Recommending music on Spotify with deep learning

AUGUST 05, 2014

Reading time ~20 minutes

This summer, I'm interning at Spotify in New York City, where I'm working on content-based music recommendation using convolutional neural networks. In this post, I'll explain my approach and show some preliminary results.

1. Overview

This is going to be a long post, so here's an overview of the different sections. If you want to skip ahead, just click the section title to go there.

- *Collaborative filtering*
  A very brief introduction, its virtues and its flaws.
- *Content-based recommendation*
  What to do when no usage data is available.
- *Predicting listening preferences with deep learning*
  Music recommendation based on audio signals.
- *Scaling up*
  Some details about the convnets I've been training at Spotify.
- *Analysis: what is it learning?*
  A look at what the convnets learn about music, with **lots of audio examples**, yay!
- *What will this be used for?*
  Some potential applications of my work.
- *Future work*
- *Conclusion*

2. Collaborative filtering

Traditionally, Spotify has relied mostly on collaborative filtering approaches to power their recommendations. The idea of collaborative filtering is to **determine the users' preferences from historical usage data**. For example, if two users listen to largely the same set of songs, their tastes are probably similar. Conversely, if two songs are listened to by the same group of users, they probably sound similar. This kind of information can be exploited to make recommendations.

Pure collaborative filtering approaches do not use any kind of information about the items that are being recommended, except for the consumption patterns associated with them: they are **content-agnostic**. This makes these approaches widely applicable: the same type of model can be used to recommend books, movies or music, for example.

Unfortunately, this also turns out to be their biggest flaw. Because of their reliance on usage data, popular items will be much easier to recommend than unpopular items, as there is more usage data available for them. This is usually the opposite of what we want. For the same reason, the recommendations can often be rather boring and predictable.

Another issue that is more specific to music, is the **heterogeneity of content with similar usage patterns**. For example, users may listen to entire albums in one go, but albums may contain intro tracks, outro tracks, interludes, cover songs and remixes. These items are atypical for the

artist in question, so they aren't good recommendations. Collaborative filtering algorithms will not pick up on this.

But perhaps the biggest problem is that **new and unpopular songs cannot be recommended**: if there is no usage data to analyze, the collaborative filtering approach breaks down. This is the so-called **cold-start problem**. We want to be able to recommend new music right when it is released, and we want to tell listeners about awesome bands they have never heard of. To achieve these goals, we will need to use a different approach.

3. Content-based recommendation

Recently, Spotify has shown considerable interest in incorporating other sources of information into their recommendation pipeline to mitigate some of these problems, as evidenced by their acquisition of music intelligence platform company The Echo Nest a few months back. There are many different kinds of information associated with music that could aid recommendation: tags, artist and album information, lyrics, text mined from the web (reviews, interviews, …), and the audio signal itself.

Of all these information sources, the audio signal is probably the most difficult to use effectively. There is quite a large **semantic gap** between music audio on the one hand, and the various aspects of music that affect listener preferences on the other hand. Some of these are fairly easy to extract from audio signals, such as the genre of the music and the instruments used. Others are a little more challenging, such as the mood of the music, and the year (or time period) of release. A couple are practically impossible to obtain from audio: the geographical location of the artist and lyrical themes, for example.

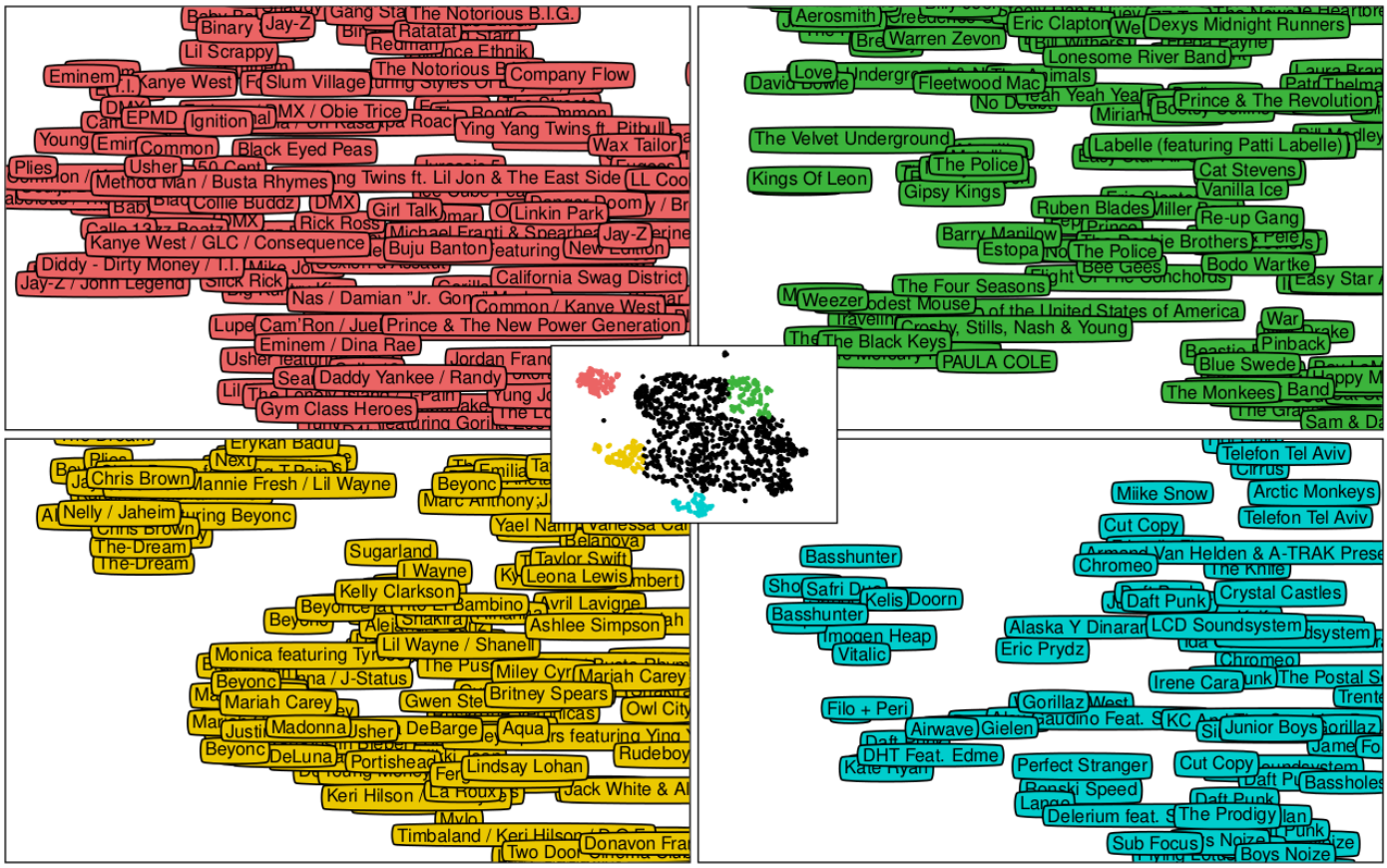
Despite all these challenges, it is clear that the actual *sound* of a song will play a very big role in determining whether or not you enjoy listening to it - so it seems like a good idea to try to predict who will enjoy a song by analyzing the audio signal.

4. Predicting listening preferences with deep learning

In December last year, my colleague Aäron van den Oord and I published a paper on this topic at NIPS, titled **'Deep content-based music recommendation'**. We tried to tackle the problem of predicting listening preferences from audio signals by training a regression model to predict the **latent representations** of songs that were obtained from a collaborative filtering model. This way, we could predict the representation of a song in the collaborative filtering space, even if no usage data was available. (As you can probably infer from the title of the paper, the regression model in question was a deep neural network.)

The underlying idea of this approach is that many collaborative filtering models work by projecting both the listeners and the songs into a shared low-dimensional **latent space**. The position of a song in this space encodes all kinds of information that affects listening preferences. If two songs are close together in this space, they are probably similar. If a song is close to a user, it is probably a good recommendation for that user (provided that they haven't heard it yet). If we can predict the position of a song in this space from audio, we can recommend it to the right audience without having to rely on historical usage data.

We visualized this in the paper by projecting the predictions of our model in the latent space down to two dimensions using the t-SNE algorithm. As you can see below on the resulting map, similar songs cluster together. Rap music can be found mostly in the top left corner, whereas electronic artists congregate at the bottom of the map.



t-SNE visualization of the latent space (middle). A few close-ups show artists whose songs are projected in specific areas. Taken from *Deep content-based music recommendation, Aäron van den Oord, Sander Dieleman and Benjamin Schrauwen, NIPS 2013.*

5. Scaling up

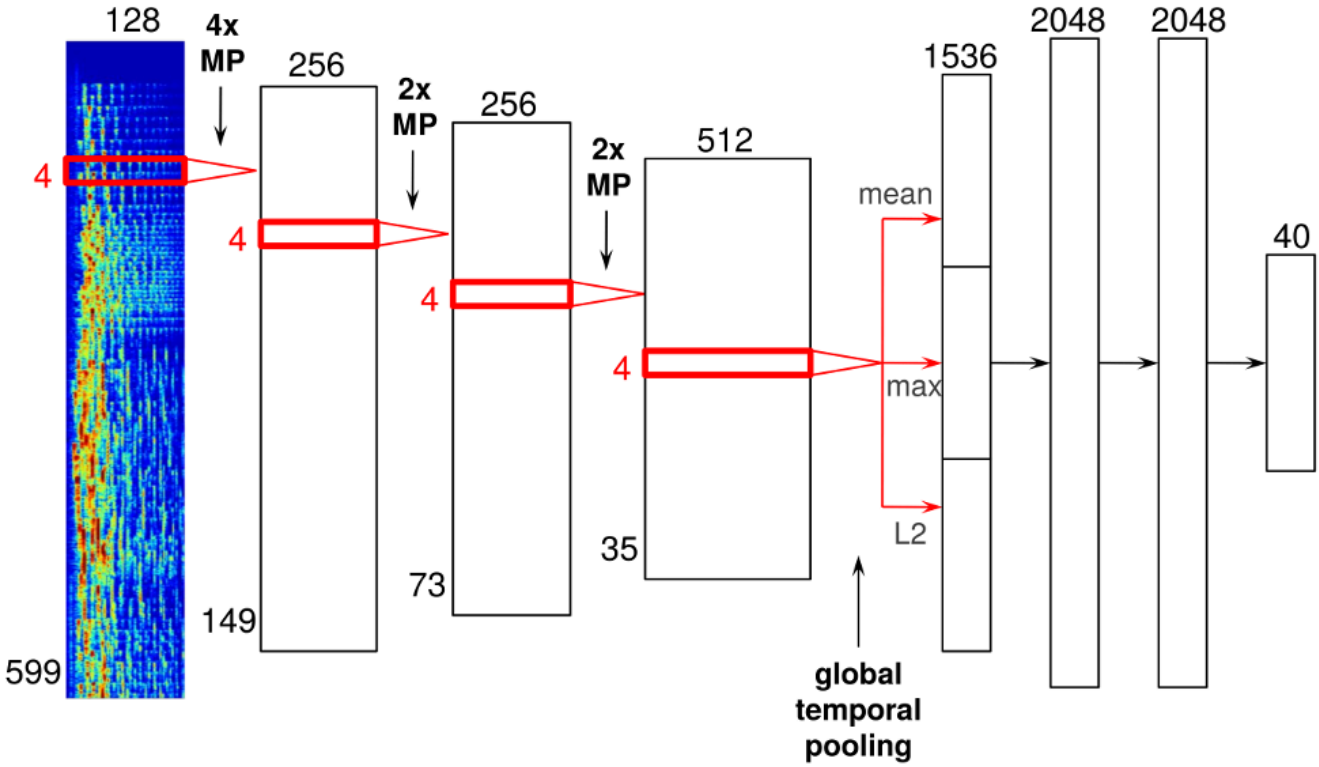
The deep neural network that we trained for the paper consisted of two convolutional layers and two fully connected layers. The input consisted of spectrograms of 3 second fragments of audio. To get a prediction for a longer clip, we just split it up into 3 second windows and averaged the predictions across these windows.

At Spotify, I have access to a larger dataset of songs, and a bunch of different latent factor representations obtained from various collaborative filtering models. They also got me a nice GPU to run my experiments on. This has allowed me to scale things up quite a bit. I am currently training convolutional neural networks (convnets) with 7 or 8 layers in total, using much larger intermediate representations and many more parameters.

5.1. Architecture

Below is an example of an architecture that I've tried out, which I will describe in more detail. It has four convolutional layers and three dense layers. As you will see, there are some important differences between convnets designed for audio signals and their more traditional counterparts used for computer vision tasks.

Warning: gory details ahead! Feel free to skip ahead to 'Analysis' if you don't care about things like ReLUs, max-pooling and minibatch gradient descent.



One of the convolutional neural network architectures I've tried out for latent factor prediction. The time axis (which is convolved over) is vertical.

The input to the network consists of **mel-spectrograms**, with 599 frames and 128 frequency bins. A mel-spectrograms is a kind of **time-frequency representation**. It is obtained from an audio signal by computing the Fourier transforms of short, overlapping windows. Each of these Fourier transforms constitutes a *frame*. These successive frames are then concatenated into a matrix to form the spectrogram. Finally, the frequency axis is changed from a linear scale to a mel scale to reduce the dimensionality, and the magnitudes are scaled logarithmically.

The **convolutional layers** are displayed as red rectangles delineating the shape of the filters that slide across their inputs. They have rectified linear units (ReLUs, with activation function `max(0, x)`). Note that all these convolutions are **one-dimensional**; the convolution happens only in the time dimension, not in the frequency dimension. Although it is technically possible to convolve along both axes of the spectrogram, I am not currently doing this. It is important to realize that the two axes of a spectrogram have different meanings (time vs. frequency), which is not the case for images. As a result, it doesn't really make sense to use square filters, which is what is typically done in convnets for image data.

Between the convolutional layers, there are **max-pooling operations** to downsample the intermediate representations in time, and to add some time invariance in the process. These are indicated with '**MP**'. As you can see I used a filter size of 4 frames in every convolutional layer, with max-pooling with a pool size of 4 between the first and second convolutional layers (mainly for performance reasons), and with a pool size of 2 between the other layers.

After the last convolutional layer, I added a **global temporal pooling layer**. This layer pools across the entire time axis, effectively computing statistics of the learned features across time. I included three different pooling functions: the mean, the maximum and the L2-norm.

I did this because the absolute location of features detected in the audio signal is not particularly relevant for the task at hand. This is not the case in image classification: in an image, it can be useful to know roughly where a particular feature was detected. For example, a feature detecting clouds would be more likely to activate for the top half of an image. If it activates in the bottom half, maybe it is actually detecting a sheep instead. For music recommendation, we are typically only interested in the overall presence or absence of certain features in the music, so it makes sense to perform pooling across time.

Another way to approach this problem would be to train the network on short audio fragments, and average the outputs across windows for longer fragments, as we did in the NIPS paper. However, incorporating the pooling into the model seems like a better idea, because it allows for this step to be taken into account during learning.

The globally pooled features are fed into a series of **fully-connected layers** with 2048 rectified linear units. In this network, I have two of them. The last layer of the network is the **output layer**, which predicts 40 latent factors obtained from the **vector_exp** algorithm, one of the various collaborative filtering algorithms that are used at Spotify.

5.2. Training

The network is trained to minimize the **mean squared error** (MSE) between the latent factor vectors from the collaborative filtering model and the predictions from audio. These vectors are first normalized so they have a unit norm. This is done to reduce the influence of song popularity (the norms of latent factor vectors tend to be correlated with song popularity for many collaborative filtering models). Dropout is used in the dense layers for regularization.

The dataset I am currently using consists of mel-spectrograms of 30 second excerpts extracted from the middle of the 1 million most popular tracks on Spotify. I am using about half of these for training (0.5M), about 5000 for online validation, and the remainder for testing. During training, the data is augmented by slightly cropping the spectrograms along the time axis with a random offset.

The network is implemented in **Theano**, and trained using minibatch gradient descent with Nesterov momentum on a NVIDIA GeForce GTX 780Ti GPU. Data loading and augmentation happens in a separate process, so while the GPU is training on a chunk of data, the next one can be loaded in parallel. About 750000 gradient updates are performed in total. I don't remember exactly how long this particular architecture took to train, but all of the ones I've tried have taken between 18 and 36 hours.

5.3. Variations

As I mentioned before, this is just one example of an architecture that I've tried. Some other things I have tried / will try include:

- More layers!
- Using maxout units instead of rectified linear units.
- Using stochastic pooling instead of max-pooling.
- Incorporating L2 normalization into the output layer of the network.
- Data augmentation by stretching or compressing the spectrograms across time.
- Concatenating multiple latent factor vectors obtained from different collaborative filtering models.

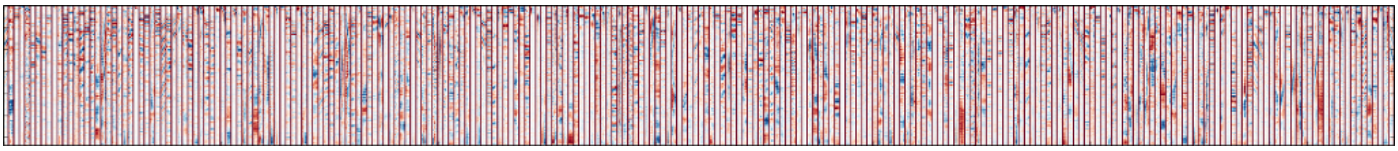
Here are some things that didn't work quite as well as I'd hoped:

- Adding 'bypass' connections from all convolutional layers to the fully connected part of the network, with global temporal pooling in between. The underlying assumption was that statistics about low-level features could also be useful for recommendation, but unfortunately this hampered learning too much.
- Predicting the conditional variance of the factors as in mixture density networks, to get confidence estimates for the predictions and to identify songs for which latent factor prediction is difficult. Unfortunately this seemed to make training quite a lot harder, and the resulting confidence estimates did not behave as expected.

6. Analysis: what is it learning?

Now for the cool part: **what are these networks learning? What do the features look like?** The main reason I chose to tackle this problem with convnets, is because I believe that music recommendation from audio signals is a pretty complex problem bridging many levels of abstraction. My hope was that successive layers of the network would learn progressively more complex and invariant features, as they do for image classification problems.

It looks like that's exactly what is happening. First, let's take a look at the first convolutional layer, which learns a set of filters that are applied directly to the input spectrograms. These filters are easy to visualize. They are shown in the image below. Click for a high resolution version (5584x562, ~600kB). Negative values are red, positive values are blue and white is zero. Note that each filter is only four frames wide. The individual filters are separated by dark red vertical lines.



Visualization of the filters learned in the first convolutional layer. The time axis is horizontal, the frequency axis is vertical (frequency increases from top to bottom). Click for a high resolution version (5584x562, ~600kB).

From this representation, we can see that a lot of the filters pick up harmonic content, which manifests itself as parallel red and blue bands at different frequencies. Sometimes, these bands are are slanted up or down, indicating the presence of rising and falling pitches. It turns out that these filters tend to detect human voices.

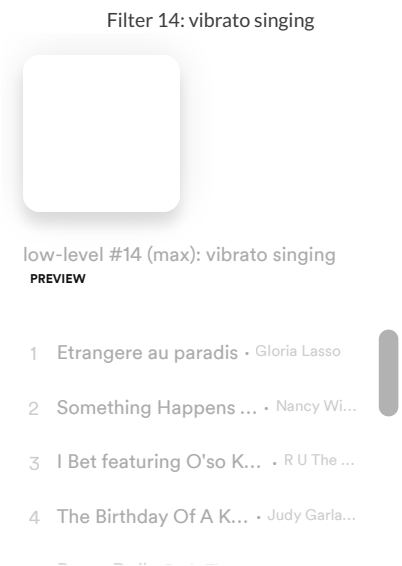
6.1. Playlists for low-level features: maximal activation

To get a better idea of what the filters learn, I made some playlists with songs from the test set that maximally activate them. Below are a few examples. There are 256 filters in the first layer of the network, which I numbered from 0 to 255. Note that this numbering is arbitrary, as they are unordered.

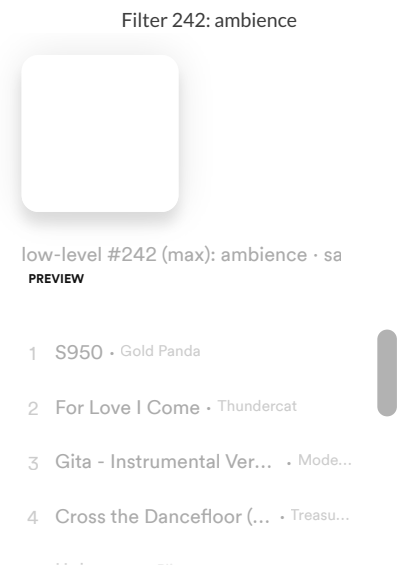
These four playlists were obtained by finding songs that maximally activate a given filter in the 30 seconds that were analyzed. I selected a few interesting looking filters from the first convolutional layer and computed the feature representations for each of these, and then searched for the maximal activations across the entire test set. **Note that you should listen to the middle of the tracks to hear what the filters are picking up on, as this is the part of the audio signal that was analyzed.**

All of the Spotify playlists below should have 10 tracks. Some of them may not be available in all countries due to licensing issues.

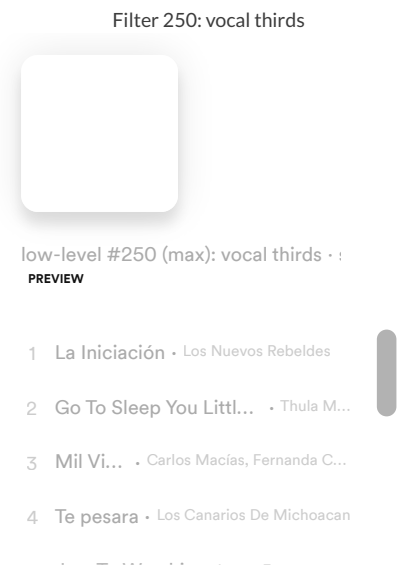
Filter 14: vibrato singing



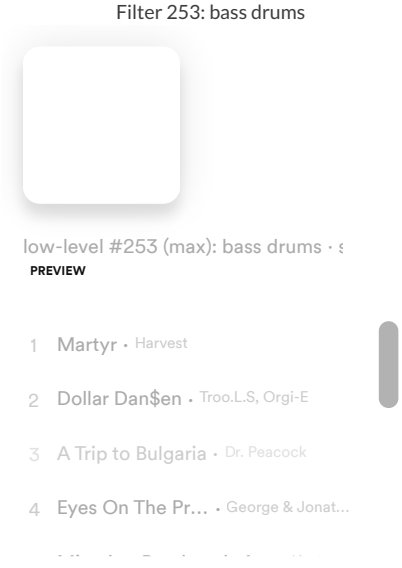
Filter 242: ambience

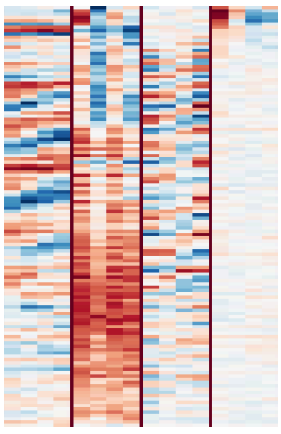


Filter 250: vocal thirds



Filter 253: bass drums





Closeup of filters 14, 242, 250 and 253. Click for a larger version.

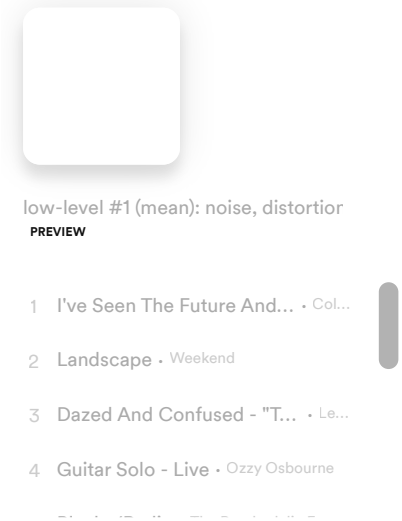
- Filter 14 seems to pick up **vibrato singing**.
- Filter 242 picks up some kind of **ringing ambience**.
- Filter 250 picks up **vocal thirds**, i.e. multiple singers singing the same thing, but the notes are a major third (4 semitones) apart.
- Filter 253 picks up various types of **bass drum sounds**.

The genres of the tracks in these playlists are quite varied, which indicates that these features are picking up mainly low-level properties of the audio signals.

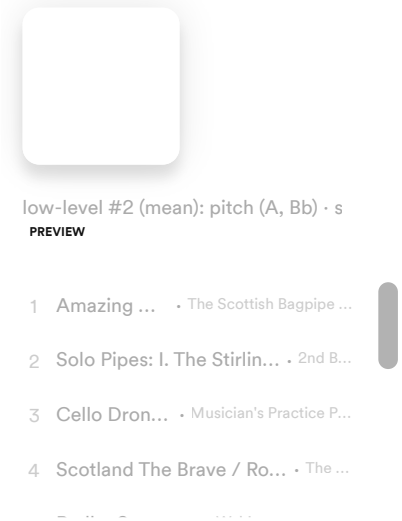
6.2. Playlists for low-level features: average activation

The next four playlists were obtained in a slightly different way: I computed the **average activation of each feature across time** for each track, and then found the maximum across those. This means that for these playlists, the filter in question is constantly active in the 30 seconds that were analyzed (i.e. it's not just one 'peak'). This is more useful for detecting harmonic patterns.

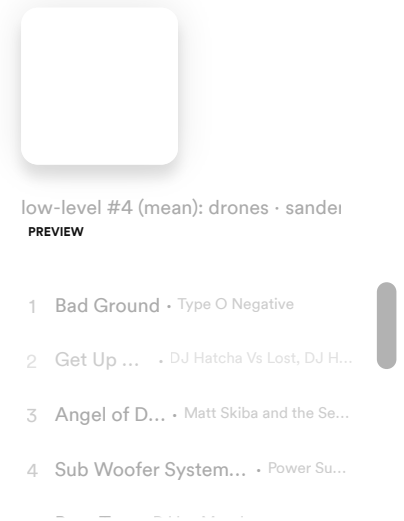
Filter 1: noise, distortion



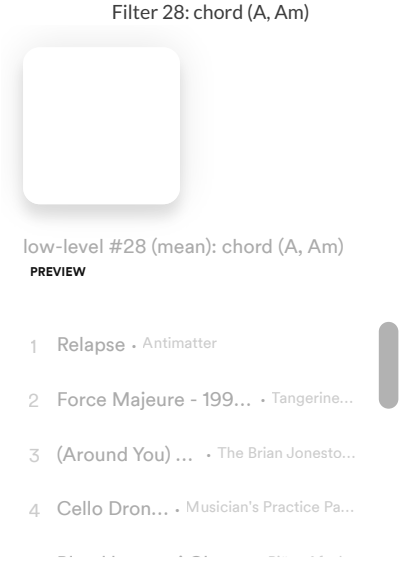
Filter 2: pitch (A, Bb)

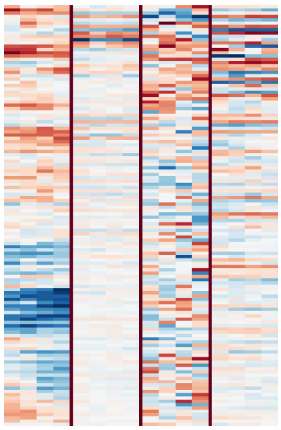


Filter 4: drones



Filter 28: chord (A, Am)





Closeup of filters 1, 2, 4 and 28. Click for a larger version.

- Filter 1 picks up **noise** and (guitar) **distortion**.
- Filter 2 seems to pick up a **specific pitch: a low Bb**. It also picks up A sometimes (a semitone below) because the frequency resolution of the mel-spectrograms is not high enough to distinguish them.
- Filter 4 picks up various low-pitched **drones**.
- Filter 28 picks up the **A chord**. It seems to pick up both the minor and major versions, so it might just be detecting the pitches A and E (the fifth).

I thought it was very interesting that the network is learning to detect specific pitches and chords. I had previously assumed that the exact pitches or chords occurring in a song would not really affect listener preference. I have two hypotheses for why this might be happening:

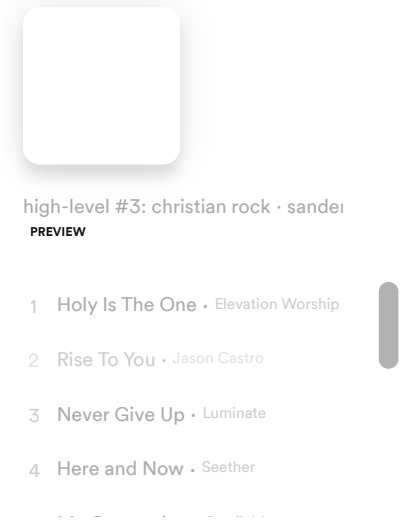
- The network is just learning to detect **harmonicity**, by learning various filters for different kinds of harmonics. These are then pooled together at a higher level to detect harmonicity across different pitches.
- The network is learning that some **chords and chord progressions** are more common than others in certain genres of music.

I have not tried to verify either of these, but it seems like the latter would be pretty challenging for the network to pick up on, so I think the former is more likely.

6.3. Playlists for high-level features

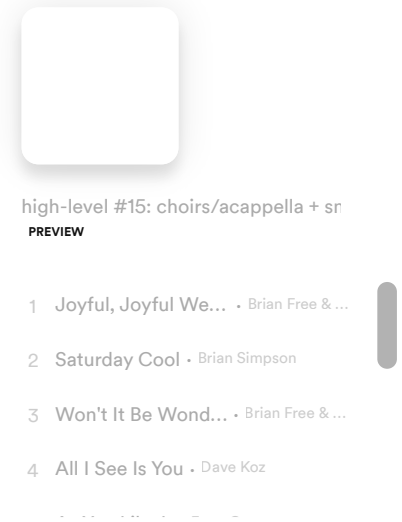
Each layer in the network takes the feature representation from the layer below, and extracts a set of higher-level features from it. **At the topmost fully-connected layer of the network, just before the output layer, the learned filters turn out to be very selective for certain subgenres**. For obvious reasons, it is non-trivial to visualize what these filters pick up on at the spectrogram level. Below are six playlists with songs from the test set that maximally activate some of these high-level filters.

Filter 3: christian rock



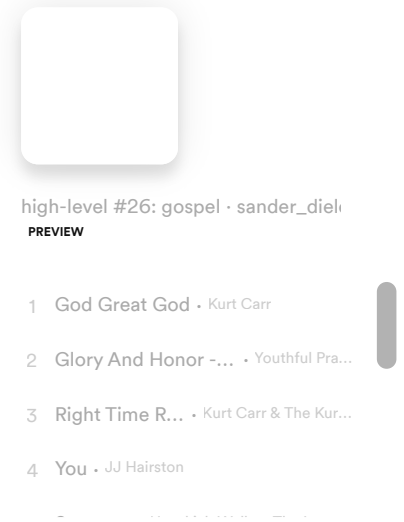
Filter 37: Chinese pop

Filter 15: choirs / a cappella + smooth jazz

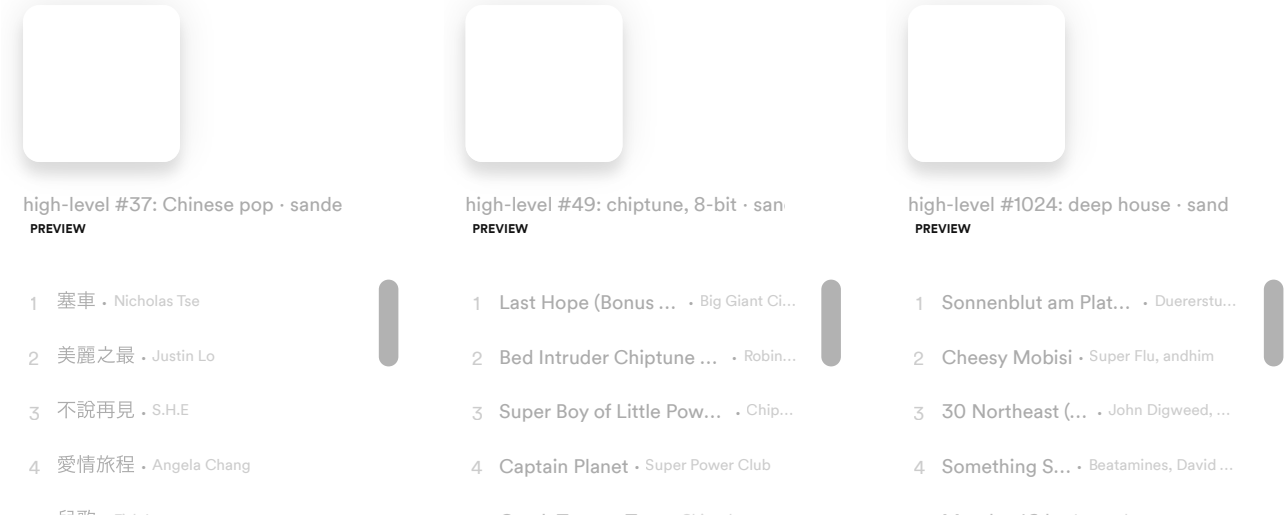


Filter 49: chiptune, 8-bit

Filter 26: gospel



Filter 1024: deep house



It is clear that each of these filters is identifying specific genres. Interestingly some filters, like #15 for example, seem to be **multimodal**: they activate strongly for two or more styles of music, and those styles are often completely unrelated. Presumably the output of these filters is disambiguated when viewed in combination with the activations of all other filters.

Filter 37 is interesting because it almost seems like it is **identifying the Chinese language**. This is not entirely impossible, since the phoneme inventory of Chinese is quite distinct from other languages. A few other seemingly language-specific filters seem to be learned: there is one that detects rap music in Spanish, for example. Another possibility is that Chinese pop music has some other characteristic that sets it apart, and the model is picking up on that instead.

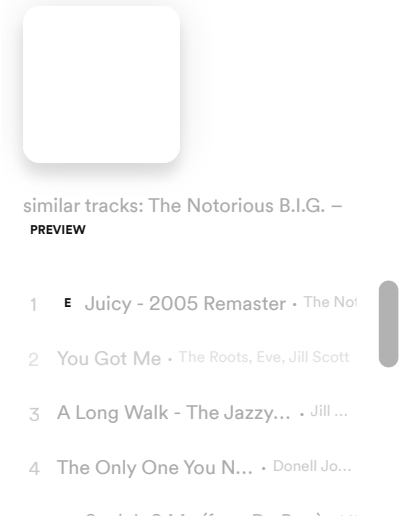
I spent some time analyzing the first 50 or so filters in detail. Some other filter descriptions I came up with are: lounge, reggae, darkwave, country, metalcore, salsa, Dutch and German carnival music, children's songs, dance, vocal trance, punk, Turkish pop, and my favorite, 'exclusively Armin van Buuren'. Apparently he has so many tracks that he gets his own filter.

The filters learned by Alex Krizhevsky's ImageNet network have been reused for various other computer vision tasks with great success. Based on their diversity and invariance properties, it seems that these filters learned from audio signals may also be useful for other music information retrieval tasks besides predicting latent factors.

6.4. Similarity-based playlists

Predicted latent factor vectors can be used to find songs that sound similar. Below are a couple of playlists that were generated by predicting the factor vector for a given song, and then **finding other songs in the test set whose predicted factor vectors are close to it** in terms of cosine distance. As a result, the first track in the playlist is always the query track itself.

The Notorious B.I.G. - Juicy (hip hop)



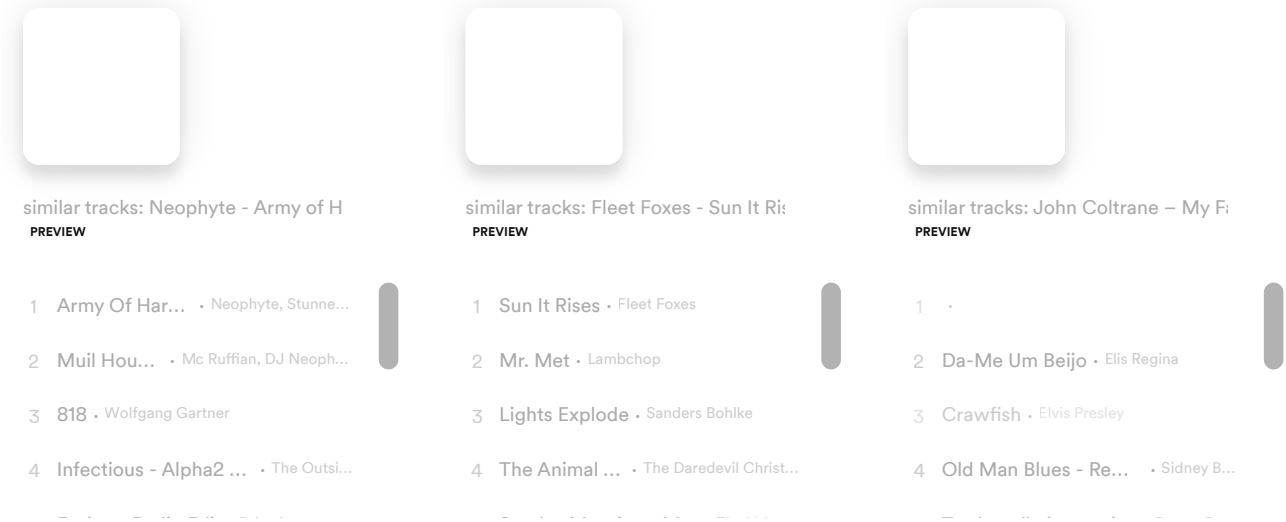
Neophyte - Army of Hardcore (hardcore techno, gabber)

Cloudkicker - He would be riding on the subway... (post-rock, post-metal)

Fleet Foxes - Sun It Rises (indie folk)

Architects - Numbers Count For Nothing (metalcore, hardcore)

John Coltrane - My Favorite Things (jazz)



Most of the similar tracks are pretty decent recommendations for fans of the query tracks. Of course these lists are far from perfect, but considering that they were obtained based only on the audio signals, the results are pretty decent. One example where things go wrong is the list for 'My Favorite Things' by John Coltrane, which features a couple of strange outliers, most notably 'Crawfish' by Elvis Presley. This is probably because the part of the audio signal that was analyzed (8:40 to 9:10) contains a crazy sax solo. Analyzing the whole song might give better results.

7. What will this be used for?

Spotify already uses a bunch of different information sources and algorithms in their recommendation pipeline, so the most obvious application of my work is simply to include it as an extra signal. However, it could also be used to filter outliers from recommendations made by other algorithms. As I mentioned earlier, collaborative filtering algorithms will tend to include intro tracks, outro tracks, cover songs and remixes in their recommendations. These could be filtered out effectively using an audio-based approach.

One of my main goals with this work is to make it possible to recommend new and unpopular music. I hope that this will help lesser known and up and coming bands, and that it will level the playing field somewhat by enabling Spotify to recommend their music to the right audience. (Promoting up and coming bands also happens to be one of the main objectives of my non-profit website got-djent.com.)

Hopefully some of this will be ready for A/B testing soon, so we can find out if these audio-based recommendations actually make a difference in practice. This is something I'm very excited about, as it's not something you can easily do in academia.

8. Future work

Another type of user feedback that Spotify collects are the **thumbs up** and **thumbs down** that users can give to tracks played on radio stations. This type of information is very useful to determine which tracks are similar. Unfortunately, it is also quite noisy. I am currently trying to use this data in a 'learning to rank' setting. I've also been experimenting with various distance metric learning schemes, such as DrLIM. If anything cool comes out of that I might write another post about it.

9. Conclusion

In this post I've given an overview of my work so far as a machine learning intern at Spotify. I've explained my approach to using convnets for audio-based music recommendation and I've tried to provide some insight into what the networks actually learn. For more details about the approach, please refer to the NIPS 2013 paper 'Deep content-based music recommendation' by Aäron van den Oord and myself.

If you are interested in deep learning, feature learning and its applications to music, have a look at my research page for an overview of some other work I have done in this domain. If you're interested in Spotify's approach to music recommendation, check out these presentations on Slideshare and Erik Bernhardsson's blog.

Spotify is a really cool place to work at. They are very open about their methods (and they let me write this blog post), which is not something you come across often in industry. If you are interested in recommender systems, collaborative filtering and/or music information retrieval, and you're looking for an internship or something more permanent, don't hesitate to get in touch with them.

If you have any questions or feedback about this post, feel free to leave a comment!

- Post on Hacker News

- Post on r/machinelearning
- Post on the Google+ deep learning community
- Post on the Google+ music information retrieval community

View of NYC from the Spotify deck.

DEEP LEARNING CONVOLUTIONAL NEURAL NETWORKS CONVNETS THEANO CONVOLUTION MIR MUSIC INFORMATION RETRIEVAL MUSIC RECOMMENDATION SPOTIFY INTERNSHIP MUSIC COLLABORATIVE FILTERING COLD START PROBLEM

132 Comments

Login ▾

G

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

28 Share

Best Newest Oldest

T

Tobias

10 years ago

Very impressive indeed! I am starting in September with a Machine Learning masters at UCL in London. Reading your blog post reassured me of my decision to study this exciting field.

I am now listening to the playlist of Filter 1024, loving it ;)

Keep up the good work!

Tobias

5 0 Reply Share ›

B

bumelant

10 years ago

I did my master thesis on collaborative filtering few years back and I'm a huge fan of music. I'm also an avid user of Spotify (kind of love hate relationship) and it was really entertaining to read on all you guys do in terms of recommendations. Great article! I think this area still needs huge improvements in Spotify, especially with bringing less obvious recommendations in, so keep up the good work!

2 0 Reply Share ›

S

Sander Dieleman Mod ↗ bumelant

10 years ago

Hey,

If you're interested in how Spotify does recommendations, definitely check out my supervisor Erik Bernhardsson's blog, he writes about this regularly: http://erikbern.com/

Thanks for the kind words!

Sander

4 0 Reply Share ›

Vinicius ↗ bumelant

10 years ago

Hey! Would you mind to share your thesis with us? I'm writing my own thesis right now, about recommendation systems and more sources is always useful :-)

2 0 Reply Share ›

D

David King ↗ Vinicius

8 years ago

I'm also starting my dissertation on music recommendation using machine learning. could you share your thesis? as you say, more sources are always useful.

0 0 Reply Share ›

S

Sander Dieleman Mod ↗ David King

8 years ago

You can find it here: https://www.dropbox.com/s/2...

0 0 Reply Share ›

D

David King ↗ Sander Dieleman

8 years ago

Sander - thank you. I was hoping for Vinicius's thesis, but yours looks like a fantastic source of information. A thousand thank yous!

0 0 Reply Share ›

jamey_22
7 years ago

Fascinating work.

Quick (and perhaps very elementary) question, Sander: I can't quite wrap my head around the visualization of the filters - what exactly does it mean for one of the filters to have a negative value (i.e. show up as red) at a particular frequency & time?

2 1 Reply Share ›

Sander Dieleman Mod → jamey_22
7 years ago

It means that the weights connecting to this part of the spectrogram are negative. As a result, if there is a lot of energy in this part of the spectrogram, the activation of the filter will be inhibited. On the contrary, if the area is blue (positive values), energy in this part of the spectrogram will increase the activation of the filter.

So filters with parallel blue bands are in fact looking for specific patterns of harmonics, and if there is additional signal content in frequencies outside of these harmonics, that will actually end up inhibiting the activation of the filters.

1 0 Reply Share ›

Michele
10 years ago

This post is awesome. This work is awesome. And I really appreciate Spotify lets you publish this.
I listened to some playlists and they are very good!

Hope to see this in Spotify very soon.

Good work!

1 0 Reply Share ›

nphippen
10 years ago

Thanks for sharing, a very cool read!

1 0 Reply Share ›

Neil
10 years ago

I'm looking to get started with Machine Learning, but its frustrating that so many write-ups on neural nets and deep learning just describe black boxes. I absolutely love that you have tried to interpret what the network is learning at each layer!

1 0 Reply Share ›

Sander Dieleman Mod → Neil
10 years ago

I feel that trying to understand what the network is learning has helped me a lot in finding new ways to improve their performance. I used to be too lazy to do that kind of analysis, but it has really paid off. It's also really cool to see what kind of invariances it discovers.

One thing I haven't tried is to visualize the salient parts of the input, i.e. the parts that are contributing the most to the activations in the top layer. Matthew Zeiler et al. have a very nice paper about that: http://arxiv.org/abs/1311.2901

That could reveal some interesting patterns about what defines a given genre, for example. If I have some time left I definitely want to give that a try.

Thanks for the kind words!

Sander

1 0 Reply Share ›

Ms Informed
10 years ago

I think spotify should curate more songs instead of suggesting Karaoke version of the songs they removed. No recommendation engine would save subpar content.

1 0 Reply Share ›

i_dont_discus
10 years ago

Finally, someone is doing recommendations the right way. I built something similar a few years back using python. This is how recommendations should be done. You're on the right track, something else is being able to distinguish instruments in the track, which as far as I know is only possible with midi files (unsure about converting them from wav to midi to be able to separate instrument layers). I know a lot of people approach them with a DFT/IFFT but that can only carry you so far

lot of people approach them with a DFT/IFFT but that can only carry you so far.

Melodyne has something called "Direct Note Access" which can give you chord analysis, and polyphonic audio material information. Which can give you the instruments. However this was after what I built, and I'm sure since 2008 there have been many advances.

Knowing the instruments, and comparing the full audio file will allow you to be able to give better recommendations.

However there will still be outliers, some bands enjoy placing instruments / noises outside of the norm. Think people like Martin Zero who use insane things like tons of ambient sounds / spray cans for the cymbal.

Nice step in the right direction though.

1 0 Reply Share ›

S **Sander Dieleman** Mod → i_dont_discus
10 years ago

Personally I don't think there is a 'right way' when it comes to music recommendations. Using the audio signal is great (and it's definitely my favorite approach), but I think the 'ultimate music recommender system' is probably going to follow a holistic approach, integrating every possible source of information (including audio signals, collaborative filtering, web mining, ...). Just CF or just audio is definitely not enough :)

Thanks for your comment!

Sander

4 0 Reply Share ›

Chae Yoon Lee
5 years ago

One of the best writing of recommender system algorithm!!!

0 0 Reply Share ›

Michael Johnson
5 years ago

Awesome. Someone pointed me to your post based on a similar blog I made. Did you consider using an autoencoder to create a compressed representation of the music? Check out my blog post here https://minimizeuncertainty...

0 0 Reply Share ›

A **Alex Torex**
6 years ago

I think it is not possible to do personalized content-based music recommendation with your approach. As a solution you could try to train with top popular songs vs less popular songs in order at least to discern qualities of popular songs. I read somewhere that someone already did this. After this you can reuse this pre-trained network and do transfer learning to personalize based on user preferences. This is what Baidu did to achieve user voice cloning in 20 seconds. Next you need something like a headset with integrated electrods to read brainwaves in order to extract pleasure brain signals and get rich rewards to do online training (Here Neurala has tehnology to do life long training). Same song can vary a lot and it is important to get rich info.

0 0 Reply Share ›

Arash Jamshidi
6 years ago

Hello
Very impressive! did you get aware of the A/B testing and the influence it had on recommendations? I mean does it improve them for the users or no change?

0 0 Reply Share ›

S **Sander Dieleman** Mod → Arash Jamshidi
6 years ago

Hi Arash,

Unfortunately there wasn't enough time during my internship to look at A/B tests, so I don't have this information!

1 0 Reply Share ›

E **Eliott Iticsohn**
6 years ago

Hello,
I am currently working on a research project for my studies about Playlist Prediction. As in the article from Cornell University called "Playlist Prediction via Metric Embedding", I would like to use your API to improve my model . Unfortunately, I just saw your are no longer delivering keys for using the API.
My work aims only to advance the science in this field and I would be honored if you can do an exception by letting me using the API.
Thank you,
Eliott

0 0 Reply Share ›

odrigo
6 years ago

This could be better the usual recommendations that are like "you like X (or gave a score of Y to X) and someone else like X (or gave a similar score to X) and also like Z, so you will problably like Z"

One of the problems with this method is that it is limited by what people know.
Imagine Black Sabbath, now imagine a famous band A, from the guys that know AND LIKE black sabbath, 150000 KNOW this band A. Now imagine a underground band B, from the guys that know AND LIKE Black Sabbath, 2000 KNOW band B.

Lets imagine, band A has a "similarity score" of 25% with black sabbath, this means 25% of the listeners that KNOW AND LIKE Black Sabbath and also Know the band A, will like band A. Band B has a similarity score of 80%, this means that 80%of the guys that KNOW AND LIKE Black Sabbath and know band B exist, will like band B.

Anyway 25% of 150000 listeners is 37500 listeners. And 80% of 2000 is 1600.
Band B had a similarity score of 80% compared with 25% of band A, but will be considered less similar (37500 vs 1600) to black sabbath.

This problem could be lessened by allowing what you don't like influence recommendations, so with more people that like and know black sabbath and know band A, will come more people that like black sabbath but don't like band A. Giving a better weight for people that know more bands (and so are more sure of his taste) could help to lessen the problem.

This problem happens alot at last.fm, with mainstream metal bands having mainstream nu metal bands as recommendations. Or mainstream rock bands from countries that arent japan, uk or usa, having pop artists as recommendations.

Deep Learning method would solve those problems.

0 0 Reply Share ›

Sander Dieleman Mod → odrigo
6 years ago

Hi odrigo, I actually implemented a rudimentary recommender system for a website I used to run, based on these ideas. I'm not sure if deep learning would actually be necessary to tackle this :) I wrote about it here: http://got-djent.com/conten...

0 0 Reply Share ›

Generic ID
7 years ago edited

Disqus is telling me that my comment from 2 weeks ago to this article was automatically marked as spam:

> genericid 16 days ago
> Detected as spam

Can you please look into it? I really put a lot of effort on it. Thanks a lot!

0 0 Reply Share ›

David King
8 years ago

fantastic article. thank you!

I am trying to implement a similar program for my masters dissertation. you mentioned you used Theano. could this instead be implemented in TensorFlow?

0 0 Reply Share ›

Sander Dieleman Mod → David King
8 years ago

Thanks! Sure, why not :) I work at Google DeepMind so I'm mostly using TensorFlow myself these days.

0 0 Reply Share ›

David King → Sander Dieleman
8 years ago

Great. I'll see what I can do (:

0 0 Reply Share ›

Zhu Harry
8 years ago

Hi, benanne. I am pretty interesting in your research. Could I translate it into Chinese?

0 0 Reply Share ›

Alec

8 years ago

Very awesome!!
Hi, Benanne, I am now trying to replicate your algorithm on my dataset (roughly 10 000 songs ~ 260GB, and a user-songRating matrix). But I am somewhat confused..
1. Is that an alternative way to factorize the User-SongRating matrix to get the latent factor of all these 10,000 songs? (I am not clear about how to decide the dimension of the feature factor, in your blog, you just let it be 40.)
2. After step 1, the training data are in the format like:
song_0, feature vector_0.
song_1, feature vector_1.
....

3. How to deal with varied size audios? In your paper, you mentioned that: The networks were trained on windows of 3 seconds sampled randomly from the audio clips. Is these data randomly generated before training or during the training period? How to decide the number of windows randomly sampled in each audio(sizes are varied).

Best regards, :)
Alec

0 0 Reply Share ›

Sander Dieleman Mod → Alec
8 years ago

1. the number of factors is a hyperparameter that you should tune. But if you don't have time to do that, 40 will probably work reasonably well.

3. For the work described in this post, I trained on 30 second clips, which were extracted from the middle of the songs. This was mainly for convenience (the code is a lot cleaner if all clips are the same length). You could also sample 30 second clips randomly from all songs.

0 0 Reply Share ›

John Yang
8 years ago

I have learned so much about the sincere implementation of neural network on music through your post. I am currently mimicking your method, but it seems like the link of the 'vector_exp algorithm' with which you had obtained your 40 latent factors for the class labels has been crashed, and no longer available.
Is it possible for you to fix the link?

0 0 Reply Share ›

Sander Dieleman Mod → John Yang
8 years ago

I don't remember exactly which post I was linking to, but I'm fairly confident it must have been one of these:
http://erikbern.com/2013/12...
http://erikbern.com/2013/11...

0 0 Reply Share ›

John Yang → Sander Dieleman
8 years ago edited

I have been trying to reconstruct your method myself for the past months, and I can't seem to find the values for the WMF parameters i.e. alpha and epsilon [http://yifanhu.net/PUB/cf.pdf]. I understand this project was done long time ago for you to remember all the details of your algorithm, but could you please explain some tips which range of values for alpha and epsilon should be selected during the WMF?

0 0 Reply Share ›

Sander Dieleman Mod → John Yang
8 years ago

Those are hyperparameters that need to be tuned carefully. They are extremely dependent on the size of the dataset (number of users, number of items) and its sparsity, so it's hard to give guidelines. Unfortunately I don't remember which values we used. The number of latent factors is another hyperparameter that you should also tune (although something like 40-50 will usually work fine).

0 0 Reply Share ›

John Yang → Sander Dieleman
8 years ago

Yes, I am extracting 50 features for the latent vectors as you did in the post. Thank you for your reply:)

0 0 Reply Share ›

Quennie
8 years ago

hi, Benanne,I was wondering do you still study in deep learning about music. I want to paly a game between AI and famous composer. Do you have interest?

0 0 Reply Share ›

Arjen Oudheusden

8 years ago

Very interesting indeed! Did you look at / compare results with Pandora's 'music genome' music discovery technology?

0 0 Reply Share ›

S **Sander Dieleman** Mod → Arjen Oudheusden

8 years ago

No, it's pretty hard to compare to Pandora's technology when working at a competitor ;) The music genome project represents an enormous investment and more than a decade of work by a team of many music experts, so that's not exactly reproducible within the constraints of a 3-month internship either.

0 0 Reply Share ›

Julian

8 years ago

Thanks for this blogpost, really enjoyed reading it as it connects to a problem we're trying to tackle at kollekt.fm as well (in collaboration with CrowdRec)!

0 0 Reply Share ›

F **Fender**

8 years ago

Very impressive Sander. Is Max-pooling in the first step of your convnet basically just used for dimensionality reduction? So you basically divide the input-dimensions by max-pooling over every 4 consecutive frames? I'm asking since I also used pooling more as a pre-processing step to reduce training-time (of an unsupervised architecture) under the assumption that it's not necessary to keep every FFT time-bin. Another option for spectrograms I see a lot is e.g. vectorize 4 time bins (concatenate them) but I'm not sure if this level of detail is really necessary to detect musical features or if a temporal aggregation in the beginning does the job as well (+speeds everything up).

0 0 Reply Share ›

S **Sander Dieleman** Mod → Fender

8 years ago

Note that the max-pooling step happens after the first convolution, maybe this isn't really clear in the figure (I drew the feature maps and not the layers themselves). So this step is also backpropagated through and the filters of the first convolutional layer "know" that it will happen. I could not achieve the same effect through pre-processing, that would only be possible if there was no learning happening before the pooling.

0 0 Reply Share ›

F **Fender** → Sander Dieleman

8 years ago

Ah okay, interesting! Did you nevertheless validate if this level of time-detail is necessary or if an aggregation e.g. over the 4 consecutive frames right at the beginning is feasible as well? I'm finding no good literature about the validity of pooling AFTER feature extraction only, since it also sounds feasible to use before feature learning to be honest.

0 0 Reply Share ›

S **Sander Dieleman** Mod → Fender

8 years ago

Pooling before feature extraction seems pointless: just extract a more coarse-grained spectrogram representation, that's computationally cheaper. As for what timescale is optimal for a given task, I can't answer that question in general. The only way to find out is to try out different resolutions, keeping in mind that increasing the resolution also substantially increases the amount of computation required.

0 0 Reply Share ›

vintermann

8 years ago edited

Hello,

I found this blog after being wow-ed by Discover Weekly, suspecting that they must have been using analysis of the actual sound data (and not just collaborative filtering), and googling "neural net spotify".

I notice that Discover Weekly, in addition to impressing me as much as anyone, has a tendency to offer me some smooth Jazz with my a cappella.

I'm pretty sure it's the technology you worked on that they have rolled out!

0 0 Reply Share ›

Sander Dieleman Mod → vintermann
8 years ago

It's possible :) no idea though!

One thing I've noticed is that intro/outro tracks or interludes often sneak into the DW playlists, and that would actually be something a content-based approach would be great at detecting. So I'm not 100% convinced they are using it (or maybe just not in this way).

0 0 Reply Share ›

vintermann → Sander Dieleman
8 years ago

I really love some tiny Norwegian all-women folk a cappella groups. They have less than 1000 listens, and probably most of those are me, lol. Spotify managed to find a small Swedish all-women folk a cappella group in my DW.

That's the sort of thing that I was pretty sure mere collaborative filtering couldn't do - I've certainly not seen Google or Amazon or anyone else do something comparably impressive with recommender systems.

But I'll try just asking them :) If they're still as open as they were when you wrote this post, they will probably just answer truthfully! Although with the huge edge they have on e.g. Google Play's recommender systems now, I couldn't exactly blame them for trying to keep secret the details of what they do.

0 0 Reply Share ›

Sander Dieleman Mod → vintermann
8 years ago

Turns out you were right :) http://qz.com/571007/the-ma...

0 0 Reply Share ›

Hamid
8 years ago

Hi Sander, impressive work. I am wondering if the code is publicly available.

0 0 Reply Share ›

Sander Dieleman Mod → Hamid
8 years ago

The code belongs to Spotify (I wrote it during an internship), so it's not mine to share unfortunately.

0 0 Reply Share ›

Load more comments

Read More

The paradox of diffusion distillation

Thoughts on the tension between iterative refinement as the thing that makes diffusion models work, and our continual attempts to make it _less_ iterative. Continue reading

The geometry of diffusion guidance

Published on August 28, 2023

Perspectives on diffusion
Published on July 20, 2023