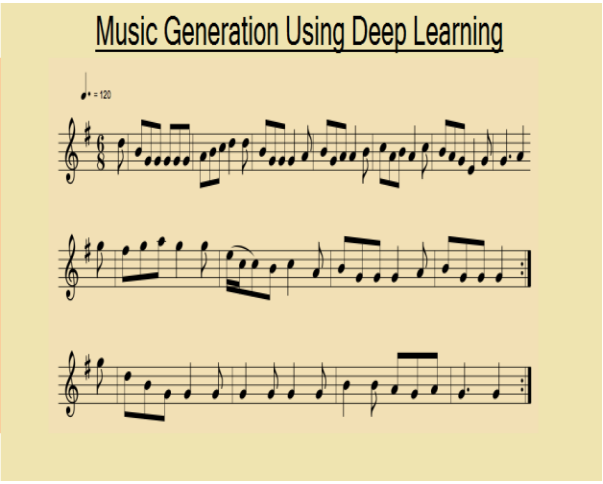


امروزه این شبکه ها کاربرد های متنوعی دارند و جزو شبکه های محبوب می باشند. از جمله این کاربرد ها میتوان به چند مورد زیر اشاره کرد:

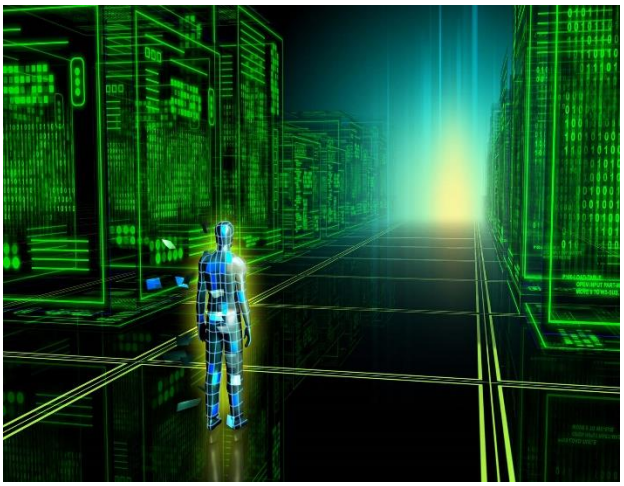
Deep fake



Domain translation



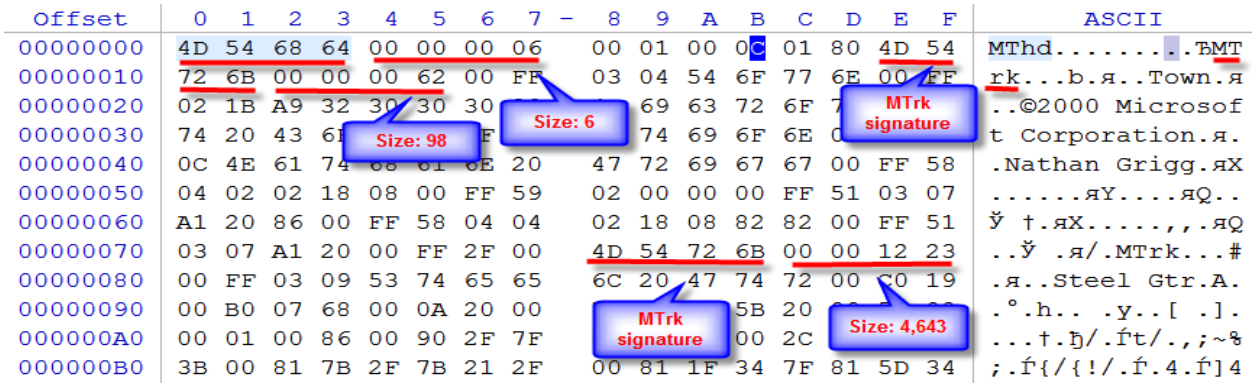
Music generation



Simulation

تولید موسیقی:

در دنیای دیجیتال فرمت های زیادی برای فایل های موسیقی و صوتی وجود دارد. اگرچه برای اهداف این پروژه همه ی این فرمت ها مناسب نمی باشند. MIDI یک نوع فرمت است که در آن یک موسیقی یا مثلاً آهنگ به صورت ساختار موسیقی مثل نت، اکتاو، کورد، رست، اکسیدنتال، آفست و ... برای هر لحظه از آهنگ تعریف شده است. این فرمت در مقایسه با فرمت هایی همچون mp3 برای machine learning و هوش مصنوعی بسیار مفید تر می باشد. در واقع ما برای آنکه بتوانیم به بهترین شکل یک شبکه را با یک آهنگ یا صوت آموزش دهیم، نیاز است تا ساختار آن را برای شبکه روشن سازیم. اگرچه این امر ضروری نیست اما برای بالا بردن هوش شبکه و دقت کار و در ادامه نیز دادنِ توانایی های بیشتر به آن برای تولید موسیقی بهتر و پیشرفته تر، قطعاً این روش بسیار مفید خواهد بود. همانطور که در شکل زیر می بینید، این فرمت صوتی کاملاً از قوانین موسیقی که در آهنگ وجود دارند ساخته شده است. در واقع شبکه ی ما با آموزش دیدن توسط این نوع داده ی صوتی مثل کودکی است که در حال یادگیری نواختن یک ساز یا موسیقی است و پس از یادگیری کامل اجزای آن می تواند خود نیز یک موسیقی بنوازد.



کتابخانه MUSIC21

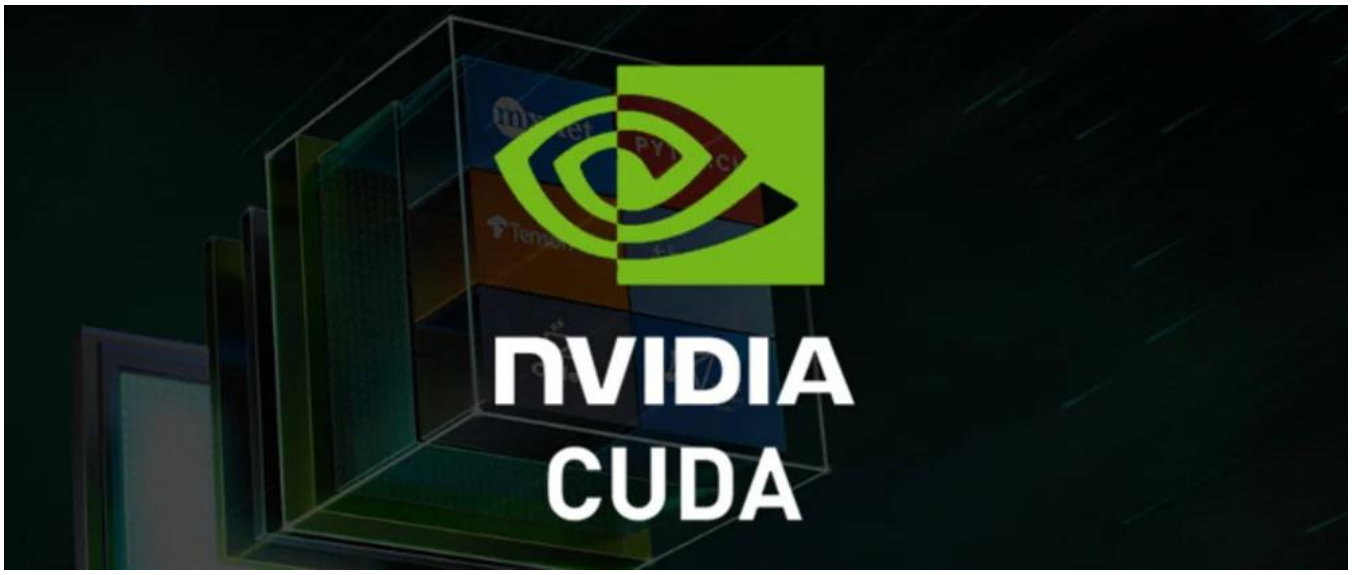
حال که با فرمت MIDI آشنا شدیم، باید شیوه ی استفاده از آن برای آموزش شبکه را نیز بدانیم. با وجود اینکه یک فایل صوتی با این فرمت در خود اطلاعات موزیکالِ گفته شده را دارد، اما نیاز است که با ابزاری این اطلاعات از این فایل استخراج شود (Parsing) . ما برای اینکار از یک کتابخانه ی قدرتمند و معروف در زبان برنامه نویسی پایتون استفاده کردیم. کتابخانه ی MUSIC21 نه تنها دارای قابلیت استخراج این اطلاعات می باشد بلکه می توان از آن برای دسته بندی، جدا سازی، تغییر و ویژگی دادن به این اطلاعات نیز استفاده کرد. همانطور که در شکل زیر (سمت راست) میبینید، اطلاعات یک دیتاست پس از استخراج شدن توسط این کتابخانه به صورت زمانی (بر حسب کوارتر نت) چیده شده و ساختار موزیکال فایل را نمایش میدهد:

```
'G5',
'E5.G2',
'G1',
'G5.D5',
'G5',
'C6.C3',
'C5',
'C2',
'G5',
'D5',
'E4',
'G5',
'E5',
'C4.G3',
'C5',
'G5',
'G6',
'G5.C4',
'D5',
'C3',
'G5',
'E5',
'E6',
'B-4.C5',
'B-3.B-2',
'D5',
'E5',
'F#5.A3',
'A2',
'F#4.A3',
'F#5.D5',
'A4',
```

```
{0.0} <music21.stream.Part 0x7f2d9eb92f98>
    {0.0} <music21.tempo.MetronomeMark animato Quarter=120.0>
    {0.0} <music21.meter.TimeSignature 4/4>
    {0.0} <music21.stream.Voice 0x7f2da7488e10>
        {0.0} <music21.note.Rest rest>
        {19.75} <music21.note.Note C#>
        {19.75} <music21.note.Rest rest>
        {20.0} <music21.note.Note G#>
        {25.0} <music21.note.Note B->
        {25.75} <music21.chord.Chord E-5 B-4>
        {26.75} <music21.chord.Chord B-4 G#4>
        {27.3333} <music21.note.Note E->
        {29.6667} <music21.note.Note B->
        {32.0} <music21.note.Note G#>
        {34.5} <music21.note.Note G#>
        {39.75} <music21.note.Note C#>
        {44.0} <music21.note.Note E->
        {44.25} <music21.note.Note F>
        {49.5} <music21.note.Note B->
        {52.0} <music21.chord.Chord B-3 G#4>
        {52.5} <music21.note.Note G#>
        {55.3333} <music21.note.Note F>
        {58.75} <music21.chord.Chord E-5 F#4>
        {62.6667} <music21.note.Note C#>
        {63.0} <music21.note.Note F>
        {66.25} <music21.note.Note C#>
        {67.6667} <music21.chord.Chord G#5 B-3>
        {70.0} <music21.chord.Chord F#5 F#4>
        {72.25} <music21.note.Note B->
        {72.25} <music21.note.Note F>
        {76.6667} <music21.note.Note B->
        {79.5} <music21.chord.Chord G4 G3>
```

در شکل سمت چپِ بالا هم نوع هر نت به تنهایی نشان داده شده است.

حال ما باید توسط یک الگوریتم خاص این داده ها را به صورت عددی در آورده و سپس روی یک بازه اسکیل کنیم تا بتوانیم آن را به شبکه بدهیم. برای اینکار کافیست که برای هر المان در این داده ها یک عدد خاص نسبت دهیم و سپس تبدیل را انجام دهیم. اگرچه ما در این پروژه به تبدیل نت (با اکتاو)، کورد و رست بسنده کرده ایم، اما هرچه که تنوع داده های موزیکال که بهشان عدد نسبت داده و به شبکه میدهیم بیشتر باشد، عملکرد شبکه در تولید موسیقی هوشمند تر و نزدیک تر به انسان خواهد بود و در عین حال به شبکه ی پیچیده تری نیز نیاز است.



پلتفرم CUDA و پردازنده‌ی گرافیکی

خیلی از مردم CUDA را با یک زبان برنامه نویسی یا حتی یک API اشتباه می‌گیرند در صورتی که اینطور نیست. CUDA یک پلتفرم محاسبات موازی و یک مدل برنامه نویسی است که استفاده از پردازنده‌های گرافیکی (GPU) را برای محاسبات موازی بسیار ساده می‌کند. در واقع برنامه‌نویس نیاز نیست تا زبان جدیدی یاد بگیرد، صرفا با اضافه کردم افزونه‌ی CUDA در همان زبان و برنامه، از محاسبات موازی روی GPU بره می‌برد.

کتابخانه‌ی keras که بر پایه‌ی کتابخانه‌ی Tensorflow ساخته شده است، به صورت پیشفرض از این افزونه استفاده می‌کند. در حالتی که برنامه نتواند به صورت کامل به پردازنده‌ی گرافیکی وصل شود، به اجبار از پردازنده‌ی مرکزی (CPU) برای محاسبات بهره می‌برد. ما در این پروژه با نصب فایل‌های مورد نیاز و راه اندازی پیشنیازهای این پلتفرم، نهایتا موفق شدیم تا شبکه‌ی عصبی خود را با کمک کارت گرافیکی 1080ti که به صورت remote به آن وصل بودیم آموزش دهیم که سرعت بسیار بسیار بالایی دارد. در مقایسه با یک پردازنده‌ی COREi7 نسل هفتم، تقریبا ۱۰۰ برابر سریعتر عمل کرد.

دیتاست‌ها

با جستجوی فراوان و با کمک لینک معرفی شده در فایل تعریف پروژه، نهایتا به سه دیتاست کلی دست پیدا کردیم:

۱- دیتاست GiantMIDI:

این دیتاست در اصل ماحصل یک پروژه‌ی بسیار جذاب هوش مصنوعی است که توسعه دهنده‌ی آن یک مدلی ساخته است که می‌تواند فایل‌های mp3 را به فایل MIDI تبدیل کند! وی بعد از جمع آوری تمام موزیک‌های نواخته شده با پیانو در یوتیوب، آن‌ها را به فایل MIDI تبدیل کرده است. اما فقط چهار فایل نمونه از این کار را در لینک موجود در تعریف پروژه قرار داده است. ما پس از مکاتبه با توسعه دهنده توانستیم به دیتاست اصلی که شامل ۱۱۰۰۰ فایل MIDI است دست پیدا کنیم.

۲- دیتاست سازهای ایرانی:

بعد از جستجو در اینترنت توانستیم چندین فایل MIDI از سازهای ایرانی پیدا کنیم اما به دلیل تعداد کم این داده‌ها، برای آموزش مدل ما مناسب نبود و نتیجه‌ی مطلوبی نداشت.

۳- دیتاست Lakh:

این یک دیتاست غنی از چندین فایل MIDI است که یک وجه تمایز مهم با دیتاست قبلی دارد، آن هم این است که این فایل‌ها شامل پنج ساز همزمان است نه فقط یک ساز مثل پیانو. این دیتاست می‌تواند برای آموزش شبکه‌ای استفاده شود که قصد دارد یک موزیک با چند ساز تولید کند.