**Final Report: Requirements and Outline**

**Introduction / Background / Motivation:**

|  |  |  |
| --- | --- | --- |
| **Question** | **Member** | **Status** |
| (5 points) What did you try to do? What problem did you try to solve? Articulate your objectives using absolutely no jargon. | Bowen |  |
| Alison |  |
| Iman |  |
| Yilun |  |
| Yufei |  |
| (5 points) How is it done today, and what are the limits of current practice? | Bowen |  |
| Alison |  |
| Iman |  |
| Yilun |  |
| Yufei |  |
| (5 points) Who cares? If you are successful, what difference will it make? | Bowen |  |
| Alison |  |
| Iman |  |
| Yilun |  |
| Yufei |  |
| (5 points) What data did you use? Provide details about your data, specifically choose the most | Bowen |  |
| Alison |  |
| Iman |  |
| Yilun |  |
| Yufei |  |

**Approach:**

|  |  |  |
| --- | --- | --- |
| **Question** | **Member** | **Status** |
| (10 points) What did you do exactly? How did you solve the problem? Why did you think it would be successful? Is anything new in your approach? | Bowen |  |
| Alison |  |
| Iman |  |
| Yilun |  |
| Yufei |  |
| (5 points) What problems did you anticipate? What problems did you encounter? Did the very first thing you tried work? | Bowen |  |
| Alison |  |
| Iman |  |
| Yilun |  |
| Yufei |  |

**Experiments and Results:**

|  |  |  |
| --- | --- | --- |
| **Question** | **Member** | **Status** |
| (10 points) How did you measure success? What experiments were used? What were the results, both quantitative and qualitative? Did you succeed? Did you fail? Why? Justify your reasons with arguments supported by evidence and data. Make sure to mention any code repositories and/or resources that you used! | Bowen |  |
| Alison |  |
| Iman |  |
| Yilun |  |
| Yufei |  |

**Additional:**

|  |  |  |
| --- | --- | --- |
| **Question** | **Member** | **Status** |
| (5 points) Appropriate use of figures / tables / visualizations. Are the ideas presented with appropriate illustration? Are the results presented clearly; are the important differences illustrated? | Bowen |  |
| Alison |  |
| Iman |  |
| Yilun |  |
| Yufei |  |
| (5 points) Overall clarity. Is the manuscript self-contained? Can a peer who has also taken Deep Learning understand all of the points addressed above? Is sufficient detail provided? | Bowen |  |
| Alison |  |
| Iman |  |
| Yilun |  |
| Yufei |  |
| (5 points) Finally, points will be distributed based on your understanding of how your project relates to Deep Learning. Here are some questions to think about:   * What was the structure of your problem? How did the structure of your model reflect the structure of your problem? * What parts of your model had learned parameters (e.g., convolution layers) and what parts did not (e.g., post-processing classifier probabilities into decisions)? * What representations of input and output did the neural network expect? How was the data pre/post-processed? * What was the loss function? * Did the model overfit? How well did the approach generalize? * What hyperparameters did the model have? How were they chosen? How did they affect * performance? What optimizer was used? * What Deep Learning framework did you use? * What existing code or models did you start with and how did these starting points help? | Bowen |  |
| Alison |  |
| Iman |  |
| Yilun |  |
| Yufei |  |

Abstract:

Introduction / Background / Motivations:

* Bowen will edit + Yufei

**Edited Version**

Music is an art and a universal language. Deep learning methods have achieved many successes in various tasks such as machine translation and image recognition. However, the field of music generation is inherently related to human expression and emotion, which is a challenging task to machine. Is it possible for artificial intelligence to aid the creative process by learning to create novel melodies in alignment with the style of a particular original melodic sequence? In recent years, researchers have made some great achievements in this field. In this project, we are going to use some existing approaches to predict and evaluate the output music sequences.

|  |
| --- |
| Bowen  Many great composers throughout history have composed pieces that were both creative and deliberate. Is it possible for artificial intelligence to aid the creative process by learning to create novel melodies in alignment with the style of a particular original melodic sequence?    Accordingly, after learning the style of the original input melodic sequence, we will predict and evaluate the success of a corresponding output music sequence. |
| Yufei  Music is an art and a universal language. Deep learning methods have achieved many successes in various tasks such as machine translation and image recognition. However, the field of music generation is inherently related to human expression and emotion, which is a challenging task to machine. In recent years, researchers have made some great achievements in this field. In this project, we are going to use some existing approaches to predict and evaluate the output music sequences. |
|  |
|  |
|  |

Data:

* Alison + Elon

|  |
| --- |
| Bowen:  We will be working with ABC music notation. ABC notation is a shorthand form of musical notation that uses the letters A through G to represent musical notes and other elements to place added values. These added values include sharps, flats, the length of a note, the key and ornamentation.  [Figure of ABC notion example]  Lines in part 1 of the music notation show a letter followed by a colon. These indicate various aspects of the tune such as the index, when there is more than one tune in a file (X:), the title (T:), the time signature (M:), the default note length (L:), the type of tune (R:) and the key (K:). The lines following the key designation represent the tune itself. |
| Alison  There are various file formats for storing music. We chose to use the ABC file format due to its ease of distribution, low overhead, and already tokenized-like representation. Notably, this musical notation language stores compositions in plain text ASCII format and already embeds various features characteristic of music, i.e. pitch, rhythm. Specifically, we chose to use a dataset of folksongs collected The Session dataset where v2\_wo\_repeats was chosen in particular because key signatures were already normalized to C and in 4/4 meter (cite: <https://github.com/IraKorshunova/folk-rnn/tree/master/data>).  The music21 library was used to convert and parse the ABC files into streams for further processing. Firstly, we computed preliminary statistics regarding song length. The final dataset was chosen to comprise of songs ranging from song length \_\_ to \_\_ where an interval was chosen to allow for a larger pool of training data. To standardize file lengths, we added padding to songs of smaller song length and decided to discard pickup measures. Due to a total number of \_\_\_ songs, a train validation test split ratio of 80:10:10 was chosen to allow for more training data. |
| Datasets - Alison |
|  |
|  |

Approach:

|  |
| --- |
| LSTM - Bowen |
| LSTM w/ attendtion - Yufei |
| In-painting LSTM - Alison / Elon |
| Transformer - Iman |
|  |

**Edited Version**

**Approach – RNN character-level language model**

Music in core root is a sequence of musical notes. As in ABC notation, it’s a sequence of characters. Therefore, we’ll give the RNN a huge chunk of text and ask it to model the probability distribution of the next character in the sequence given a sequence of previous characters. This will then allow us to generate new text one character at a time. Feeding the network musical data in ABC notation, it then generates a sequence character by character, whose output is a musical composition in ABC notation. We also want to learn information about song structure, musical measures, and required headers. We cannot expect the model to give us a professional quality music in one-go but we can expect results that are decent quality tunes and worth hearing.

RNN is used here because the length of the data doesn’t need to be fixed, sequence memory is stored and various combinations of input and output sequence lengths can be used. Moreover, RNNs combine the input vector with their state vector with a fixed (but learned) function to produce a new state vector. This can in programming terms be interpreted as running a fixed program with certain inputs and some internal variables.[ karpathy]

[Figure of LSTM]

We used LSTM networks here which are a special kind of RNN capable of learning long-term dependencies. The model is a multi-class classification, and hence we will use categorical cross-entropy as a loss function. The final layer in the model will be a linear activation layer. The number of activation units in the final layer will be equal to the number of all unique characters in all of the music in train data. A Dropout layer is added for regularization to prevent overfitting during the training. To optimize our model, we use Adaptive Moment Estimation(Adam), as it is a very good choice for RNN.

Once our char RNN model is trained, we will then give random characters from the available dataset to our trained Char RNN, it will then generate characters accordingly based on the trained dataset. With this structure, we generate music.

Experiments/Results:

|  |
| --- |
| * hyperparameters, why/how * Individual experimental results * Experiment - splitting the project into the different approaches   LSTM - Bowen |
| LSTM w/ attendtion - Yufei |
| In-painting LSTM - Alison / Elon |
| Transformer - Iman |
|  |

**Edited Version**

**RNN character-level language model**

[Loss plot]

After training for 100 epoch, we got 0.9925 for training loss and 0.7032 for validation loss. And we were able to generate some pleasant samples of music using the trained model. For every generation, the patterns will be different but similar to the training data.

[Figure]

These melodies can be used for a wide variety of purposes such as a productivity tool to develop new ideas, inspiring artists’ creativity, or completing an unfinished piece.

Future Work:

* Iman

|  |
| --- |
| LSTM - Bowen  However, this model can still be improved. Our training data consisted of a single instrument, the piano. One way we could enhance our training data is by adding music from multiple instruments. Another would be to increase the genres of music, their rhythms, and their timing signatures.  At present, our model generates a few false notes and the music is not exceptional. We could reduce these errors and increase the quality of our music by increasing our training dataset as detailed above. |
| LSTM w/ attendtion - Yufei  One future direction is to visualize the attention distribution which can help us understand how the model builds up recurring structures and how far it is attending back. |
| In-painting LSTM - Alison / Elon |
| Transformer - Iman |
|  |

Conclusion: - Iman

|  |
| --- |
| LSTM - Bowen |
| LSTM w/ attention - Yufei |
| In-painting LSTM - Alison / Elon |
| Transformer - Iman |
|  |

**Approach - Inpaint**

**Data Preprocessing:**

1. Brief intro about abc file, maybe show one sample?

2. Filter the music sample according to meters and sequence length; brief explanation

3. Slice by fraction; 0.3past-0.4target-0.3future

4. paddings

5. Brief data description (some hist, data summary, note vocab, etc,.)

6. Train\_test\_val split: 0.8; 0.1; 0.1

**Architecture: Seq2Seq:**

1. Architecture diagram

2. Encoder: bi-directional LSTM, stack hidden states,...

3. Embedding: note emb + measure emb + position emb

4. Loss function: Cross-Entropy (more advanced: focal loss due to unbalanced set?)

5. Optimizer: Adam

6. Others: problems, concerns, ...

There are various file formats for storing music. We chose to use the ABC file format due to its ease of distribution, low overhead, and already tokenized-like representation. Notably, this musical notation language stores compositions in plain text ASCII format and already embeds various features characteristic of music, i.e. pitch, rhythm. Specifically, we chose to use a dataset of folksongs collected The Session dataset where v2\_wo\_repeats was chosen in particular because key signatures were already normalized to C and in 4/4 meter (cite: <https://github.com/IraKorshunova/folk-rnn/tree/master/data>).

The music21 library was used to convert and parse the ABC files into streams for further processing. Firstly, we computed preliminary statistics regarding song length. The final dataset was chosen to comprise of songs ranging from song length \_\_ to \_\_ where an interval was chosen to allow for a larger pool of training data. To standardize file lengths, we added padding to songs of smaller song length and decided to discard pickup measures. Due to a total number of \_\_\_ songs, a train validation test split ratio of 80:10:10 was chosen to allow for more training data.

The amount of past and future context was a percentage chosen based upon the actual song length (pre-padding), namely past:target:future was 30:40:30. Thus, the target length per song is variable. All songs were padded to have equal length, where padding was applied equally to both past and future context. The target was also temporarily padded for ease of computation and reducing time complexity; it is removed after prediction on the test set.

The model architecture and task were inspired by (cite: Inpainting paper). Specifically, the inpainting group used a bidirectional LSTM model with note, measure, and positional embeddings. The past and future

Cross entropy loss was deliberately calculated with the predicted target padding to allow for the model to learn where a song should start and end.

The final prediction target on the hold-out test set is compared to the original ABC target.

Data tokenization, note vocab, measure vocab, etc. Target is note target.

Limitations:

a) Song length statistic

## The song length statistic we use to generate our cleaned dataset is based upon the “number of notes” and is not sensitive to the organization of “measures/bars” in the music. Likewise, though we attempted to control the interval of song length chosen for all songs in a dataset to reflect that of 4/4 meter to some extent, it may still not

b)

Future Work:

a) Experiments on different ratios of past and future context length

Initial experiments used past and future context lengths of the same size. However, given a larger dataset where all songs were of a single standardized song length, it would be interesting to conduct experiments on differing past and future context ratios. A useful application for this may apply to musical form and structure, where a prediction near the “end” of a song may be better conditioned on more past context, whereas a prediction near the “beginning” of a song may be better conditioned on more future context.

Experiments and Results:

**Experiments and Results - Inpaint**

1.Start with default hyperparameters

2.Report loss, perplexity, time, etc,.

3.HP tuning (Alison’s visualization, super cool!)

4. Evaluation: quantitative (CR loss) and qualitative (expert rating? Blind test (ground truth - inpainted, which is better?))

4.Show one/some sample song after inpaint

Additional:

Conclusion:

**Bowen**

**Motivation**

Many great composers throughout history have composed pieces that were both creative and deliberate. Is it possible for artificial intelligence to aid the creative process by learning to create novel melodies in alignment with the style of a particular original melodic sequence?

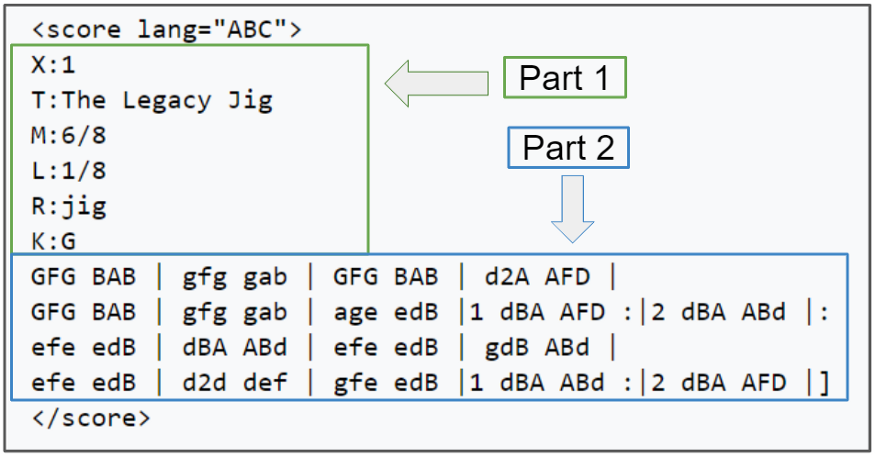
Accordingly, after learning the style of the original input melodic sequence, we will predict and evaluate the success of a corresponding output music sequence.

**Related work**

There has been considerable work regarding melody generation with RNNs, LSTMs, and GANs. Lots of existing work that has already been done such as Google’s Magenta team’s melodic\_rnn, melodic and rhythmic style transfer, and explicitly conditioned melody generation. Moreover, evaluation of the subjective results generated have been considered and discussed in favor of more objective measures. The use of monophonic MIDI melodies are generally preferred for ease of melodic analysis which involves both pitch and rhythmic components.

**Data**

We will be working with ABC music notation. ABC notation is a shorthand form of musical notation that uses the letters A through G to represent musical notes and other elements to place added values. These added values include sharps, flats, the length of a note, the key and ornamentation.



Lines in part 1 of the music notation show a letter followed by a colon. These indicate various aspects of the tune such as the index, when there is more than one tune in a file (X:), the title (T:), the time signature (M:), the default note length (L:), the type of tune (R:) and the key (K:). The lines following the key designation represent the tune itself.

**Approach**

We’ll train RNN character-level language models. We’ll give the RNN a huge chunk of text and ask it to model the probability distribution of the next character in the sequence given a sequence of previous characters. This will then allow us to generate new text one character at a time.

Feeding the network musical data in ABC notation, it generates a sequence character by character, whose output is a musical composition in ABC notation. We also want to learn information about song structure, musical measures, and required headers. We cannot expect the model to give us professional quality music in one-go but we can expect results that are decent quality tunes and worth hearing to.

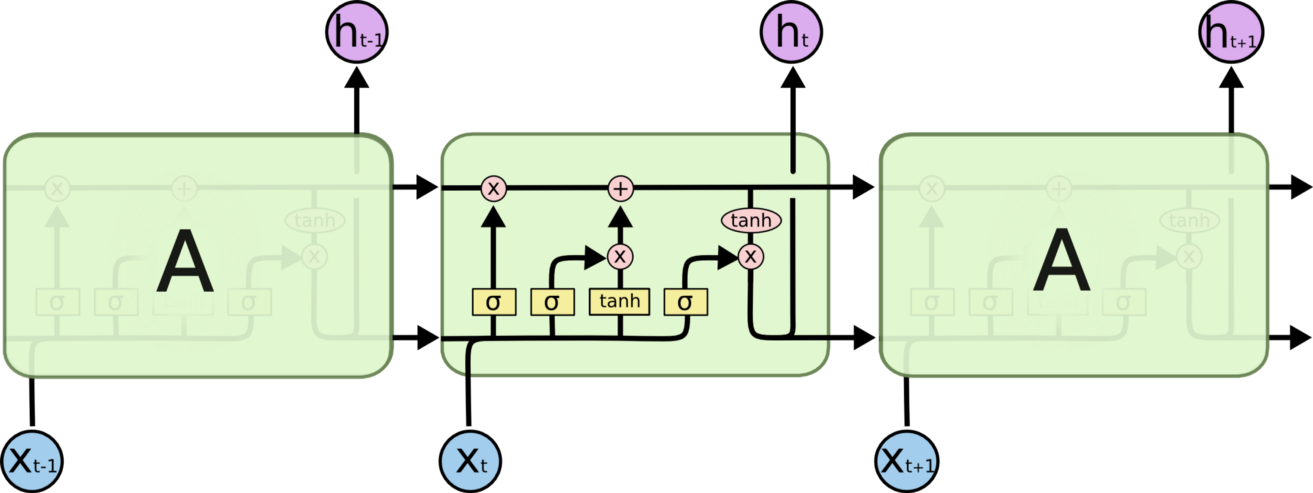
RNN is used here because:

1. The length of the data doesn’t need to be fixed. For every input, the data length can vary

2. Sequence memory is stored.

3. Various combinations of input and output sequence lengths can be used.

Moreover, RNNs combine the input vector with their state vector with a fixed (but learned) function to produce a new state vector. This can in programming terms be interpreted as running a fixed program with certain inputs and some internal variables.

We used LSTM networks here which are a special kind of RNN capable of learning long-term dependencies.

The model is a multi-class classification, and hence we will use categorical cross-entropy as a loss function. The final layer in the model will be a linear activation layer. The number of activation units in the final layer will be equal to the number of all unique characters in all of the music in train data.

A Dropout layer is added for regularization to prevent overfitting during the training. We choose 0.2 after tuning. To optimize our model, we use Adaptive Moment Estimation(Adam), as it is a very good choice for RNN.

Once our char RNN model is trained, we will then give random character- from the available dataset to our trained Char RNN, it will then generate characters accordingly based on trained dataset.

In the process of music generation, the first character is chosen randomly from the unique set of characters, the next character is generated using the previously generated character and so on. With this structure, we generate music.

We generated some pleasant samples of music using machine learning neural networks known as LSTMs. For every generation, the patterns will be different but similar to the training data. These melodies can be used for a wide variety of purposes such as a productivity tool to develop new ideas, inspiring artists’ creativity, completing an unfinished piece etc.

**Future work**

However, this model can still be improved. Our training data consisted of a single instrument, the piano. One way we could enhance our training data is by adding music from multiple instruments. Another would be to increase the genres of music, their rhythms, and their timing signatures.

At present, our model generates a few false notes and the music is not exceptional. We could reduce these errors and increase the quality of our music by increasing our training dataset as detailed above.

**Conclusion**

We looked at how to process music for use with neural networks, the in-depth workings of deep learning models like RNN & LSTMs, and we also explored how tweaking a model can result in music generation.

We’ve trained an RNN character-level language model, and we got results that are decent quality tunes and worth hearing to. We can confidently expect a large amount of innovation in the space of RNNs, and I believe they will become a pervasive and critical component to intelligent systems.

**End**

**IMAN**

**Approach:**

The Music Transformer uses a relative self-attention mechanism and follows the methodology of language modeling. The data transformation for this task took a different approach than for the other methods; the entire dataset of song information was flattened into one long super-song, and then batched to match the Transformer’s input dimensions. These batches are considered to be source sequences for the Transformer, and they are used to generate targets by incrementing the sequences by one item. A description of the data transformation is provided in Figure [X].

The key components of the Transformer Model are: an encoder, a decoder, a mask generator, and a positional encoder. The first two are implemented with Pytorch’s Embedding, TransformerEncoder, and Linear blocks. The mask generator returns an upper-triangular matrix of -inf values to ensure that the Transformer only learns based on past context and not any future context. The positional embedding is implemented as described in [Reference to VASWANI ET AL], in which sinusoidal functions of different fundamental frequencies are added to the embedding of the source sequence so that the Transformer learns the relationships of the relative positions of items in the sequence.

We believed that implementing the Music Transformer in the same way as a language modeler was a promising approach because there are many similarities between the relative importance of words in a written sentence, and notes in a melody. Additionally, there was reason to believe that the music modeling task would be simpler because the “vocabulary” of notes is very small with respect to the size of a language’s lexicon. This approach follows the lead of [Vaswani and Huang papers] and the only new feature is the use of .abc data as opposed to MIDI used by Huang.

Because the training data of all songs was completely flattened, it was not apparent what form the outputs of the Transformer would take. This difficulty carried on through testing of the model, the outputs were difficult to interpret. The best we accomplished was a model that required a start sequence of the same size as the training vectors, and would output a generated melody of the same size.

**Experiments and Results:**

The Music transformer would be evaluated based on the perplexity of the outputs and the subjective analysis of the output sequences. The trained model was tested by

Yufei

**Motivation**

Music is an art and a universal language. Deep learning methods have achieved many successes in various tasks such as machine translation and image recognition. However, the field of music generation is inherently related to human expression and emotion, which is a challenging task to machine. In recent years, researchers have made some great achievements in this field. In this project, we are going to use some existing approaches to predict and evaluate the output music sequences.

**Related work**

Various approaches have been developed for modelling music in the past few years, from RNNs and LSTMs to Bidirectional LSTMs. However, a model needs to refer to the historical elements to generate a coherent piece of music and create contrast and surprise. The sequential-processing structure of RNNs and LSTMs requires the model to process the data step by step. The longer the information travels, the more it may lose during the process. Different from RNNs, LSTMs use gate cells to decide which information should be stored. However, when the distance is too large, LSTM may not work well either. Therefore, the model performance may be affected when the needed historical elements are in the distant past. Compared with RNNs and LSTMs, the self-attention mechanism appears to be a better approach. The self-attention mechanism can “look” at all available elements simultaneously at every step of generation. It can also help capture the self-referential phenomena that exist in music.

**Approach**

Self-attention mechanism

Self-attention mechanism was first developed for NLP problems. It allows the input elements to interact with each other to find which elements they should pay more attention to. The attention layer will first transform a sequence of input into query vectors Q, key vectors K, and value vectors V by multiplying X by weight matrices WQ, WK, and WV. Then the calculated Query, Key and Value vectors will be used to calculate the attention score by:



Relative position self-attention mechanism

Since the attention mechanism is not aware of the position of each input element, a position information is added to each element before it is fed to the attention layer. Shaw et al. (2018) introduced relative position representations which allow the attention layer to know how far two elements are in a sequence. The relative attention score is calculated by:



where is a logits matrix which modulates the attention probabilities for each head.