Representation, Exploration, and Recommendation of Music Playlists

by

Piyush Nolastname

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved June 2019 by the
Graduate Supervisory Committee:

Sethuraman Panchanathan, Chair
Hemanth Kumar Demakethepalli Venkateswara
Heni Ben Amor

ARIZONA STATE UNIVERSITY

August 2019

ABSTRACT

Playlists have become a significant part of the music listening experience today because of the digital cloud-based services such as Spotify, Pandora, Apple Music. Owing to the meteoric rise in usage of playlists, recommending playlists is crucial to music services today. Although there has been a lot of work done in playlist prediction, the area of playlist representation hasn't received that level of attention. Over the last few years, sequence-to-sequence models, especially in the field of natural language processing have shown the effectiveness of learned embeddings in capturing the semantic characteristics of sequences. Similar concepts can be applied to music to learn fixed length representations for playlists and the learned representations can then be used for downstream tasks such as playlist comparison and recommendation.

In this thesis, the problem of learning a fixed-length representation is formulated in an unsupervised manner, using Neural Machine Translation (NMT), where playlists are interpreted as sentences and songs as words. This approach is compared with other encoding architectures and evaluated using the suite of tasks commonly used for evaluating sentence embeddings, along with a few additional tasks pertaining to music. The aim of the evaluation is to study the traits captured by the playlist embeddings such that these can be leveraged for music recommendation purposes. This work lays down the foundation for analyzing music playlists and learning the patterns that exist in the playlists in an end-to-end manner. This thesis finally concludes with a discussion on the future direction for this research and its potential impact in the domain of Music Information Retrieval.

DEDICATION

Dedicated to my sister for introducing me to the gift of music, and my parents for always being there through the crescendos and decrescendos of my life.

ACKNOWLEDGEMENTS

In completing this thesis, I have so many people to thank. First and foremost, I would like to thank Dr. Sethuraman Panchanathan, who made sure I was always supported in every possible way, and I could focus on my research. I would like to thank Dr. Hemanth Venkateswara for providing me with invaluable guidance and support for this thesis. This work would not have been possible without his mentorship. I am grateful to Dr. Troy McDaniel, who welcomed me into the CUbiC Lab and allowed me the freedom to do this work. This work would have been much harder to accomplish without his support and encouragement. The CUbiC Lab has been an incredible place to explore, learn, and play. I would also like to thank Dr. Heni Ben Amor for being supportive and willing to be on the committee to assess my work.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

In this age of cloud-based music streaming services such as Spotify, Pandora, Apple music among others, with millions of songs at fingertips, users have grown accustomed to, a) immediate attainment of their music demands, and b) an extended experience, as first mentioned by Choi *et al.* (2016). With millions of songs in the pocket and ever-improving technology, any song can be retrieved in a few seconds. However, this makes the decision to select an item with millions of items available, extremely overwhelming and difficult. Recommendation engines service this aspect of the change in user behavior by selecting items for users based on their calculated preferences. They help users find new music conveniently. Playlists handle the second aspect of the changing behavior, which is the need for an extended experience. An extended experience is achieved by sustaining the mood of the songs in a playlist. For e.g. Spotify has over two billion playlists (ref: spotify.com (2018a)) created for every kind of mood (sad, happy, angry, calm, etc.), activity (running, workout, studying, etc.), and genre (blues, rock, pop, etc.). Because of the rising popularity of playlists as the primary listening experience, a lot of attention is given to creating the best possible playlists to have maximum user engagement. As a result, the playlist recommendation has taken the center stage in the domain of Music Information Retrieval.

Unsurprisingly, over the past couple of years, the playlist recommendation task has become analogous to playlist prediction/creation rather than playlist discovery, comparison, and similarity. Tasks such as Automatic Playlist Continuation (APC) (Schedl *et al.* (2017)) are commonly tried problems in the literature (Volkovs *et al.* (2018), Yang *et al.* (2018), Ludewig *et al.* (2018)). However, not much focus has

been given to optimal playlist representation for discovery and exploration. Optimal playlist representation forms a significant part of the overall playlist recommendation pipeline, as it is an effective way to help users discover existing playlists on the platform by leveraging nearest-neighbor techniques. It also provides a way for the recommendation engine to capture the implicit details of the user's musical preferences such as the transition of the musical mood over time in a session.

This work borrows ideas from the field of Natural Language Processing (NLP) to achieve the task of representing playlists. Sequence to sequence (seq2seq) learning has had a huge impact in the domain of NLP in capturing the semantic meaning of sentences. Because of the fact that seq2seq networks can learn a representation of a sentence that not just captures the content and meaning of the words occurring in the sentence, but also the sequence of words means that it captures the context in a much more holistic manner. As a result, seq2seq learning has been very successful in tasks such as neural machine translation, where the task of the network is to learn translation between two different languages.

The aim of this work is to create an end-to-end pipeline for learning playlist embeddings which can be directly used for discovery and recommendation purposes. The relationship playlist:songs :: sentences:words is evident. Inspiration is derived from research in natural language processing to model playlist embeddings the way sentences are embedded.

1.1 Goals and Motivations

1.1.1 Why Playlist Embeddings?

Current research pertaining to playlists is in the areas of automatic playlist generation (Andric and Haus (2006), Logan (2002) Chen *et al.* (2012)), and continuation Chen *et al.* (2018) Volkovs *et al.* (2018) Yang *et al.* (2018). Multiple solutions have been proposed to address these problems, like reinforcement learning (Liebman *et al.* (2015)) and Recurrent Neural Network-based models (Choi *et al.* (2016)) for playlist generation and playlist continuation tasks. However, great success has been achieved in the field of natural language processing using the power of learned embeddings. These fixed-length embeddings are easier to use and manipulate and can be used for tasks such as machine translation and query-and-search. A case can be made for using similar methods for music playlists as well:

1. The semantic properties captured by the learned embeddings can be leveraged for providing with good-quality recommendations.

2. It can be easily integrated with other modes of information such as word2vec Mikolov *et al.* (2013) model, or content-analysis-based models proposed by Lee *et al.* (2009) or a combination of both, thus providing a multi-modal recommendation.

3. Another use case for projecting the playlist content onto an embedding space is easier browsing through the entire corpus as shown in the figure. MusicBox by Lillie (2008) is a great example of this.

4. A similar case can be made for searching playlists that do not exactly fit into one genre and hence are difficult to find for using the conventional query-and-search method. Queries such as *"find a playlist like the current playlist with 50*

songs, some blues, some rock, starting out with lower tempo songs and ending with higher tempo songs" can be answered with the proposed technique.

5. Lastly, variational sequence-to-sequence models such as the one proposed by Zhang *et al.* (2016) can be used for generating playlists from the embedding space (Bowman *et al.* (2015b)).

1.1.2 Why Unsupervised Learning?

One of the major challenges when working with playlists is the lack of labeled data. Natural language processing has many popular supervised datasets such as SNLI Bowman *et al.* (2015a), Microsoft's paraphrase detection Dolan *et al.* (2004), SICK dataset Marelli *et al.* (2014), that are used to learn sentence embeddings that capture distinct discriminative characteristics which would not be possible without the labeled data. In the absence of annotated playlist datasets, this work resorts to unsupervised learning to model playlist representations.

1.2 Contributions

This work is the first attempt at modeling and extensively analyzing compact playlist representations with inspirations from natural language processing. The contributions of the work are as follows:

1. A sequence -to-sequence learning-based model is proposed for learning a fixed-length representation for the playlists that capture not just the content details, but the information related to the sequence of the items as well.

2. A comprehensive comparison is made with other sentence-embedding models to benchmark the performance of different models for different tasks and analyze the relevance of the models for each task.

3. A suite of evaluation tasks is created to evaluate the proposed model. This set of tasks not only contains the evaluation tasks used to evaluate sentence embeddings, but new tasks pertaining to music are proposed which test the extent of the properties captures by the embeddings relevant to music.

4. A new dataset (*playlists-tracks*) is introduced for the purpose of this work. The dataset consists of 1 million playlists and 13 million tracks.

1.3 Thesis Outline

The thesis is structured in the following manner.

Chapter 2 provides an overview of playlists, a brief historical background, and discusses their significance in the domain of music information retrieval.

Chapter 3 .takes a look at the literature for music playlist representation. It discusses various models and techniques which have been used over the years to represent playlists. This chapter also surveys the literature for the neural machine translation domain. It finally ends the chapter briefly discussing how the current work differs from the previous work.

Chapter 4 discusses the background to Sequence to Sequence Learning. It begins with describing the basic building blocks of the seq2seq learning, and how those blocks are stacked together to form the sequential encoder-decoder networks. This is followed by a discussion about the state-of-the-art networks used for neural machine translation.

Chapter 5 describes in detail the models used in this work, and how they compare

with each other in terms of architectures, use cases expected to be fulfilled and required computation complexity.

Chapter 6 focuses on the evaluation tasks and metrics for this work. In this chapter are described the evaluation tasks used for evaluating sentence embeddings which are used for this work as well. Tasks pertaining to music which can be used for evaluating music playlist embeddings are introduced in this chapter as well.

Chapter 7 discusses in detail the experimental setup for this work. It begins with describing the pipeline designed to download the data and create the dataset, followed by dataset statistics, and data filtering needed for the work. It then mentions in detail the word2vec algorithm training set up which is used to get song embeddings. Finally, it ends by discussing the training set up for the embedding models used in this work.

Chapter 8 describes the experimental results for the evaluation tasks and discusses in detail the insight gained from the results.

Chapter 9 concludes the thesis by summarizing the contributions of this work. It also takes a look at the directions in which the current work can be extended.

Chapter 2

PLAYLISTS: OVERVIEW

What else is a playlist but an eloquently composed sentence.

-Anonymous

2.1 Overview

This chapter sets up the context for playlists and their significance with regards to this work. It delves into the traits that make up a playlist and how they shape our listening experience.

2.2 What Is a Playlist?

According to Fields and Lamere (2010), a playlist can be defined as *"a set of songs meant to be listened to as a group, usually with an explicit order*". A playlist isn't supposed to act as a mere container of songs, but is also supposed to have an underlying order to the songs which plays a significant part in shaping the final user experience. Three main factors led to the emergence of the playlists today:

1. Emergence of beat matching and phase alignment by disk jockeys in clubs which led to birth of concepts such as "proper song transitions".

2. Emergence of portable devices which propelled the growth of mix tapes which contained different songs by different artists in different orders.

3. Once the music went to cloud, these mix tapes began to be shared among users and listened to, hence becoming the primary mode for listening to music.

2.3 Aspects of a Good Playlist

As mentioned in section 2.2, the two main components that make up a playlist are song content (genre information) and order information. However, According to De Mooij and Verhaegh (1997), there are several other implicit factors contribute towards making of a good playlist:

1. **Songs in the playlist**: The songs comprising the playlist are the primary components of a playlist as they shape the impact in terms of mood, emotion etc.

2. **Listeners preference for the songs**: Since music is something that is very subjective, often the impact of a particular piece of music is decided by the listener's preferences. For instance, some people prefer listening to sad music when they are sad, while others prefer listening to angry music to cope with their sadness.

3. **Listeners familiarity with the songs**; People feel more comfortable listening to music with which they familiar as compared to completely unknown music. Songs of unfamiliar type/genre tend to throw off the listeners, hence worsening the listening experience.

4. **Artist / Song variety**: Variety is another strong factor in deciding the quality of a playlist. A good playlist should have a good balance of homogeneity of songs and variety in sense that a good playlist seldom has all the songs of exactly the same type.

5. **Order of songs**: The order of songs in a playlist play a huge role in shaping the listening experience. The order of songs in a playlist shape how our emotions change over time by listening to music.

6. **Song Transitions**: Song Transition is defined as the flow between song to another. Often songs which align in their starting points and ending points tend to create much more flowing experience than songs put together which so not complement each other.

7. **Serendipity**: Serendipity means applying shuffle mode to the playlist to have a unique experience. Whilst in theory it might not sound like the best idea, in real life, shuffle is a very popular feature of a playlist as it provides listener with a new perspective an experience by reordering the songs each time the playlist is listened to.

8. **Context**: Lastly, the context of a playlist is very important as it defines the purpose of a playlist. Spotify has over 2 billion playlists created for different contexts such as roadtrip, study, meditation etc. Having a context for a playlist makes it easy for the listener to select the right playlist for the right moment.

2.4 Why Is It Important?

As mentioned in Chapter 1, in this day and age of on-demand music, users have grown accustomed to, a) immediate attainment of their music demands, and b) an extended experience. Playlists play a significant part in redefining the music experience:

1. Playlists formalize the music listening experience process. They represent a very important component in the music listening experience which can be tweaked, modified and experimented with in an attempt to the music experience better.

2. Playlists handle the need for an extended experience, which is achieved by sustaining the mood of the songs in a playlist. For e.g. Spotify has over two

billion playlists (ref: spotify.com (2018a)) created for every kind of mood (sad, happy, angry, calm, etc.), activity (running, workout, studying, etc.), and genre (blues, rock, pop, etc.).

3. Playlists also enable discovery of new artists and songs. Owing to the decline of albums as the primary mode of listening experience, playlists have gained huge significance in discovery of new artists and songs.

Chapter 3

PLAYLIST REPRESENTATION: LITERATURE SURVEY

3.1 Overview

With regards to this work, natural language processing and music have important similarities. Both have sequential structure in their constituent parts - words in a sentence are akin to audio segments in a song or songs in a playlist. Both have semantic relationships between the elements of the sequence. This chapter focuses on highlighting the related work with regards to playlist representation, and natural language processing for sentence representation.

Owing to the aforementioned similarities, there have been many works in the literature which employ techniques from the field of natural language processing by translating the problem at hand to an already solved-problem in natural language processing, like McFee and Lanckriet (2011), which uses this analogy for evaluating automatically generated playlists. The playlists generation algorithms in this work are evaluated based on how *likely* it is to produce *naturally occurring playlists*, meaning the candidate algorithms are assumed to follow Markov property:

$$\mathbf{P}\left[(x_0, x_1, \ldots, x_k)\right] = \mathbf{P}\left[X = x_0\right] \prod_{i=1}^{k} \mathbf{P}\left[X_{t+1} = x_i | X_t = x_{i-1}\right] \tag{3.1}$$

The best performing algorithms would have the maximum log likelihood of the songs occurring proposed by the algorithm to occur in the playlist.

3.2 Embedding Based Approaches

3.2.1 LDA Based Models

Embedding models are often used alongside the aforementioned approach in Music Information Retrieval (MIR) to project the data onto a compact space. Introduced by Blei *et al.* (2003), Latent Dirichlet Allocation (LDA) *"is a three-level hierarchical Bayesian model, in which each item of a collection is modeled as a finite mixture over an underlying set of topics. Each topic is, in turn, modeled as an infinite mixture over an underlying set of topic probabilities"*. Zheleva *et al.* (2010) create a statistical model for capturing user taste using (LDA) . This work attempts to create a playlist for each user by selecting songs from the joint distribution of different media clusters, i.e. a distribution of songs, with each cluster representing its own unique *taste*.

3.2.2 Collaborative Filtering

Collaborative Filtering (CF) Herlocker *et al.* (1999) is a widely used method for recommendation, where the aim it to predict the choice of item selection for a user based on the of information about selection choices of all the users. This technique is able to predict how well a user would like an item that he/she has not rated. A subset of users is chosen based on their similarity to the active user and a weighted aggregate of their ratings is used to generate prediction for the current user. An example of collaborative filtering is shown in the table 3.1 One of its shortcomings is lack of consideration for order of items in the list and that there is no way to adjust the search results based on query.

	Star wars	Hoop Dreams	Contact	Titanic
Joe	5	2	5	4
John	2	5	4	3
Al	2	2		2
Nathan	3	1	5	?

Table 3.1: Collaborative Filtering can be represented as the problem of predicting missing values in a user-item matrix. This is an example of user-item rating matrix where each fixed cell represents a user's rating for an item. The prediction engine is attempting to provide Nathan a prediction for the movie Titanic.

3.2.3 Neural Network Based Approaches

Similar to CF, there have been many **neural network based works** Chen *et al.* (2016) Van den Oord *et al.* (2013) Lee *et al.* (2018) which project the corpus and the user profiles to a low dimensionality vector and then recommend items based on cosine similarity between the query and the corpus items. But this thesis focuses on works which do not have users in the loop.

1. Volkovs *et al.* (2018) create a playlist embedding, albeit with a task-tailored objective function for automatic playlist continuation. A concatenated song sequence is passed through multiple layers of gated linear unit (GLU) convolution blocks to output the playlist embedding as shown in the figure 3.1

2. Yang *et al.* (2018) use a custom autoencoder with an aim to make it easier to include multiple modalities in the input. The proposed model takes in the playlist content and the artist information, randomly setting either of these two off enforcing the model to learn both the marginals and joint information across playlists and contents.

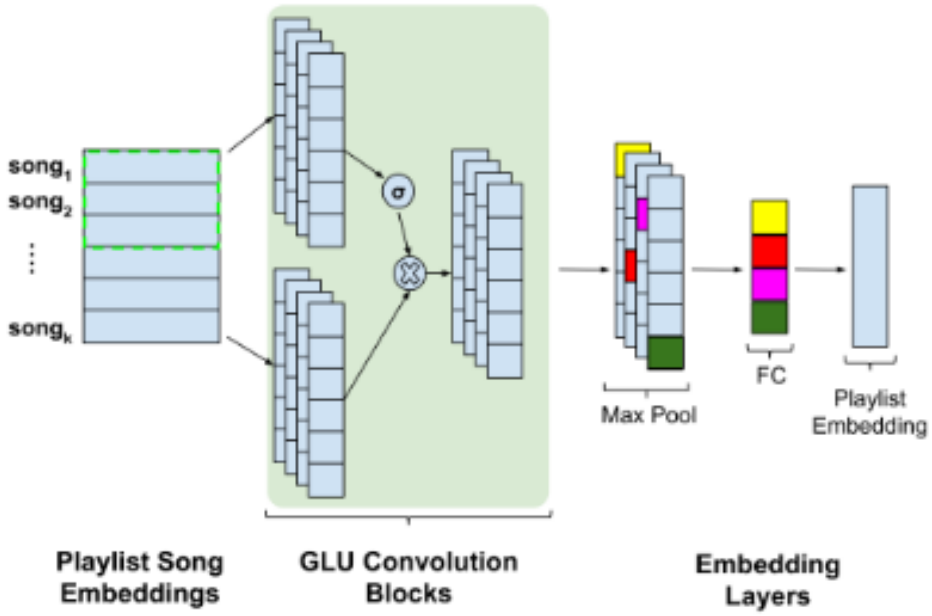

Figure 3.1: CNN model architecture for producing playlist embeddings as discussed in Volkovs *et al.* (2018)

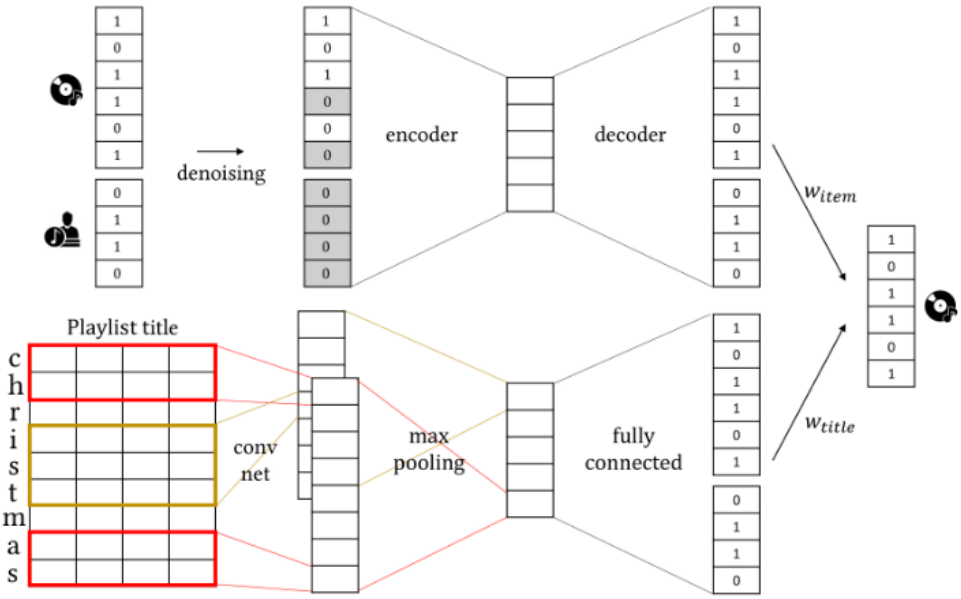

Figure 3.2: An architecture of the proposed model for Yang *et al.* (2018) consisting of (1) the content-aware autoencoder Uusing both the playlist and the artist and (2) the charCNN using the playlist title.

3.2.4 Other Non-neural Network Approaches

1. Ludewig *et al.* (2018) uses tf-idf Ramos *et al.* (2003) to create playlist embeddings.

2. Aalto (2015) uses eigenvectors from the playlist to create a representation and further uses cosine similarity between the playlists to compare playlists. However, one of the major limitations of this approach is that it doesn't take into account the order of items in the playlist.

3.3 Neural Machine Translation

This section briefly looks at the work done with regards to neural machine translation since it applies this technique to this work. Cho *et al.* (2014) used an RNN -based network to create fixed length representation of variable length source and target sequences. This work proved to be a breakthrough in the field of neural machine translation as prior approaches were much more complicated, mostly rule-based, and not end-to-end. Work done by Cho *et al.* (2014) meant that neural machine translation became end-to-end and owing to the fixed-representation size irrespective of the input sequence size, the embeddings became a lot more easier to access and manipulate. However, basic RNN units were not good enough for capturing long-term dependencies. Sutskever *et al.* (2014) improved on this by using LSTM units instead of RNN, and using separate networks for encoder and decoder to increase the model capacity. Further improvement was made on this in Bahdanau *et al.* (2014) by enabling the model to translate even longer sequences by introduction of Attention mechanism, which is a technique used by the decoder to make use of source sequence in making the output prediction.

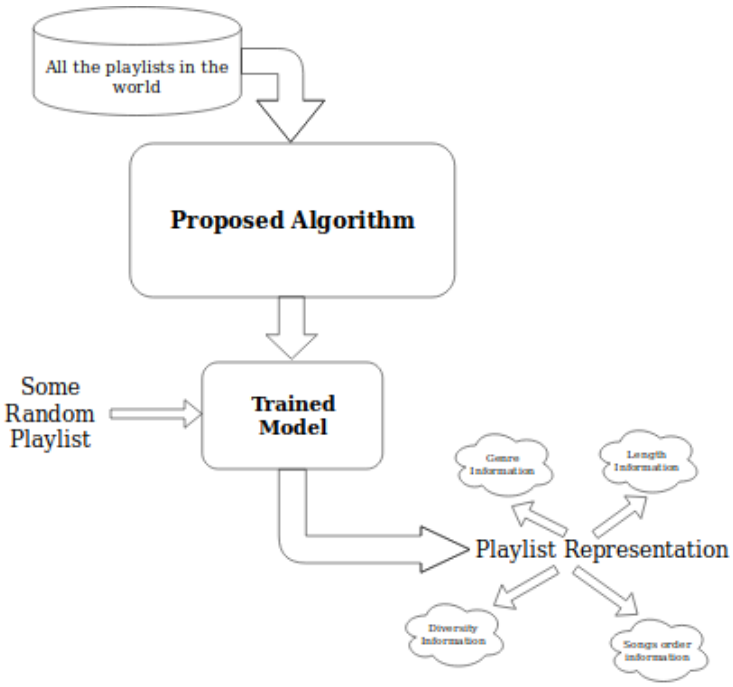

Figure 3.3: Overview of the Proposed Work Architecture

3.4 About Current Work

This thesis work differs from the previously mentioned works in sense that the aim is to leverage the playlist embeddings and use those for discovering, visualizing, and recommending existing playlists, hence making playlists as the focal point of the work, and their discovery as the prime purpose.

In addition to that, this work also focuses on creating a framework of evaluation tasks for the playlist embeddings, which being source agnostic, can be used to evaluate other models in the future as well. Many such tasks exists for evaluating embeddings in other domains such as natural language processing, but no such suite of tasks exists for evaluating playlist embeddings. This work attempts to be the first to create such a framework.

Chapter 4

SEQUENCE TO SEQUENCE LEARNING: BACKGROUND

4.1 Overview

The name sequence-sequence learning in its very core implies that the network is trained to take in sequences and output sequences. So instead of predicting single word, the network outputs the entire sentence, which could be a translation in a foreign language, could be the next predicted sentence from the corpus, of the same sentence if the network is trained as a autoencoder.

4.2 Recurrent Neural Networks

A recurrent neural network (RNN) is a component of a neural network where the current input x_t along with the previous previous hidden state h_{t-1} outputs next hidden state h_{t+1}. RNNs are called recurrent neural networks because of the recurrent nature of its operation in the sense that same operation is applied on every input of a sequence. Because of their design and operational nature, RNNs work well for sequential inputs such as text,speech etc. Figure 4.1 shows an unfolded RNN.

This unfolded network shown in Figure 4.1 can be used to predict the next word in a sequence given current word. For instance, for an input sentence "Sky is blue, grass is —", RNN can be used to predict the green. For predicting "green", the network is unfolded into an n-layer network where n is the length of the sentence, with a layer for each word. Some important points about 4.1 are:

1. U and V are the activation functions and W is a weight vector. U, V, and W are same for all the layers.

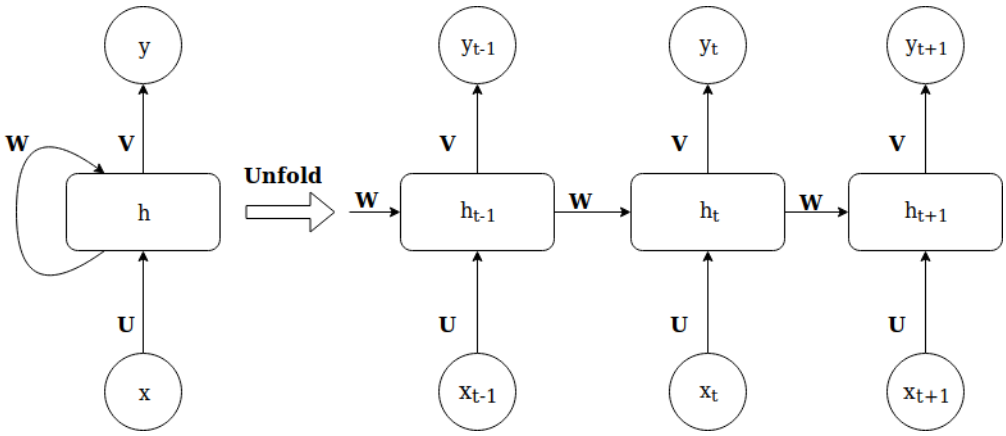

Figure 4.1: Simple Recurrent Neural Network Diagram

2. x_t is an input to the RNN at time t.

3. h_t is a hidden state, At time step t, it is the memory state of the network. h_t is calculated based on the current input and the previous hidden state. The formula to calculate h_t is given by:

$$h_{\mathrm{t}} = f\left(x_{\mathrm{t}} * \mathrm{U} + \mathrm{W}^{*} h_{\mathrm{t}-1}\right) \tag{4.1}$$

4. f is a nonlinear function such as Rectified Linear unit (ReLU), tanh. h_{t-1} is initialized to zero for the first hidden state.

5. At time t, the output state y_t is:

$$y_{\mathrm{t}} = \mathrm{Softmax}\left(\mathrm{V}^{*} h_t\right) \tag{4.2}$$

One of the major limitations of using RNN is the insufficient learning of long term dependencies with gradient descent, as first mentioned by Bengio *et al.* (1994). Also known as the vanishing gradient problem, it occurs when the weight of the neurons in the network becomes too small during the back-propagation and as a

result, the majority of the network stops training. RNN's don't seem to work well with longer length sequences, failing to capture long term sequential dependencies in the sequences. This considerably limits the application of RNN's to real life sequential problems.

4.2.1 Long Short Term Memory Unit

Long Short-Term Memory Networks, a special kind of RNNs, were introduced by Hochreiter and Schmidhuber (1997) to solve the aforementioned problem of vanishing gradient. LSTM architecture as shown in figure 4.2 consists of four main gates:

1. input gate (i)
2. forget gate (f)
3. output gate (o)
4. memory cell (c)

The newly introduced "gating mechanism", where each cell makes decisions about what to store, read and write via gates that open or close. The information is passed by the gates based on a set of weights. Equations 4.3 - 4.8 show the the operation of LSTM:

$$F_t = \sigma\left(W_F x_t + U_F h_{t-1} + b_F\right) \tag{4.3}$$

$$I_t = \sigma\left(W_I x_l + U_I h_{t-1} + b_I\right) \tag{4.4}$$

$$O_t = \sigma\left(W_O x_t + U_o h_{t-1} + b_O\right) \tag{4.5}$$

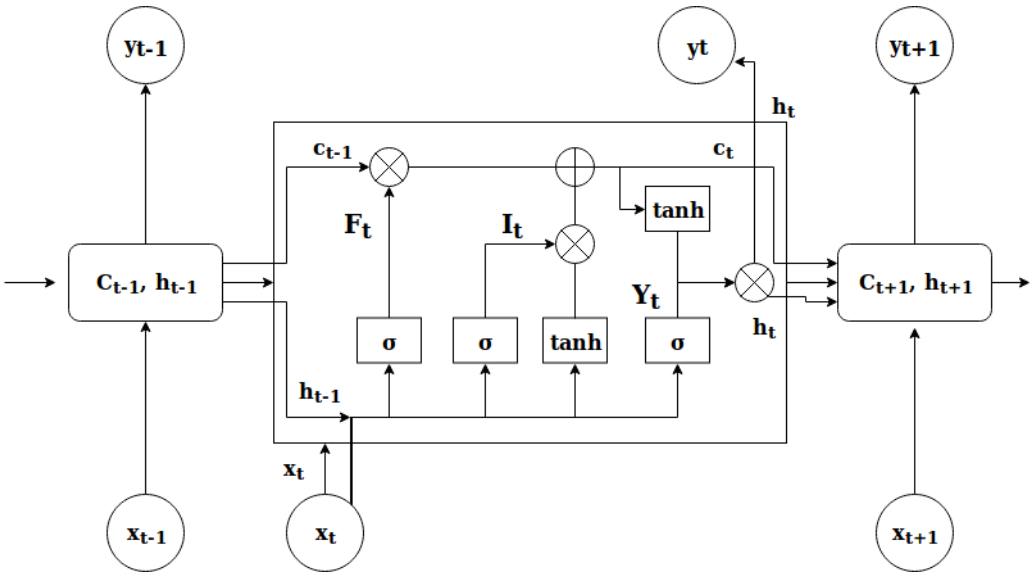

Figure 4.2: LSTM Diagram

$$c_t = F_t \odot c_{t-1} + I_I \odot \tanh\left(W_c x_t + U_c h_{I-1} + b_c\right) \tag{4.6}$$

$$h_t = O_t \odot \tanh\left(c_t\right) \tag{4.7}$$

$$o_t = f\left(W_o h_t + b_o\right) \tag{4.8}$$

Where σ is a sigmoid function, x_t is an input vector at time t, h_t is a hidden state vector at time t, W is an input to hidden weight matrix, U is a hidden to hidden weight matrix, and b_t is the bias term.

4.2.2 Gated Recurrent Unit

Gated Recurrent Unit, introduced by Chung *et al.* (2015), is a variant of LSTM . GRU doesn't have an output gate, hence it writes the contents from its memory cell to the larger net at each time-step. Owing to its comparatively simpler structure, GRU is considered to be faster to train as it it needs fewer computations to make

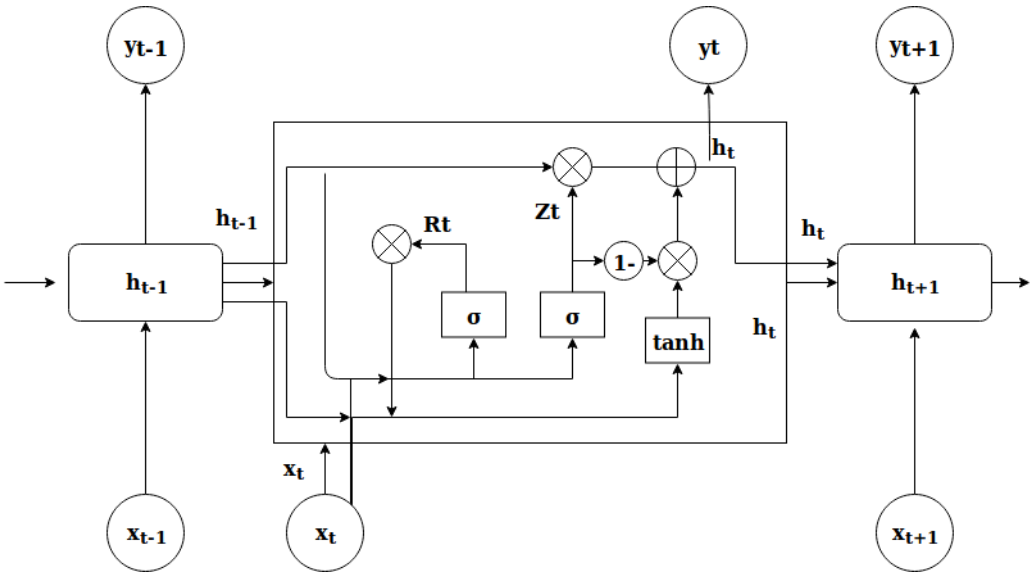

Figure 4.3: GRU Diagram

hidden state updates.

Equations 4.9 - 4.11 show the operations of a GRU unit:

$$Z_t = \sigma\left(W_z x_t + U_z h_{t-1} + b_z\right) \tag{4.9}$$

$$R_t = \sigma\left(W_R x_t + U_R h_{t-1} + b_R\right) \tag{4.10}$$

$$h_t = Z_t \odot h_{t-1} + (1 - Z_t) \odot \tanh\left(W_h x_t + U_h\left(R_t \odot h_{t-1}\right) + b_h\right) \tag{4.11}$$

Where Z_t is the update gate, R_t is the reset gate, h_t is the activation function, $\odot$ is element wise multiplication, and σ is the sigmoid function. U and W are the learned weight matrices.

4.2.3 Bidirectional RNN

A Bidirectional RNN (shown in figure 4.4) is another variant of RNN, introduced to capture more information before making a prediction at each time step. For each

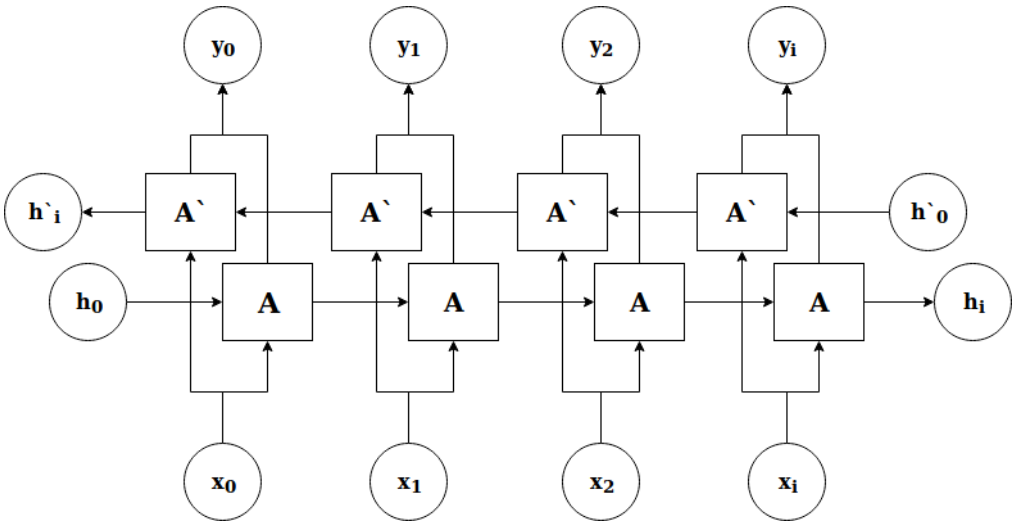

Figure 4.4: A simplified architecture of Bidirectional RNN

time step, input for all the time steps before the current time step and all the time steps ahead of the current time step are taken into consideration. In other words, it means stacking two RNNs together in which the input sequence is fed in normal time order for one network, and in reverse time order for another. The outputs of the two networks are concatenated at each time step.

4.3 Encoder-decoder Networks

In this section is described the RNN EncoderDecoder framework (shown in figure 4.5), proposed first in Cho *et al.* (2014) and later improved in Sutskever *et al.* (2014), upon which the proposed model for this work is based. Given a sequence of input vectors $x = \{x_1, x_2, x_3...x_T\}$, the encoder is an RNN that reads each symbol of the input sequentially. After reading the end of the sequence (marked by an end-of-sequence symbol), the the hidden state of the RNN is a summary c of the whole input sequence. This c is called context vector.

The decoder of this network is another RNN which generates the output sequence by predicting the next symbol y_t given the hidden state h_t. Both y_t and h_t are also

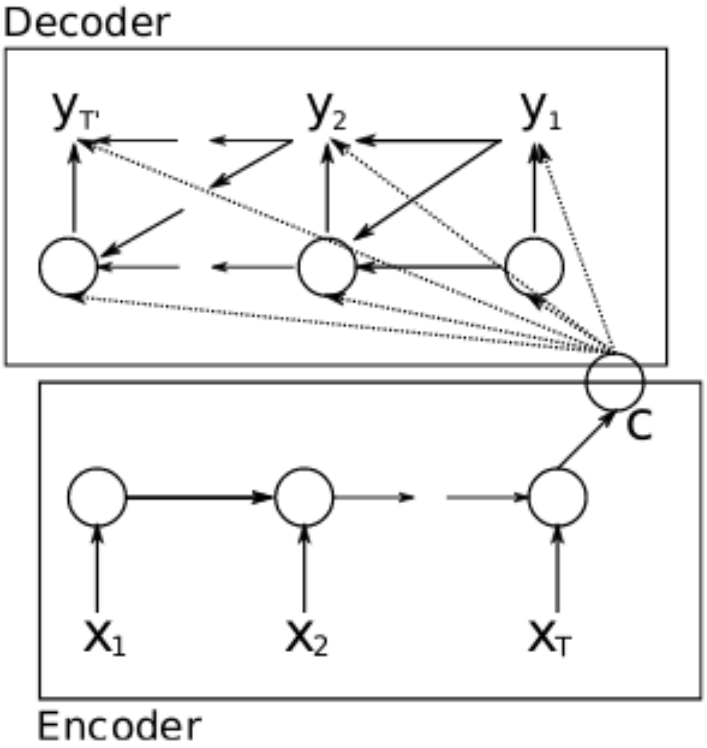

Figure 4.5: Overview diagram of encoder -decoder network. Image source: Cho *et al.* (2014)

conditioned on y_{t1} and on the context vector c of the input sequence. The hidden state of the decoder at time t is computed by:

$$\mathbf{h}_{\langle t \rangle} = f\left(\mathbf{h}_{\langle t-1 \rangle}, y_{t-1}, \mathbf{c}\right) \tag{4.12}$$

and conditional distribution of the next output symbol is given by:

$$P\left(y_t | y_{t-1}, y_{t-2}, \ldots, y_1, \mathbf{c}\right) = g\left(\mathbf{h}_{(t)}, y_{t-1}, \mathbf{c}\right) \tag{4.13}$$

where f and g are activation functions. The encoder and the decoder of the network are trained together to maximize the following conditional log likelihood:

$$\max_{\boldsymbol{\theta}} \frac{1}{N} \sum_{n=1}^{N} \log p_{\boldsymbol{\theta}}\left(\mathbf{y}_n | \mathbf{x}_n\right) \tag{4.14}$$

where θ is the set of the network parameters and (x_n, y_n) is the (input sequence, output sequence) pair from the training set.

4.3.1 Attention Mechanism

One of the significant limitations of network described in section 4.3 is that the network is not able to capture long term dependencies for relatively longer sequences Bengio *et al.* (1994), because the entire onus of capturing the whole meaning of the input sequence lies on context vector c, which is unable to do so. This problem is partially mitigated in Sutskever *et al.* (2014) by using LSTM Sundermeyer *et al.* (2012) units instead of vanilla RNN units and feeding the input sequence in the reversed order to solve for lack of long-term dependency capture.

Bahdanau *et al.* (2014) introduced the attention mechanism to solve this problem which involved focusing on a specific portion of the input sequence when predicting the output at a particular time step. The attention mechanism ensures the encoder doesn't have to encode all the information into a single context vector. In this setting, the context vector c is calculated using weighted sum of hidden states h_{j}:

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j \tag{4.15}$$

where α_{ij} is calculated as follows:

$$\alpha_{ij} = \frac{\exp\left(e_{ij}\right)}{\sum_{k=1}^{T_x} \exp\left(e_{ik}\right)} \tag{4.16}$$

where $e_{ij} = a\left(s_{i-1}, h_j\right)$ and $s_{\mathrm{i\text{-}1}}$ is the decoder state at time step $i-1$ and h_j is the encoder state at time step j. $a(.)$ is the alignment model which scores how well the output at time step i aligns with the input at time step j. The alignment model a is a shallow feed forward neural network which is trained along with the rest of the

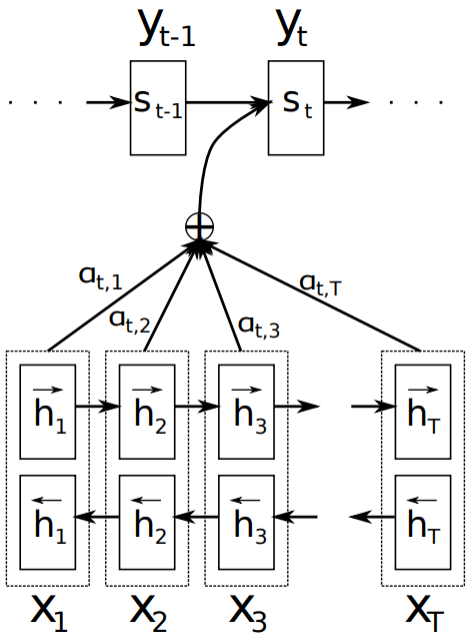

Figure 4.6: Diagram showing the Attention module. Image source: Bahdanau *et al.* (2014)

network.

Chapter 5

EMBEDDING MODELS USED IN THIS WORK

5.1 Introduction

There are broadly two ways in which a representation for a sentence can be derived from its constituting components:

1. Just considering the content of the components and discarding the order in which the components appear.

2. Taking into account the content, as well as the order of the components.

Deriving an embedding from just the components is much simpler to achieve in terms of required procedural and computation complexity, and yet surprisingly good results can be obtained from such methods as shown in Adi *et al.* (2016). Learning an embedding by taking into account not just the content, but the order of the components as well is computationally complex, but yields very good results in cases where the sequence really matters as shown in Adi *et al.* (2016) and Conneau *et al.* (2018)

5.2 Models

For this work, both these types of models are compared with along with some of their variations:

1. **Bag of words Model (BOW):** Bag of words is a simple, yet highly effective technique of feature extraction in the fields of information retrieval and natural

language processing, and even computer vision as shown by Fei-Fei (2007). It derives its simplicity by throwing away the information related to the sequence of the sub-components and only making sense from the multiplicity of the composing components. Due to these reasons, It is especially good at capturing the basic characteristics of the entities which can be decomposed into smaller components such as sentences, documents (composed of words), images (composed of histograms) etc.

A BOW representation can be calculated using arithmetic mean of components representations. For this work, given$\{p_1, p_2, \ldots, p_n\}$ a collection of playlists with each sentence being a collection of songs $\{s_1, s_2, \ldots, s_m\}$, sentence embedding is calculated using a simple arithmetic mean of its constituent word embeddings. The effectiveness of this approach coupled with the simplicity of computation makes it a very competitive baseline for comparison.

2. **Smooth Inverse Frequency Weighted Scheme based Model (SIF):** This technique uses a weighted averaging scheme to get the playlist embedding vectors followed by their modification using singular-value decomposition (SVD). Inspired by random walk model for describing corpus generation and the discourse vector for representing the current context of the text in Arora *et al.* (2016), this work models improves on the aforementioned modeling by taking into account:

   (a) Some words can occur out of context.

   (b) Some frequent words such as "the", "etc" appear often regardless of the current topic being discussed.

Based on these considerations, two modifications are made:

(a) An additive term $\alpha p(w)$ in the log-linear model is introduced, where $p(w)$ is the unigram probability of the word and α is a scalar.

(b) A common discourse vector c is introduced which serves as a correction term for the most frequent discourse.

Concretely, given the discourse vector c_s, the probability of a word w is emitted in the sentences is modeled by:

$$\Pr\left[w \text{ emitted in sentence } s|c_s\right] = \alpha p(w) + (1-\alpha)\frac{\exp\left(\langle \tilde{c}_s, v_w \rangle\right)}{Z_{\tilde{c}_s}} \qquad \text{where } \tilde{c}_s = \beta c_0 + (1-\beta)c_s, c_0 \perp c_s \tag{5.1}$$

where α and β are scalar hyperparameters, and $Z_{\tilde{c}_s} = \sum_{w \in \mathcal{V}} \exp\left(\langle \tilde{c}_s, v_w \rangle\right)$

The maximum likelihood estimator for c_s is:

$$\arg\max \sum_{w \in s} f_w\left(\tilde{c}_s\right) \propto \sum_{w \in s} \frac{a}{p(w)+a} v_w, \text{ where } a = \frac{1-\alpha}{\alpha Z} \tag{5.2}$$

That is, the maximum likelihood estimator is approximately a weighted average of the vectors of the words in the sentence.

To estimate c_s, the direction of c_0 is estimated by computing the first principal component of c_s's set of sentences. In other words, the final sentence embedding is obtained by subtracting the projection of c_s's to the first principal component.

This method of generating sentence embeddings proves to be a stronger baseline compared to the traditional averaging.

3. **NMT-based Models:** This model is based on the the RNN-Encoder-Decoder framework discussed in Section 4.1. Given a sequence of n words $x_{t=1,\ldots,n}$, an

RNN-encoder generates hidden states $h_{t=1,\ldots,n}$. The attention mechanism takes in these hidden states along with the decoder states and outputs context vectors $c_{\mathrm{t=1,\ldots,n}}$, which are then fed to the decoder to predict the output $y_{\mathrm{t=1,\ldots,n}}$.

(a) **Base Unidirectional Encoder:** This is considered as the base NMT model for this work. For this model, a unidirectional RNN, and global attention model Luong *et al.* (2015) is used where score $\left(h_t, \overline{h}_s\right) = h_t^T W_a \overline{h}_s$.

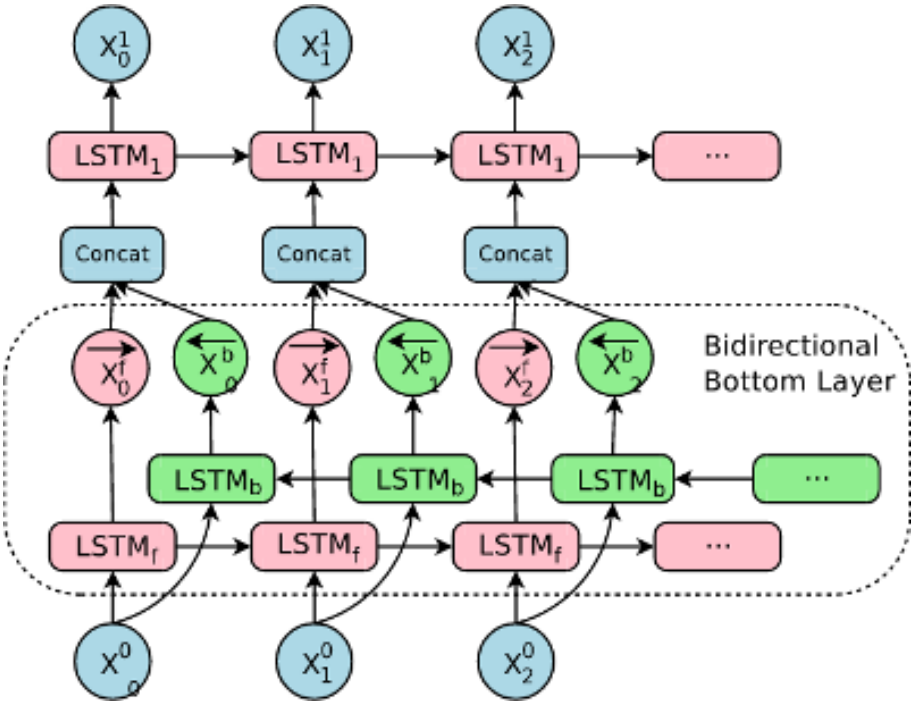

Figure 5.1: Google NMT Encoder

(b) **Google NMT Encoder:** This model was introduced in Wu *et al.* (2016). The first layer of encoder in this model is bidirectional, while the rest of the layers are unidirectional. This model also uses residual connections introduced by Szegedy *et al.* (2017) to improve the gradient flow in the network, and the attention model used is the one mentioned in Luong *et al.* (2015).

Residual Connections Training very deep networks is susceptible to

running in problems such as vanishing and exploding gradients as shown in Pascanu *et al.* (2012) and Hochreiter *et al.* (2001). A way to mitigate the this issue is residual connections, which are simply connections between a layer and layers after the next. Residual connections improve the gradient flow in the backward pass, which allows very deep encoder and decoder network to be trained effectively.

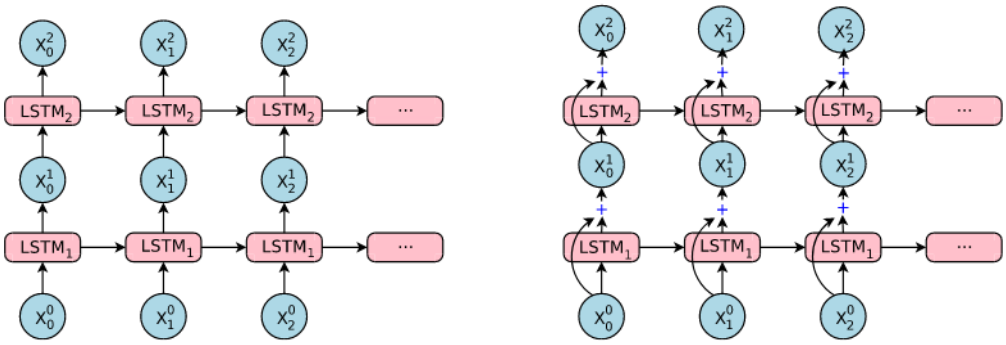

Figure 5.2: Image showing a comparison of simple stacked connections (on the left) with Residual connections (on the right). Image source : Wu *et al.* (2016)

Let $LSTM_i$ and $LSTM_{i+1}$ be the i^{th} and $(i+1)^{th}$ layer, whose parameters are W_i and W_{i+1} respectively. At the t^{th} time step, for the stacked LSTM with residual connections:

$$\begin{aligned} \mathbf{c}_t^i, \mathbf{m}_t^i &= \text{LSTM}_i\left(\mathbf{c}_{t-1}^i, \mathbf{m}_{t-1}^i, \mathbf{x}_t^{i-1}; \mathbf{W}^i\right) \\ \mathbf{x}_t^i &= \mathbf{m}_t^i + \mathbf{x}_t^{i-1} \\ \mathbf{c}_t^{i+1}, \mathbf{m}_t^{i+1} &= \text{LSTM}_{i+1}\left(\mathbf{c}_{t-1}^{i+1}, \mathbf{m}_{t-1}^{i+1}, \mathbf{x}_t^i; \mathbf{W}^{i+1}\right) \end{aligned} \tag{5.3}$$

where x_t^i is the input to $LSTM_i$ at time step t, and m_t^i and c_t^i are the hidden states and memory states of $LSTM_i$ at time step t, respectively.

(c) **Deep Bidirectional RNN Encoder:** For bidirectional encoder, hidden state $\mathrm{h_t}$ where $\mathrm{t} \in \{1, ..n\}$ is the concatenation of a forward RNN and a backward RNN that read the sequences in two opposite directions. Its

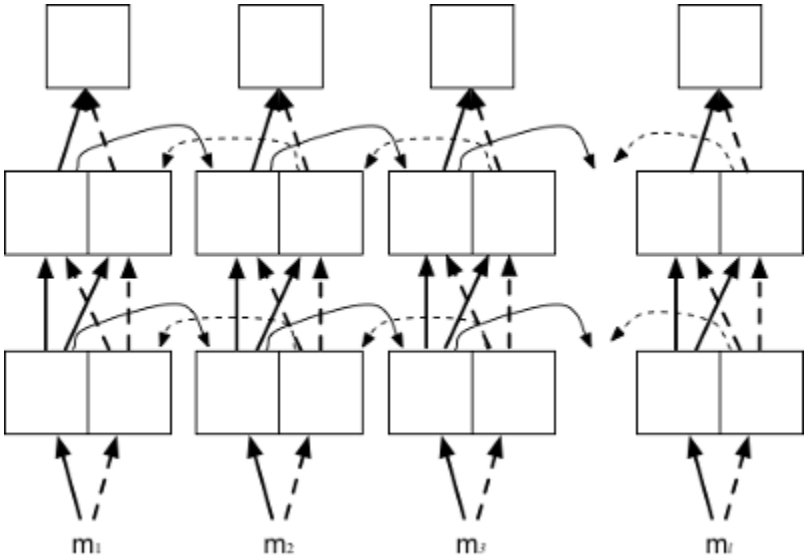

Figure 5.3: DBRNN Encoder

called a deep bidirectional encoder when multiple layers of bidirectional layers are stacked together, making it a deep network.

5.3 Training Tasks: Problem Statement Formulation

For the current work, a Neural Machine Translation System (a sequence to sequence network with attention module) is trained as an autoencoder where the target sequence is the same as the source sequence, and the goal of the model is to reconstruct the input sequence. The corpus for the current work consists of playlists with each playlist consisting of song ids. The context vector c would then represent the playlist embedding generated by the encoder.

Chapter 6

EVALUATION: TASKS AND METRICS

6.1 Evaluation Tasks

The defining characteristics of any playlist are **type, length and order**. Type signifying the genre information of the playlist, length, the number of songs in the playlist and order capturing implicit characteristics of how the songs are ordered in the playlist. Hence, a good playlist embedding should encode information about the genre of the songs it contains, the order of songs, length of playlist and songs themselves. The experiments for this work are divided into four parts; the first part examines the effectiveness of the embeddings learned in encoding the genre information of the songs constituting the playlist. The second part examines the quality of the playlist length information encoded in the learned embedding. The third part deals with evaluating the extent of the song information encoded in the learned embedding, and the fourth part deals with evaluating the songs-order information.

6.1.1 Genre Related Tasks

6.1.1.1 Genre Prediction (Multi-Class Prediction)

This task **(GPred)** measures to what extent a playlist embedding encodes the genre-related information of the songs it contains. Given a playlist embedding, the goal of the classifier is to predict the correct genre for the playlist. The task is formulated as multi-class classification, with nine output classes. The ground truth labels (genres) are assigned to only those playlists for which all of the songs having

a genre[1] and more than 70% of the songs agree on the genre, which results in 35527 playlists for evaluation. Training samples are weighted-by-class as the dataset is skewed with the majority class (electronic) having 18138 samples while the minority class (classical) just having 75 samples.

6.1.1.2 Genre Diversity Prediction

This task **(GDPred)** measures the extent to which the playlist embedding captures the sense of the homogeneity/diversity of the songs (with regards to their genre) constituting it. Given a playlist embedding, the goal of the classifier is to predict the number of genres spanned by the songs in that playlist. The task is also formulated as multi-class classification, with 3 output classes being low diversity (0-3 genres), medium diversity (3-6 genres) and high diversity (6-9 genres).

6.1.1.3 Genre Switch Prediction

A genre switch is defined here as change in genre going from one song to another. A homogeneous playlist would have fewer number of such genre shifts than a more diverse playlist. The aim of this task **(GSPred)** is to predict the number of all the genres switches given a playlist embedding. For this task, the absolute number of switches for a playlist is normalized by dividing it by the length of the playlist such that the final label lies between 0 and 1. The task is then formulated as multi-class classification, with the five output classes being low switch playlist (0-0.34), mid-switch playlist (0.34-0.67) and high-switch playlist (0.76-1.00)

[1]Only those songs are assigned genres for which song embeddings are available

6.1.2 Playlist Length Prediction Task

This task **(PLen)** measures to what extent the playlist embedding encodes its length. Given a playlist embedding, the goal of the classifier is to predict the length (number of songs) in the original playlist. Following Adi *et al.* (2016), the task is formulated as multi-class classification, with ten output classes (spanning the range [30-250]) corresponding to equal binned lengths of size 20. Training samples are class-weighted as the dataset is unbalanced, with a majority class (lengths 30-50 songs) having 78015 samples and the minority class (230-250 songs) having just 1098 samples.

6.1.3 Song Content Task

The song content **(SC)** closely follows the Word Content (WC) task described by Conneau *et al.* (2018) in testing whether it is possible to recover information about the original words in the sentence from its embedding. 750 mid-frequency songs (the middle 750 songs in a list of songs sorted by their occurrence count) are picked from the corpus vocabulary, and sample equal numbers of playlists that contain one and only one of these songs. The experiment is formulated as a 750-way classification problem where the aim of the classifier is to predict which of the 750 songs does a playlist contain, given the playlist embedding.

6.1.4 Sentence Semantic Measurement Task

The sentence semantic measurement task is an effective sentence embedding evaluation task. The goal of this task is to evaluate the embeddings based on how well they capture the semantic meaning of the sentence, and if the relationship between two sentences, given just their embeddings, can be inferred or not. The way this

task is set up is given two sentences, the goal is to classify whether the pair have a entailment, contradictory or paraphrase relationship. For the current work, this task is set up by creating a dataset of playlists sampled from the dataset. The paired playlists are chosen in the following manner:

1. A shuffled portion of the songs of the original playlist as the *entailment* playlist.
2. A completely different playlist with non overlapping songs as the *contradictory playlist*

6.1.5 Song Order Tasks

6.1.5.1 Bigram Shift Task

Text sentences are governed by language grammatical rules. These rules govern the existence of certain bi-grams in the language (e.g. *will do*, *will go*) as well as the lack of existence of others (eg. *try will*). In the field of natural language processing, the Bigram Shift **(BShift)** experiment, introduced in Adi *et al.* (2016), is a very good way to measure the extent to which word-order information is encoded in the sentence embeddings. This evaluation task is formulated as a binary classification problem, where the aim of the classifier is to distinguish between original sentences from the corpus and sentences where two adjacent words have been inverted. And even the simplistic BOW models are able to perform well on this task. Adi *et al.* (2016).
However, playlists do not have apparent grammatical rules that govern the order of songs. One of the most commonly used features on music platforms, *Shuffle* validates this claim. Hence, the Bigram shift experiment should not work for music playlists the way it works for sentences, as inverting a few songs does not make the playlist embedding very much different from the original playlist embedding. To validate the hypothesis that inverting two songs in a playlist cannot be compared to inverting two

words in sentence, bigram-shift experiment is set up.

For the bigram-shift experiment, a dataset with playlists where each playlist has a corresponding pair having adjacent songs reversed, is created. To create the dataset for this experiment, 55265 playlists are selected whose length lie in the range [50-100]. For each of these playlists, an additional playlist is created with two adjacent songs inverted. This results in a balanced dataset of size 110530.

6.1.5.2 Permute Classification Task

Through this task, this work aims to answer the questions: Can the proposed embedding models capture song order, and if they can, to what extent? This task is split into two sub tasks: **i) Shuffle Task**, and **ii) Reversal** task. To create the dataset for this experiment, a list of 38168 playlists is selected whose lengths lie in the range [50-100]. In the **Shuffle** task, for each playlist in this task-specific dataset, a fraction of all the songs in that playlist is randomly shuffled. The playlists are shuffled in two ways: a) by selecting a random block in the playlist and shuffling just that block (**shuffle type-1**), and b) Randomly selecting songs from the playlist and shuffling them (**shuffle type-2**). A binary classifier is trained where the aim of the classifier is to distinguish between an original and a permuted playlist. The **Reversal** task is similar to the Shuffle task except that the randomly selected sub-sequence of songs are reversed instead of shuffled. Both these tasks are further extended with a slight modification that from the original dataset, playlists which are inverted are not included in the dataset and vice-versa.

Chapter 7

EXPERIMENTAL SETUP

In this chapter, experimental setup details for this work are specified, starting from dataset creation, filtering, and data annotation. This is followed by the training details where the training setup and model configurations used in the experiments for this work are explained.

7.1 Data

7.1.1 Data Source

The dataset for this work has been assembled by downloading publicly available playlists from Spotify using the Spotify developer API spotify.com (2018b). 1 million playlists were downloaded from Spotify, consisting of both user-created playlists as well as Spotify-curated ones. The user created playlists are the ones created by Spotify users, whereas the Spotify curated playlists are the ones created by music editors at Spotify to create the best possible public playlists based on the listening preferences of the users. The Spotify curated playlists are famously known for their accurate contextual mapping, where the context is usually a genre (Blues, EDM, Rock etc.), activity (car rides, workouts etc) and moods (peaceful, high energy etc). To create the dataset, list of 2680 genres was extracted from everynoise.com (2012) as the search terms for getting 104935 playlists from Spotify. Data download workflow is shown in the figure 7.1 and data details are mentioned the Table 7.1.

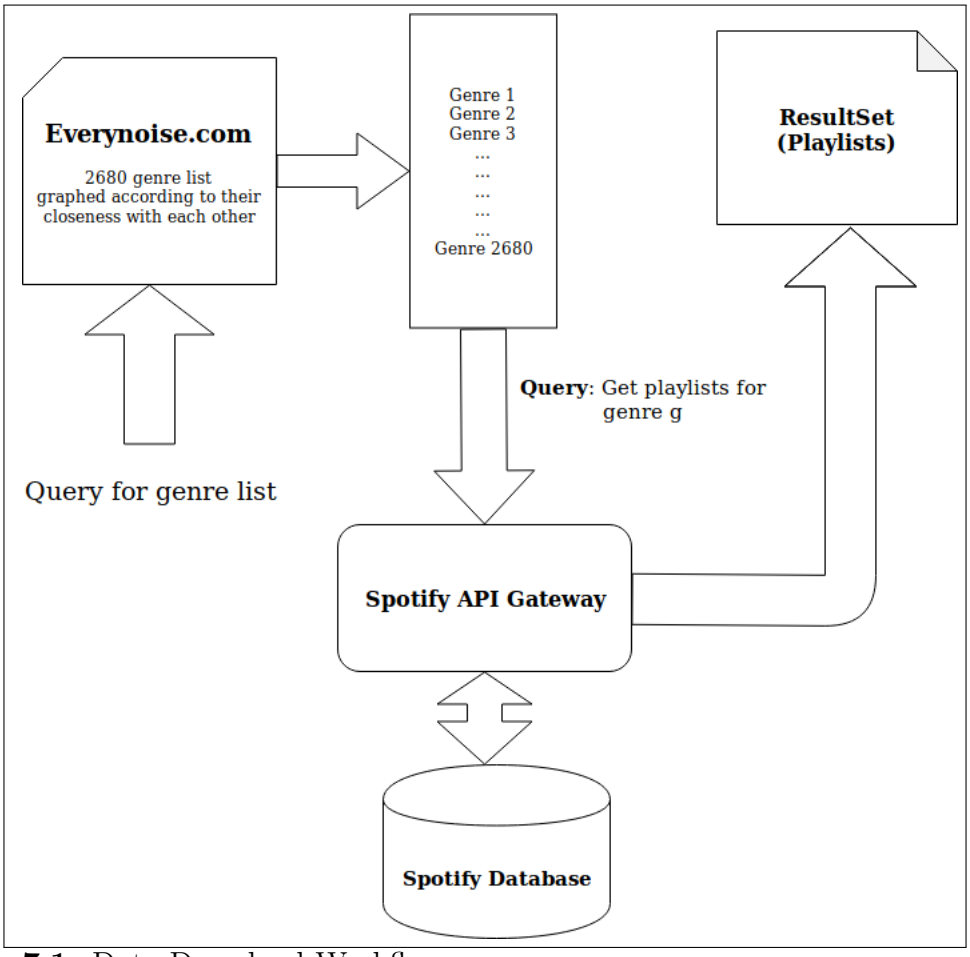

Figure 7.1: Data Download Workflow

Downloaded playlists details			
Playlists	**Artists**	**Albums**	**Songs**
1054935	1290663	2918452	13282009

Table 7.1: Downloaded Spotify Data Details

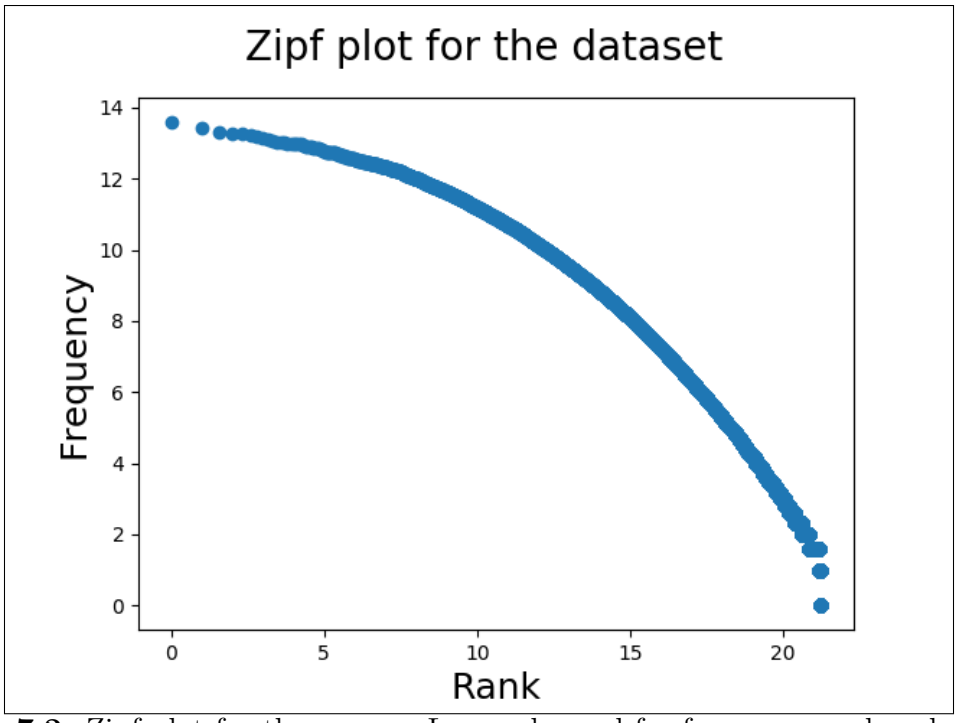

Figure 7.2: Zipf plot for the corpus. Log scale used for frequency and rank

7.1.2 Data Statistics

7.1.2.1 Zipf's Law

According to Wikipedia, Zipf's law states that given a large sample of words, the frequency of any word is inversely proportional to its rank in the frequency table. Zipf plot of the corpus is shown in Figure 7.2

7.1.2.2 Playlists Length Statistics

More than half of the playlists (401880) are of length less than 50 and another 246427 (33%) of the playlists have lengths in the range [50-150]. Statistics related to the length of the playlists are given in Table 7.2

Playlist Corpus Length statistics				
Mean	**Std**	**Median**	**Min.**	**Max.**
83.3	133	45	10	5000

Table 7.2: Corpus Length Statistics

7.1.2.3 Genre Homogeneity/Diversity Statistics

Out of 745543 playlists, 49164 playlists have all of their songs genre annotated. Out of 49164 playlists, 24422 (49%) playlists have less than or equal to 3 genres in total across all of their songs, 23162 (47%) playlists have less than of equal to 6 genres in total, and 1580 (3%) playlists have more than 6 genres.

7.1.3 Data Filtering

As a part of cleaning up the data before training, following De Boom *et al.* (2018), the less frequent and less relevant items from the dataset are discarded. The reason this step is important is because the majority of the playlists (1054200) in the corpus are user-created playlists, which are susceptible to being extremely noisy in terms of the characteristics such as playlist-length, homogeneity etc in the absence of any constraints put on the playlist creation for users. [rare words removal reason]. Having discussed the reasons for data filtering for this work, here are steps taken to filter the data:

1. Tracks occurring in less than 3 playlists are discarded.
2. Playlists with the less than 30% of tracks left after this are also removed.
3. All duplicate tracks from playlists are removed.
4. Finally, only playlists with lengths in the range [10-5000] are retained and the rest are discarded.

This leaves a total of 745543 playlists and 2470756 unique tracks.

7.2 Data Labeling: Genre Assignment

The songs in the dataset downloaded from Spotify do not have genre labels. Therefore, a word-2-vec model is trained on the corpus with playlist lengths restricted to a range [30-3000] and the minimum frequency threshold of the songs set to a value of 5. The resulting song embeddings are then clustered into 200[1] clusters, and each cluster is manually assigned one of the 9 genres:

1. Rock
2. Metal
3. Blues
4. Country
5. Classical
6. Electronic
7. Hip Hop
8. Reggae
9. Latin

To validate this approach, a classifier is trained on the annotated dataset consisting of the manually annotated song embeddings.The network used for evaluation is a 3

[1]This number was chosen with an aim to get maximum feasible localized clusters (and hence not missing less popular genres among bigger clusters) while keeping the number within a limit which was feasible for annotating the data.

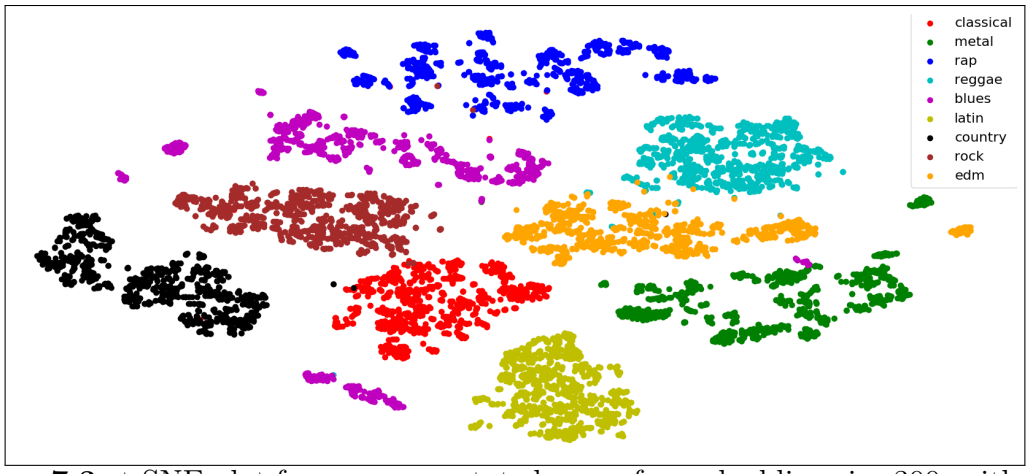

Figure 7.3: t-SNE plot for genre-annotated songs for embedding size 300, with 1000 sampled songs for each genre

layer neural network with input dimension of 300 and output dimension of 9 (which is equal to the number of genres). An accuracy of 94% is achieved on the test set. A t-SNE plot of the annotated songs is generated as well to further validate proposed approach as shown in Figure 7.3.

7.2.1 Word-2-vec: Set up and Training

A word2vec model is trained on the corpus to get the song embeddings, using the Gensim (Řehůřek and Sojka (2010)) Implementation. Skipgram (McCormick (2016)) algorithm with negative sampling value set to 5 and window size of 5 is used for the algorithm configuration. Minimum threshold for the occurrence of words is set to 5. The word vectors are trained for sizes k $\in$ {300, 500, 750, 1000}.

7.3 Training

1. **BOW Model:** Embeddings are calculated for only those playlists for which embeddings for all constituent songs are available. This leaves a total of 339998 playlists.

2. **SIF Model:** For the base configuration of the model, the value of the SIF parameter a is set to e^{-3} where the weight given to each word in the corpus is $a/(a + p(w))$ and a is the controlled parameter. Different values for a ranging from e^{-3} and e^{-5} are experimented with.

3. **NMT-based Models:** All of the NMT encoders use 3 layer network with an hidden state size is controlled for and k $\in \{500, 750, 1000\}$. LSTM and GRU units were experimented with and the hidden state size was varied from 500 to 1000. Adam and SGD are experimented with [2] optimizers. The maximum gradient norm is to 1 to prevent exploding gradients.

[2]SGD performed generally worse than Adam, hence the details for SGD are not included

Chapter 8

EVALUATION RESULTS

In this section, a detailed description of the experimental results along with their analysis and insights is provided. For each of the discussed tests genre, length, and content the performance of different embedding models across multiple embedding lengths is investigated. [1]The results for the evaluation tasks are presented in the table 8.1.

[1]Experiment details for investigating the order-related properties captured by the embeddings are discussed in the Chapter **??**

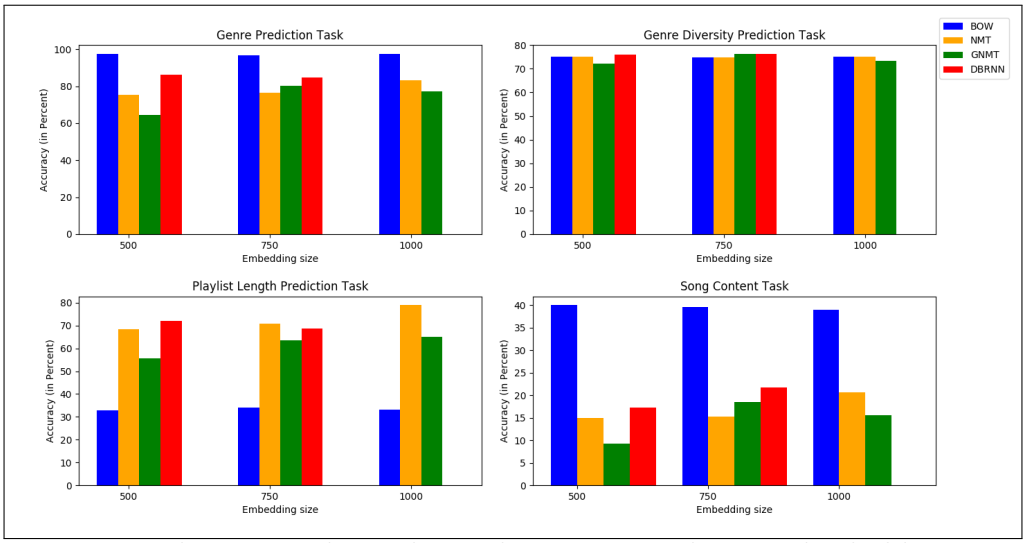

Figure 8.1: Evaluation task results with respect to the encoder hidden state size. Missing bars are for cases where it was not possible to train an embedding of that size due to memory constraints.

Evaluation Tasks					
	GPred	**GDPred**	**GSPred**	**PLen**	**SC**
BOW Model	96.8	79	82.5	34	39.6
SIF Model	97.5	80.05	84	33.4	44.3
NMT Model	76.6	75.8	80.8	70.7	15.3
GNMT Model	80.1	74.5	80.9	63.5	18.5
DBRNN Model	84.9	76.2	82.3	71.9	21.7

Table 8.1: Evaluation Task Results for the Embedding Models

8.1 Genre Related Tasks

8.1.1 Genre Prediction Task

For the **GPred** task, BoW models outperform the NMT-based models. This can be attributed to the reasoning that the playlist vector created by averaging the constituent songs is embedded in the space of the songs as their centroid. Since the genre of the playlist is the genre of its songs, the BOW outperforms the NMT at genre prediction. For the NMT-based models, the performance appears to improve with increasing RNN hidden state size in the encoder, as larger embedding sizes generally have more space to encode sequence information.

8.1.2 Genre Diversity Task

In the **GDPred task** as well BOW-based models perform better than the NMT-based models perform the best with the base SIF model achieving 80% accuracy while NMT-based models achieve an accuracy of 73%-75%.

8.1.3 Genre Switch Prediction Task

For the **GSPred** task, all models surprisingly perform quite well with an average accuracy of 80% achieved. However, even if by a comparatively smaller margin, the BoW models perform outperform the NMT-based models. Both the BoW model and the SIF model achieve an accuracy of 82.5% and 84% respectively, while the NMT models have an accuracy of 80% with the only exception of DBRNN model getting 82.3% accuracy.

Overall, for all the genre-based experiments, BoW models perform better than the NMT models.

8.2 Length Prediction Task

For the **PLen task**, the NMT-based models perform quite well, achieving 72% accuracy while the BoW models perform quite poorly managing just 35% accuracy. The performance of the NMT-based models doesn't come as any surprise as it has been widely studied that sequence to sequence based models are able to capture such characteristics of the sentences. Poor performance of the BoW models however is indeed surprising as BoW models have been shown to perform comparatively better on this task Adi *et al.* (2016) Conneau *et al.* (2018).

8.3 Song Content Task

For the **SC** task, the NMT Models performed poorly compared to the BoW models. However, the obtained results closely match the results for the unsupervised models for the same task in Conneau *et al.* (2018). The authors cite the inability of the model to capture the content-based information due to the complexity of the way

the information is encoded by the model.

8.4 Sentence Semantic Measurement Task

For the Sentence Semantic Measurement Task [2], all the models were very easily able to correctly tag the paired playlists, resulting in 99% accuracy. One of the reasons this experiment didn't work in the way it was set up was that since the entailed playlists are always going to be shorter in length than the original playlists, the model could just focus on the length for making the prediction while completely ignoring the content of the playlists. This experiment especially points towards the lack of supervised datasets in music.

8.5 Song Order Tasks

8.5.1 Bigram Shift Task

A binary classifier is trained over the set of playlists with an accuracy of 49% for all the embedding models, meaning the classifier is unable to distinguish the original playlist from bigram-inverted ones for all the encoders. Hence, the results for the current task show that another experiment is needed to assess the effectiveness of the embedding models in capturing the song-order in the playlist.

8.5.2 Permute Classification Task

As seen in Figure 8.2, the base NMT-based model is able to distinguish correctly the permuted playlists from the original playlists as the proportion of the permutation is increased. Even for the extension of the tasks where complementary playlist pairs are not added to the dataset, the classifier can still distinguish between the

[2] results not included in the results table

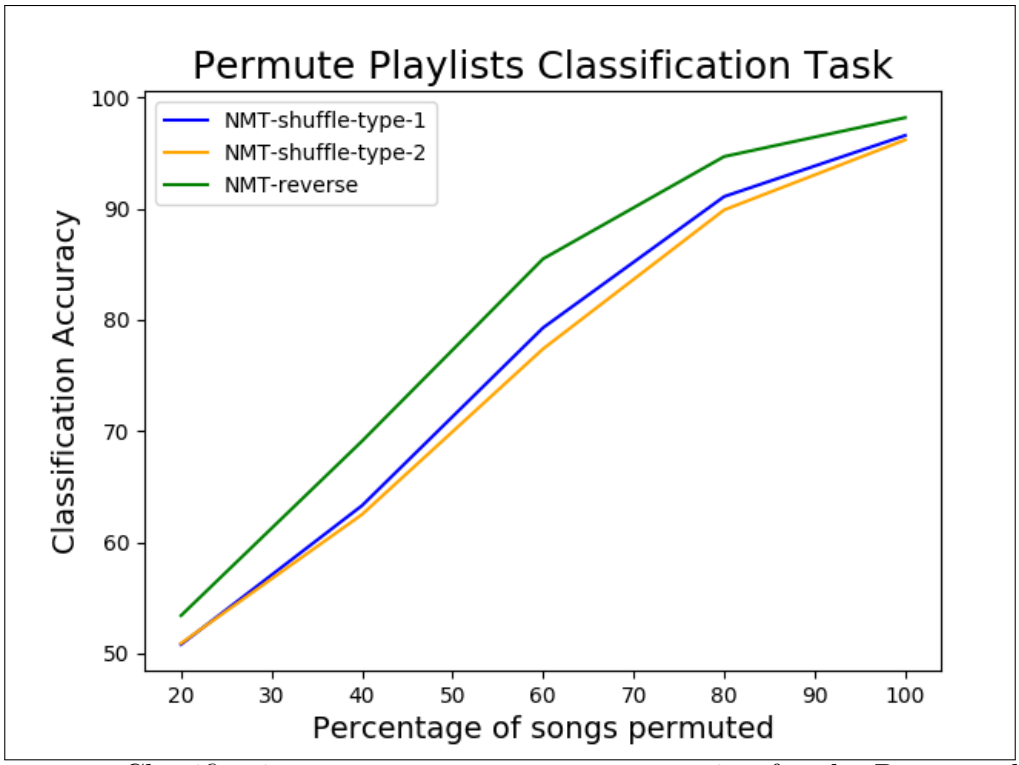

Figure 8.2: Classification accuracy vs permute proportion for the Permute-shuffle task

original and the permuted playlists. On the other hand, BoW models cannot distinguish between the original and permuted playlists, making seq2seq models better for capturing the song-order in the playlist.

Chapter 9

CONCLUSION AND FUTURE WORK

This work presents a sequence-to-sequence based approach for learning playlist embeddings, which can be used for tasks such as playlist discovery, comparison and recommendation. First the problem of learning a playlist-embedding is defined and followed by how it can be formulated as a seq2seq-based problem. The proposed model is then compared with two BoW models on a number of tasks chosen from the field of natural language processing (like sentence length prediction, bi-gram shift experiment etc) as well as music (genre prediction and genre-diversity prediction). This work shows that the proposed approach is effective in capturing the semantic properties of playlists.

This approach can also be extended in two directions:

1. **Learning better playlist representations**: The proposed technique can be used to learn even better playlist-representations by integrating additional song-embedding models such as spectrogram-based models such as Lee *et al.* (2009). This would mean combining a) co-occurrence of songs in a playlist, and b) absolute content analysis of songs using a neural network, to learn even better playlist embeddings.

2. **Creating new playlists**: The proposed approach can also be leveraged for generating new playlists by using variational models approach as shown in Zhang *et al.* (2016) for generation of new sentences.

BIBLIOGRAPHY

Aalto, E., "Learning playlist representations for automatic playlist generation", (2015).

Adi, Y., E. Kermany, Y. Belinkov, O. Lavi and Y. Goldberg, "Fine-grained analysis of sentence embeddings using auxiliary prediction tasks", arXiv preprint arXiv:1608.04207 (2016).

Andric, A. and G. Haus, "Automatic playlist generation based on tracking users listening habits", Multimedia Tools and Applications **29**, 2, 127–151 (2006).

Arora, S., Y. Li, Y. Liang, T. Ma and A. Risteski, "A latent variable model approach to pmi-based word embeddings", Transactions of the Association for Computational Linguistics **4**, 385–399 (2016).

Bahdanau, D., K. Cho and Y. Bengio, "Neural machine translation by jointly learning to align and translate", arXiv preprint arXiv:1409.0473 (2014).

Bengio, Y., P. Simard, P. Frasconi *et al.*, "Learning long-term dependencies with gradient descent is difficult", IEEE transactions on neural networks **5**, 2, 157–166 (1994).

Blei, D. M., A. Y. Ng and M. I. Jordan, "Latent dirichlet allocation", Journal of machine Learning research **3**, Jan, 993–1022 (2003).

Bowman, S. R., G. Angeli, C. Potts and C. D. Manning, "A large annotated corpus for learning natural language inference", arXiv preprint arXiv:1508.05326 (2015a).

Bowman, S. R., L. Vilnis, O. Vinyals, A. M. Dai, R. Jozefowicz and S. Bengio, "Generating sentences from a continuous space", arXiv preprint arXiv:1511.06349 (2015b).

Chen, C.-M., M.-F. Tsai, Y.-C. Lin and Y.-H. Yang, "Query-based music recommendations via preference embedding", in "Proceedings of the 10th ACM Conference on Recommender Systems", pp. 79–82 (ACM, 2016).

Chen, C.-W., P. Lamere, M. Schedl and H. Zamani, "Recsys challenge 2018: Automatic music playlist continuation", in "Proceedings of the 12th ACM Conference on Recommender Systems", pp. 527–528 (ACM, 2018).

Chen, S., J. L. Moore, D. Turnbull and T. Joachims, "Playlist prediction via metric embedding", in "Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining", pp. 714–722 (ACM, 2012).

Cho, K., B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation", arXiv preprint arXiv:1406.1078 (2014).

Choi, K., G. Fazekas and M. Sandler, "Towards playlist generation algorithms using rnns trained on within-track transitions", arXiv preprint arXiv:1606.02096 (2016).

Chung, J., C. Gulcehre, K. Cho and Y. Bengio, "Gated feedback recurrent neural networks", in "International Conference on Machine Learning", pp. 2067–2075 (2015).

Conneau, A., G. Kruszewski, G. Lample, L. Barrault and M. Baroni, "What you can cram into a single vector: Probing sentence embeddings for linguistic properties", arXiv preprint arXiv:1805.01070 (2018).

De Boom, C., R. Agrawal, S. Hansen, E. Kumar, R. Yon, C.-W. Chen, T. Demeester and B. Dhoedt, "Large-scale user modeling with recurrent neural networks for music discovery on multiple time scales", Multimedia Tools and Applications pp. 1–23 (2018).

De Mooij, A. and W. Verhaegh, "Learning preferences for music playlists", Artificial Intelligence **97**, 1-2, 245–271 (1997).

Dolan, B., C. Quirk and C. Brockett, "Unsupervised construction of large paraphrase corpora: Exploiting massively parallel news sources", in "Proceedings of the 20th international conference on Computational Linguistics", p. 350 (Association for Computational Linguistics, 2004).

everynoise.com, "Everynoise.com", URL `https://www.everynoise.com/` (2012).

Fei-Fei, L., "Recognizing and learning object categories", CVPR Short Course, 2007 (2007).

Fields, B. and P. Lamere, "Finding a path through the juke box: The playlist tutorial", in "11th International Society for Music Information Retrieval Conference (ISMIR)", (Citeseer, 2010).

Herlocker, J. L., J. A. Konstan, A. Borchers and J. Riedl, "An algorithmic framework for performing collaborative filtering", in "22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 1999", pp. 230–237 (Association for Computing Machinery, Inc, 1999).

Hochreiter, S., Y. Bengio, P. Frasconi, J. Schmidhuber *et al.*, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies", (2001).

Hochreiter, S. and J. Schmidhuber, "Long short-term memory", Neural computation **9**, 8, 1735–1780 (1997).

Lee, H., P. Pham, Y. Largman and A. Y. Ng, "Unsupervised feature learning for audio classification using convolutional deep belief networks", in "Advances in neural information processing systems", pp. 1096–1104 (2009).

Lee, J., K. Lee, J. Park, J. Park and J. Nam, "Deep content-user embedding model for music recommendation", arXiv preprint arXiv:1807.06786 (2018).

Liebman, E., M. Saar-Tsechansky and P. Stone, "Dj-mc: A reinforcement-learning agent for music playlist recommendation", in "Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems", pp. 591–599 (International Foundation for Autonomous Agents and Multiagent Systems, 2015).

Lillie, A. S., *MusicBox: Navigating the space of your music*, Ph.D. thesis, Massachusetts Institute of Technology (2008).

Logan, B., "Content-based playlist generation: Exploratory experiments.", in "ISMIR", (2002).

Ludewig, M., I. Kamehkhosh, N. Landia and D. Jannach, "Effective nearest-neighbor music recommendations", in "Proceedings of the ACM Recommender Systems Challenge 2018", p. 3 (ACM, 2018).

Luong, M.-T., H. Pham and C. D. Manning, "Effective approaches to attention-based neural machine translation", arXiv preprint arXiv:1508.04025 (2015).

Marelli, M., L. Bentivogli, M. Baroni, R. Bernardi, S. Menini and R. Zamparelli, "Semeval-2014 task 1: Evaluation of compositional distributional semantic models on full sentences through semantic relatedness and textual entailment", in "Proceedings of the 8th international workshop on semantic evaluation (SemEval 2014)", pp. 1–8 (2014).

McCormick, C., "Word2vec tutorial-the skip-gram model", (2016).

McFee, B. and G. R. Lanckriet, "The natural language of playlists.", in "ISMIR", vol. 11, pp. 537–541 (2011).

Mikolov, T., K. Chen, G. Corrado and J. Dean, "Efficient estimation of word representations in vector space", arXiv preprint arXiv:1301.3781 (2013).

Pascanu, R., T. Mikolov and Y. Bengio, "Understanding the exploding gradient problem", CoRR, abs/1211.5063 **2** (2012).

Ramos, J. *et al.*, "Using tf-idf to determine word relevance in document queries", in "Proceedings of the first instructional conference on machine learning", vol. 242, pp. 133–142 (2003).

Řehůřek, R. and P. Sojka, "Software Framework for Topic Modelling with Large Corpora", in "Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks", pp. 45–50 (ELRA, Valletta, Malta, 2010), `http://is.muni.cz/publication/884893/en`.

Schedl, M., H. Zamani, C.-W. Chen, Y. Deldjoo and M. Elahi, "Recsys challenge 2018: Automatic playlist continuation", in "Late-Breaking/Demos Proc. 18th Int. Soc. Music Information Retrieval Conf.", (2017).

spotify.com, "Million playlist dataset", URL `https://labs.spotify.com/2018/05/30/introducing-the-million-playlist-dataset-and-recsys-challenge-2018/` (2018a).

spotify.com, "Spotify developer api", URL `https://developer.spotify.com/` (2018b).

Sundermeyer, M., R. Schlüter and H. Ney, "Lstm neural networks for language modeling", in "Thirteenth annual conference of the international speech communication association", (2012).

Sutskever, I., O. Vinyals and Q. V. Le, "Sequence to sequence learning with neural networks", in "Advances in neural information processing systems", pp. 3104–3112 (2014).

Szegedy, C., S. Ioffe, V. Vanhoucke and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning", in "Thirty-First AAAI Conference on Artificial Intelligence", (2017).

Van den Oord, A., S. Dieleman and B. Schrauwen, "Deep content-based music recommendation", in "Advances in neural information processing systems", pp. 2643–2651 (2013).

Volkovs, M., H. Rai, Z. Cheng, G. Wu, Y. Lu and S. Sanner, "Two-stage model for automatic playlist continuation at scale", in "Proceedings of the ACM Recommender Systems Challenge 2018", p. 9 (ACM, 2018).

Wu, Y., M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation", arXiv preprint arXiv:1609.08144 (2016).

Yang, H., Y. Jeong, M. Choi and J. Lee, "Mmcf: Multimodal collaborative filtering for automatic playlist continuation", in "Proceedings of the ACM Recommender Systems Challenge 2018", p. 11 (ACM, 2018).

Zhang, B., D. Xiong, J. Su, H. Duan and M. Zhang, "Variational neural machine translation", arXiv preprint arXiv:1605.07869 (2016).

Zheleva, E., J. Guiver, E. Mendes Rodrigues and N. Milić-Frayling, "Statistical models of music-listening sessions in social media", in "Proceedings of the 19th international conference on World wide web", pp. 1019–1028 (ACM, 2010).