Modeling Discrete Count Data with Continuous Probability Distributions: With An Application In Music Generation

Bas Maat[1 [2749086]]

Vrije Universtiteit, Amsterdam NH, NL, b.j.maat@student.vu.nl

Abstract. This paper explores the use of continuous probability distributions for modeling discrete count data, specifically in the context of music generation. We propose a novel continuous distribution function called CONTIN, alongside established distributions like the Discrete Weibull and Discretized Normal Distributions. They are compared against the second proposed representation, Bitwise, an unsigned binary bit-string. These are evaluated on two regression tasks (predicting bicycle counts and social media upvotes) and a music generation task using a transformer model. Results suggest that CONTIN is a viable option for modeling data, particularly with larger datasets, while Bitwise consistently outperforms other methods. Future research should include loss balancing between different data tokens.

Keywords: Music Transformer · Distributing Count Data · MIDI Faithful

1 Introduction

Modeling discrete count data present a unique challenge in data analysis and machine learning, for instance modeling the number of bicycles in a parking garage for bikes, or upvotes on a social media post. Traditionally, for modeling discrete count data tasks the Mean Squared Error (MSE) is used, but it is not always suitable since count data is often skewed or over-dispersed. Several approaches have been explored to address this challenge, such as the Poisson Distribution that is often used in statistics for modeling count data [5] and the Discrete Weibull Distribution [16] which performs better than the Poisson Distribution for count data in Machine Learning regression as proven by Kalktawi et al. [14]. However, the Discrete Weibull Distribution still suffers from limitations with modeling over-dispersed data [9]. To address the challenge of modeling discrete count data and the shortcomings of the aforementioned distributions, it is required to implement a distribution function that fulfills three essential criteria: (1) the distribution must produce ordinal outputs, reflecting the inherent ordinal nature of count data; (2) the use of continuous parameters ensures a smooth loss surface, which is crucial for effective modeling and optimization; and (3) differentiable distributions are fundamental for gradient-based optimization algorithms commonly used in modeling tasks.

Considering these requirements, two innovative approaches are introduced: CONTIN and Bitwise. CONTIN is a novel distribution function specifically designed for discrete count data with continuous parameters. It draws inspiration from the Discrete Weibull (Dweib) [16] and Discrete Normal (Dnormal) [22] distributions, improving upon both by establishing a fair mapping between continuous parameters and discrete values, while maintaining sufficient stability to model over-dispersed data. On the other hand, Bitwise relies on the Bernoulli Distribution, using a binary bit-string to represent count data, where each individual bit relies on the Bernoulli Distribution. Bitwise has continuous probabilities as the input and converts the bit-string to an integer value as the output. CONTIN and Bitwise offer the flexibility to combine Categorical Cross Entropy loss with the loss specific to each distribution, a feature not readily achievable when using MSE loss. In this paper, a practical application of CONTIN and Bitwise is explored in the context of music modeling, where this combined loss can improve modeling time.

In essence, music can be distilled into two fundamental elements: pitch and time. Pitch is commonly represented in discrete frequency bins (categorical), while time in ticks is intrinsically continuous (regressive) and can not be properly modeled by current day models. Music Instrument Digital Interface (MIDI) [3] is used as a simplified representation of these musical elements in the digital realm, which has become the standard for modeling music. However, current MIDI-based models typically simplify the time domain by discretizing it into multiple bins and often quantizing it, transforming time from a continuous into a discrete representation. This simplification necessitates the use of multiple consecutive tokens to advance time, leading to an overrepresentation of time tokens compared to other musical elements. Consequently, the expressiveness of these models is limited, particularly when generating non-Western music, which often features complex rhythms and microtiming variations. Moreover, these models often rely exclusively on twelve-tone equal temperament to represent pitch, introducing a Western-music bias.

To address these biases in rhythm and pitch representation, a more faithful adherence to the MIDI standard is required, leveraging its full capabilities to represent diverse musical styles. This entails developing tokenization methods that enable models to fully comprehend the nuances of MIDI data, capturing both continuous time variations and a wider range of pitch possibilities. Models that can comprehend these nuances of MIDI data are referred in this paper as MIDI Faithful models. MIDI Faithful models not only fully use elements of MIDI as pitch and control messages, but it also represents time in a single event. This is possible by modeling the tick token as discrete count data. For this purpose, we will use the aforementioned distribution functions since they can easily be combined with the other elements of MIDI, which are categorical features.

In short, this paper investigates whether: (1) a distribution for discrete numbers using continuous parameters can be made; (2) whether this distribution works for modeling purposes; and (3) whether it works within a musical context.

2 Related work

2.1 Discrete Integer Modeling

One prominent example of distributions with continuous parameters is the Discrete Weibull distribution [16]. Collins et al. [6] models over-dispersed count data using this discrete distribution. The research shows the great potential of the Discrete Weibull distribution for regression tasks, especially in comparison with the Poisson distribution for regression [5]. Kalktawi et al. [14] stands out as a significant contribution, bridging a gap in the existing methodology of the Discrete Weibull Distribution. Kalktawi introduces a versatile and computationally efficient approach to address the common challenges in count data modeling, particularly dispersion and the presence of excessive zeros. The application of their model to diverse real-world datasets underscores its practical relevance and potential to significantly improve the analysis of count data in various fields, setting a new precedent for future research and application in discrete data analysis.

Similar to CONTIN, Tovissodé et al. [24] covers balanced discretization for modeling count data, specifically addressing the challenges encountered in regression analysis due to various levels of data dispersion equi-, under-, and over-dispersion. Balanced discretization offers a solution by discretizing continuous probability distributions in a way that expectations are preserved, enabling the exact inference for count data across all dispersion levels. The core contribution of this work is the introduction of the balanced discrete gamma distribution, a family of distributions that can accurately model count data with different dispersion levels. This method simplifies the generation of pseudo-random variate through a probabilistic rounding mechanism, making it especially suitable for count regression models where data do not adhere to the strict equi-dispersion assumption of the Poisson model.

2.2 MIDI Representation

Music Transformer [12] marked a significant advancement in music modeling, demonstrating the power of the Transformer architecture for generating music with improved long-term structure and coherence. This model was the first attempt at using a transformer model to model MIDI data. As a music representation, the researchers used the encoding proposed in Oore et al. [18]. This encoding represents MIDI in individual blocks as a sequence of: note on, velocity, time shift and note off. In between these blocks, various time shifts can be given with a maximum of 10 milliseconds per token. The velocity value is binned to 32 different dynamics. Transformers are very effective at modelling MIDI using this encoder, in comparison to an LSTM. Another interesting introduction is the use of relative position encoding for the transformer model, which hugely improves the capabilities of modeling time.

Newer models, such as REMI [13] and REMI+ [23] represent MIDI as a sequence of note and time tokens. The note tokens consist of pitch, velocity and duration tokens, while the time is represented with bar and position tokens. The

music is split up into separate bars where each bar is further divided into sub steps. The position token defines in which sub-step of the bar following note should be placed. In addition to REMI, REMI+ was introduced to add program tokens to handle multiple instruments.

Finally, the Octuple tokenizer was first introduced in Music Bert [26]. This representation uses a matrix to represent the MIDI events, so multiple tokens at once over time. At each token there is a pitch, velocity, duration, program, position (relative to bar), bar number, tempo and time signature.

Each of the aforementioned implementation are MIDI unfaithful since they still discretize and quantize the time domain across multiple tokens. However, Octuple's representation is the most similar to the tokenizer introduced in the current paper, as it uses multiple tokens over time.

3 Distribution Modeling

3.1 CONTIN

CONTIN exists in three main forms, unbounded, two-way bounded and one-way bounded. For the purposes of this research, only the one-way bounded form is explored, since ticks are only lower bounded by zero. Furthermore, CONTIN is characterized by two continuous parameters: μ (mean) and $\gamma \in \mathbb{R}|0 < \gamma < 1$ (dispersion). The distribution assigns probabilities to discrete integer values centered around μ.

The probability mass of μ is balanced between the two nearest integers, $\lfloor \mu \rfloor$ and $\lceil \mu \rceil$. If the μ value is set to exactly a discrete integer value the $\lfloor \mu \rfloor$ would be equal to $\lceil \mu \rceil$, this would cause instability in the equation. To circumvent this issue, we define $\lceil \mu \rceil$ as $\lfloor \mu \rfloor + 1$. This is achieved using a convex combination, where the weights are determined by the distance (d) of μ to its floor and ceiling, as shown in Equation 1. For instance, if μ lies exactly halfway between $\lfloor \mu \rfloor$ and $\lceil \mu \rceil$ and γ is 0, then both would have a probability of 0.5.

$$\begin{aligned} \lceil \mu \rceil &= \lfloor \mu \rfloor + 1 \\ d_f &= \mu - \lfloor \mu \rfloor \\ d_c &= \lceil \mu \rceil - \mu \\ b_f &= d_c + \gamma * (1 - d_c) \\ b_c &= d_f + \gamma * (1 - d_f) \end{aligned} \tag{1}$$

When the dispersion parameter γ is 0 all the probability is concentrated on $\lfloor \mu \rfloor$ and $\lceil \mu \rceil$. As γ increases, the probability mass of these values decreases and the distribution spreads more across the surrounding integers. To calculate the probabilities for the remaining discrete values in the distribution, separate geometric series are employed for the left and right tails. The left tail begins with the probability assigned to $\lfloor \mu \rfloor$, while the right tail starts with the probability assigned to $\lceil \mu \rceil$.

Equation 2 provides the log probability mass function (PMF) for the one-way CONTIN distribution, calculating the logarithmic probability of observing the

value x, given parameters μ and γ. To further illustrate CONTIN Fig. 1 provides a clear illustration of the discretization of the μ value.

$$
\begin{aligned}
x &= \text{target value} \\
b_p &= \begin{cases} b_f & \text{if } x \leq \mu \\ b_c & \text{else } x > \mu \end{cases} \\
d &= \begin{cases} \lfloor \mu \rfloor - x & \text{if } x \leq \mu \\ x - \lceil \mu \rceil & \text{else } x > \mu \end{cases} \\
\log P(X = x|\mu, \gamma) &= \log(b_p) + d \cdot \log(\gamma) \\
\log P(X = x|\mu, \gamma) &= \log(1 - \gamma) - \log(b_f \cdot (1 - \gamma^{\lfloor \mu \rfloor + 1}) + b_c) + \log P(X = x|\mu, \gamma)
\end{aligned}
\tag{2}
$$

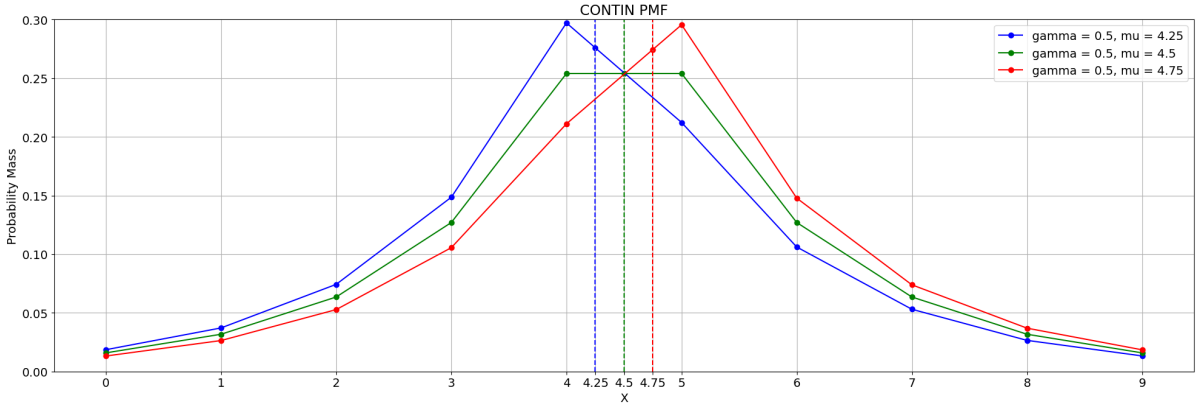

Fig. 1. Visual example of the convex combination of μ.

3.2 Modeling with CONTIN

For modeling purposes, CONTIN is employed to predict discrete integer values. To accomplish this, the model outputs values for μ and γ. The loss is then calculated by determining the negative logarithmic probability of the observed value x under the CONTIN distribution, given the μ and γ values predicted by the model. However, the raw output of a model cannot be directly used as input to the distribution function. Negative μ values would lead to an invalid distribution with a total area not equal to 1. Furthermore, γ must be strictly bounded between 1e-5 and 1 - (1e-5) to ensure stability and proper behavior of the distribution.

To address these issues, we introduce three activation functions to scale the raw model outputs into the correct representation for μ and γ, ensuring they meet the required constraints of the CONTIN distribution.

3.3 Activation functions

For each of the different distribution functions, it is important to represent the outputs of the model adequately. We propose three different activation functions for the one-way bounded approach for the μ parameter and one for the γ parameter.

First, the absolute activation function (ABS). It behaves similarly to ReLU [15], but it has an expanded search space in the direction where ReLU becomes negative. ABS is calculated with the parameters α, which determines the slope, and β, which determines the distance from zero. The final parameter l is the lower bound value of the function found in equation 3.

$$\mu = |\mu \cdot \alpha + \beta| + l \tag{3}$$

Secondly, the parabola activation function (PARA). It provides an improvement over ABS due to its softened lower bound at the bottom. PARA also uses the parameters α and β similarly to ABS. On the contrary, PARA's α value is more sensitive than in ABS and better kept below 2. l is the parameter that controls the lower bound, equation 4.

$$\mu = \alpha \cdot (\mu + \beta)^2 + l \tag{4}$$

Finally, the Sigmoid linear function (SIGLIN). The SIGLIN function is a hybrid between a sigmoid function and a linear activation function, inspired by SiL [8] and Swish [21]. Both of these activation functions improve upon the performance of ReLU [2] in deep learning tasks, However, neither fits the requirements of holding to a strict lower bound. SIGLIN incorporates two parameters, namely α and β, where α determines the slope of the linear part and the sharpness of the Sigmoid curve, and β determines the crossover point from the Sigmoid function to linear. l is the parameter that controls the lower bound. The SIGLIN function should give the model more fine-grained control over values closer to the lower bound in contrast to the values further from the lower bound, equation 5

$$\begin{aligned} z &= \frac{\mu\alpha - l - \beta}{0.5\beta} \\ \mu &= \min(\frac{1}{1+e^{-z}} \cdot 2\beta, \beta) + \max(\mu\alpha, \beta + l) - \beta \end{aligned} \tag{5}$$

Each of these activation functions are used with Dweib, Dnormal and CONTIN.

3.4 Bitwise

Bitwise presents a novel approach to integer regression by decomposing it into a series of binary classification tasks. With Bitwise, integer numbers are transformed into unsigned binary bit-strings, where the numbers of bits (up to n) determine the maximum representable integer. While this imposes a two-way

boundary, increasing the bit count significantly raises the upper limit. The representation is also flexible enough to accommodate different encodings to allow for different boundary limitations, for example encoding negative numbers through two's complement encoding.

For this study, we focus on a 32-bit unsigned binary encoding. This choice is motivated by the absence of negative values in our dataset and the sufficient range of 32 bits to encompass all data points.

To assess model performance, we employ Bitwise Binary Cross Entropy (BBCE). This loss function compares the model's output bit-string to the target integer value (which is also encoded in bit-string form). Before the model's output bit-string is computed in BBCE it is first activated using Log Sigmoid, similarly to regular activation for Binary Cross Entropy with Logits. BBCE is computed for each individual bit and then averaged across all bits.

3.5 Model

To implement these regression tasks, we employ a multi-layer perceptron (MLP) architecture with intermediate ReLU activation layers. The model consists of three layers: an input layer with n nodes, a hidden layer with 128 nodes, and an output layer with either 2 nodes (for CONTIN, Dweib and Dnormal distributions) or 32 nodes (for the Bitwise method). The 2-node output represents the input parameters for the respective distributions, while the 32-node output corresponds to the probabilities of each bit in a 32-bit string.

Loss calculation is performed using the negative log-likelihood for CONTIN, Dweib, and Dnormal, and the BBCE method for Bitwise. This is reported as the average compression loss in bits for each method.

These approaches are benchmarked against the root mean squared error (RMSE) between the model's output (single output node) and the target integer value. For comparison, the RMSE of each distribution is computed using its mean as a proxy for the predicted integer. For Bitwise, the RMSE is calculated directly by decoding the output bit-string and comparing it to the target integer.

3.6 Hyper Parameter Tuning

For each task a good α, β and learning rate has to be found. Hence, hyperparameter tuning is applied to select the best performing model, as shown in Table 1. The best performing hyper parameters are then used for the test set. Each of the models iterated over six different learning rates: 3.4e-3, 1e-3, 3.4e-4, 1e-4, 3.4e-5 and 1e-5.

3.7 Datasets

The Bicycles dataset [10] focuses on predicting the hourly number of bicycles parked in a designated space within the Capital Bike Sharing System of Washington D.C. nitially comprising 12 features, one-hot encoding of nominal features

Table 1. Tuned hyperparameter ranges

Loss function	A	B
ABS	2, 4, 10, 16	25, 50, 100, 200
SIGLIN	2, 4, 8, 16	25, 50, 100, 200
PARA	0.1, 0.25, 0.5, 1.0, 2.0	-25, 0, 25

expands the dataset to 58 features. The dataset is divided into a training set (10,948 samples, 63%), a test set (5,214 samples, 30%), and a validation set (1,217 samples, 7%).

The Upvotes dataset [17] aims to predict the number of upvotes a social media post will receive. It employs a similar train/test/validation split as the Bicycles dataset, with 207,927 training samples, 99,014 test samples, and 23,104 validation samples. The primary goal is to predict upvotes based on the post's tag, number of answers, and the poster's reputation. One-hot encoding of the tag type (the only nominal feature) and standard scaling of other features results in a total of 12 features. As shown in Figure 2, the Upvotes dataset presents a greater challenge for modeling due to the higher variance in target values compared to the Bicycles dataset.

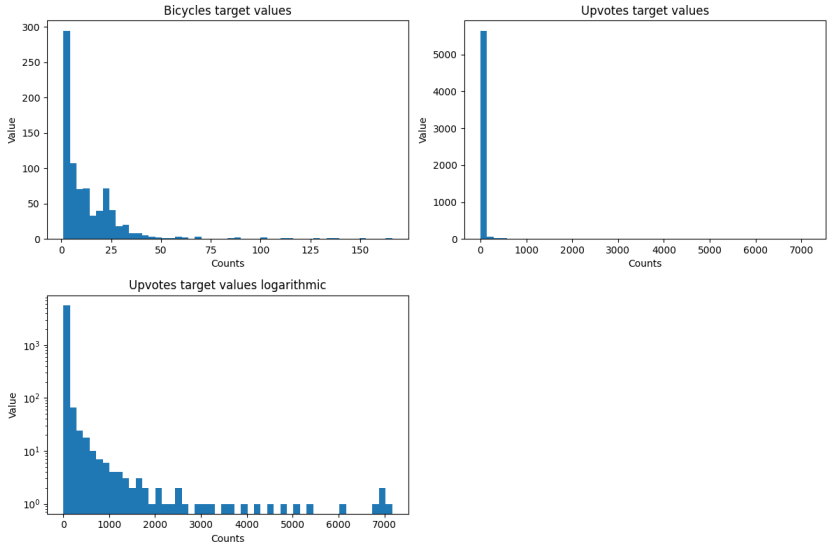

Fig. 2. Target variance

4 Distributed Modeling Results

Hyperparameter tuning for both the Bicycles and Upvotes datasets reveals that the optimal α value is 4, the optimal β value is 50. However, these values vary slightly depending on the chosen activation function. Especially, PARA stands out with an optimal α of 0.1 and a β value of 0. A comprehensive overview of the best hyperparameter values for each dataset and activation function combination can be found in Table 2.

Table 2. Best Hyperparameters Bicycles

Loss Function	Activation	Bicycles			Upvotes		
		Learning Rate	α	β	Learning Rate	α	β
CONTIN	ABS	3.4e-3	4	25	1e-4	2	25
	PARA	3.4e-3	0.1	0	3.4e-4	2.0	0
	SIGLIN	1e-3	4	50	1e-3	2	25
Dnormal	ABS	3.4e-4	2	25	3.4e-3	2	50
	PARA	3.4e-4	0.1	0	3.4e-3	0.1	0
	SIGLIN	3.4e-3	10	25	1e-3	16	25
Dweib	ABS	3.4e-4	2	25	1e-5	2	200
	PARA	3.4e-4	0.1	0	3.4e-3	2.0	25
	SIGLIN	1e-4	4	200	1e-5	16	100
Bitwise	Log Sigmoid	1e-5	na	na	3.4e-3	na	na
SQE	na	3.4e-3	na	na	3.4e-3	na	na

4.1 Results

Across both the Bicycles and Upvotes datasets, Bitwise consistently outperforms other methods, achieving both the lowest compression in bits and the lowest RMSE loss. An overview of the results can be found in Table 3.

The Dweib distribution function, when paired with the ABS activation function, demonstrates the strongest performance on the Bicycles dataset. Interestingly, CONTIN and Dnormal exhibit similar results when combined with SIGLIN.

Dnormal emerges as the most effective distribution function on the Upvotes dataset, especially when paired with the PARA activation function. While Dnormal with ABS shows comparable RMSE. CONTIN also performs reasonable well, but slightly underperforms compared to Dnormal. In contrast, Dweib proves unstable and yields high and NaN loss on this dataset.

4.2 Results Analysis

Bitwise consistently outperforms all other combinations in terms of both bits and RMSE on both the Bicycles and Upvotes datasets. This suggests that trans-

Table 3. Test set results bicycles and upvotes, best in bold

		Bicycles		Upvotes	
Loss Type	Activation	Bits	RMSE	Bits	RMSE
CONTIN	ABS	15.787	206.861	9.197	2194.675
	PARA	14.316	226.400	8.685	2182.036
	SIGLIN	10.334	229.121	9.889	2699.537
Dnormal	ABS	10.274	168.463	9.347	1642.377
	PARA	12.425	228.179	8.465	2128.102
	SIGLIN	10.774	189.500	23.232	69414.511
Dweib	ABS	7.64	148.741	nan	nan
	PARA	96.982	268.212	143.434	2170.820
	SIGLIN	9.592	259.153	138.297	2360.165
Bitwise	Log Sigmoid	**0.300**	**4.728**	**0.214**	**9.14**
Sqe	na	na	44.620	na	654.100

forming integer regression into a multi-target binary classification task is highly effective.

While CONTIN and Dnormal distribution functions show good performance, particularly with high-dimensional data, Dweib exhibits instability in those scenarios. Although distribution functions like CONTIN and Dnormal offer the advantage of calculating the explainable variance, in these experiments they did not outperform the simpler bitwise or RMSE methods in terms of predictive accuracy.

This research demonstrates the potential of Bitwise as a powerful tool for integer regression tasks, while also highlighting the importance of careful consideration when selecting distribution functions for specific datasets and problem contexts.

5 MIDI Faithful

Building upon our previous findings, where Bitwise demonstrated superior performance in integer regression tasks, the focus shifts to investigate how Bitwise, along with CONTIN and Dnormal, fare in the context of music modeling. As discussed in the introduction, these distributions are chosen for their compatibility with categorical features in music, allowing for seamless integration of their loss with Categorical Cross Entropy. This integration is not as straightforward when employing continuous distributions or MSE loss.

5.1 MIDI Faithful Representation

MIDI Faithful is a way of tokenizing MIDI that closely follows general MIDI [3]. The tokenizer represents the data across six different events: event, data1, data2, channel, instrument and tick, as described in 4. In contrast to other tokenizers, MIDI Faithful is able to capture each MIDI event without requiring multiple

timeshift events. The tokens per event are represented in a similar stacked form as Octuple [26].

Table 4. MIDI representation

Event	Data 1	Data 2	Channel	Instrument	MIDI Tick
Note on	0-127	0-127	0-15	0-127	0-∞
Note off	0-127	0-127	0-15	0-127	0-∞
Polytouch	0-127	0-127	0-15	0-127	0-∞
Pitch Bend	0-127	0-127	0-15	0-127	0-∞
Channel Pressure	0-127	0-127	0-15	0-127	0-∞

There are five different event tokens: note on, note off, channel pressure, aftertouch, and pitch. There are 128 data1, data2 and instrument tokens and 16 channel tokens. This research only uses note on and note off events, which means that data1 is only representing pitch and data2 only representing velocity.

Ticks are smaller subdivisions of time of a single beat, which is normally set to 120 Beats Per Minute (BPM) at 96 Pulses Per Quarter (PPQ). This results in a discretization of the time domain into bins of 5.2ms, the amount of milliseconds in a minute divided by the product of BPM and PPQ. In this research a resolution of 220 PPQ is used resulting in a bin size of 2.27ms. Multiple smaller bins are advantageous as it offers more information about time to models. Nonetheless, both the standard and the adjusted bin sizes are well below the Just Noticeable Difference threshold of 24 ms that humans can perceive [1]. Any note with a lower duration than this threshold cannot be easily distinguished as an individual beat in a rhythmic pattern. However, it is still possible that these shorter notes can portray additional context in music. Thus, it is desirable to keep the bin size considerable small. The ticks have an infinite number of tokens represented as integers.

Similarly to general MIDI, MIDI events are rolled out over time with the tick token relatively in length to the previous event, when an event has to start at the same time as the previous event it will be set at 0 ticks. This is illustrated in Fig 3.

5.2 Dataset

To primarily evaluate the performance of tick prediction, various synthetic datasets with increasing levels of difficulty are used, culminating in the MAESTRO V3 dataset [11] as the gold standard. MAESTRO V3 presents the most challenging scenario out of the selected dataset due to its large collection of classical piano music featuring diverse levels of virtuosity.

The synthetic datasets are randomly sampled during training, while MAESTRO is randomly sliced since it is a fixed dataset. The synthetic dataset contain 44 different tonal scales (see Appendix C Table 8) and 31 distinct rhythms (see Appendix C Table 9). The order of the scales, rhythms, and tonics are randomly

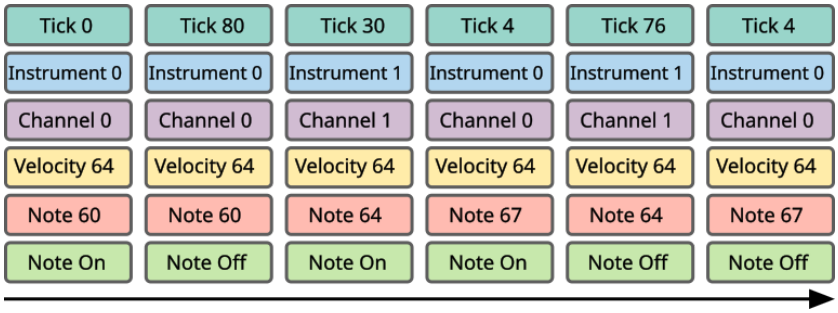

Fig. 3. Example of midi tokens over time.

shuffled after each epoch. Each of these synthetic datasets have a resolution fo 220 ticks at 120 BPM. For MAESTRO the files are loaded in a random order and the sliced subsequences are randomly shuffled.

Random Scales : The random scales dataset is a dataset with a fixed rhythm and only varying the pitch (data1). The tick for a note on event relative to the previous note is 80 MIDI ticks, and the note off event is 140 MIDI ticks. Making a single note a bit longer than half the tick resolution of 220 MIDI ticks, and giving the note a brief pause before the next note starts.

Scales : The second pitch based dataset is the scales dataset. This dataset uses the same rhythmic vocabulary as the random scales dataset. The only difference is that the scales follows one direction from tonic to tonic (the starting tone of a scale). It does a pattern up and then down in the range of the scale, when it is at its lowest point it has a 50/50 chance of either walking up an octave extra or down an extra octave.

Scales Extending : The scales extending dataset uses the same note choice rules as the scales dataset, it has an addition in the rhythmic section. Every time the scales does an octave walk in the same direction twice the note and pause length will double. Effectively slowing the song every time the generator goes up a couple of times. This is to test the effects of applying rhythmic changes at the same time as note changes.

Rhythm Only : This dataset keeps the pitches consistent, but applies different rhythms over time, similarly as changing the scales per "song". The consistent note is set at 60 (central c). This dataset is chosen to see the effectiveness of the modelling of just the rhythm.

Rhythmic Random Scales : Rhythmic random scales is a combination of random scales and rhythm only. This is to test the full capabilities of the model to create a rhythm while playing a pattern of notes.

MAESTRO : Finally, MAESTRO contains 13,424,811 training samples, 4,274,905 test samples and 1,437,884 validation samples. MAESTRO is a curated set of classical piano music with various levels of expressively. This dataset is considered the golden standard for testing any generative music models.

During training, model validation occurs every 20,000 steps. For each synthetic dataset, 30% of the data is reserved for testing. Specifically, if only scales are used in a dataset, 30% of the scales are held back for testing. In the case where both scales and rhythms are incorporated, approximately 15% of each are reserved for testing.

Given the random sampling of the synthetic datasets during training, validation can be performed on the same scales and rhythms as the training set, as the random sampling ensures sufficient diversity in both sets. Each of the tests use a random seed of 42 to ensure reproducibility.

5.3 Transformer model

Similarly to Music Transformer [12] a Transformer model [25] is employed for its scalability and aptitude in processing sequential data. The architecture of Generative Pre-Training (GPT) is adopted, using the same hyperparameters outlined in Radford et al. [19]:

- 12 heads
- 12 layers
- 512 sequence length
- 768 embedding dimension
- 0.1 dropout

In line with Music Transformer, we implement relative positional encoding to enhance the model's ability to capturing temporal information. The latest implementation of relative positional encoding by Zihang Dai et al. [7]is used, which offers improvements over earlier versions. The transformer model has two output variations, similar to the preliminary research. One variant has two output nodes for predicting the μ and σ or γ for CONTIN and Dnormal. The other has 32 output nodes with 32 bits for the Bitwise representation. The former having 93.6 M parameters and the latter 112 M parameters.

5.4 Embedding and Encoding

Each event type is processed through its own dedicated embedding layer, except tick values. As ticks are not categorical, they are not suitable for direct embedding. To achieve a similar representational effect, the ticks are encoded as a 32

bit bit-string. Furthermore, to capture the magnitude of tick values, the natural log of the tick value is added as an additional feature. This encoding is applied for Bitwise as well as CONTIN and Dnormal.

6 MIDI Faithful Results

As described in Table 5 Bitwise outperforms other methods across all datasets. Notably, CONTIN generally surpasses Dnormal, with ABS proving most effective for CONTIN, except on the MAESTRO dataset, where SIGLIN excels. This suggests that SIGLIN may be better suited for capturing complex data patterns, while underperforming on simpler datasets. In contrast, Dnormal remains largely unaffected by different activation functions, except for the Rhythmic Scales and MAESTRO datasets. This implies that datasets with rhythmic variation may necessitate careful selection of the activation function for Dnormal.

Table 5. MIDI Faithful validation results in bits, best in bold.

Loss Type	Activation	Random Scales	Scales	Scales Extending	Rhythm Only	Rhythmic Random Scales	MAESTRO
CONTIN	ABS	4.593	8.521	16.302	2.951	3.950	33.238
	PARA	3.070	9.874	16.184	5.503	5.559	54.728
	SIGLIN	11.238	14.365	18.471	5.801	11.735	14.200
Dnormal	ABS	16.810	21.990	21.843	16.382	14.509	24.489
	PARA	16.810	21.990	21.843	16.326	17.329	25.398
	SIGLIN	16.810	21.990	21.843	16.327	18.818	24.558
Bitwise	Log Sigmoid	**0.592**	**6.128**	**5.770**	**0.096**	**0.972**	**8.951**

The total loss comprises a combination of Cross Entropy for embedded events and the specific loss function employed for tick prediction. Among the datasets, Scales and Scales Extending exhibit weaker performance, likely attributable to their distinct note generation algorithm. To further investigate this, Table 6 compares the Random Scales dataset against the Scales and Scales Extending datasets.

Dnormal consistently demonstrates inferior performance when modeling ticks compared to learning the notes, indicating its unsuitability for tick modeling in this context. Conversely, CONTIN varies: it balances both components in the Random Scales dataset, excels at tick modeling in the Scales dataset, but underperforms in the more complex Scales Extending dataset. Bitwise proves consistently effective at tick modeling across all datasets. However, its data1 loss remains relatively high for most datasets.

As Bitwise outperforms the other distributions, we only report the results of the test set on Bitwise. Table 7 shows a consistent pattern where the test loss mirrors the validation loss across all datasets, indicating that the model is not overfitting.

Table 6. MIDI Faithful validation results in bits of Ticks against Data1 in Random Scales, Scales and Scales Extending. best in bold.

		Random Scales		Scales		Scales Extending	
Loss Type	Activation	Tick	Data1	Tick	Data1	Tick	Data1
CONTIN	ABS	2.563	**2.023**	**2.421**	6.097	10.474	**5.819**
	PARA	1.755	**1.311**	**3.784**	6.086	10.389	**5.788**
	SIGLIN	5.784	**5.257**	**5.784**	8.389	10.245	**8.179**
Dnormal	ABS	16.325	**0.479**	16.325	**5.663**	16.325	**5.517**
	PARA	16.325	**0.479**	16.325	**5.663**	16.325	**5.517**
	SIGLIN	16.325	**0.479**	16.325	**5.663**	16.325	**5.517**
Bitwise	Log Sigmoid	**0.002**	0.587	**0.001**	6.127	**0.030**	5.738

Table 7. MIDI Faithful test results in bits.

	Random Scales	Scales	Scales Extending	Rhythm Only	Rhythmic Random Scales	MAESTRO
Bitwise Validation	0.592	6.128	5.770	0.096	0.972	8.951
Bitwise Test	0.868	5.121	4.932	0.093	1.242	7.110

7 Conclusion

In this research, we demonstrated the challenges of modeling discrete count data. While both CONTIN and Dnormal outperformed Dweib when it comes to distributed modeling, they underperform in comparison to the standard RMSE loss. This demonstrates that while it is possible to create a distribution function for discrete numbers using continuous parameters, it does not work well for modeling purposes over existing methods.

Conversely, Bitwise not only demonstrates efficacy in modeling, but also has continuous parameters for each of its output bits, which use an underlying Bernoulli distribution. Furthermore, it outperforms the commonly used RMSE loss. A potential limitation of Bitwise lies in its two-way bounded nature. However, this constraint might be mitigated by exploring alternative decoding methods like Two's Complement, which would enable Bitwise to predict negative numbers. Additionally, investigating other decoders could potentially extend its capabilities to for instance decimal numbers.

Our findings further reveal that Bitwise is most suitable among the tested distribution functions for modeling MIDI data, bringing MIDI Faithful modeling a step closer to reality. While CONTIN exhibits performance improvements over Dnormal, suggesting greater stability for combined learning tasks like MIDI faithful modeling, it still falls short of Bitwise's performance gains.

A notable trade-off exists between modeling ticks and other events, with Bitwise consistently excelling at tick modeling. One potential improvement to the loss of other events could be to use Bitwise for modeling these events. Since these events typically require fewer bits for representation (e.g., 7 bits for Data1 and

Data2), this approach could be feasible. Alternatively, employing larger Transformer models like GPT-2 variations [20] might also enhance Data1 modeling.

In conclusion, MIDI Faithful modeling appears to be within reach, particularly with the use of Bitwise and the proposed tokenizer. However, this tokenizer could benefit from improvements, such as incorporating pitch bends and other control events that are used in general MIDI. Pitch bending is a possible event token, which is not fully implemented in the current tokenizer, this would require the Data1 and Data2 tokens to be converted into a bit-string and converted back into a pitch bend integer.

Finally, it is important to acknowledge that this study did not directly compare the proposed MIDI Faithful tokenizer against other existing tokenizers like REMI+, Octuple, or Music Transformer due to the complexities involved. However, it is worth noting that current MIDI tokenizers often suffer from information loss during tokenization. To facilitate a comprehensive comparison, future research could incorporate this information loss into the total compression loss (in bits) for a more accurate evaluation.

Acknowledgments. This study would not have been possible without the support of DAS6 [4] and Snellius. Both supercomputer clusters were of great help running the experiments.

References

1. Adelstein, B., Begault, D., Anderson, M., Wenzel, E.: Sensitivity to haptic-audio asynchrony. pp. 73–76 (11 2003). https://doi.org/10.1145/958432.958448
2. Agarap, A.F.: Deep learning using rectified linear units (relu) (2019)
3. Association, M.M.: MIDI 1.0 Detailed Specification (1996), https://midi.org/midi-1-0-detailed-specification, 1996 Revision
4. Bal, H., Epema, D., de Laat, C., van Nieuwpoort, R., Romein, J., Seinstra, F., Snoek, C., Wijshoff, H.: A medium-scale distributed system for computer science research: Infrastructure for the long term. Computer **49**(05), 54–63 (may 2016). https://doi.org/10.1109/MC.2016.127
5. Cameron, A.C., Trivedi, P.K.: Regression Analysis of Count Data, vol. 53. Cambridge University Press (2013)
6. Collins, K., Waititu, A., Wanjoya, A.: Discrete weibull and artificial neural network models in modelling over-dispersed count data. Int J Data Sci Anal **6**(5), 153–62 (2020)
7. Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q.V., Salakhutdinov, R.: Transformer-xl: Attentive language models beyond a fixed-length context (2019), https://arxiv.org/abs/1901.02860
8. Elfwing, S., Uchibe, E., Doya, K.: Sigmoid-weighted linear units for neural network function approximation in reinforcement learning (2017)
9. Englehardt, J., Li, R.: The discrete weibull distribution: An alternative for correlated counts with confirmation for microbial counts in water. Risk analysis : an official publication of the Society for Risk Analysis **31**, 370–81 (11 2010). https://doi.org/10.1111/j.1539-6924.2010.01520.x
10. Fanaee-T, H.: Bike Sharing. UCI Machine Learning Repository (2013), DOI: https://doi.org/10.24432/C5W894

11. Hawthorne, C., Stasyuk, A., Roberts, A., Simon, I., Huang, C.Z.A., Dieleman, S., Elsen, E., Engel, J., Eck, D.: Enabling factorized piano music modeling and generation with the MAESTRO dataset. In: International Conference on Learning Representations (2019), https://openreview.net/forum?id=r1lYRjC9F7
12. Huang, C.A., Vaswani, A., Uszkoreit, J., Shazeer, N., Hawthorne, C., Dai, A.M., Hoffman, M.D., Eck, D.: An improved relative self-attention mechanism for transformer with application to music generation. CoRR **abs/1809.04281** (2018), http://arxiv.org/abs/1809.04281
13. Huang, Y.S., Yang, Y.H.: Pop music transformer: Beat-based modeling and generation of expressive pop piano compositions. In: Proceedings of the 28th ACM International Conference on Multimedia. p. 1180–1188. MM '20, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3394171.3413671, https://doi.org/10.1145/3394171.3413671
14. Kalktawi, H., Vinciotti, V., Yu, K.: A simple and adaptive dispersion regression model for count data. Entropy **20** (11 2015). https://doi.org/10.3390/e20020142
15. Maas, A.L.: Rectifier nonlinearities improve neural network acoustic models (2013), https://api.semanticscholar.org/CorpusID:16489696
16. Nakagawa, T., Osaki, S.: The discrete weibull distribution. IEEE Transactions on Reliability **R-24**(5), 300–301 (1975). https://doi.org/10.1109/TR.1975.5214915
17. Naseer, U.: Predict the number of upvotes a post will get. https://www.kaggle.com/datasets/umairnsr87/predict-the-number-of-upvotes-a-post-will-get (2020), accessed: March 20, 2024
18. Oore, S., Simon, I., Dieleman, S., Eck, D., Simonyan, K.: This time with feeling: Learning expressive musical performance (2018)
19. Radford, A., Narasimhan, K., Salimans, T., Sutskever, I.: Improving language understanding by generative pre-training (2018)
20. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I.: Language models are unsupervised multitask learners (2019)
21. Ramachandran, P., Zoph, B., Le, Q.V.: Searching for activation functions (2017)
22. Roy, D.: The discrete normal distribution. Communications in Statistics-theory and Methods - COMMUN STATIST-THEOR METHOD **32**, 1871–1883 (01 2003). https://doi.org/10.1081/STA-120023256
23. von Rütte, D., Biggio, L., Kilcher, Y., Hofmann, T.: Figaro: Generating symbolic music with fine-grained artistic control (2024)
24. Tovissodé, C.F., Honfo, S.H., Doumatè, J.T., Glèlè Kakaï, R.: On the discretization of continuous probability distributions using a probabilistic rounding mechanism. Mathematics **9**(5) (2021). https://doi.org/10.3390/math9050555, https://www.mdpi.com/2227-7390/9/5/555
25. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need (2023)
26. Zeng, M., Tan, X., Wang, R., Ju, Z., Qin, T., Liu, T.Y.: Musicbert: Symbolic music understanding with large-scale pre-training (2021)

A Activation Functions

A.1 Activation functions visualization

A visualization of the activation functions can be found under Fig. 4.

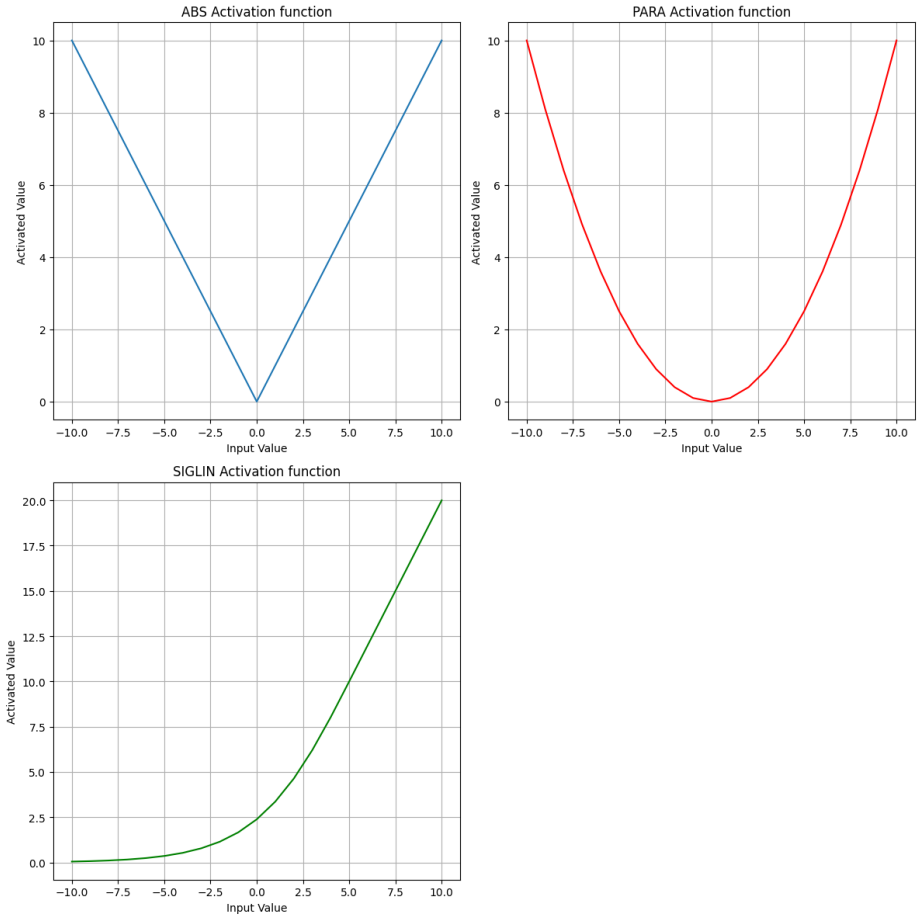

Fig. 4. Activation functions ABS, PARA and SIGLIN

A.2 ABS Backward

For ABS we implemented a stable version of the backward pass in PyTorch as described in Equation 6

$$\frac{\partial L}{\partial x} = (a \cdot \text{sign}(x + b) + l) \cdot \frac{\partial L}{\partial y} \tag{6}$$

A.3 PARA Backward

For PARA we implemented a stable version of the backward pass in PyTorch as described in Equation 7

$$\frac{\partial L}{\partial x} = ((2ax + \frac{b}{2}) + l)\frac{\partial L}{\partial y} \tag{7}$$

A.4 SIGLIN Backward

For SIGLIN we implemented a stable version of the backward pass in PyTorch as described in Equation 8

$$\begin{aligned} s &= 2b \cdot \text{sigmoid}(\frac{2ax - 2l - 2b}{b}) \\ \frac{\partial L}{\partial x} &= \begin{cases} 2a \cdot s \cdot (1-s) \cdot \frac{\partial L}{\partial y}, & \text{if } s < b \\ 0, & \text{otherwise} \end{cases} \end{aligned} \tag{8}$$

A.5 SIGLIN

SIGLIN could be further refined for better gradients. Later in the research, we discovered that Log Sigmoid could be modified to have a similar effect, but with smoother gradients due to Log Sigmoid not containing Min and Max functions. The main difference is the steepness of the cross-over point. This is not used in the research, but valuable for further research. As described in Equation 9, the α parameter determines the slope, the β parameter the crossover point and the l parameter the lower bound of the function.

$$\mu = -\log(\frac{1}{1 + e^{\mu\alpha}}) \cdot b + l \tag{9}$$

B CONTIN

As part of the early preliminary research of CONTIN the Two-Way bounded version of the distribtion has been explored by the means of a Variational Auto Encoder (VAE). Besides this early exploration phase, the unbounded version of CONTIN has also been fully implemented, but has never been properly used. In Equation 10 the unbounded version of CONTIN can be found, while in Equation 12 the two-way version can be found. Both variants use the same variables from Equation 1.

$$\begin{aligned} x &= \text{target value} \\ b_p &= \begin{cases} b_f & \text{if } x \leq \mu \\ b_c & \text{else } x > \mu \end{cases} \\ d &= \begin{cases} \lfloor \mu \rfloor - x & \text{if } x \leq \mu \\ x - \lceil \mu \rceil & \text{else } x > \mu \end{cases} \\ \log P(X = x|\mu, \gamma) &= \log(b_p) + d \cdot \log(\gamma) \\ \log P(X = x|\mu, \gamma) &= \log(1 - \gamma) - \log(1 + \gamma) + \log P(X = x|\mu, \gamma) \end{aligned} \tag{10}$$

$$
\begin{aligned}
x &= \text{target value} \\
l &= \text{lower bound} \\
u &= \text{upper bound} \\
b_p &= \begin{cases} b_f & \text{if } x \leq \mu \\ b_c & \text{else } x > \mu \end{cases} \\
d &= \begin{cases} \lfloor \mu \rfloor - x & \text{if } x \leq \mu \\ x - \lceil \mu \rceil & \text{else } x > \mu \end{cases}
\end{aligned} \tag{11}
$$

$$\log P(X = x|\mu, \gamma) = \log(b_p) + d \cdot \log(\gamma)$$

$$\log P(X = x|\mu, \gamma) = \log(1 - \gamma) - \log(b_f \cdot (1 - \gamma^{\lfloor \mu \rfloor - l + 1}) + b_c \cdot (1 - \gamma^{u - \lceil \mu \rceil + 1})) + \log P(X = x|\mu, \gamma)$$

B.1 PMF

The PMF function of CONTIN can be described as Equation 12 and seen in Fig. 5.

$$\text{PMF} = \exp(\log P(X = x|\mu, \gamma)) \tag{12}$$

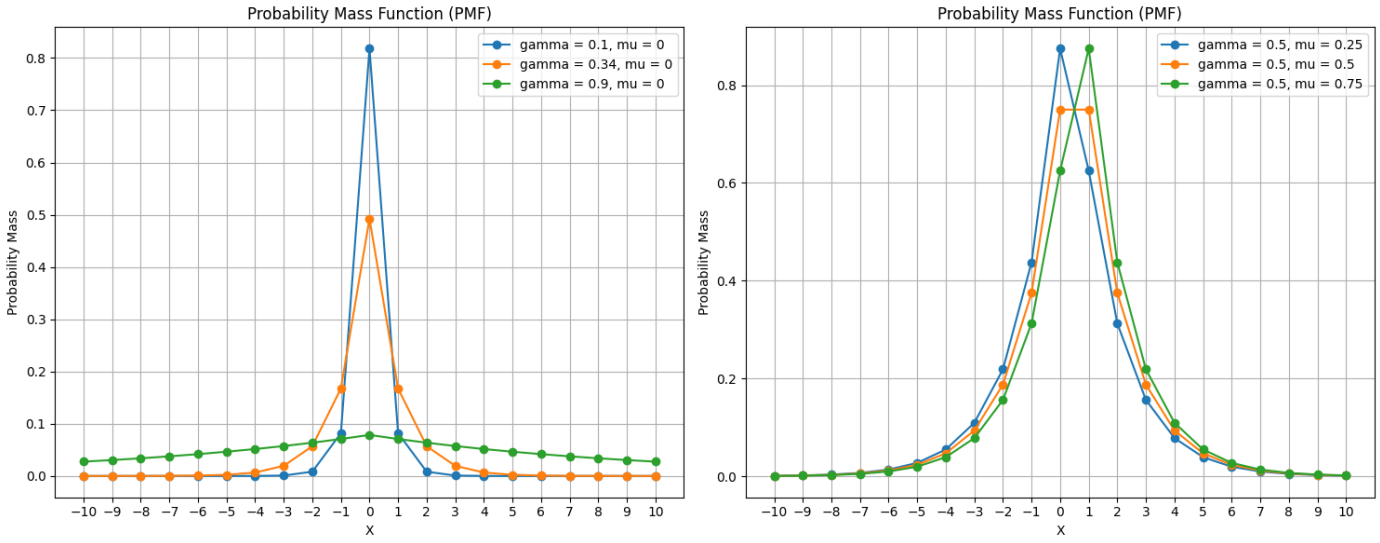

Fig. 5. CDF of CONTIN

B.2 CDF

The CDF function of CONTIN can be seen in Fig. 6.

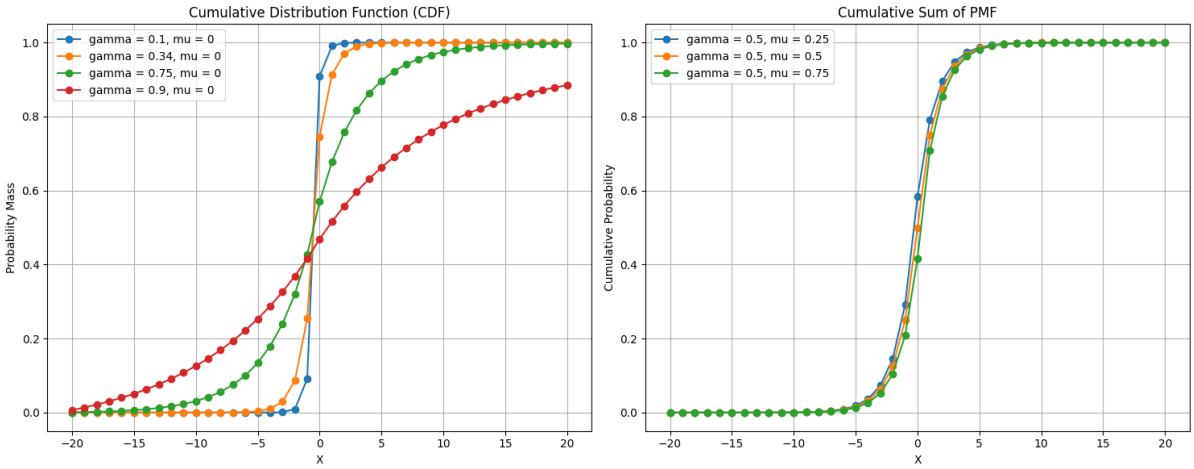

Fig. 6. CDF of CONTIN

B.3 One-way CONTIN with $\mu = 0$ and γ of 0.5

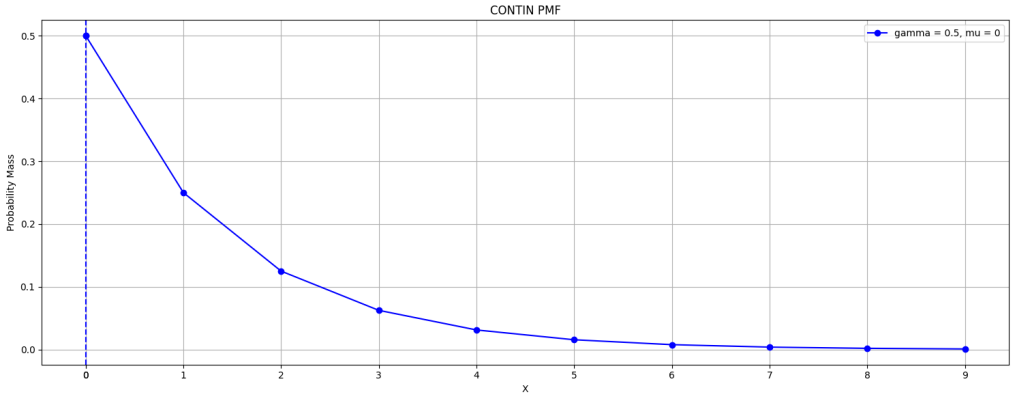

Fig. 7. One-way CONTIN with $\mu = 0$ and γ of 0.5

B.4 CONTIN for VAE

The VAE is trained on the MNIST dataset, since this is a simple example that has been proven before to work with VAE. CONTIN as a distribution function is compared to the Gumbel Softmax distribution as well as the Discrete Weibull distribution.

This research has not been completed due to time constraints and the shift in focus to the one-way bounded version of CONTIN that is used for MIDI Faithful. Nonetheless, there are some findings in regard to using CONTIN for image generation. CONTIN in this version uses a sigmoid activation on the mu variable that is scaled to the upper and lower bound of the data.

In Fig. 8 the flow of the model can be found. On the left the layers of the encoder and decoder, in the middle the calculation of the posterior and prior, and on the right side the respective three loss calculations.

Training has been inconclusive, but a strong indication that CONTIN is a good option for image generation using VAE gaining similar loss values to the Gumbell Softmax implementation. These results are not reported due to time constraints limiting the ability to properly interpret these results. Furthermore, it is suggested that applying batch normalization in the final layer improves the results of CONTIN significantly. This was tested to discover whether this addition could have been the reason for the better performance of CONTIN in Transformer models in comparison to other distribution methods. In Figures 9 and 10, different visual results can be found of the different loss functions after training the VAE.

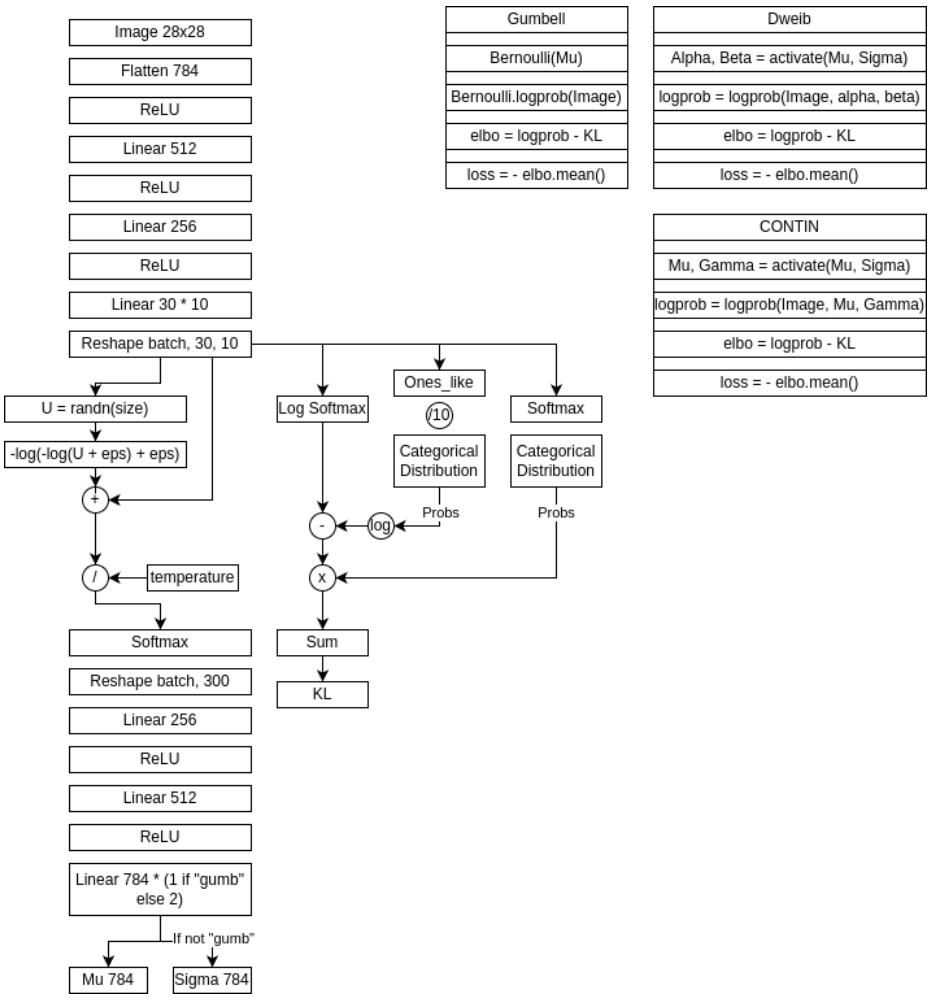

Fig. 8. Flowchart of VAE

Fig. 9. Generated images of CONTIN, left step 540, right step 2000

Fig. 10. Generated images of Gumbell, left step 540, right step 2000

C Datasets

Table 8 describes a list of scales and Table 9 the different rhythmic patterns used by the synthetic datasets.

Table 8. Different scales used in the synthetic datasets

Acoustic scale	Aeolian mode or natural minor scale
Algerian scale	Altered scale or Super Locrian scale
Augmented scale	Bebop dominant scale
Blues scale	Chromatic scale
Dorian mode	Double harmonic scale
Enigmatic scale	Flamenco mode
Gypsy scale	Half diminished scale
Harmonic major scale	Harmonic minor scale
Hirajoshi scale	Hungarian minor scale
Hungarian major scale	In scale
Insen scale	Ionian mode or major scale
Istrian scale	Iwato scale
Locrian mode	Lydian augmented scale
Lydian mode	Major bebop scale
Major Locrian scale	Major pentatonic scale
Melodic minor scale	Minor pentatonic scale
Mixolydian mode or Adonai malakh mode	Neapolitan major scale
Neapolitan minor scale	Non-Pythagorean scale
Octatonic scale	Persian scale
Phrygian mode	Quarter tone scale
Scale of harmonics	Slendro
Tritone scale	Whole tone scale

Table 9. Different rhythms used in the synthetic datasets, 1=quarter note, 0.5=eight note, 0.25=sixteenth note, 0.75=eight dotted, 1.5=quarter dotted

1, 0.5, 0.5, 1, 0.5, 0.25, 0.25	0.5, 1, 1, 0.5, 1
1, 0.5, 0.5, 1, 1	1.5, 0.5, 0.5, 0.5, 1
0.5, 1, 0.5, 1, 0.5, 0.5	1, 1, 1, 1
0.5, 1.5, 1, 0.5	1, 0.5, 1.5, 1
1, 1, 0.5, 0.5, 1	0.5, 1, 1, 1, 0.5
0.75, 0.75, 0.75, 0.75, 0.5, 0.5	1.5, 0.5, 1, 1
0.5, 0.5, 1, 1, 1	0.5, 0.5, 0.5, 1.5, 1
1, 1, 0.5, 0.5, 0.5, 0.5	0.5, 1, 0.5, 0.5, 1, 0.5
1, 1, 0.5, 1, 0.5	0.5, 1, 1, 0.5, 0.5, 0.5
0.5, 0.5, 0.5, 1, 1, 1	0.5, 0.5, 0.5, 0.5, 0.5, 1, 0.5
0.5, 1, 1, 0.5, 1	1, 1, 1, 0.5, 0.5
1.5, 0.5, 0.5, 0.5, 1	1, 0.5, 0.5, 0.5, 1, 0.5
1, 1, 1, 1	1, 0.5, 0.5, 0.5, 0.5, 1
1, 0.5, 1.5, 1	0.5, 1, 0.5, 1, 1
0.5, 1, 1, 1, 0.5	0.5, 1.5, 0.5, 0.5, 1
1.5, 0.5, 1, 1	1, 0.5, 1, 0.5, 1
0.5, 0.5, 0.5, 1.5, 1	0.5, 0.5, 1.5, 1, 0.5
0.5, 1, 0.5, 0.5, 1, 0.5	0.5, 1.5, 1, 1
0.5, 1, 1, 0.5, 0.5, 0.5	1.5, 1.5, 1
0.5, 0.5, 0.5, 0.5, 0.5, 1, 0.5	0.5, 0.5, 0.5, 0.5, 1, 1
0.5, 0.5, 1, 1, 0.5, 0.5	