Major in Computer Science - AP

Major in Computer Science - JG

Major in Computer Science - AS

**A Hopfield Network for Musical Sequence Memory and Completion**

**Anthony Porturas**

**Jessie George**

**Affaan Shariff**

12/15/17

525 Brain Inspired Computing

Prof. Konstantinos Michmizos

# Abstract

We use Hopfield Networks to create a piano app that enables the user to save songs, remember songs and complete songs. Saving songs stores the user’s input as an 8-bit binary state for each note, where each bit corresponds to a note in a single octave. Remembering songs matches the user’s input to the saved song that contains the same sequence of musical notes. Completing songs takes a partial sequence of musical notes from the user, and then plays the rest of the song by matching it to a saved song that contains the same sequence of musical notes. To increase the accuracy of our system, we tweak the Hebbian Learning process. First, we force the system to not remember the inverse of the saved states by ignoring weight change when two neurons are off together. Second, the weight increase between two firing neurons is magnified by the number of measures to account for weight decay between the measures. By modifying the learning process, we achieve a higher accuracy rate of properly attained states with more states saved into the system.

# 1. Introduction

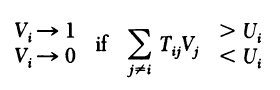
The growing music industry gives rise to the need for applications which enable the user to search for songs based on the content of the song instead of just the name of the song. This gives more flexibility to the user because they can search for a song by a partial tune even if they forget the name of the song. Currently available music recognition apps, like Shazam use spectrogram matching. Our research explores how brain inspired computing can tackle the problem of music recognition without the need for spectrogram matching in simplistic song data covering one octave. We used the theory of Hopfield Networks with a spiking neural net consisting of 8 Leaky Integrate-and-Fire neurons in order to model content-addressable memory for musical sequence recognition and completion.

The advantage of our musical content-addressable memory is that the user does not need to upload any audio files, but they can play the notes themselves. The piano app requires no former knowledge of music theory, and it can be connected to the qwerty keyboard for ease of use. When notes are pressed together quickly, they come together to form a chord as an input.

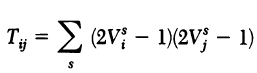
# 2. Theory

The theory for the Hopfield Network (as cited in reference paper 1) is based on a system with coordinates X1…XN which has locally stable limit points Xa, Xb and so on. If the system is started sufficiently near any Xa, as at X = Xa + ∆ then it will proceed in time until X approximately equals Xa. Therefore, we can regard the information stored in the system as vectors Xa, Xb and so on. The starting point X = Xa + ∆ represents a partial knowledge of the item Xa and the system generates the total information Xa. Any physical system who dynamics in phase space is dominated by a substantial number of locally stable states to which it is attracted can therefore be regarded as a general content-addressable memory. In our piano app, the locally stable states are the songs saved by the user. The content-addressable memory is accessed whenever a user wants to remember a saved song or complete a song with a partial sequence of notes.

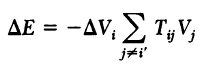
The theory of Hopfield Networks defines the state of a model as an N bit binary word where N is the number of neurons in the spiking neural net. Each digit of the N bit binary word represents Vi where Vi = 1 if the neuron is firing at maximum rate and Vi = 0 if the neuron is not firing.



In the equation given above, i is the post-synaptic neuron. j is the pre-synaptic neuron, Tij is the effectiveness of the synapse between neuron i and neuron j, and Ui is the fixed threshold for i which determines whether the neuron i is firing maximally or not. The information storage algorithm given below describes s as the state number such that s = 1 … n, and Vis is the state of neuron i. In this model a neuron is not connected to itself so Tii = 0.

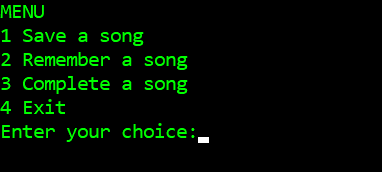


The Hopfield Network uses a smooth input-output step which neglects the details of individual action potentials and considers a neuron as on or off depending on whether the neuron is firing maximally (i.e. above the threshold), or not firing (i.e. below the threshold). The algorithm for altering Vi causes E to be a monotonically decreasing function as shown below. State changes will continue until a least local E is reached.

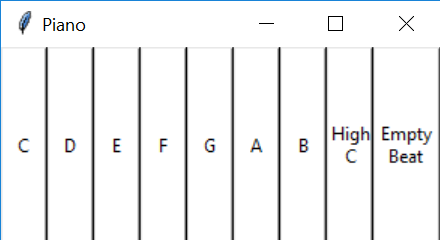


The idea for our piano app was inspired by Google Duet by Yotam Mann (as cited in reference paper 2) which allows computer generated melodies. Unlike the Google Duet app, we use Hopfield Networks, and we generate the melody based on the user’s saved songs.

# 3. Modeling Design

****

**Figure 1:** The Menu



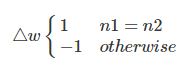
**Figure 2:** The Piano

We coded a piano app in Python which uses wav files for 8 musical notes in the C scale. The app shows a menu (Fig. 1) where the user can choose whether to save a song, remember a song, or complete a song. The menu loops until the user exits. Saving a song involves playing notes on the piano (Fig. 2) and each note that the user plays is stored as an 8-bit binary state in the spiking neural net. We use an 8-bit binary state because we have restricted the notes to one musical scale for the sake of simplicity. The spiking neural net has 8 neurons and each neuron spikes for a given musical note in the scale. According to the theory of Hopfield Networks we set the bit to 1 when the neuron is considered to be firing maximally, and 0 when the neuron is considered to be not firing.

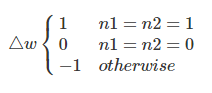
Once the user has stored songs, the user can choose to remember a song by playing a sequence of musical notes on the piano, and by using the Hopfield Network, we match the given song to one of the saved songs that most closely resembles it based on the similar sequence of musical notes. Although the theory of Hopfield Networks allows for simultaneous training of the target state and the inverse state, we have restricted the Hopfield Network to only train for the target state and to ignore the inverse state. For example, if the user plays E, F, G then the inverse state would be C, D, A, B, High-C (i.e. all notes other than what the user played). If we were to return the inverse state, it would not sound anything like the song that the user had saved. Therefore, if two neurons are both inactive at the same time then the weights stay the same instead of increasing. This allows for the Hopfield Network to only train for the target state, and return the saved song which most resembles the user’s input.

The user can also choose to complete a song by playing a partial sequence of notes, and with the given input, we use the Hopfield Network to complete the song. For example, a user could play the first half of a song on the piano app, and the output will play the entire song. The Hopfield Network will find the first saved song that contains the partial information given by the user and use the saved song to complete the musical sequence. Our piano app also has the feature of an ‘empty beat’ (silence for 1 note). So the user can click empty beats and then play musical notes on the piano app, and the output will fill in the empty beats with the corresponding notes from the saved song.

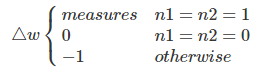
The original model of the Hebbian Learning process had only increments and decrements by a value of 1.



To remove storing of the inverse of states (“G” translates to “00001000”, where the inverse is “11110111”), we modify the model to only increment when both neurons are firing.



Now when two neurons don’t fire together, it’s not considered an association between the two. The potential to store additional states is increased as well because of the removal of the not important inverse states. However, due to the nature of our system, we must also account for the weight decay between neurons for each measure. This happens because there are 8 possible associations for each note to another note. As the number of measures increase, the weights decay to zero. Thus, we get our third model accounts for this neutrality by increasing the weight gain of two activating neurons by a factor of the number of measures, causing an increase in accuracy.



The weight changes are the only true deviations from the original Hopfield Network. The weight matrices operate similarly, where each note has a weight with the other notes. For the purposes of simplicity, we keep track of higher energy states rather than lower. When a state is found to have higher energy total, it’s a preferable transition. Theoretically, the energy states should be the negative of the respective values. It works exactly the same without the extra steps. The notes chosen for turning on and off are randomized every time, but keep track of which flips were not successful. When a flip is preferred, a recursion is called to retry previous notes which might be more successful now.

# 4. Results and Discussion

Given below is an example text output for the app. (The audio plays the corresponding notes). In this example, we set the measure = 1 and the number of beats per measure = 4. If the user wants to play longer songs, they must change the measure and number of beats per measure accordingly. In this example, we saved two songs: [‘C’, ‘E’, ‘G’, ‘^C’] and [‘B’, ‘B’, ‘F’, ‘F’]. We then played a song and it remembered the correct song from the saved songs. We then played a partial song and it completed the song.

$ python mainproj.py

MENU

1 Save a song

2 Remember a song

3 Complete a song

4 Exit

Enter your choice:1

The song you played is:

['C', 'E', 'G', '^C']

MENU

1 Save a song

2 Remember a song

3 Complete a song

4 Exit

Enter your choice:1

The song you played is:

['B', 'B', 'F', 'F']

MENU

1 Save a song

2 Remember a song

3 Complete a song

4 Exit

Enter your choice:2

The song you played is:

['B', 'B', 'F', 'F']

Remembered song:

B B F F

MENU

1 Save a song

2 Remember a song

3 Complete a song

4 Exit

Enter your choice:3

The song you played is:

['C', 'E', 'G']

The song completion is:

['C', 'E', 'G', '^C']

MENU

1 Save a song

2 Remember a song

3 Complete a song

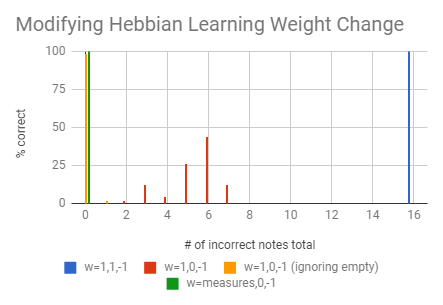
4 Exit

Enter your choice:4

To test the accuracy of our system, we create saved states where every note is randomly selected. To create the tests, we randomly pick a saved state and randomly alter a certain number of notes in the sequence to a different note.

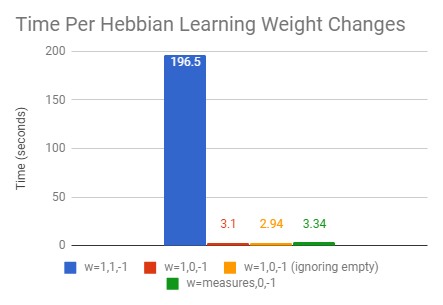
First we compare the accuracy of our new model to the accuracy of the other models. For the purpose of these comparisons, we defaulted to 4 measures, 5 saved states and 1 wrong starting note. There are four runs and three models being looked at. The “w” value represent the weight change for two neurons firing, two neurons not firing, and one of two neurons firing, respectively. The x-axis represents the number of wrong notes per test case. The y-axis represents the number percentage of test cases falling into the category of number of incorrect ending notes. Fifty test cases were randomly generated for every saved state.

The first dataset (blue) is the original Hebbian Learning process. Weights increase for two neurons firing and not firing together. The second dataset (red) is the model that ignores weight modifying if two neurons do not fire simultaneously. The third dataset (orange) is the same as the previous model, however, we ignore empty notes that are returned. Because of the nature of our application, whenever a note activates with multiple other notes, it could cause the weight change to go down to 0. For example, [A, E, G] and [A, C, D] as inputs would cause the weights between E and G and, C and D to increase. However, first the weights between A and E increase as the weight between A decreases with the other notes. Then they decrease between A and E (as with the other notes) and increase with A and C. This causes the overall weight change to be 0, and thus, empty states appear. The fourth dataset is our third model, where we magnified the weight association increase by a factor of the number of measures.

.

The accuracy of the original model failed completely. The largest factor is the number of saved states. Essentially, the number of states saved are doubled, one extra inverse for each save.So for this set of parameters, there are actually 10 saved states. Because of this, wrong states could be reached, where half of the saved states are incorrect to begin with. The accuracy of the second model increased, however, the empty notes caused large amounts of errors. When looking at the accuracy of the model after ignoring empty notes, the accuracy is near 100% for 0 wrong notes. When comparing to our third model, the accuracies are similar. Thus, it’s proven to be necessary to account for weight decay.

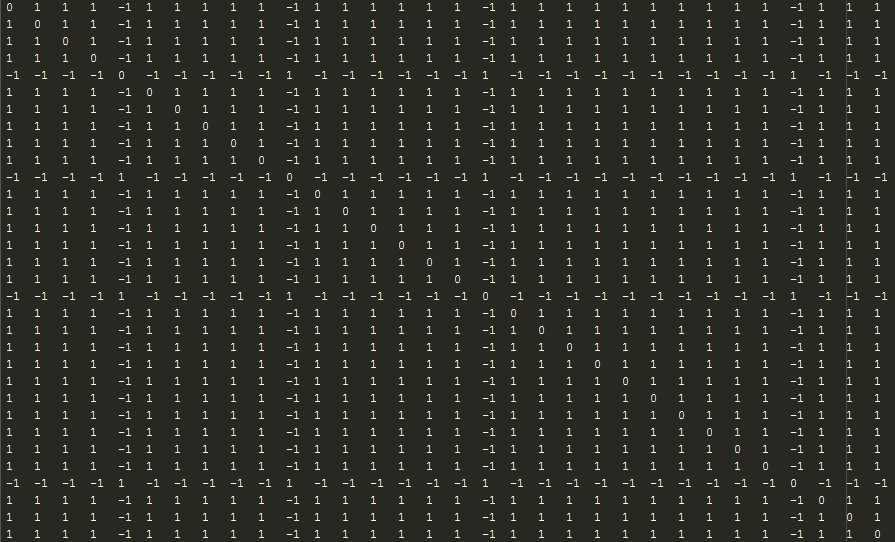
The computing time of the models were then tested for the same parameters as above.



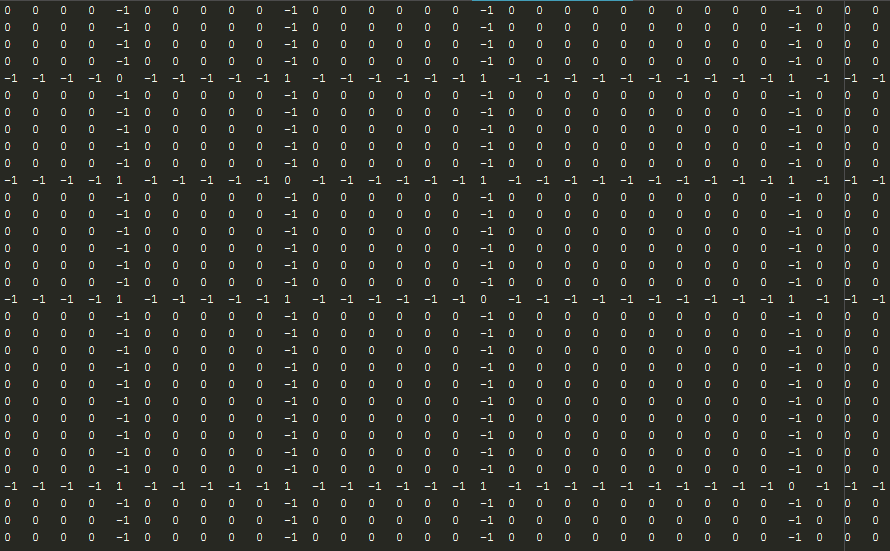
The original model performed the worst, probably due to the inverse states being remembered, increasing the number of states to flip and check to increase. Our working model performed a bit slower than the second model. By accounting for the weight decay, we increase the number of note associations, which causes more recursion calls of flipping states.

The weight matrices of the old model vs the new model look obviously different. These are matrices after training “G E D G” for a one measure parameter.

Regular Model:

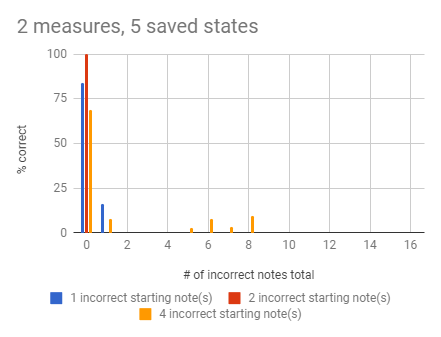


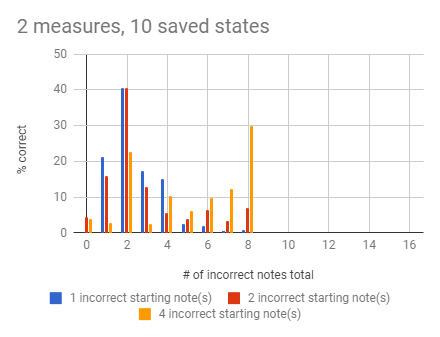
New Model:

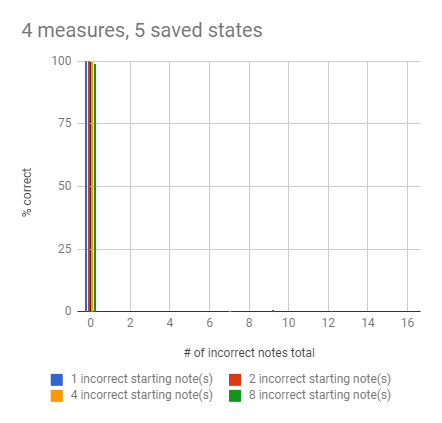


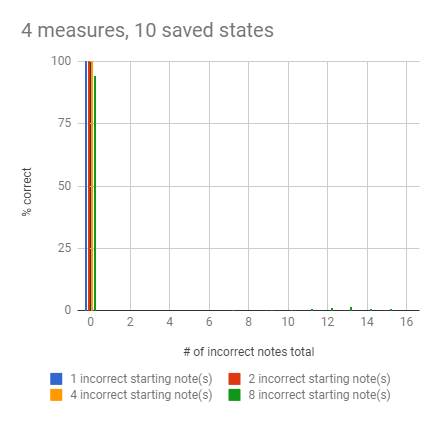
As seen, the older model creates associations between neurons not firing increase by 1, creating noisey states. The newer model weight matrix is filled with zeros instead. Thus, no unneeded associations are born.

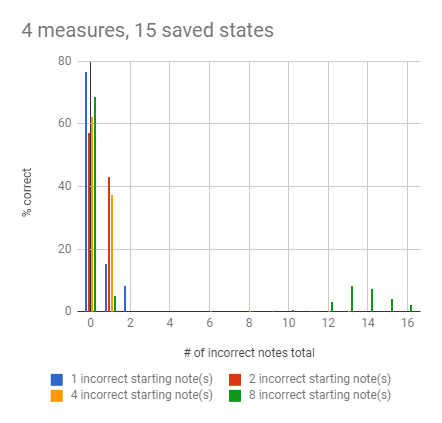
We then analyze results of varying number of measures, number of incorrect starting notes and number of saved states.

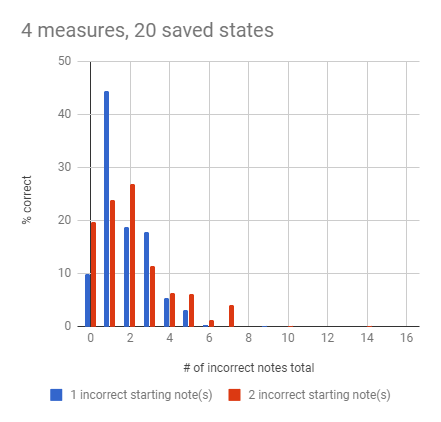








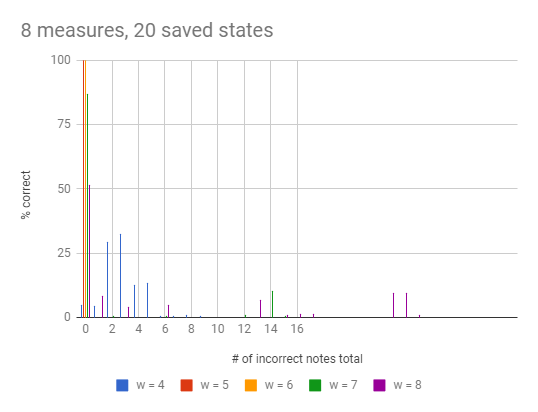




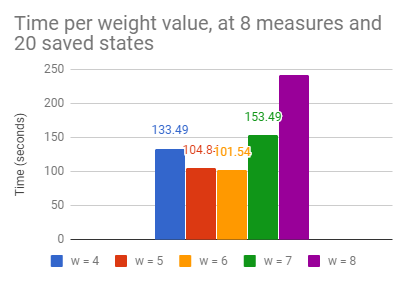
At about 4 measures, the system performs almost perfectly at 10 saved states. When that number is increased to 15 saved states, the correct saved states are still more likely than not to be remembered. At 20 saved states, the percentages decrease to below 50%.

The number of incorrect starting notes also affects the correctness of the system. When there are more incorrect starting notes than not, the percentage correct decreases as the system is more unsure of which stored state it is.

However, when the number of measures is around 8 and saved states are at 20, keeping all other parameters the same as in the previous examples, the accuracy of our model decreases severely. At this point, having the weight gain be equal to the number of measures is detrimental.



The best weight values were 5 and 6 for this case. This data suggests that the weight gain is not a static mapping to the number of measures, but rather a function in need of deriving. The speed of calculating the saved states also varies on the correctness.



The more wrong number of notes that are gathered increase the number of recursive calls of note flipping. When there are more correct states found, the number of flips is reduced.

# 5. Conclusions

Overall, by tweaking the Hebbian Learning process for our specific Hopfield Network case, we exponentially increased the processing time. For one measure, there is no speed increase. For 4 measures, the new speed was about 1.6% of the old speed. By eliminating the inverse states from the model, the system has less to needlessly process. The accuracy also increased by 100% for 5 saved states. The inverse saved states are also stored into the system, causing unnecessary cluttering. Thus, without them, we increase the storing potential.

By modifying the weight increase by a factor of the number of measures, we improve accuracy by accounting for the weight decay of weights being associated to other measures. As shown in the comparison, if the empty spaces are ignored, the accuracy is closer to 100%. This new model we propose implies that for learning music sequences, neurons that do not fire together, should not wire together.

It is no surprise that keeping track of unnecessary information could cause performance issues. However, an important takeaway is noting that depending on the nature of the task, modifying weight changes dynamically is important. When stronger associations are needed there is more information and noise. However, the weight multiplying needs to be increased not by the number of measures, but rather by a function of the number of measures and beats.

Future work should be done to increase accuracy and performance. Rather than the weight increase being directly associated to the number of measures, it should be treated more as a function. So it is important to derive the function to apply it to increase performance at large numbers of notes. Furthermore, as the number of measures and saved states increase, the system slows down considerably. Parallelizing the code would increase performance. The state flipping could be done in parallel and then merged. When these two aspects are explored and integrated, the system would be a more practical tool.

# Acknowledgments

The authors would like to thank Prof. Konstantinos Michmizos, Guangzhi Tang, and Jeff Ames for their guidance.

# References

1. Hopfield, J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8), pp.2554-2558.
2. <https://experiments.withgoogle.com/ai/ai-duet>. (2017)