Denver Persinger
Dr. Stephen Hutt
COMP 3501 - Intro. To Artificial Intelligence
19 November 2024

# Project Report

Utilizing Artificial Intelligence (AI) systems to produce or assist in the music creation process is a sector of AI that is gaining a lot of traction. Developments like IBM's Watson Beat, which produces beats and audio from simple input melodies, and Staccato, which assists in songwriting, are both strong indications of the strides that have occurred in the area as well as the increase in investment into such technologies from major corporations [4, 21]. The developments relating to AI and music lead to an interesting discussion into its implications and how it impacts artistry as a whole. With incidents such as the anonymous user that used Drake and The Weekend's voices without their consent and the debate of who owns material created with generative AI, this form of AI is making headlines [5, 17].

The problem at the core of this project is understanding how AI can be used to generate music that replicates human creativity. Music generation involves producing sequences of notes, rhythms, and dynamics that are coherent, harmonic, and aesthetically pleasing. Unlike tasks where there are clear metrics for success, music creation is inherently subjective, making it a unique and challenging problem to address, and an interesting test of the creativity and artistry of AI.

Music, unlike other forms of data, is highly complex, with dependencies that can span across measures or even entire songs. Capturing this long-range context is particularly difficult and it is extremely important in the development of quality music, as transitions between notes or chords depend on the transitions that occurred before them. This makes music generation a complicated issue, there is a large technical challenge of designing and training a model capable of learning patterns, structures, and when to introduce shifts in structure from musical data.

There are also philosophical questions attached to this problem: how can the generated music reflect the emotional depth of human created music? Can AI-generated music really be considered "creative," or is it merely replicating learned patterns? Is human-developed music just the implementation of learned patterns as well, and therefore can AI be just as good or even better at making music? These questions go beyond the technical and delve into the philosophical and ethical realms, adding a layer of richness to the problem, a layer of richness which I am not qualified to speak on [2, 10]. However, I hope the result of this paper, and the product of this report can aid in the answers to some of these questions in some way based on the quality or feeling produced.

**Methods:**

The first approach I attempted in tackling this issue was through the use of a genetic algorithm (GA). I have found that GAs are especially useful when working in a field in which you are not an expert as the long-term dependencies and needed structures to form a solution are learned and not needed to be necessarily hard-coded in. I did, however, need a fitness function. As I do not have a deep understanding of music theory, I decided to build a classifier, a Random Forest, that would be trained on scored MIDI files that were broken down into data-points, such as consonance counts, dissonance counts, pitch range, silence counts, and more. I then had a folder of ~700 classical music MIDI files and ran them through preprocessing that would break them into 4 folders. \`decimatedMidis\`, \`badMidis\`, \`tweakedMidis\`, and \`goodMidis,\` these directories are full of MIDI files that have been changed or tweaked in order to create diverse datasets. I then scored them based on which directory they were in, zero to six, and trained the Random Forest on that data.

The first issue I ran into was that the GA never produced music classified higher than a zero starting from a random population, even after 500 generations using the classifier as a fitness function, variable crossing overrates, and a low, 0.01, mutation rate. After switching to a non-random population, starting with the tweaked MIDIs, I still got bad results. I realized the reason behind this failure is potentially due to the complexities of musical structure addressed before and my lack of a good enough fitness function that takes into account all of those necessary complexities. Crossing over a good song by splicing it and combining it with another would detriment the overall quality of a composition as it completely changes the flow. As I do not have a PhD in music theory, and since the time needed to run 500 generations which only produced one song was immense I decided to attempt a different solution rather than attempting to fortify my fitness function and algorithm.

I decided to explore how a Recurrent Neural Network (RNN) with Gated Recurrent Unit (GRU) layers could be used to generate music note-by-note, instead of a GA that generated entire populations at a time, which would lose the dependencies and structure of music. The use of a RNN over a traditional Neural Network (NN) is a more viable solution as it has the ability to use previous inputs as information that influences the output of the current input, while a NN assumes all inputs are independent and hold no information on previous passes. Since music can be interpreted as time-series data where the next note depends on all of the notes before it, similar to language processing, we can incrementally build a solution by passing in an incomplete sequence and training it on predicting the next note, and as it will build from a single to a full song, it will be able to learn and remember complex dependencies and structures that are needed in music. This approach will address both concerns that lead to my failure through the GA approach. While the model might take a long time to train, creating a song should be easy with a saved model. Also, the model will be able to better understand and implement the necessary structures of music that a GA or normal free-forward NN can.

My model consists primarily of GRU layers instead of other modern RNN layers, such as Long Short-Term Memory (LSTM), as they are computationally more efficient. Below is an image of a LSTM versus GRU node and the gates associated with them.

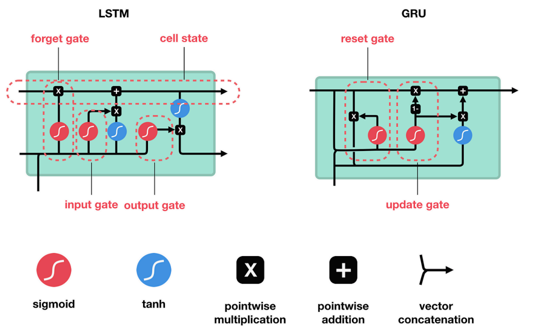


*Figure 1: Comparison between LSTM (left) and GRU (right) nodes with their respective gates [11]*

The gates shown in this figure within each node are fundamental to RNNs and are the means in which a RNN is able to hold memory about previous inputs. The core of RNNs is based on the feedback loops held within each node, where there are hidden units that are used in computation along with the input values, allowing memories of inputs from past time steps to be used in computation with the current timestamp. Through the use of activation functions, such as sigmoid or  tangent hyperbolic (tanh), the RNNs learn what data is important, and what data is good to forget [15]. Music is fundamentally sequential time-series data, and due to RNNs ability to remember past timestamps, this solution should be very effective.

An LSTM node has a forget gate, cell state, input gate, and output gate while a GRU node only has two gates, the reset gate and the update gate, leading to less parameters needed and a faster computation time. While GRUs are more architecturally simple, they perform close to as well as their more complicated counterparts in most situations while being able to learn quickly and adapt to new dependencies, which is why I opted to use GRUs over LSTMs or other RNN layers [19].

To train my RNN model, I compiled Musical Instrument Digital Interface (MIDI) files from four genres: Classical, Jazz, Christmas, and Lo-Fi. Each set of MIDI files were put into separate directories by name. These MIDI files were compiled from various sources on the web such as Cymantics for Lo-Fi, Kunstderfuge for Classical, Midkar for Jazz, and MidiWorld for Christmas [3, 6, 14, 22]. These collections were then incrementally tuned by removing and adding MIDI files based on quality and impact on training. In order to process the MIDI files, I first had to translate them into a comprehensible format. In my `get_data.py` file I iterate over all files in a given genre and convert them into a list of songs, where each song is represented as a list of notes and chords such as “C4” and “2.6.9” respectively, where chords are represented as pitches in normal order. I then write that list of notes in songs to a file to mitigate having to reparse the MIDI files when training again.

Even though I now have data from each song, the data is not in a format that a RNN can process. I break each song into fixed-length input sequences with a minimum of 32 notes and a maximum of the length of the shortest song, and output sequences containing the rest of the song, where the input data is then normalized to be passed into th RNN.

The RNN I implemented contains eight layers through Keras: an input layer, three GRU layers with 256 nodes each, a Dense layer with 256 nodes using a tanh activation function, Dropout and Batch Normalization layers to help prevent overfitting and aid in learning, and a final output Dense layer using a softmax activation function with as many nodes as the number of notes are possible so that any note can be produced, with the generated note being the output with the highest probability. A softmax activation function was used as it is more suited for multi-class classification tasks [18]. These layers and parameters were tweaked through a testing process on subsets of data. When compiling the model, I used a Categorical Cross-Entropy loss function with an AdamW optimizer. The Categorical Cross-Entropy loss function is more well-suited for multi-class classification similarly to the softmax activation function and the AdamW optimizer uses stochastic gradient descent (SGD) with momentum that yields better loss values and generalizes better than the traditional optimizers like the commonly used Adam optimizer [8, 9]. The parameters and functions used in training a RNN are extremely important, and using a far from optimal function can drastically influence your output. For example, looking at sigmoid compared to tanh, it is clear that tanh is more suited for RNN tasks as it maps values from -1 to 1 rather than the more limited 0 to 1 range a sigmoid function maps to. This behavior allows for the gradient of tanh to be four times as strong as the gradient for sigmoid, allowing us to converge faster and take larger learning steps [1].

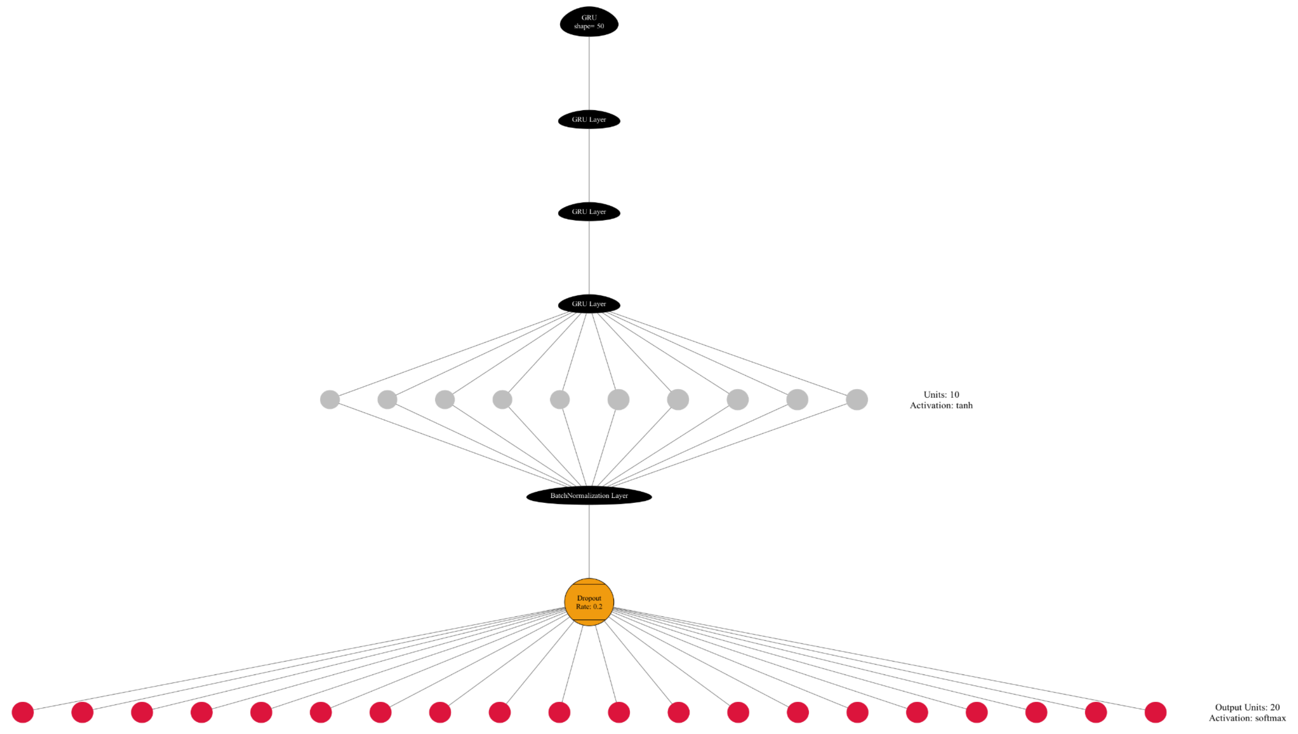


*Figure 2: a visual representation of the layers in the RNN*

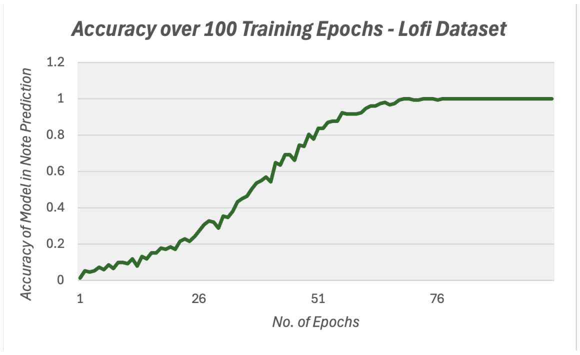
I trained the model for 200 epochs and a variable batch size depending on how much data I had in the genre I was training. The batch size determines the number of training samples used in one iteration of training, and due to diverse sizes in my datasets, I tuned down the batch sizes to fit the amount of available data [7]. 200 epochs were used as I included a ModelCheckpoint callback that saved the current weights of the model after every epoch, allowing me to kill the training process when I find the loss and accuracy to be values that I find to work. I also added a TensorBoard callback that provides me with data throughout the training process to analyze my results over the training process. The training times varied immensely for each model, with the classical model training for ~10 hours and the Lo-Fi model taking less than 10 minutes. The jazz model took about 4 hours to train and the christmas model took roughly 30 minutes. This was the time needed to converge on an accuracy greater than 95% or run 200 epochs, whichever came first. These differences stemmed from variations in data complexity, breadth, average length, and batch sizes.

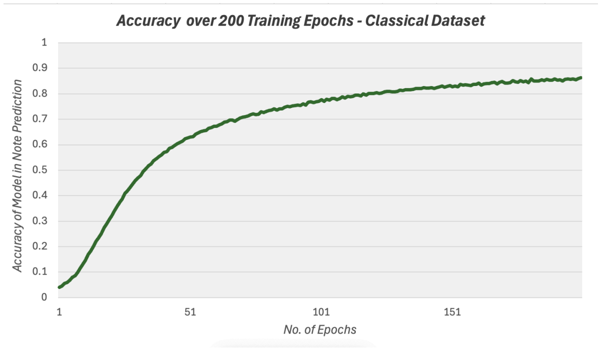
Now that I had a trained model, I still needed to generate the music note by note. In order to achieve this I needed to provide a random starting sequence of notes from the input data for the model to incrementally build off of. I then incrementally generate notes, add them to a sequence of generated notes and continue to build off of those generated notes. I convert the generated sequence back into a MIDI file and save the output. I compiled these functions into a main function that allows users to create and listen to generated music created by a saved model by interacting with inputs that indicate which genre of music it should create a song in.

Many assumptions were made in the development of this system. I assume that the data compiled is complete and diverse enough to where the model can make proper generalizations and not fall into set patterns and overfit. I assume that in encoding the music as a set of notes and nothing more, I encompass enough of the intricacies of music to meaningfully generate a piece. Lastly, I assume the GRU layers can properly pick up the long-term dependencies of the data and can therefore reproduce music that incorporates those dependencies.

**Results:**

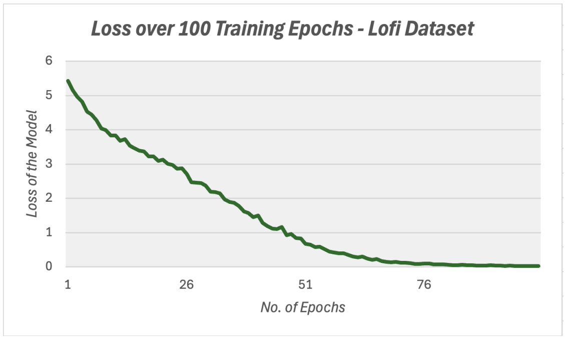
Using a TensorBoard callback, I am able to get data concerning the training and success of the model over time. This callback and its associated data can be found in the \`logs\` directory within the project. The data presented below is in terms of the training of my Lo-Fi and Classical music datasets, my smallest and largest datasets, respectively, to compare and contrast the impact of data and complexity on the result. Overall, the model performed well, yielding a high accuracy and generating some very pleasant music, of which the subjective best outputs for each genre were submitted along with the project. Before analyzing the quality of the product, I would like to analyze the results of the models training in terms of potential overfitting, accuracy, and loss.

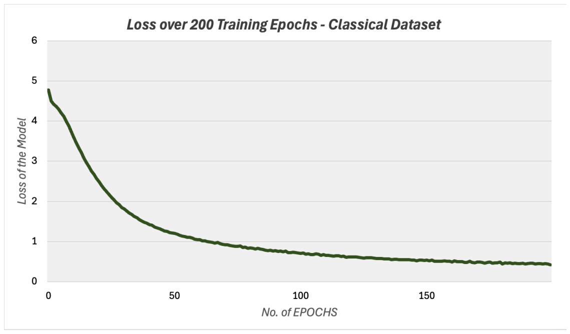




*Figures 3 and 4: graphs displaying accuracy improvements over the training period*

The accuracy for the Lo-Fi and Classical datasets reveal large differences in the model's training behavior caused by dataset size and complexity. For the Lo-Fi data, the accuracy rapidly increased to near-perfect levels at a steep rate (~100%) within just 100 epochs, indicating that the model easily captured the simpler and repetitive patterns characteristic of Lo-Fi music, while also only having to ingest 44 short MIDI files compared to the 108 Classical MIDI files that sometimes spanned minutes in length for a single composition. The accuracy for the Classical dataset improved more gradually, converging to a final value of approximately 90% after 200 epochs, taking about 180 seconds an epoch compared to the 1 second needed per epoch when training on the Lo-Fi data. This slower convergence reflects the increased complexity and variation within the Classical dataset, which requires the model to learn intricate dependencies and long-term structures over a longer training period. While the Lo-Fi dataset allowed the model to achieve near-perfect accuracy, it may also suggest a risk of overfitting due to the limited data diversity. The Classical dataset's slightly lower accuracy indicates a balance between learning and generalization, as the model was challenged to adapt to a broader range of musical patterns, as seen when comparing the breadth of the music created in the \`main.py\` file. This comparison highlights the importance of dataset characteristics in shaping the training dynamics and final performance of the model.





*Figures 5 and 6: graphs displaying loss improvements over the training period*

The loss metrics for the Lo-Fi and Classical datasets outline the differences in how the model learned from datasets. For the Lo-Fi dataset, the loss decreased rapidly almost at a constant rate from an initial value of ~5.5 to nearly 0 within 100 epochs, reflecting the model's ability to quickly learn the relatively simple, short, and repetitive patterns characteristic of Lo-Fi music. This rapid decline in loss suggests that the dataset was less challenging for the model to fit. In contrast, the loss for the Classical dataset exhibited a steadier and more gradual decline, starting around ~5 and plateauing at approximately 0.2 after 200 epochs. This slower convergence indicates the model's need to process and adapt to the intricate structures and dependencies inherent in Classical music. The near-zero loss for the Lo-Fi dataset suggests that the model could be at risk of overfitting, especially given its accuracy of 1.0, whereas the higher final loss for the Classical dataset demonstrates a balance between minimizing error and maintaining generalization. The lack of a smooth loss and accuracy plot in the Lo-Fi model indicates that the model is having a difficult time remaining consistent, outlining that the lack of data and variability in musical dependencies forces the model to memorize the training data rather than learn it, making it very likely overfit, a feature that is reinforced when generating music with the model [12].

While looking strictly at the numerical loss and accuracy data compiled during the training process, it would be seemingly safe to assume that both models will perform very well in generating new musical sequences note by note. However, when creating music, there is a clear discrepancy in quality produced by the models. For example, the Lo-Fi model often falls into long-term and redundant patterns while producing many off-pitch chords/notes, a feature that majorly suggests the model is overfit and therefore is lacking the ability to generalize. While it occasionally produces good results, a majority of the time, it generalizes poorly. This is likely due to the datasets limitations and the lack of diversity and structure that the model was exposed to. While it does produce music in most instances that has the Lo-Fi feel, it has a feeling like it is a perversion of what Lo-Fi music should be, providing an uncanny valley effect. The Classical music model, on the other hand, is very consistent, with most song generations having a strong amount of diversity while also being structurally sound and audibly appealing.

When evaluating the models as a whole, the subjective, personal rankings in terms of the quality of the music they produced are as follows: Lo-Fi, Christmas, Jazz, and Classical. This ranking reflects both consistency and overall quality of the generated music and closely aligns with the size and quality of the datasets used to train the models. Larger and more diverse datasets, such as those for classical music, allowed the model to capture more complex patterns and produce higher-quality outputs, while smaller datasets with simpler patterns, like Lo-Fi, resulted in less sophisticated yet still sometimes, yet very infrequently, enjoyable music.

To assess the subjective quality of the generated music, I conducted a survey of students on campus (*N = 15*), asking them to rate the music from 1 to 10. The survey was prefaced with the explanation that I created the music myself, with no musical background, instead of telling them it was AI generated. This framing was important, as it set appropriate expectations for the students. Without this context, individuals with more extensive musical training, such as students from the Lamont School of Music, might have rated the pieces much more critically. The goal was to determine whether the AI-generated music was impressive to the average listener, given my limited musical expertise. Telling students it was AI-generated may set the bar higher than telling them I made it as the media perception of AI frames it to be extremely powerful in all contexts [13].

Selected students rated pieces for each genre generated on the spot, and the average scores were as follows: 4.4 for Lo-Fi, 6.0 for Christmas, 7.5 for Jazz, and 8.2 for Classical. The classical music model, trained on the largest and most complex dataset, produced compositions that were both intricate and harmonically sound, earning the highest average score. Jazz followed behind, with its outputs demonstrating a balance of complexity and style. The Christmas and Lo-Fi models, while enjoyable, were rated lower, reflecting their simpler patterns and the limitations of their smaller training datasets, sometimes repeating a harsh pattern over and over again. These models were by far the least successful, a feature of which is evident when generating music. This evaluation reinforces the idea that dataset size and quality are critical factors in shaping the effectiveness and perceived quality of AI-generated music, even if using an effective model.

The results of this project demonstrate that the implementation of AI for music generation using RNNs with GRU layers was successful, producing music that was both stylistically consistent and audibly pleasing for most of the models, as seen in the good outputs attached in the `GoodOutputs` directory in my submission. By analyzing training metrics, dataset features, and the subjective personal and peer evaluations of generated outputs, it is clear that the model effectively learned patterns and common dependencies from the datasets and generalized well to create new, coherent songs.

**Discussion and Conclusion:**

The implementation of AI for music generation using Recurrent Neural Networks (RNNs) with Gated Recurrent Unit (GRU) layers was a big success. It generated music that was both stylistically consistent and subjectively pleasing, while minimizing and maximizing loss and accuracy. The results show the model's ability to learn and generalize musical patterns from datasets of varying sizes and complexities. Larger datasets, such as the Classical and Jazz datasets, enabled the model to capture intricate long-term dependencies and produce complex and better compositions, while smaller datasets, like the Lo-Fi and Christmas datasets, resulted in quicker convergence but were more prone to overfitting.

While the project achieved its primary goals, several limitations came to light. First, the smaller datasets, particularly for Lo-Fi and Christmas music, lacked the data diversity needed for quality generalization, leading to repetitive patterns and harsh tones appearing in the generated outputs. Within the parsing of the data, I also did not encode empty space or silence into the dataset, meaning that there are very few points in created compositions where the model meaningfully used breaks in the music. This lack of encoding was understood during the development process, but was not implemented as my understanding of how to extract it is limited. Lastly, the music created was limited to one track, meaning that complex compositions were not truly possible. Training on multitrack data while only producing one may have been an issue as well.

Additionally, while the GRU-based RNN was computationally efficient and effective for time-series data, further optimization through hyperparameter tuning and experimentation with other models such as simple RNNs and LSTMs may have improved the performance across all genres. Finally, the subjective survey, though insightful, was limited in scale (*N=15*) and framing, which may have influenced participants' perceptions and ratings. Also, it was not a random sample but rather a convenience sample due to time and resource constraints, which most likely skewed the results.

The results of this project underscore the importance of dataset characteristics in shaping model performance, a common theme throughout machine learning as a whole, as with bad data comes bad results [16]. The classical dataset, with its size and complexity, enabled the model to produce compositions that were diverse, harmonically rich, and structurally sound, earning the highest subjective ratings. It utilized long-term structural dependencies very well, allowing for pieces to flow better. In contrast, the Lo-Fi dataset's smaller size and simpler patterns resulted in outputs that were somewhat stylistically appropriate but often repetitive, less dynamic, and jarring. An interesting feature is in the feel of the music. While the best model, the classical model, produces harmonic and most of the time, quality music, it still feels like it is missing something. While that may or may not be the emotion or feeling behind the composition, it is an interesting sensation to listen to music that is definitely music, with something seeming off.

This project demonstrates the potential of AI to contribute meaningfully to creative fields like music composition. The ability to learn and replicate musical patterns opens avenues for AI as a collaborative tool in the artistic process. Even with my limited dataset, relatively smaller model, and shorter training periods, my output for my best model is still successful, indicating the potential for much more impressive music creation. Beyond music, this project outlines the value of selecting the right models and data for specific problems. GRUs, with their ability to model time-series data efficiently, proved to be a practical choice for this task, as my initial attempt with a GA was needlessly complicated. The lessons learned here can inform broader AI applications, particularly in areas requiring long-term dependency modeling on time-series data, such as natural language processing.

**Bibliography**

[1] Panagiotis Antoniadis. 2022. Activation Functions: Sigmoid vs Tanh | Baeldung on Computer Science. *www.baeldung.com*. Retrieved from https://www.baeldung.com/cs/sigmoid-vs-tanh-functions

[2] Melissa Avdeeff, Claire Benn, Lindsay Brainard, Alice Helliwell, Adam Linson, Elliot Samuel Paul, Dustin Stokes, and Steffen Steinert. 2023. Eight Scholars on Art and Artificial Intelligence. *Aesthetics for Birds*. Retrieved from https://aestheticsforbirds.com/2023/11/02/eight-scholars-on-art-and-artificial-intelligence/

[3] Bob BobE. 2024. Jazz MIDI Index Page. *Midkar.com*. Retrieved November 19, 2024 from https://midkar.com/Jazz/Jazz.html

[4] Anna Chaney. 2018. The Watson Beat. *Medium*. Retrieved from https://medium.com/@anna_seg/the-watson-beat-d7497406a202

[5] Joe Coscarelli. 2023. An A.I. Hit of Fake "Drake" and "The Weeknd" Rattles the Music World. *The New York Times*. Retrieved from https://www.nytimes.com/2023/04/19/arts/music/ai-drake-the-weeknd-fake.html

[6] Cymatics. Cymatics.fm - The #1 Site For Serum Presets, Samplepacks & More! *Cymatics.fm*. Retrieved from https://cymatics.fm/

[7] GeeksforGeeks. 2024. How to Calculate Optimal Batch Size for Training Neural Networks. *GeeksforGeeks*. Retrieved November 18, 2024 from https://www.geeksforgeeks.org/how-to-calculate-optimal-batch-size-for-training-neural-networks/

[8] Raúl Gómez. 2018. Understanding Categorical Cross-Entropy Loss, Binary Cross-Entropy Loss, Softmax Loss, Logistic Loss, Focal Loss and all those confusing names. *Github.io*. Retrieved from https://gombru.github.io/2018/05/23/cross_entropy_loss/

[9] Fabio M. Graetz. 2020. Why AdamW matters. *Medium*. Retrieved from https://towardsdatascience.com/why-adamw-matters-736223f31b5d

[10] David Henkin. 2023. Orchestrating the Future—AI in the Music Industry. *Forbes*. Retrieved from https://www.forbes.com/sites/davidhenkin/2023/12/05/orchestrating-the-future-ai-in-the-music-industry/

[11] Tiew Kee Hui. 2019. Tensorflow 2.0 / Keras - LSTM vs GRU Hidden States. *Tiew Kee Hui's Blog*. Retrieved November 19, 2024 from https://tiewkh.github.io/blog/gru-hidden-state/

[12] Mostafa Ibrahim. 2023. A Deep Dive Into Learning Curves in Machine Learning. *W&B*. Retrieved from

https://wandb.ai/mostafaibrahim17/ml-articles/reports/A-Deep-Dive-Into-Learning-Curves-in-Machine-Learning--Vmlldzo0NjA1ODY0

[13] Rasmus Kleis Nielsen. 2024. How news coverage, often uncritical, helps build up the AI hype | Reuters Institute for the Study of Journalism. *reutersinstitute.politics.ox.ac.uk*. Retrieved from https://reutersinstitute.politics.ox.ac.uk/news/how-news-coverage-often-uncritical-helps-build-ai-hype

[14] Kunstderfuge. kunstderfuge.com | Free Classical MIDI files. Classical Music. *kunstderfuge.com*. Retrieved from https://kunstderfuge.com/

[15] Wenyi Pi. 2024. An Introduction to Recurrent Neural Networks (RNNs). *Medium*. Retrieved from https://medium.com/@researchgraph/an-introduction-to-recurrent-neural-networks-rnns-802fcfee3098

[16] Thomas Redman. 2018. If Your Data Is Bad, Your Machine Learning Tools Are Useless. *Harvard Business Review*. Retrieved from https://hbr.org/2018/04/if-your-data-is-bad-your-machine-learning-tools-are-useless

[17] Kristin Robinson. 2023. Ghostwriter, the Mastermind Behind the Viral Drake AI Song, Speaks For the First Time. *Billboard*. Retrieved from https://www.billboard.com/music/pop/ghostwriter-heart-on-my-sleeve-drake-ai-grammy-exclusive-interview-1235434099/

[18] Syed Ameer John SK. 2023. Comparison between Sigmoid Function and Softmax Function with Python Code Implementation. *Medium*. Retrieved from https://medium.com/@sksyedroshan007/comparison-between-sigmoid-function-and-softmax-function-with-python-code-implementation-df05a978060d

[19] Prudhviraju Srivatsavaya. 2023. LSTM vs GRU. *Medium*. Retrieved from https://medium.com/@prudhviraju.srivatsavaya/lstm-vs-gru-c1209b8ecb5a

[20] Cole Stryker. 2023. What are Recurrent Neural Networks? | IBM. *www.ibm.com*. Retrieved from https://www.ibm.com/topics/recurrent-neural-networks

[21] Staccato | Create Your Best Music with Generative AI Tools for Music & Lyrics. *staccato.ai*. Retrieved from https://staccato.ai/

[22] Christmas Carols MIDI files - Download for free. *www.midiworld.com*. Retrieved from https://www.midiworld.com/files/1130/