**Rhythm Abstraction and Generation with Simple RNNs**

**Background**

Music consists of cued associations that unfold over time. Its temporal nature makes it likely that humans encode musical information in varying cycles at different points in time. Empirical evidence demonstrates the unlikelihood of encoding and integration of sequences accomplished by single neurons, indicating that storage of complex musical associations can span from seconds to minutes.[[1]](#footnote-1) Leaver et al. fMRI studies indicated that the brain structures recruited for learning sound sequences are similar to those for learning motor sequences, therefore retrieving stored sequences involves the anticipation of upcoming information about an event before the event occurs.[[2]](#footnote-2) These studies provide strong evidence that support neural networks can be used model how increased familiarity to music strengthens associations between small and large structural units which become more complex over time.

Elman worked with simple neural networks to simulate the emergence of words from a stream of letters and the sentences from a stream of words. His work indicated that real structure is not in the symbols used but in the input stream (or sequence) itself.[[3]](#footnote-3) Both music and language are comprised of sequences generated from a finite set of sounds organized into discrete categories which facilitate representation and memory. The many parallels lead between the music and language lead to the assumption that a similar recurrent neural network model to the ones used in Elman’s studies in *Finding structure in time* could simulate musical sequences. Drum patterns can be thought of as simple sentences, where each point in time encapsulates information about each drum sound being on or off. If we treat the information at each time step as part of a word, then we should be able to use a recurrent neural network model to predict the next rhythmic word using the information from the current input and the internal representation of the state of the network from the previous time-step.

**Aim**

Use a recurrent neural network in a predictive setting to explore rhythm abstraction and generation. Understand the network model by varying the input sequence stream.

**Network Task**

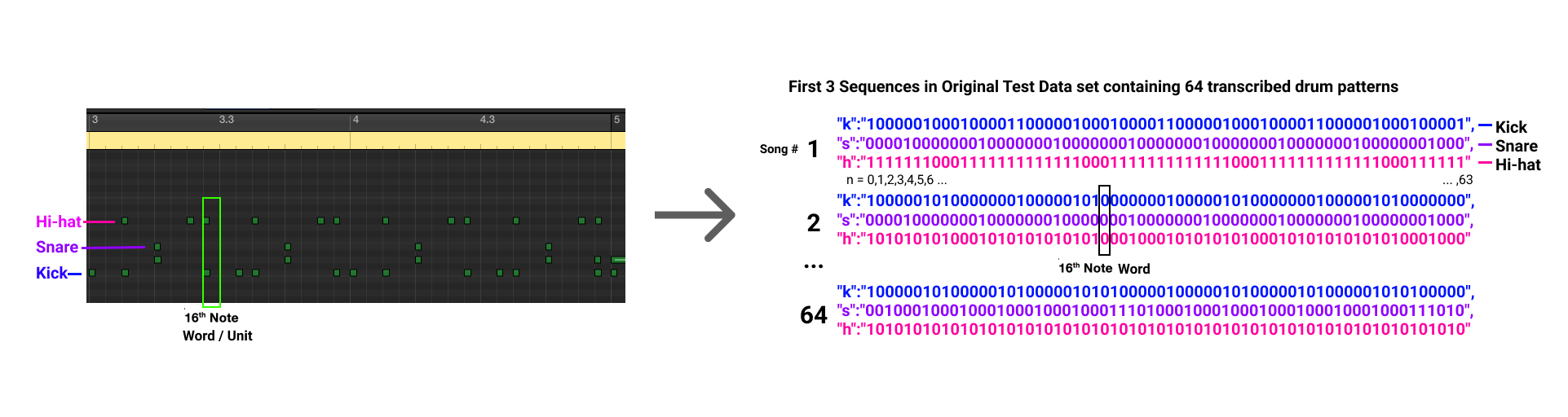
Predict the next word item in the string, using the item at time n, plus an internal representation of the state of a set of hidden units from the previous time step.

**Stimuli Creation**

64 rhythmic sequences were transcribed from different hip-hop song MIDI files (the majority from the artist Tupac). MIDI files were loaded into a digital audio workstation (DAW) and quantized to a 16th note grid. The kick, snare and hi-hat drum note onset patterns were transcribed from the DAW grid into separate sequences of 1s and 0s. (See Figure 1.0 below) Only note patterns for these main three drum types, featured in all songs, were transcribed. Any additional percussive sounds, eg. the cymbal or sound effects, were ignored.

Figure 1.0 shows the original first three sequences where 0’s and 1’s represent kick, snare and hi-hat note onsets at 64 time steps representing 16th notes (four time steps = single beat in one measure). The information at a given time step was considered part of a word unit to be inputted into the model at time n. In first sequence [1,0,1] at n = 0, [0,0,1] at n = 1, would be a three-unit word input that encapsulated relational information between the kick, snare and hi-hat drum hits. Before fed into the model, a fourth, periodic input unit was added to each word to assist with keeping track of time. The a fourth counter input was intended to assist the model as counting one-e-and-a does for beginner drummers.

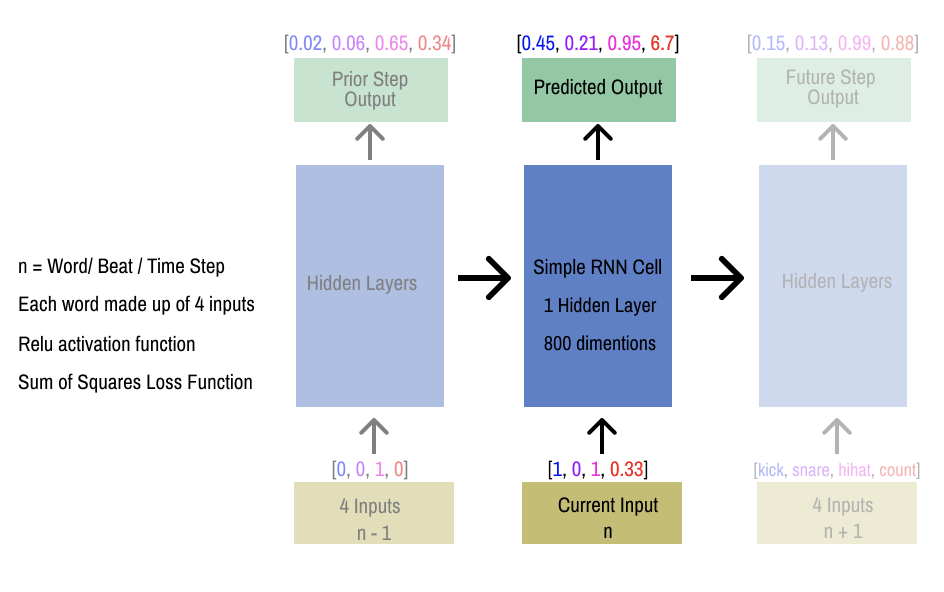
**Figure 1.0**



A single word consisted of a list of four numbers: the first three, either 0’s or 1’s signified kick, snare and hi-hat note onsets at a given time-step; and the fourth, the added counter input with a fixed period of n = 4 (counting up to one quarter note) to help keep track of time. This counter value was normalized by dividing by the period (4) to scale the number between 0 and 1.

**Model Description and Training**

A simple RNN model built by Steven Hanson in Python using the TensorFlow was modified to fit the rhythmic sequence prediction task. An input vector, consisting of a list of four numbers at time n, was passed through a hidden layer with 1,000 dimensions that also received input from the weights of the hidden layer from the previous time step to produce an output vector the same size as the input, that signified the networks prediction of the next input word unit at n+1 (or target). Figure 2.0 provides a general overview of the recurrent network structure.

**Figure 2.0**

The flexible TensorFlow Python API allowed for the assessment of different types of recurrent network cells used with TensorFlow’s core RNN methods (line 45 in multi\_pred\_rnn.py). Setting

cell = tf.contrib.rnn.**BasicRNNCell**(hid\_dim,inp\_dim,activation=tf.nn.relu)

implemented the most basic RNN cells where, output = .

Cell = tf.contrib.rnn.**LSTMCell**(hid\_dim,inp\_dim,activation=tf.nn.relu)

implemented long short-term memory unit RNN cells, expected to increase performance of the network.

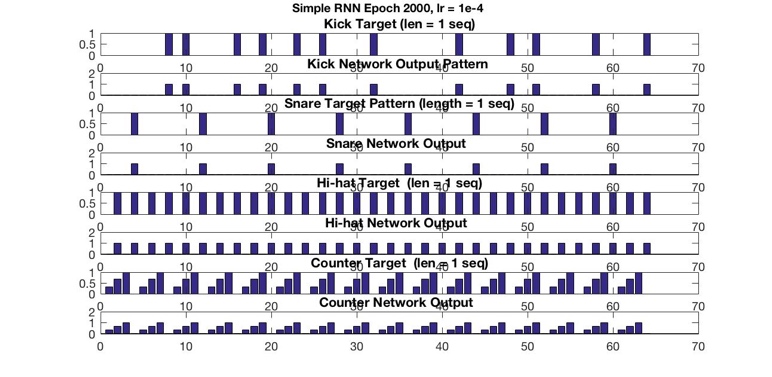
A Rectified Linear Unit (ReLU) activation function, defined as where x in the input to the unit, was used instead of a sigmoid or tanh non-linear functions to be more computationally efficient and accelerate the convergence of stochastic gradient descent.[[4]](#footnote-4) The hidden to output layer reused the same weights for each word in a given sequence and a fully connected layer that computed each output linearly: . The loss was computed using sum-of-squares differences between the predicted value (output) and true value (target = input at n+1) at each time step.

**Experiments and Results**

5 separate experiments were designed to methodologically test the networks ability to learn one or two simple sequences. In all experiments, the simple RNN and LSTM models were assessed over at least 7 separate runs and detailed reports of their behavior can be found in the tables in Appendix A, B, C, D and E outlining the general learning behavior of the networks with varied input sequence form.

**Experiments 1, 2 and 3: Varying Input Number and Length Without Run-up**

In Experiment 1, a simple RNN and RNN with LSTM cells with the same parameters were trained on a single sequence of length 64 words units. Both models successfully fitted the target sequence almost perfectly with the exception of short interpolated ramp up present at beginning of sequences, slightly more present in the LSTM RRN. Figure 1.0 shows the simple RNN output perfectly fitting the single target sequence by epoch 2,000.

**Figure 3.0**

Experiment 2 compared the behavior of the simple RNN and the LSTM RNN trained on two input sequences. Discrepancies at the beginning of output patterns in both network models reflected a difficulty distinguishing the starting point of a sequence. Experiment 3 consisted of networks trained on one single long sequence consisting of the two length 64 patterns concatenated. The LSTM RNN rapidly identified larger structural features first which can be seen by a division of the output sequences in half in the hi-hat (see C2 in Appendix). Results overall indicated that the networks had more difficulty reproducing the details of a longer pattern but was able to pick up on larger structural qualities.

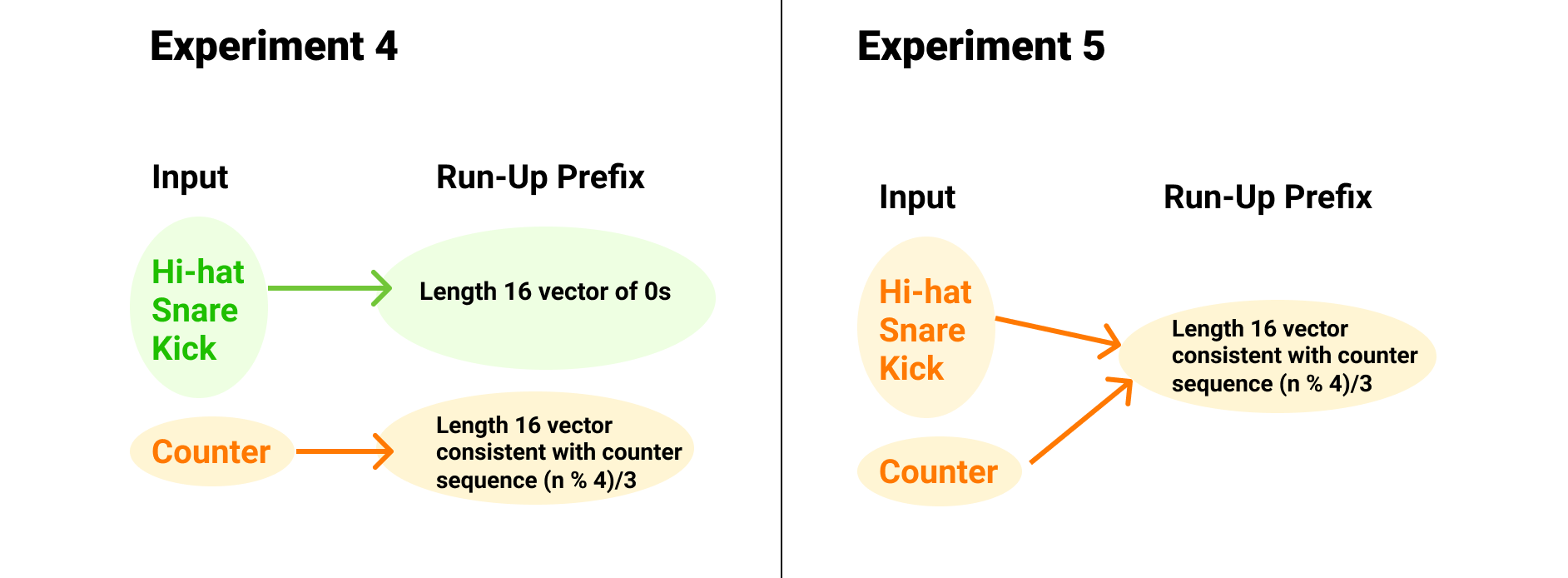
A ramp up at the beginning of the output sequences was visibly present to some degree in the output patterns for all settled networks in Experiments 1, 2 and 3, trained on two sequences. This ramp up, varying between 2-6 word units, was characterized by increasing values, like a fade in, of the interpolated output at the start of a sequence before it settled into a consistent form featuring patterns abstracted from the input stream.

This common behavior reflected a difficulty in handling the beginning of sequences and lead to the question of whether the network would be improved by additional inputs marking a new sequence start. The option to insert “run up” inputs to the beginning of sequences was implemented in Experiments 4 and 5 to act as a marker preparing the network for each sequence (see def gen\_runup lines 8-16 in lists.py). The run up inputs were inspired by the anticipatory function of a count off or metronome to orient a band before beginning playing.

**Experiments 4 and 5: Exploring the effect of adding run up inputs**

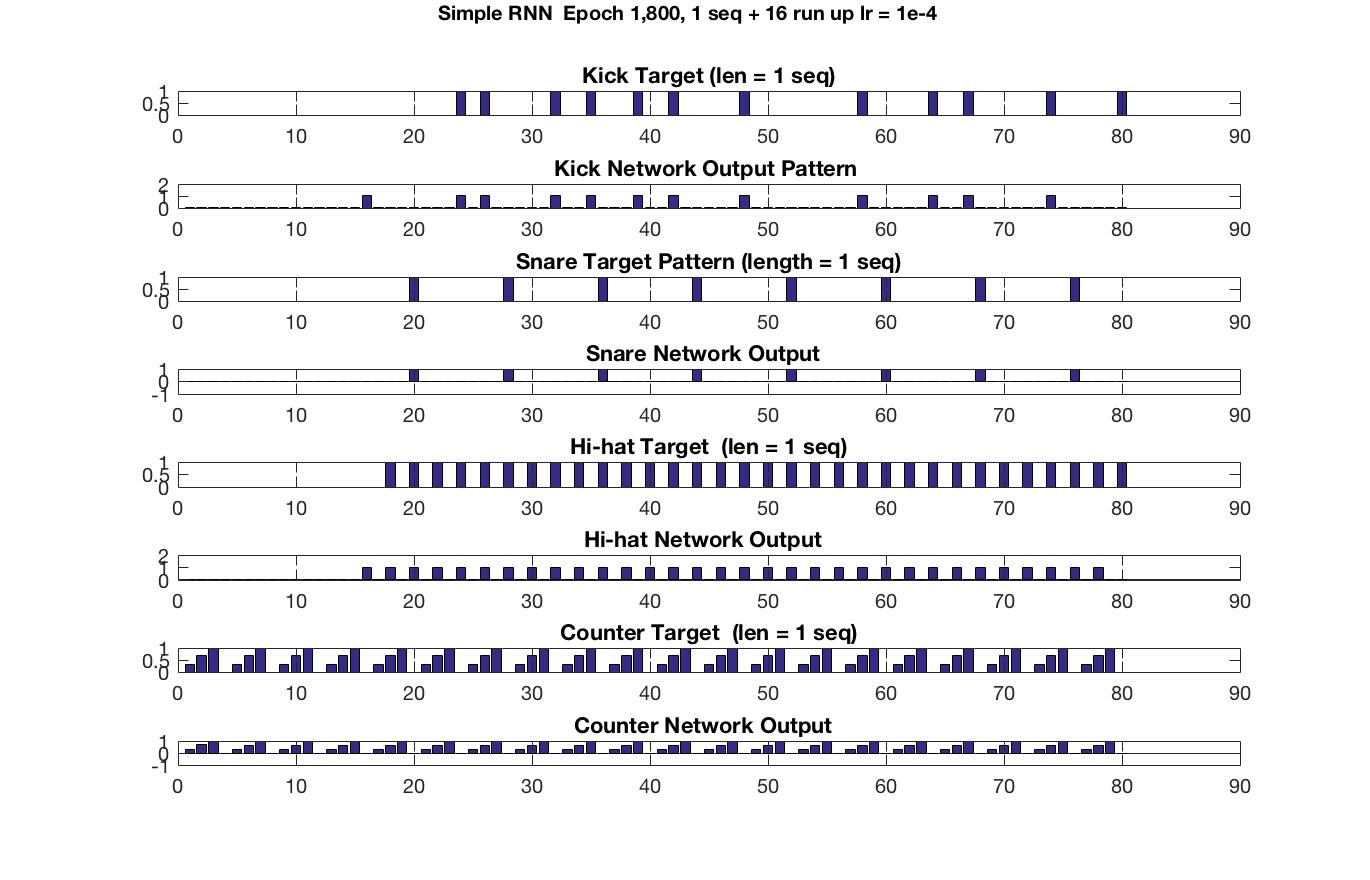
To be consistent with traditional count offs in music, 16 time steps, equaling one measure, was the length chosen for the run up. The run up sequences appended to the beginning of the four input patterns took the form outlined in Figure 4.0 below.

**Figure 4.0: Values for Run Up Inputs**



Both the simple RNN and the LSTM models were able to reproduce a single target sequence using the zeroes run up format from Experiment 4. Figure 4.5 displays the target and output sequences for a simple RNN trained on a single sequence length 64 + 16 counter inputs that fir the data slightly more accurate then the LSTMs output.

**Figure 4.5: Simple RNN with**



Results for two sequences + length 16 zeroes run up varied much more between the RNN and LSTM. The RNN more frequently picked up on details of the input sequence while the LSTM output patterns resembled more, higher level, periodic features found early on in learning.

Network behavior during Experiment 5 indicated changing all of the run up inputs to match the continuous cyclic counter unit switched the direction of the ramp up, making it go from high to low before the start of the sequence. Overall, both the simple RNN and LSTM models performed marginally better with the zero run up values in Experiment 4 than Experiment 5 with uniform run up values mimicking the periodic counter cycling [0, .33, .67, 1, 0, …].

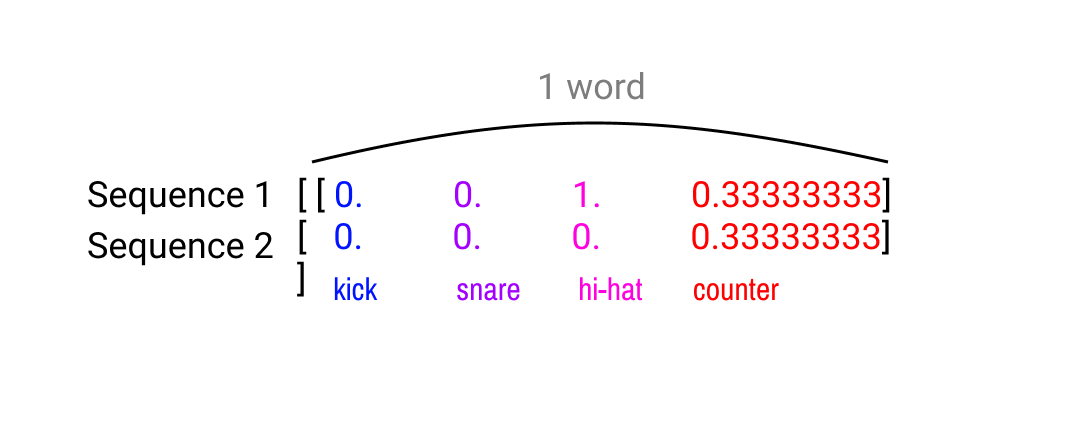
With a fixed hidden unit size of 1,000 dimensions and learning rate of 1e-4 across all experiments, networks with simple RNN cells performed more efficiently then with LSTMs. This contradicts most literature discussing using RNNs for music generation. [[5]](#footnote-5) Due to the vanish gradient, RNNs are limited to looking back only a few steps, and there have been many reported difficulties training RNNs to perform tasks with temporal contingencies in input sequences spanning over long intervals. [[6]](#footnote-6) This explains the poorer performance with a single longer sequence in Experiment 3.

When dealing with two sequences the LSTM network often went awry either suddenly outputting nans or having a large jump in the error which never settles down again. This suggests that the methods implemented from the TensorFlow library may be more optimized for the simple RNN cells over LSTM. Adjusting the learning rate sometimes improved the LSTM models, but inconsistently across different experiments. Higher success in some runs with the learning rate adjusted slightly up and down reflected that the LSTM need more carefully tuned parameters then the simple RNN cells. Although there was a large amount of variance in results, learning rates much lower then 1e-5 found to be too slow and learning rates faster then 1e-4 often resulted in outputted nans and void error functions.

**Discussion**

LSTMs had more difficulty with the task of fitting two sequences and stunted learning often lead to interpolated outputs with a periodic nature. These results could be explained by the way the network processes input sequences in parallel. With two sequence inputs, at a given time step the network is processing one word made up of two four-unit parts. Figure 5.0 shows a word input at each time step into a network with a training set composed of two input sequences.

**Figure 5.0**



The offset and interpolated outputs as well as the networks inability to reduce its error could be linked to how difficult the task is of finding patterns between two sequences which directly contrast each other at certain time steps. In the case of the hi-hat, if the model used solely the knowledge of one sequence to guess the values of the second, every other guess would be incorrect. Too few to generalize but too many to over fit, two sequences are hard to model in a single output sequence.

The output patterns of successful network runs achieving low error in Experiments 2, 4 and 5, trained off of two sequences, were characterized by offset and overlaid versions of 2 input sequences. The doubled counter output suggested a simple procedure could rearrange the overlaid sequences into sequential form. A closer look revealed a systematic remapping. The transformation of the target into the network output was reproduced by a Matlab script **remap\_two\_seq.m** (see E1 for results) making sense of the networks output arrangement. The same remapping worked for the snare and counter (the same in both sequences) but there were some inconsistencies in the kick and hi-hat.

**Future Steps and Motivations**

There is still a lot to be explored about the effect of the run up. How would the output sequence pattern change if we varied the run up values for kick, snare and hi-hat mimicking the counter in Experiment 5? Similarly, how would the kick, snare and output patterns change if we altered the period of the counter? This study brings fourth many opportunities to fine tune the simple model and optimize the LSTM RNN cells by exploring different activation functions and parameters specific to the unique cell structure such as drop out. In this study two simple input sequences were used to understand the model. How would varying one of these two input sequences change the output mapping?

Note that all results discussed in this paper were from models trained with one or two sequences only. Once network parameters are optimized further, ideally a much large training dataset would be used. Originally the model was tested on the original training of 64 different sequences in an arbitrary order (F1 in Appendix). The 64 sequences were reformatted into a cyclic form where the snare and hi-hat sequences repeat at different periods while the kick stays relatively random (F2 in Appendix). Although the network performed better with the cyclic data set, confusing results and ambiguity in how to assess the musicality of output sequences lead to the need to drastically simplify the input a single sequence to understand the network behavior. Since the songs picked to form the custom training data set were arbitrary, it is would difficult to assess the output.

The definition of good music varies greatly from person to person since it is heavily influenced by an individual’s experience and background. The lack of a concrete definition makes it very difficult to evaluate what a “good” a piece of music is. The broad definition music in this paper was structured sounds events occurring in cyclic patterns at different periods which evolve over time. In future studies, using a larger number of input drum sequence patterns, we could evaluate the model by measuring the extent to which there is there repetition and logical embellishment (individualism). With this method, we could describe simple drum sequences as more “good” if they balance between 1) sounding like a familiar rhythm and 2) featuring inconsistences that emulate natural embellishments of live drummers. It takes a long time for musicians to learn to improvise or be creative within boundaries, therefore, the same concept should apply to neural networks where a large number of input sequences with shared features (and a long time to learn them) are necessary to improve the model.

**Significance**

Due to limited temporal or spatial resolution in research tools today for measuring brain activity, neural networks can provide useful insight from a different angle on the neurological process of encoding and integration of musical sequences over time. This simple model of learning rhythms also has many other applications.

The large number of affordable digital tools for music production available today have made it possible for anybody with a computer to make electronic music, regardless of musical expertise. Beginning in the late 19th century, composers started to move away from the tonal system to create music that took new forms. Today, many armature producers do not bother to learn music theory, believing it is too time consuming or may destroy the magic of their music. This could relate to recent trends in electronic dance music that emphasize percussion and rhythm over tonality and traditional counterpoint conventions. The blurred line between the listener and the producer has resulted in a lot of computer music, composed mainly from looped sections, lacking a human presence.

These trends make for an increasing need of a tool that facilitates generating rhythms that evolve and possess human-like embellishments. Since neural networks optimize pattern recognition tasks and mimic specific aspects of human learning behavior, they can be used to abstract patterns within rhythmic sequences, to generate new interesting sequences while providing insight into how humans learn rhythm.

**References**

Asif A. Ghazanfar, Miguel A. L. Nicolelis; Feature Article: The Structure and Function of Dynamic Cortical and Thalamic Receptive Fields. Cereb Cortex 2001; 11 (3): 183-193. doi: 10.1093/cercor/11.3.183

Y. Bengio, P. Simard, and P. Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. Trans. Neur. Netw. 5, 2 (March 1994), 157-166. DOI=http://dx.doi.org/10.1109/72.279181

Elman, J. L. (1990), Finding Structure in Time. Cognitive Science, 14: 179–211. doi:10.1207/s15516709cog1402\_1

Koutník, Jan, Greff, Klaus, Gomez, Faustino, and Schmidhuber, Jürgen. A clockwork rnn. arXiv preprintarXiv:1402.3511, 2014.

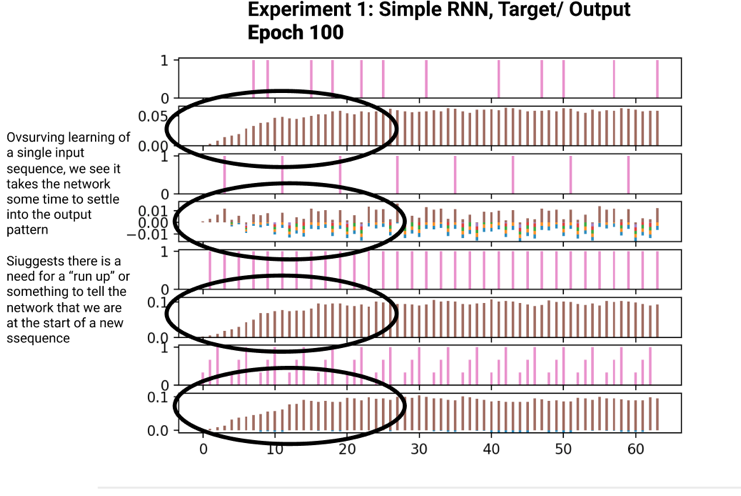
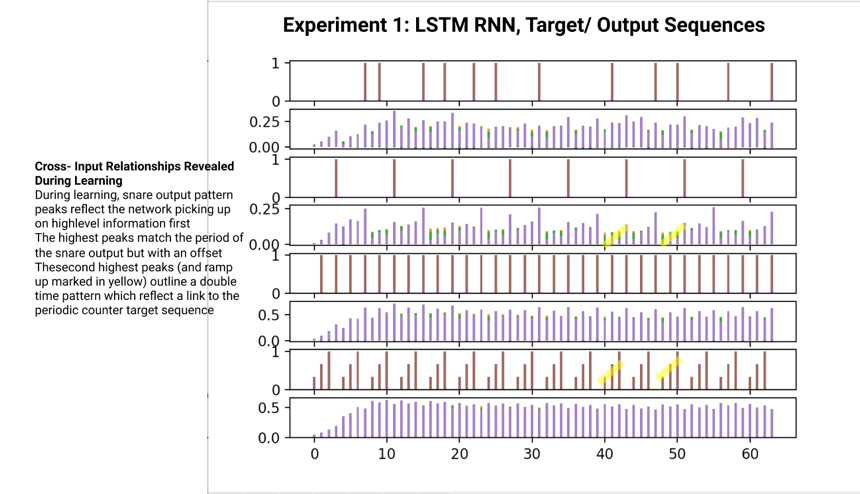
Leaver, A.M., Van Lare, J.E., Zielinski, B.A., Halpern, A. & Rauschecker, J.P. Brain activation during anticipation of sound sequences. *J. Neurosci.* **29**, 2477–2485 (2009).

|  |  |  |
| --- | --- | --- |
| **Training set = 1 sequence length 64 word units**  Learning rate = 1e-4  Hidden layer dimension = 1,000 | | |
|  | **Simple RNN cells** | **LSTM RNN cells** |
| Speed | Fast | Medium/ Fast |
| Error Curve | Smooth exponential decay dropping below 10 after 200 epochs | Characterized by a large initial drop around epoch 50  Consistent gradual decrease with one more significant drop in error below 5 around epoch 800 |
| Learning Behavior | Most interesting behavior occurring during first 100 epochs  Extra peaks added in kick output but are low values compared to the peaks matching the target sequence very close to 1 | Early snare pattern reflects cross between snare and counter suggesting detected relationship between inputs |
| Epoch 2,000 | Network output fits target sequence accurately for all four inputs  Short ramp up/ warm up length 1-2 words present | Network output fits target for all four inputs  Small ramp up between 2-4 values present |

**Appendix A: Experiment 1**

**A1**

**A2**



Kick

Snare

Hi-hat

Counter

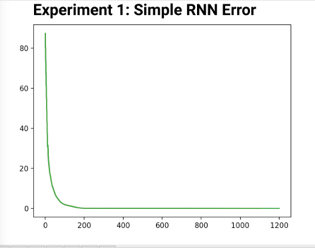
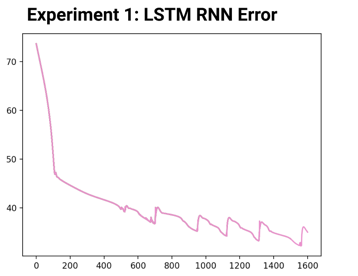
Kick

Snare

Hi-hat

Counter

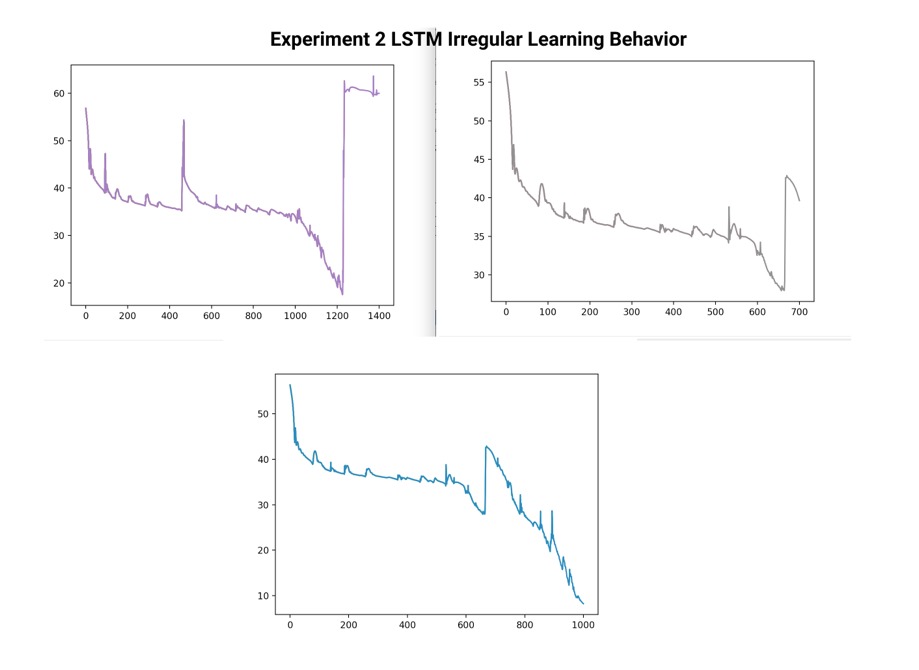
**A3**



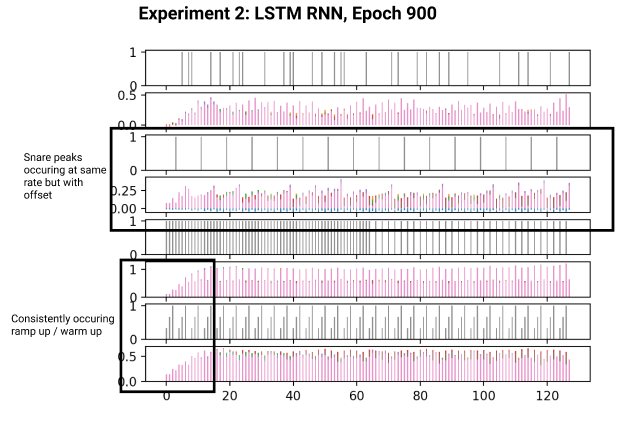
**A4**

|  |  |  |
| --- | --- | --- |
| **Training set = 2 sequences each length 64 word units**  Learning rate = 1e-4  Hidden layer dimension = 1,000 | | |
|  | **Simple RNN cells** | **LSTM RNN cells** |
| Speed | Medium/ Slow | Slow |
| Error Curve | Smooth exponential decay dropping below 20 after 100 epochs | Characterized by a large drop |
| Learning Behavior | Ramp up to snare opposite to counter in epoch 200  Doubled/ offset hi-hat patterns outlined by epoch 400 (B3)  Values reflect stretched, interpolated input sequences  Kick output composed of structures that repeat but not periodically | Varied performance  Output is characterized by a ramp up that settles into the more consistent output pattern form  Gradual change across hi-hat output detected in early epochs |
| Error | Rough exponential decay with a few spiky jumps  Flattens around epoch 500 with error < 5 | Sometimes error suddenly skyrockets and the network takes a turn for the worst while other times it gradually descends  Other runs successful relative gradual descent with drop below 20 followed by large spike (B1) |
| Additional | Slightly different offsets in kick, snare and hi-hat output sequences |  |

**Appendix B: Experiment 2**

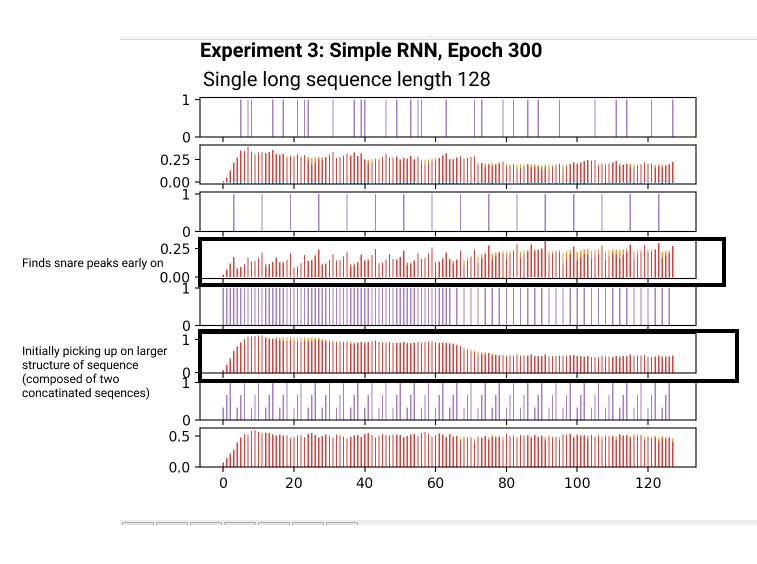


**B1**



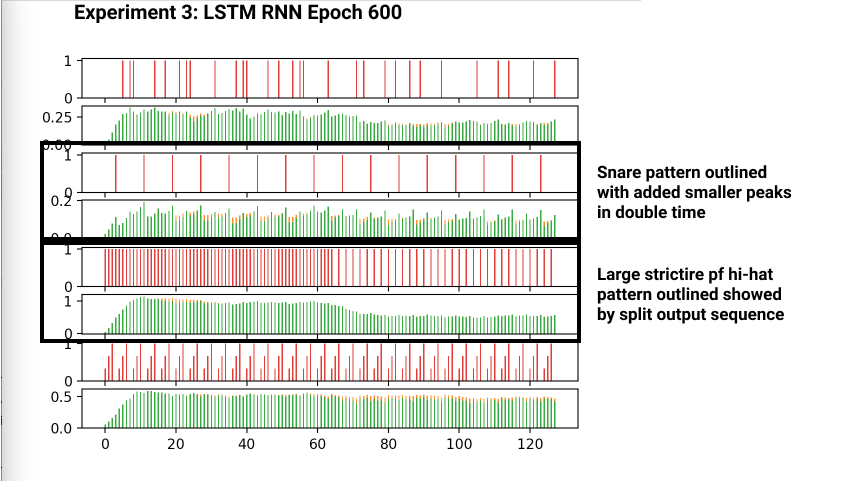
**B2**

**Appendix C: Experiment 3**

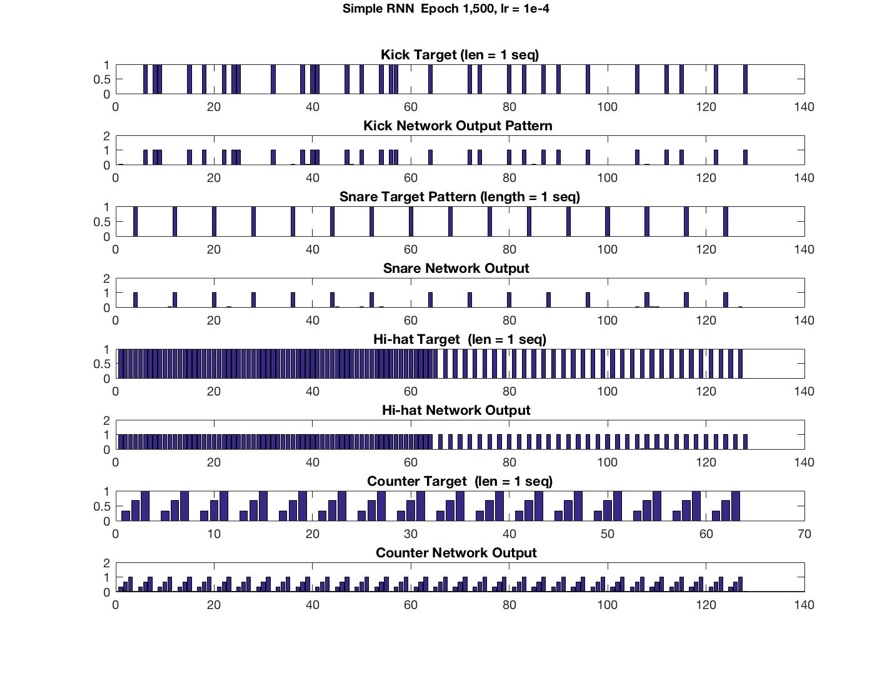


**C2**

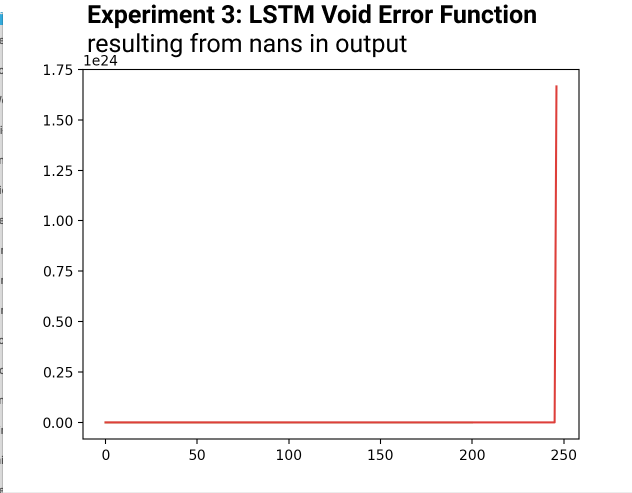
**C1**



|  |  |  |
| --- | --- | --- |
| **Training set = 1 long sequence length 128 word units (2 concatenated sequences)**  Learning rate = 1e-4  Hidden layer dimension = 1,000 | | |
|  | **Simple RNN cells** | **LSTM RNN cells** |
| Speed | Medium/ Slow | Slow |
| Error Curve | Jagged, gradually decreasing error which frequent jumps ranging in size | Very gradual descent with large spikes |
| Learning Behavior | Large range between runs  Picks up on high level patterns quickly  Snare peaks outline snare pattern with additional smaller peaks double time (C1) | Slower learning then simple RNN  Irregular learning ability which sometimes goes awry (C4) |
| Epoch 2,000 | Input sequence is over fitted by epoch 1,200 (C3)  Clear doubled periodic nature in counter output suggests simple remapping can explain output pattern | Output success varies, similar to simple RNN but in general network finds patterns in input sequences and remaps them with an offset |

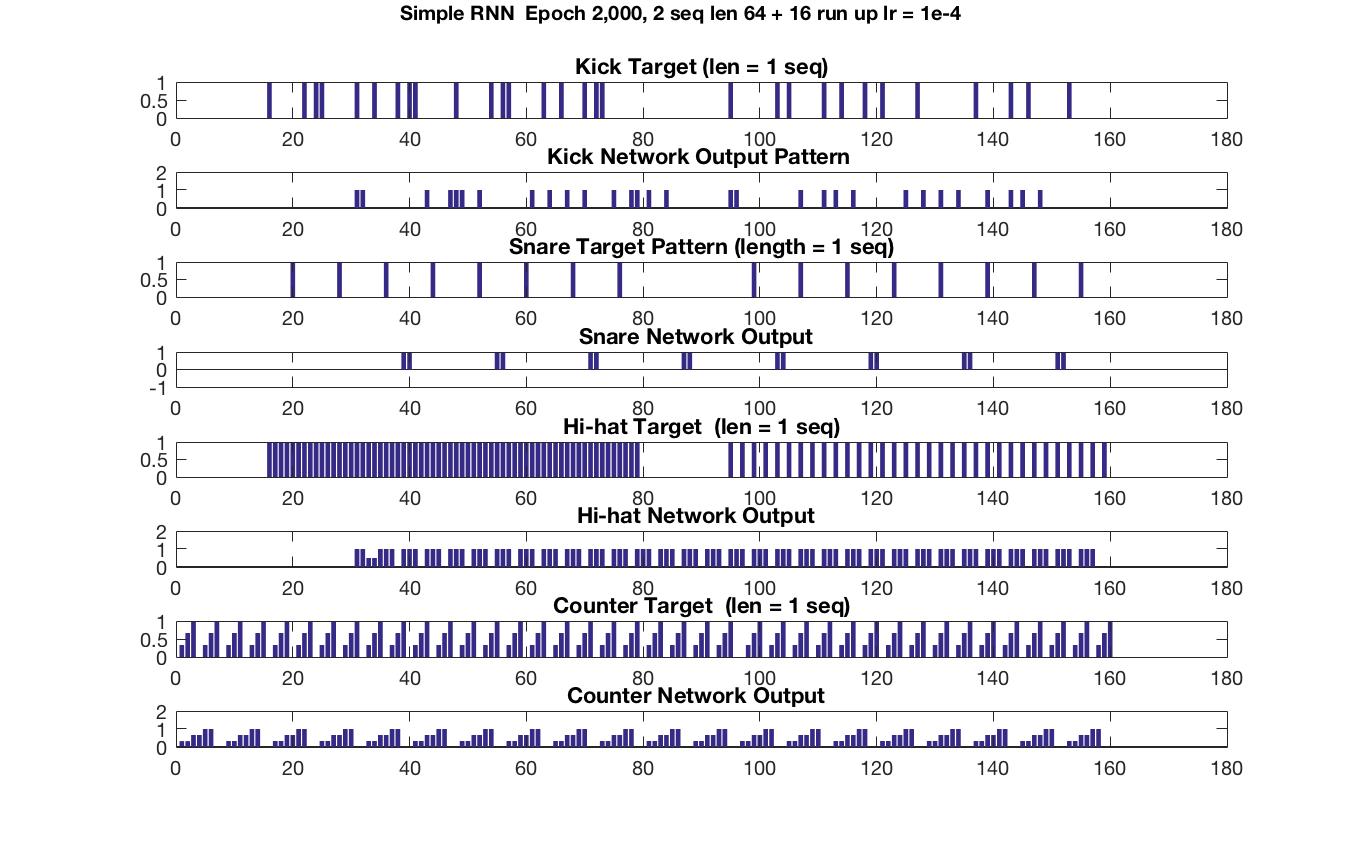


**C3**



**C4**

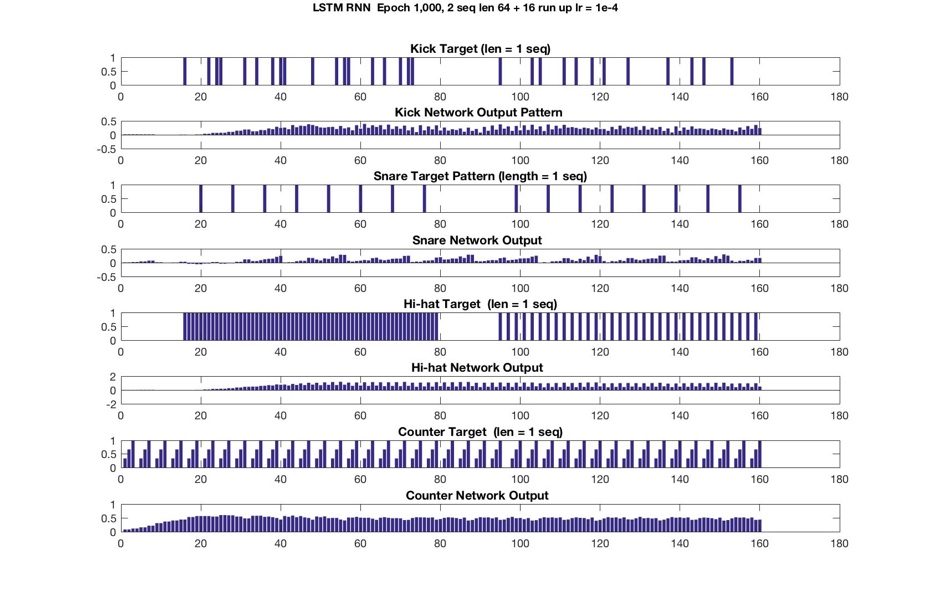
**Appendix D: Experiment 4**



**D2**

**D1**

|  |  |  |
| --- | --- | --- |
| Training set = 2 sequence length 64 word units + 16 run up units  Run up inputs = **0 for kick, snare and hi-hat** and regular periodic counter values for counter  Learning rate = 1e-4  Hidden layer dimension = 1,000 | | |
|  | Simple RNN cells | LSTM RNN cells |
| Speed | Fast | Medium/ Fast |
| Error Curve | Smooth exponential decay dropping below 20 after 100 epochs  Exponentially decaying error with many large jumps  By epoch 1,000, error is below 5 | Characterized by a large initial drop then gradual, jerky decay  Second larger drop to ~ 5 after epochs 800-1,000 sometimes occurring |
| Learning Behavior | Period of snare outlined by epoch 300  Reproduces recognizable features from input sequence by epoch 900  Network is able to find patterns in the data but remaps them with an offset resembling a doubled output in the counter and a larger offset in the snares | Large drop in error coupled with clear patterns outlined in output  Reversed ramp up behavior where values go from high to low lead up to sequence start |
| Effect of run up | Reduces ramp up so seems to assist network in finding start of sequence but exact function needs clarification | Network does not fit sequences but generalizes periodic nature with outputs resembling layered, interpolated inputs  Varied but better performance than when using uniform run up inputs |

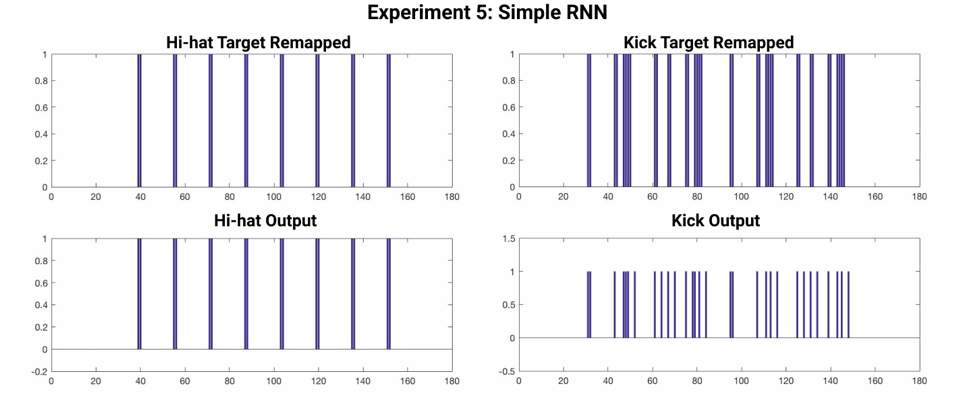


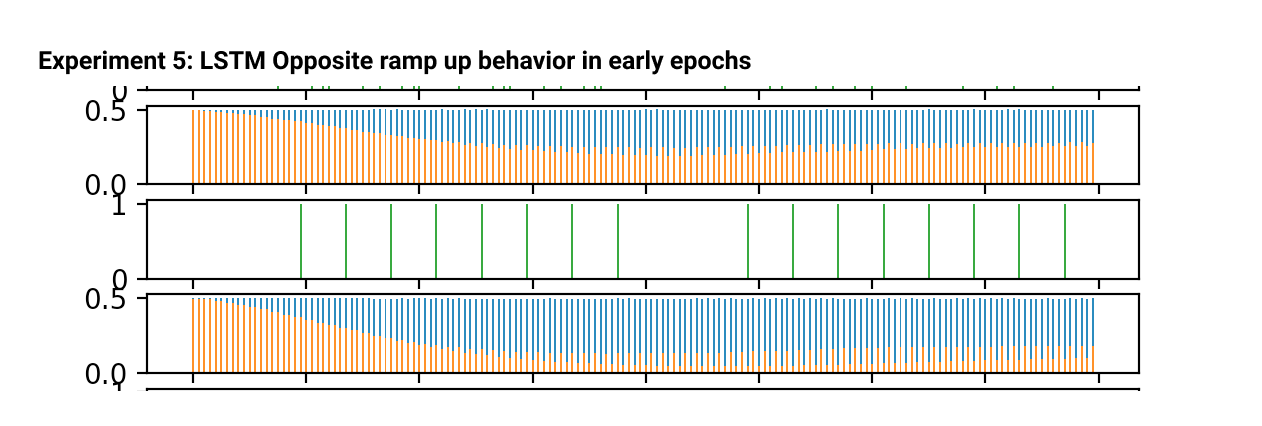
**Appendix E: Experiment 5**

|  |  |  |
| --- | --- | --- |
| **Training set = 2 sequence length 64 word units + 16 run up units**  Run up inputs = periodic counter values for all four inputs  Learning rate = 1e-4  Hidden layer dimension = 1,000 | | |
|  | **Simple RNN cells** | **LSTM RNN cells** |
| Speed | Medium/ Fast | Medium/ Slow |
| Error | Exponentially decaying error plateauing around 5  Low error rates suggest discrepancies in figures reflect reordering of input | Initial rapid decrease from > 100 to 50s after epoch 50  Further epochs characterized by gradual descent with sporadic upward jumps ranging in height between runs inhibiting learning  Error doesn’t drop below 40 in most runs |
| Learning behavior | Mode rapidly finds input sequence patterns | After certain point error jumps very high and plateaus |
| Epoch 2,000 | Counter output outlines stretched doubled version of counter target.  Outputs resemble layered offset target sequences  Number of hits matches hi-hat, and snare  Number of kicks close  Remapping target(n) to target mod(n\*2,len(output)) in Matlab script indicates the network is learning the sequences  an offset by one at the start of second sequence producing a doubling effect | Network never converges  Hi-hat output pattern values reflect blending between two sequences (1111…) -> (1010…)  In other runs periodic but interpolated hi-hat output pattern  Snare pattern not found  Ramp up output |
| Test remapping theory | Run up appears to have been moved to the start of the first sequence  Adding 1 unit offset the second sequence (moving index 1 at the start of second sequence to end) results in Offset / overlapping still appearing in output | No large noticeable effects improving networks ability to fit data |
| Effect of run up | Ramp up that matches counter removes ramp up period (no reduction early in values) but improves network less then zero run up values for kick, snare and hi-hat | Reversed ramp up during learning (E2) |

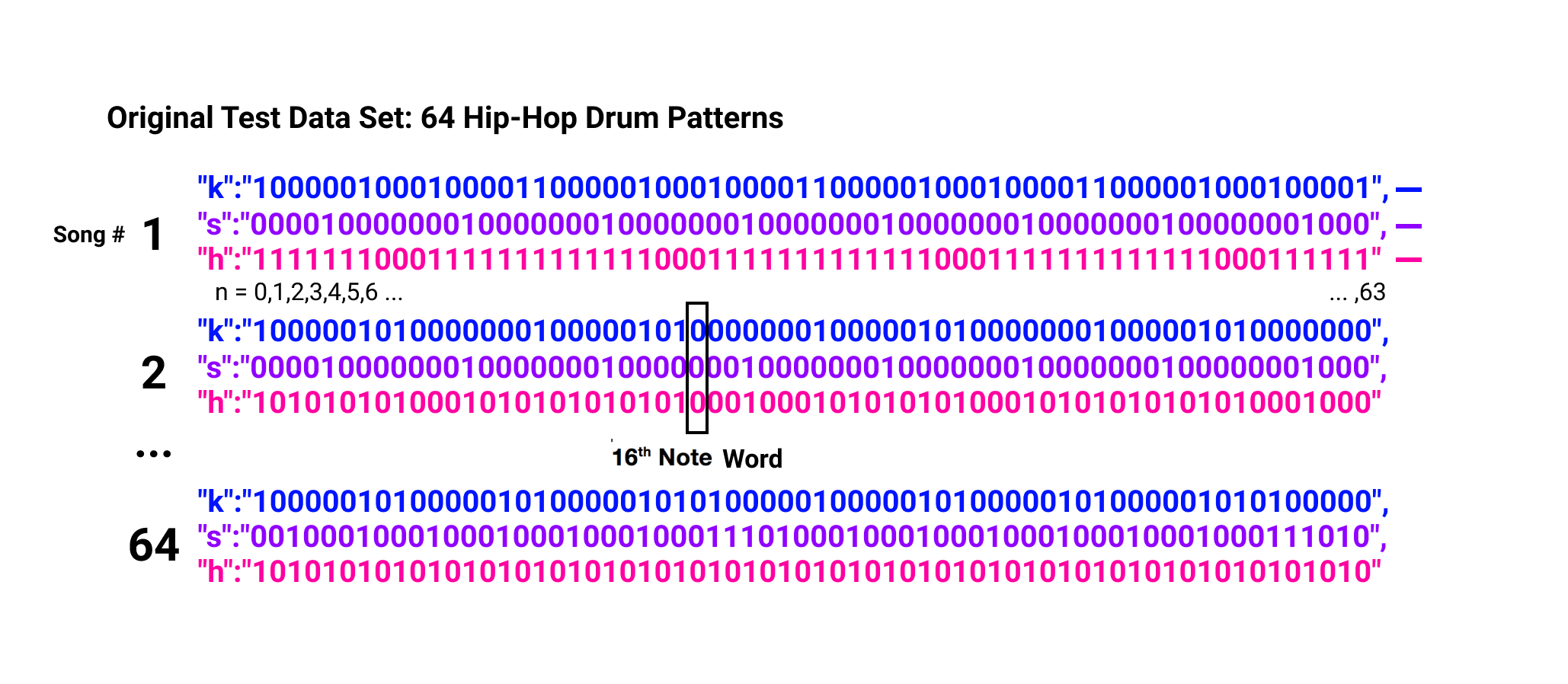
**E2**

**E1**

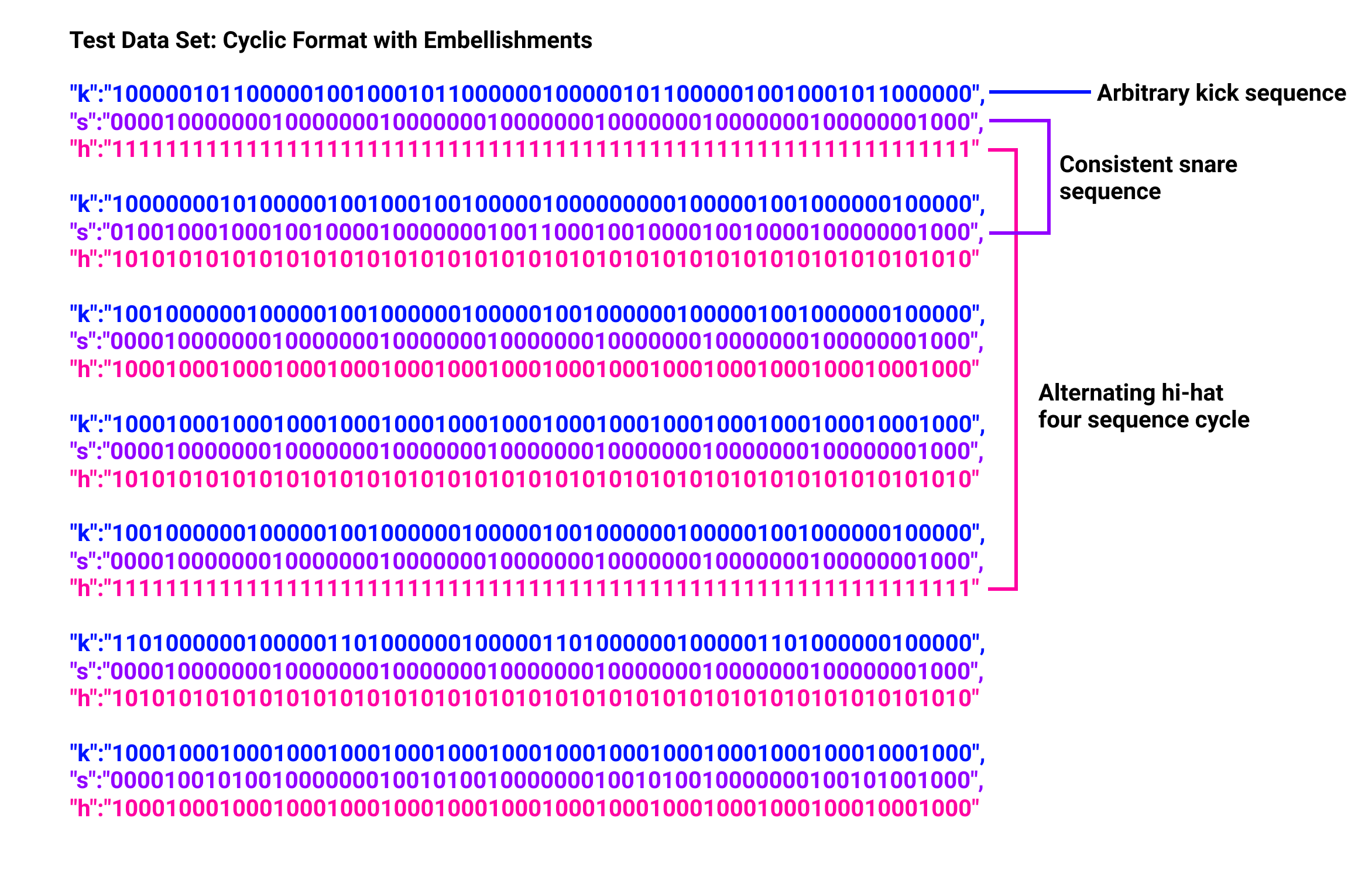




**Appendix F: Original Data Sets (unused for current study)**

****

**F1**



**F2**

1. Suga et al., 1978; Margoliash and Fortune, 1992; Rauschecker et al., 1995. [↑](#footnote-ref-1)
2. Leaver et al., 2009. [↑](#footnote-ref-2)
3. Elman, 1990. [↑](#footnote-ref-3)
4. http://cs231n.github.io/neural-networks-1/ [↑](#footnote-ref-4)
5. Koutník, et al., 2014. [↑](#footnote-ref-5)
6. Bengio, 1994 [↑](#footnote-ref-6)