STOCHASTIC PROCESSES

(4450:693)

Title: Stochastic Processes Project – MIDI Note Prediction of Time Series Data using Recurrent Neural Networks

Author Name: Mackenzie Hawkins

Student ID: 2691549

Date submitted: 12/13/2018

Instructor: Dr. Jin Kocsis

Department of Electrical and Computer Engineering

The University of Akron

Akron, OH 44325

Table of Contents

INTRO

Artificial neural networks (ANNs) are computing models that intend to mimic the neuron structures of brains in organic beings. In organic life, neurons are connected to each other via synapses. These synapses act as a communication channel that enables neurons to communicate with other neurons. This network of neurons and synapses creates a neural pathway and it's this complex interwoven neural pathway that enables brains to think. This structure is mimicked in ANNs by creating computing blocks called nodes that function as neurons. These nodes are connected to each other and the strength of the connections is modeled as a weight.

FEEDFORWARD NEURAL NETWORK

The diagram in Figure 1 illustrates a basic ANN structure. The input layer of the ANN consists of 3 input nodes, a hidden layer of 4 nodes, and an output layer of 2 nodes. This structure in known as a feedforward neural network. Feedforward neural networks only propagate information in the direction of input to output. This means that no cycles exist within the network structure and data is never propagated backwards in the network.

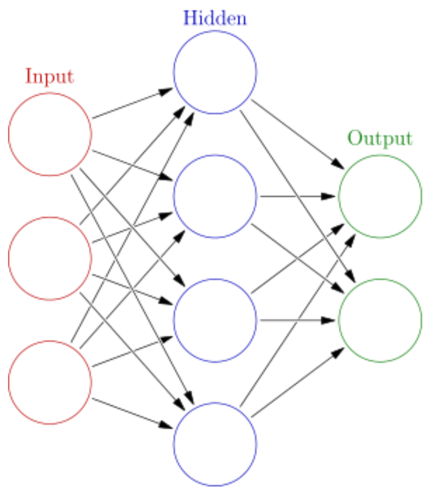


Figure 1: ANN Basic Structure

There can be any number of nodes in the hidden layer and there can be any number of hidden layers. The number of nodes in the input layer is equal to the number of features of the inputted signal that is to be processed by the ANN. For example, in an ANN that classifies hand written numbers written on a 8x8 grid will have 64 input nodes in the input layer. The number of output nodes is equal to the number of features of the outputted signal. In the hand-written number classifier example, the output layer would have 10 output nodes for numbers 0 through 9. The structure of a single node can be seen in Figure 2. The inputs from all the previous nodes are first multiplied by a weight, $w_{i,j}$, then all summed together with the addition of a bias, b. The value at n is evaluated as $n_j(t) = \sum_i p_i w_{i,j} + b_j$ where j is the j^{th} node in a given layer. The output of the node, a, passes the value n through a transfer function known as an activation function to compute the final output. Thus, the output is $a = f\left(\sum_i w_{i,j} p_i + b_j\right)$. The activation function, $f(\cdot)$, is used to bound the output. Typically, this activation function bounds the output between 0 and 1 as in the case of sigmoid activation function or between -1 and 1 as in the case of the tan-sigmoid activation function.

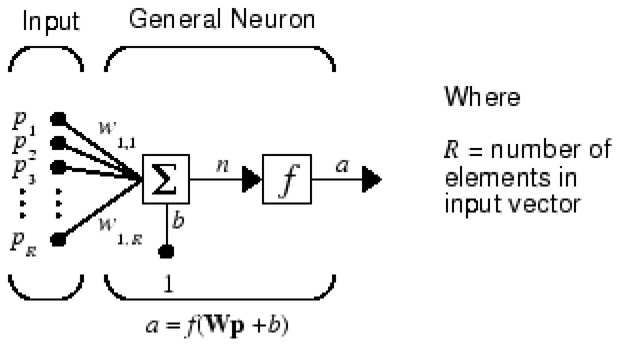


Figure 2: Node Structure

The illustration in Figure 3 shows the structure of multiple nodes in a single layer. Note that the weight is of size [ixj] where j is the number of nodes of the layer under inspection and i is the number of nodes of the previous layer. The bias and output matrix is of size [jx1], again where j is the number of nodes of the layer under inspection. The matrix representation of such a single layer of a neural network is

$$h_j = f\left(\begin{bmatrix} w_{1,1} & \cdots & w_{1,j} \\ \vdots & \ddots & \vdots \\ w_{i,1} & \cdots & w_{i,j} \end{bmatrix} \times \begin{bmatrix} p_1 \\ \vdots \\ p_j \end{bmatrix} + \begin{bmatrix} b_1 \\ \vdots \\ b_j \end{bmatrix}\right)$$

where $w_{i,j}$ is the weight matrix, p_j is the input matrix, b_j is the bias matrix, and h_j is the output matrix of size [ix1]. The output matrix h_j of the previous layer becomes the input matrix of the following layer and a new weight and bias matrix is implemented for each layer.

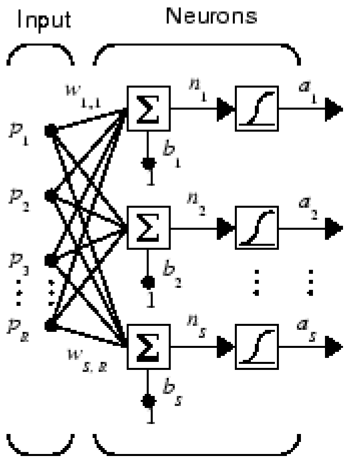


Figure 3: Node Structure of Multi-Node Neural Network

RECURRENT NEURAL NETWORK

In systems in which time dependent series information is involved, a method must be implemented in which the system has memory. This is accomplished in neural networks in the form of recurrent neural networks in which the outputs of a layer may backpropagate to previous layers. The basic structure of such a network can be seen in Figure 4 in which the output of the hidden layer is also connected to the input of the hidden layer.

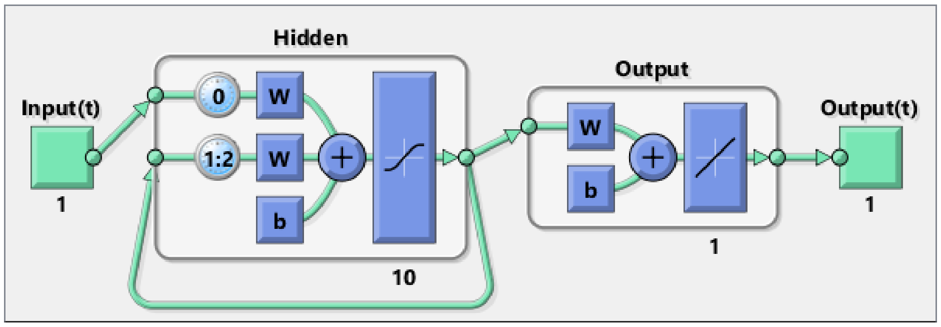


Figure 4: Recurrent Neural Network Structure

The illustration in Figure 5 shows a single recurrent neuron and its structure unfolded in time. The input to a single neuron is the same as in the feedforward neural network with the addition of the output from the previous computation. Having the output of a neuron connected to its input creates feedback in the system. This feedback is what enables the ANN to train on time series data. The weight matrix for the feedback doesn't need to be the same as the weight matrix of the feedforward weight matrix. This additional weight matrix adds more complexity to the system but allows for time series data to be computed.

$$h_j = f\left(\left(\begin{bmatrix} w_{1,1} & \cdots & w_{1,j} \\ \vdots & \ddots & \vdots \\ w_{i,1} & \cdots & w_{i,j} \end{bmatrix} \times \begin{bmatrix} x_1 \\ \vdots \\ x_j \end{bmatrix} + \begin{bmatrix} b_1 \\ \vdots \\ b_j \end{bmatrix} + \sum_{k=1}^{n}\left(\begin{bmatrix} w_{1,1} & \cdots & w_{1,j} \\ \vdots & \ddots & \vdots \\ w_{i,1} & \cdots & w_{i,j} \end{bmatrix} \times \begin{bmatrix} h_1^{(k)} \\ \vdots \\ h_j^{(k)} \end{bmatrix} + \begin{bmatrix} b_1 \\ \vdots \\ b_j \end{bmatrix}\right)\right)\right)$$

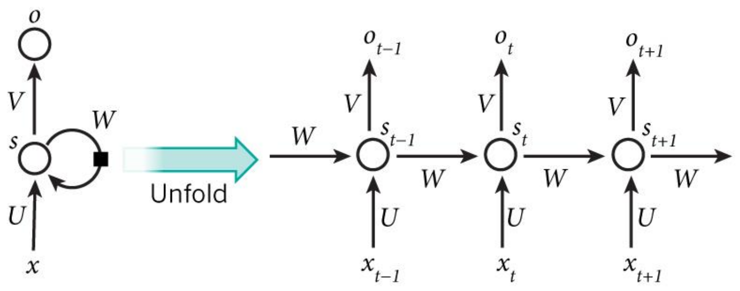


Figure 5: Recurrent Neural Network Single Neuron

GRADIENT DESCENT TRAINING

The training process of an ANN is when the weights and bias matrices are tuned so that the output of the neural network is meaningful to the input. The training process requires a training set with inputs and correlated outputs. During the training process, the weights and biases are tuned so that the error between the output of the ANN and the correlated training set output is small. This is a difficult process because there are many parameters that must be adjusted during the training process and the tuning of one parameter influences the behavior of the entire ANN. The difference between the output of a single element of the training set of the ANN and the correlated training set solution is tracked as an error. As the ANN trains, the error needs to decrease. The smaller the error is, the more accurate the ANN is at computing a meaningful output. The training process adjusts thousands of parameters and must do so in a way to improve performance of the system. The method typically used is gradient descent.

Gradient descent is an optimization technique used to find local minimums of multivariable functions. This is exactly the type of problem that is posed by the training of an ANN. Gradient descent searches for the steepest negative slope at a given point by calculating the gradient of the loss function and moves in the direction of the that slope. The distance that the loss function is stepped, in the direction of the greatest negative slope, is a hyperparameter known as the learning rate. Setting this parameter too small with make the training process unnecessarily slow but making the training rate too large has the potential of stepping passed the local minimum. This process is done until a local minimum is located at which the slope in all direction is non-negative.

This direction of descent is tracked through the mean square error of the loss function. The mean square error averages the error of all the parameters in all directions.

$$J = \frac{1}{n}\sum_{i}\left(y_i - f(x_i)\right)^2$$

The gradient of the mean square error is calculated as $\nabla J(W) = \left(\frac{\partial \mathrm{J}}{\partial w_1}, \frac{\partial \mathrm{J}}{\partial w_2}, \cdots, \frac{\partial \mathrm{J}}{\partial w_n}\right)$ and the step towards the local minimum, denoted as $W := W - \alpha \nabla J(W)$ where α is the learning rate.

This method is useful to find absolute minimums of simple functions but won't necessarily find absolute minimums of high dimensional functions like the one seen in Figure 6. A function such as the one seen in the example has many local minimums and as such using the gradient descent method used above won't guarantee that the located minimum is absolute. Different training algorithms may be used to attempt to circumvent this issue but isn't always necessary. Finding a local minimum will often yield results that are sufficient.

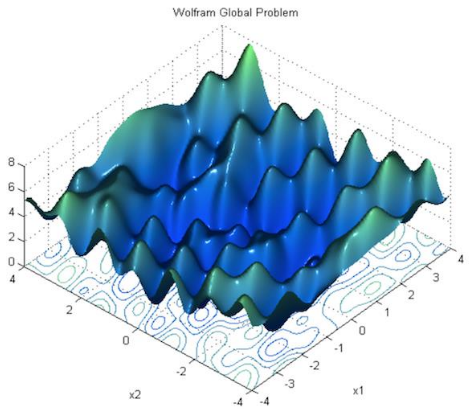


Figure 6: High Dimensional Function Example

ADAPTIVE MOMENTUM STOCHASTIC GRADIENT DESCENT

The previously mentioned method evaluated the entire data set to determine the gradient. This is known as batch gradient descent (BGD). This process is very slow and computationally intensive as the entire dataset must be processed. Doing so also has the issue of only finding a local minimum rather than an absolute minimum. To overcome this, stochastic gradient descent (SGD) can be used. Unlike BGD, SGD updates the weights of the system after randomly sampling a single data point in the set. By sampling a single data point, instead of the entire set, the gradient descent path is more irregular but still tends towards a minimum without getting stuck in a local minimum. Instead of sampling a single point, the data set can be divided into several smaller sample sets and the gradient of these smaller subsets of data can be calculated. This method of dividing the data set into many smaller sets is known as mini-batch gradient descent (MB-GD). MB-GD has the benefits of faster computations like SGD but follows a much smoother and reliable path like BGD.

To further improve the performance of finding a minimum, the update parameter can be modified to include momentum. Momentum on the update parameter makes the update parameter behave as a function of the previous updates. This means that the update parameter will tend towards the direction of the previous update.

$$W_t := W_t - \alpha \nabla J(W_t)\ , Parameter\ Update\ without\ Momentum$$

$$W_t := W_{t-1} - \alpha z_t\ , Parameter\ Update\ with\ Momentum$$

$$z_t := \beta_{t-1} + \nabla J(W_{t-1})\ , Momentum\ Adjustment\ Parameter$$

In this modified version of the update parameter, the current update is dependent on the previous update and the value of β. By setting β to 0, the update function once again becomes the update function with no momentum. Increasing β makes the update function more sensitive to the previous update.

Another issue that arises from the established method is that by setting the training rate α to a fixed value, an assumption is made that this value is sufficient for all parameters of the function. This is rarely true thus a method of modifying the training rate as the ANN trains can yield better training performance. Modifying the learning rate during the training process is known as adaptive gradient. A popular and simple adaptive gradient technique used is adaptive sub-gradient in which the parameters of the function are adjusted according to their own gradient. The expression for adaptive sub-gradient is as follows

$$w_i := w_i - \frac{\alpha}{\sqrt{G_i - \in}} \cdot \sqrt{\frac{\partial \mathrm{J}}{\partial w_i}}$$

The term $\in$ is a small value used to prevent division by 0. The intent of adaptive sub-gradient descent is to adjust the learning rate α so that the large gradients are slowed down and small gradients are speed up.

Combing both the momentum method and adaptive method of training yields a training method known as Adam. This technique combines the benefits of both methods by adapting the learning rate according to past gradients as well as including momentum to smooth the path of descent. This is done by calculating the average of the past gradient (m_t) and the past squared gradient (v_t). These values are an estimate of the first and second moment of the gradients respectively.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

To avoid these vectors from tending towards 0 as a result of them being initialized as 0 vectors, both the first and second order moments are approximated as follows.

$$\widehat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Once these moments are calculated, the update parameter is then calculated as

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{\hat{v}_t} + \in} \widehat{m}_t$$

PROJECT

For the project, I have attempted to train a RNN on time series MIDI data so that the RNN can predict the next notes of a song given preceding notes. MIDI (Musical Instrument Digital Interface) data contains a track number, channel, number, note number, velocity, start time, end time, message number of note on, and message number of note off.

Track Number – Which track the note is being played to

Channel Number –The instrument the note is played on

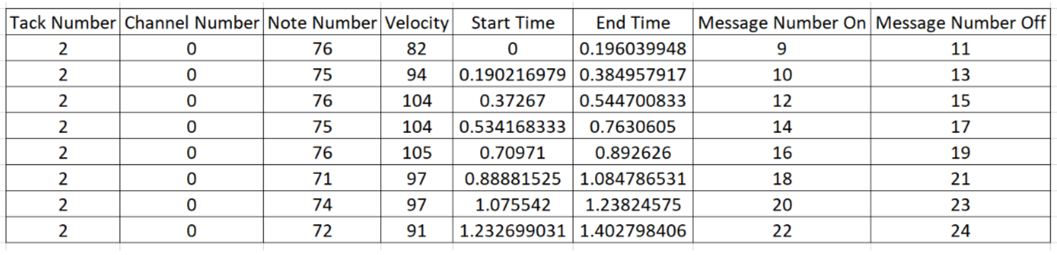
Note Number – Which note is to be played

Velocity – The loudness of the note being played

Start Time – When to begin playing the note

End Time – When to stop playing the note

Message Number of Note On and Off – These values are used to track the sequences of events of the notes being played.



Tack Number	Channel Number	Note Number	Velocity	Start Time	End Time	Message Number On	Message Number Off
2	0	76	82	0	0.196039948	9	11
2	0	75	94	0.190216979	0.384957917	10	13
2	0	76	104	0.37267	0.544700833	12	15
2	0	75	104	0.534168333	0.7630605	14	17
2	0	76	105	0.70971	0.892626	16	19
2	0	71	97	0.88881525	1.084786531	18	21
2	0	74	97	1.075542	1.23824575	20	23
2	0	72	91	1.232699031	1.402798406	22	24

Figure 7: MIDI Data Format Example

This data structure works well for training an ANN because all of the data is discretized, and each discretized sample corresponds into a single time event note. Each note occurs sequentially in the MIDI file with a time step thus the data can be used as a time series data set.

For this project, the music the ANN was trained on is of the solo pianist type. This was selected because the music is simpler from a composition perspective. This reduction in entropy of the training set will improve training time and require a less complex ANN.

The MIDI information is first extracted from the MIDI file using the midiInfo() function. This function creates an nx8 matrix from the midi file where n is the number of notes in the same format of that seen in Figure 7. The extracted midi information from each song is then combined together into a single matrix.

```
Notes1 = midiInfo(midi1,0);
Notes2 = midiInfo(midi1,0);
Combined_Notes = [Notes1;Notes2];
```

Next, the note data is extracted from the combined MIDI matrix to form a new matrix that will be used for training the ANN.

```
k=1;
[Data_Points Rows] = size(Combined_Notes);
for i=1:1:Data_Points
    if (Combined_Notes(i,2) == 0) |(Combined_Notes(i,2) == 1)
|(Combined_Notes(i,2) == 2))
        Combined_Notes_Cleaned(k,1) = Combined_Notes(i,3);
        k=k+1;
    end
end
k=1;
[Data_Points Rows] = size(Combined_Notes);
```

The data then needs to be separated into a training set and a test set. The training set consists of 90% of the data from the original data set, and the remaining 10% is used as a test set once the ANN is trained. The training set is standardized to prevent the training from diverging. This is done by adjusting the data set to have zero mean, and unit variance.

```
Sample_Interval = floor(size(Combined_Notes,1)*0.85);
Sample_Start_Note = 1;
% Load Sequence Data
data =
transpose(Combined_Notes_Drumless(Sample_Start_Note:Sample_Start_Note
+Sample_Interval));
% Partition to training and test data
numTimeStepsTrain = floor(0.9*numel(data));
dataTrain = data(1:numTimeStepsTrain+1);
dataTest = data(numTimeStepsTrain+1:end);
% Standardize the Data
mu = mean(dataTrain);
sig = std(dataTrain);
dataTrainStandardized = (dataTrain - mu) / sig;
% dataTrainStandardized = dataTrain;
% Prepare Predictor and Resonses
XTrain = dataTrainStandardized(1:end-1);
YTrain = dataTrainStandardized(2:end);
```

The features of the ANN are next defined. To reduce complexity, this ANN has a single input and a single output. The ANN has 4 layers of size 200, 100, 200, and 100. The ANN architecture is defined as a fully connected long short-term memory neural network with a sequence and regression layer. This network architecture enables the ANN to be trained on time series data because of the recurrent data path.

```
numFeatures = 1;
numResponses = 1;
numHiddenUnits1 = 200;
numHiddenUnits2 = 100;
numHiddenUnits3 = 200;
numHiddenUnits4 = 100;

layers = [ ...
    sequenceInputLayer(numFeatures)
    lstmLayer(numHiddenUnits1, 'OutputMode', 'sequence')
    lstmLayer(numHiddenUnits2, 'OutputMode', 'sequence')
    lstmLayer(numHiddenUnits3, 'OutputMode', 'sequence')
    lstmLayer(numHiddenUnits4, 'OutputMode', 'sequence')
    fullyConnectedLayer(numResponses)
    regressionLayer];
```

The training parameters are next configured. The ANN is trained using the ADAM method discussed earlier. Epochs is a hyperparameter that is chosen by the user to identify the number of times the ANN is trained using the entire training dataset. The execution environment was chosen as GPU to reduce training time. The use of GPU training allows for parallel computing resulting in shorter training times. Initial learning rate was chosen to be 0.002. This value was settled on experimentally as it resulted in successful training sessions without training needlessly slow. The training rate decreased as the ANN trained. The training rate was recued by 10% the current value 5 times throughout the training process.

```
Itterations = 1000;
options = trainingOptions('adam', ...
    'MaxEpochs',Itterations, ...
    'ExecutionEnvironment','gpu', ...
    'GradientThreshold',1, ...
    'InitialLearnRate',0.002,
    'LearnRateSchedule','piecewise', ...
    'LearnRateDropPeriod',Itterations/5, ...
    'LearnRateDropFactor',0.9, ...
    'Verbose',0, ...
    'Plots','training-progress');
```

Once training has completed the actual output is compared to the ANN predicted output. These outputs are compared by their value and the root mean squared error of the actual notes and the predicted note of the trained ANN. An example of the plots of this data from a training session can be seen in Figure 8 through Figure 10. The orange plot is the forecasted notes that the ANN predicted given the blue plot in Figure 8.

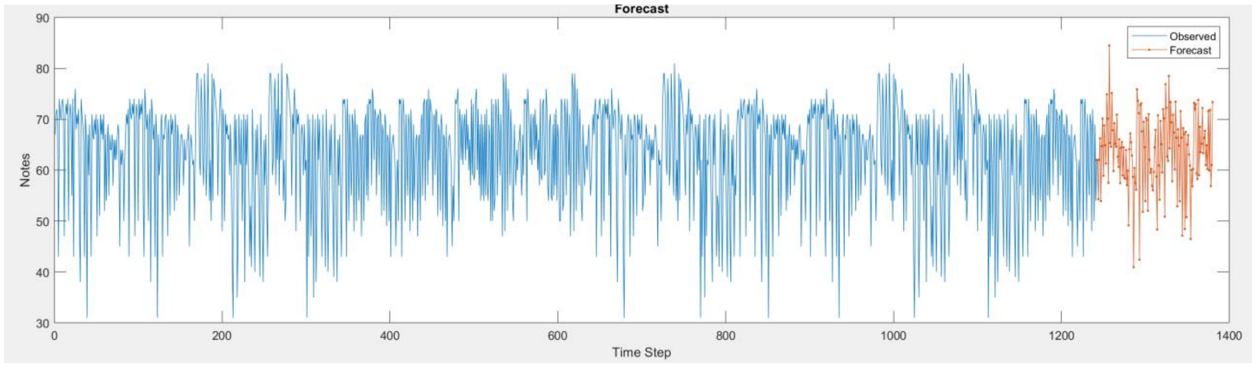


Figure 8: Observed Notes and Forecasted Notes

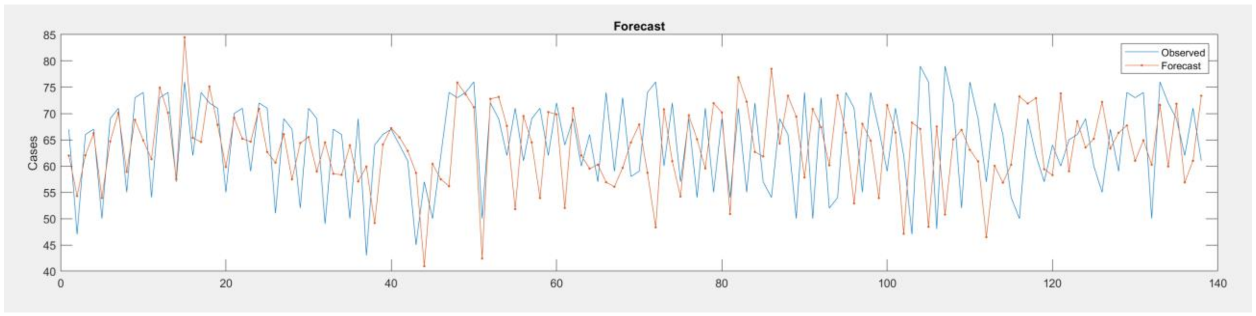


Figure 9: Observed Notes VS Forecasted Notes

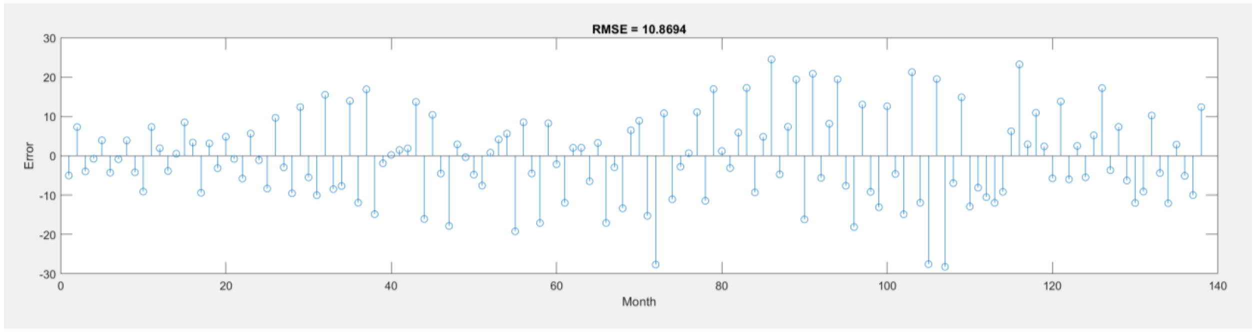


Figure 10: RMSE of Observed Notes VS Forecasted Notes

A new MIDI file is then created with the predicted notes from the ANN and the actual notes that are intended to be played. The time information from the original dataset is used to reconstruct both of these newly generated MIDI files as the ANN was not trained to predict both time and note information. An example of this outputted data in the form of sheet music from a training session can be seen in Figure 11 and Figure 12.

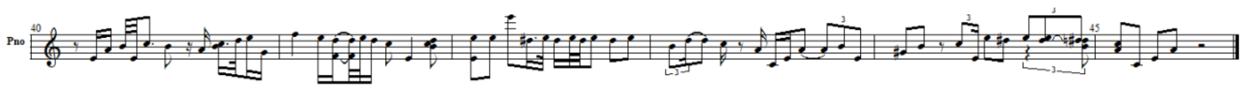


Figure 11: Observed Future Sheet Music

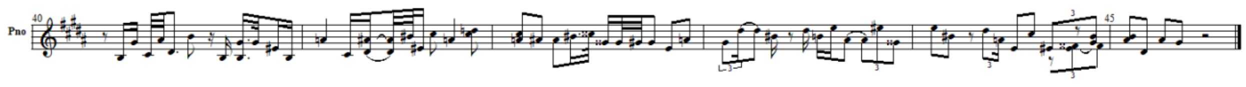


Figure 12: ANN Forecasted Future Sheet Music

CONCLUSION

From the plot seen in Figure 9 and Figure 10, note that the forecasted notes from the ANN have a relatively low RMSE. The RMSE of the forecasted notes has a general trend of increasing as the notes are further from the training set. This is likely due to the ANN "assuming" that the next several notes will continue the pattern of a small set of the previous notes thus the uncertainty is lower. As notes further from the trained set are predicted, the uncertainty increases and the RMSE also increases. This behavior is as expected if the time series data is analyzed as a Markov chain. If each subsequent note if a function of preceding notes, mutual information decreases as notes become further from the training set. This means entropy and thus uncertainty increases as each next note is predicted. This can be expresses as

$$N_1 \to N_2 \to N_3 \to \cdots \to N_n$$

Where each note N is expressed as N_n and n is the nth note is the series. The mutual information in the set can then be expressed as

$$I(N_1; N_2) \leq I(N_1; N_3) \leq \cdots \leq I(N_1; N_n)$$

From this expression, it can be observed that mutual information decreases and thus entropy and uncertainty increase as subsequent notes are predicted.

Proformance could likely be improved by increasing the complexity of the neural network by adding additional layers and adding more depth to each layer. This comes at a cost of increased training time but can result in a lower RMSE for notes predicted further from the training set. A larger data set could also improve performance as this would provide more data for the ANN to train on. This additional data could be from additional MIDI files of similar style.

The sheet music from the forecasted notes is not pleasant to listen to. This is likely do to human hearing and music experience being very sensitive to subtleties. Even at a very low RMSE, the forecasted notes can still sound bad. If one note is off by a single position, a person would be able to identify that note as being out of place. This situation would result in a low RMSE but to a human listener, it would not sound correct. A possible solution to this could be to train the ANN with additional hyper parameters about the music as it's training. If the ANN is trained with the key that the song is being played in, the entropy of the next note in the series is reduced and the likelihood that it will sound pleasant to a person is also increased.

Overall the project was successful. The trained ANN was able to forecast the next values in time series data with a relatively low RMSE. With additional time and resources, the performance of such an ANN could be improved and results could be pleasant to listen to.

REFERENCES

Britz, D. (2016, July 08). Recurrent Neural Networks Tutorial, Part 1 – Introduction to RNNs. Retrieved from http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

Colin Raffel. "Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching". PhD Thesis, 2016.

Gupta, T. (2017, January 05). Deep Learning: Feedforward Neural Network – Towards Data Science. Retrieved from https://towardsdatascience.com/deep-learning-feedforward-neural-network-26a6705dbdc7

How neural networks are trained. (n.d.). Retrieved from https://ml4a.github.io/ml4a/how_neural_networks_are_trained/

Kingma, D. P., & Ba, J. L. (2015). ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION. ICLR. Retrieved November 29, 2018, from https://arxiv.org/abs/1412.6980.

Kulbear. (n.d.). Kulbear/deep-learning-nano-foundation. Retrieved from https://github.com/Kulbear/deep-learning-nano-foundation/wiki/Introduction-to-Neural-Network:-Feedforward

Multilayer Shallow Neural Network Architecture. (n.d.). Retrieved from https://www.mathworks.com/help/deeplearning/ug/multilayer-neural-network-architecture.html

Ross, M. (2017, September 10). Under The Hood of Neural Network Forward Propagation - The Dreaded Matrix Multiplication. Retrieved from https://towardsdatascience.com/under-the-hood-of-neural-network-forward-propagation-the-dreaded-matrix-multiplication-a5360b33426

Schutte, K. (n.d.). MATLAB and MIDI. Retrieved from http://kenschutte.com/midi

Sebastian Ruder. (2018, November 29). An overview of gradient descent optimization algorithms. Retrieved from http://ruder.io/optimizing-gradient-descent/index.html#adam

Thierry Bertin-Mahieux, Daniel P. W. Ellis, Brian Whitman, and Paul Lamere. "The Million Song Dataset". In Proceedings of the 12th International Society for Music Information Retrieval Conference, pages 591–596, 2011.

MATLAB CODE

```
clc
clear all
close all
%%
addpath('D:\Akron\2018 Fall\Stochastic Processes\Project\Midi_Functions')
addpath('D:\Akron\2018 Fall\Stochastic Processes\Project\clean_midi\Ludwig
van Beethoven')
addpath('D:\Akron\2018 Fall\Stochastic Processes\Project\clean_midi\Claude
Debussy')

midi1 = readmidi('Menuet in G.mid');
midi2 = readmidi('Fur Elise.mid');
%% Combine Data
% Channel 9 is percusion
% https://pjb.com.au/muscript/gm.html
Notes1 = midiInfo(midi1,0);
Notes2 = midiInfo(midi2,0);
Combined_Notes = [Notes1;Notes2];
%% Clean the Data
% Extract the notes
k=1;
[Data_Points Rows] = size(Combined_Notes);
for i=1:1:Data_Points
    if ((Combined_Notes(i,2) == 0) |(Combined_Notes(i,2) == 1)
|(Combined_Notes(i,2) == 2))
        Combined_Notes_Cleaned(k,1) = Combined_Notes(i,3);
        k=k+1;
    end
end
k=1;
[Data_Points Rows] = size(Combined_Notes);
%%
% https://www.mathworks.com/help/deeplearning/examples/time-series-
forecasting-using-deep-learning.html
Sample_Interval = floor(size(Combined_Notes_Cleaned,1)*0.85);
Sample_Start_Note = 1;
% Load Sequence Data
data =
transpose(Combined_Notes_Cleaned(Sample_Start_Note:Sample_Start_Note+Sample_I
nterval));
% Partition to training and test data
numTimeStepsTrain = floor(0.9*numel(data));
dataTrain = data(1:numTimeStepsTrain+1);
dataTest = data(numTimeStepsTrain+1:end);
% Standardize the Data
mu = mean(dataTrain);
sig = std(dataTrain);
dataTrainStandardized = (dataTrain - mu) / sig;
% dataTrainStandardized = dataTrain;
% Prepare Predictor and Resonses
XTrain = dataTrainStandardized(1:end-1);
YTrain = dataTrainStandardized(2:end);
%% Define LSTM Network Architecture
```

```
% https://www.mathworks.com/help/deeplearning/ug/list-of-deep-learning-
layers.html
numFeatures = 1;
numResponses = 1;
numHiddenUnits1 = 200;
numHiddenUnits2 = 100;
numHiddenUnits3 = 200;
numHiddenUnits4 = 100;

layers = [ ...
    sequenceInputLayer(numFeatures)
    lstmLayer(numHiddenUnits1, 'OutputMode', 'sequence')
    lstmLayer(numHiddenUnits2, 'OutputMode', 'sequence')
    lstmLayer(numHiddenUnits3, 'OutputMode', 'sequence')
    lstmLayer(numHiddenUnits4, 'OutputMode', 'sequence')
    fullyConnectedLayer(numResponses)
    regressionLayer];


Itterations = 1000;
options = trainingOptions('adam', ...
    'MaxEpochs',Itterations, ...
    'ExecutionEnvironment','gpu', ...
    'GradientThreshold',1, ...
    'InitialLearnRate',0.002, ... % 0.005 is default, 0.02 did a thing with
the layers above
    'LearnRateSchedule','piecewise', ...
    'LearnRateDropPeriod',Itterations/5, ...
    'LearnRateDropFactor',0.9, ...
    'Verbose',0, ...
    'Plots','training-progress');

% Train LSTM Network
net = trainNetwork(XTrain,YTrain,layers,options);
%%
% Forcast Future Time Steps
% dataTestStandardized = (dataTest - mu) / sig;
dataTestStandardized = dataTest;
XTest = dataTestStandardized(1:end-1);

net = predictAndUpdateState(net,XTrain);
[net,YPred] = predictAndUpdateState(net,YTrain(1,:));

numTimeStepsTest = numel(XTest);
for i = 2:numTimeStepsTest
    [net,YPred(:,i)] = predictAndUpdateState(net,YPred(:,i-
1),'ExecutionEnvironment','gpu');
end

% Unstandardize the prediction
YPred = YPred(:,1:numTimeStepsTest);
YPred = sig*YPred + mu;
% Calculate the RMSE from unstandardized predictions
YTest = dataTest(2:end);
rmse = sqrt(mean((YPred-YTest).^2));
%%
% Plot the training series with the forecasted values
```

```
close all
figure
f1 = plot(dataTrain(1:end-1));
hold on
idx = numTimeStepsTrain:(numTimeStepsTrain+numTimeStepsTest);
plot(idx,[data(numTimeStepsTrain) YPred],'.-')
% plot(idx,[data(numTimeStepsTrain) YPred(1:numTimeStepsTest)],'.-')
hold off
xlabel("Time Step")
ylabel("Notes")
title("Forecast")
legend(["Observed" "Forecast"])
% Compare Forecasted values with test data
figure
subplot(2,1,1)
plot(YTest)
hold on
plot(YPred,'.-')
hold off
legend(["Observed" "Forecast"])
ylabel("Cases")
title("Forecast")

subplot(2,1,2)
stem(YPred - YTest)
xlabel("Month")
ylabel("Error")
title("RMSE = " + rmse)
%%
%% Create Output Midi File of the Future Notes
Output_Midi_Future = zeros(numTimeStepsTest,8);
Output_Midi_Future(:,1) = Combined_Notes(Sample_Interval-
numTimeStepsTest+2:Sample_Interval+1,1);
Output_Midi_Future(:,2) = Combined_Notes(Sample_Interval-
numTimeStepsTest+2:Sample_Interval+1,2);
Output_Midi_Future(:,3) = transpose(YTest(1,1:numTimeStepsTest)); % The
testeed Notes
Output_Midi_Future(:,4) = Combined_Notes(Sample_Interval-
numTimeStepsTest+2:Sample_Interval+1,4);
Output_Midi_Future(:,5) = Combined_Notes(Sample_Interval-
numTimeStepsTest+2:Sample_Interval+1,5);
Output_Midi_Future(:,6) = Combined_Notes(Sample_Interval-
numTimeStepsTest+2:Sample_Interval+1,6);
% Output_Midi_Future(:,5) =
Combined_Notes(numTimeStepsTrain+1:numTimeStepsTrain+numTimeStepsTest,5);
% Output_Midi_Future(:,6) =
Combined_Notes(numTimeStepsTrain+1:numTimeStepsTrain+numTimeStepsTest,6);
Output_Midi_Future(:,7) = Combined_Notes(Sample_Interval-
numTimeStepsTest+2:Sample_Interval+1,7);
Output_Midi_Future(:,8) = Combined_Notes(Sample_Interval-
numTimeStepsTest+2:Sample_Interval+1,8);
New_Midi_Future =  matrix2midi(Output_Midi_Future);
writemidi(New_Midi_Future, 'Midi_Future.mid');
%% Create Output Midi File of the Predicted Notes
% Output_Midi_Test = zeros(numTimeStepsTest,6);
Output_Midi_Prediction = zeros(numTimeStepsTest,8);
```

```
Output_Midi_Prediction(:,1) = Combined_Notes(Sample_Interval-
numTimeStepsTest+2:Sample_Interval+1,1);
Output_Midi_Prediction(:,2) = Combined_Notes(Sample_Interval-
numTimeStepsTest+2:Sample_Interval+1,2);
Output_Midi_Prediction(:,3) = transpose(floor(YPred(1,:))); % The predicted
Notes
Output_Midi_Prediction(:,4) = Combined_Notes(Sample_Interval-
numTimeStepsTest+2:Sample_Interval+1,4);
Output_Midi_Prediction(:,5) = Combined_Notes(Sample_Interval-
numTimeStepsTest+2:Sample_Interval+1,5);
Output_Midi_Prediction(:,6) = Combined_Notes(Sample_Interval-
numTimeStepsTest+2:Sample_Interval+1,6);
Output_Midi_Prediction(:,7) = Combined_Notes(Sample_Interval-
numTimeStepsTest+2:Sample_Interval+1,7);
Output_Midi_Prediction(:,8) = Combined_Notes(Sample_Interval-
numTimeStepsTest+2:Sample_Interval+1,8);
New_Midi_Prediction =  matrix2midi(Output_Midi_Prediction);
writemidi(New_Midi_Prediction, 'Midi_Prediction.mid');
```

MATLAB FUNCTIONS

MIDI-INFO()

```
function [Notes,endtime] = midiInfo(midi,outputFormat,tracklist,verbose)
% [Notes,endtime] = midiInfo(midi,outputFormat,tracklist)
%
% Takes a midi structre and generates info on notes and messages
% Can return a matrix of note parameters and/or output/display
%   formatted table of messages
%
% Inputs:
%  midi - Matlab structure (created by readmidi.m)
%  tracklist - which tracks to show ([] for all)
%  outputFormat
%   - if it's a string write the formated output to the file
%   - if 0, don't display or write formatted output
%   - if 1, just display (default)
%
% outputs:
%   Notes - a matrix containing a list of notes, ordered by start time
%     column values are:
%      1     2    3  4   5  6  7       8
%     [track chan nn vel t1 t2 msgNum1 msgNum2]
%   endtime - time of end of track message
%

% Copyright (c) 2009 Ken Schutte
% more info at: http://www.kenschutte.com/midi
if nargin<4
  verbose = 0;
end
if nargin<3
  tracklist=[];
  if nargin<2
    outputFormat=1;
  end
end
if (isempty(tracklist))
  tracklist = 1:length(midi.track);
end

current_tempo = 500000;  % default tempo


[tempos, tempos_time] = getTempoChanges(midi);

% What to do if no tempos are given?
%  This seems at leat get things to work (see eire01.mid)
if length(tempos) == 0
  tempos = [current_tempo];
  tempos_time = [0];
```

```
end


fid = -1;
if (ischar(outputFormat))
  fid = fopen(outputFormat,'w');
end

endtime = -1;

% each row:
%  1     2     3  4    5  6  7       8
% [track chan nn vel t1 t2 msgNum1 msgNum2]
Notes = zeros(0,8);

for i=1:length(tracklist)
  tracknum = tracklist(i);

  cumtime=0;
  seconds=0;

  Msg = cell(0);
  Msg{1,1} = 'chan';
  Msg{1,2} = 'deltatime';
  Msg{1,3} = 'time';
  Msg{1,4} = 'name';
  Msg{1,5} = 'data';

  for msgNum=1:length(midi.track(tracknum).messages)

    currMsg = midi.track(tracknum).messages(msgNum);

    midimeta  = currMsg.midimeta;
    deltatime = currMsg.deltatime;
    data      = currMsg.data;
    type      = currMsg.type;
    chan      = currMsg.chan;

    cumtime = cumtime + deltatime;
    seconds = seconds + deltatime*1e-
6*current_tempo/midi.ticks_per_quarter_note;

    [mx ind] = max(find(cumtime >= tempos_time));
    if numel(ind)>0 % if only we found smth
        current_tempo = tempos(ind);
    else
        if verbose
            disp('No tempos_time found?');
        end
    end

    % find start/stop of notes:
    %    if (strcmp(name,'Note on') && (data(2)>0))
    % note on with vel>0:
    if (midimeta==1 && type==144 && data(2)>0)
      % note on:
       Notes(end+1,:) = [tracknum chan data(1) data(2) seconds 0 msgNum -1];
```

```
        %    elseif ((strcmp(name,'Note on') && (data(2)==0)) ||
strcmp(name,'Note off'))
        % note on with vel==0 or note off:
    elseif (midimeta==1 && ( (type==144 && data(2)==0) || type==128 ))

      % note off:
      %       % find index, wther tr,chan,and nn match, and not complete

      ind = find((...
        (Notes(:,1)==tracknum) + ...
        (Notes(:,2)==chan) + ...
        (Notes(:,3)==data(1)) + ...
        (Notes(:,8)==-1)...
        )==4);

      if (length(ind)==0)
      %% was an error before; change to warning and ignore the message.
    if verbose
        warning('ending non-open note?');
    end

      else
      if (length(ind)>1)
        %% ??? not sure about this...
        %disp('warning: found mulitple matches in endNote, taking first...');
        %% should we take first or last? should we give a warning?
        ind = ind(1);
      end

      % set info on ending:
      Notes(ind,6) = seconds;
      Notes(ind,8) = msgNum;

      end


      % end of track:
    elseif (midimeta==0 && type==47)
      if (endtime == -1)
      endtime = seconds;
      else
          if verbose
            disp('two "end of track" messages?');
          end
      endtime(end+1) = seconds;
      end


    end

    % we could check to make sure it ends with
    %  'end of track'


    if (outputFormat ~= 0)
      % get some specific descriptions:
      name = num2str(type);
      dataStr = num2str(data);

      if (isempty(chan))
      Msg{msgNum,1} = '-';
      else
      Msg{msgNum,1} = num2str(chan);
      end

      Msg{msgNum,2} = num2str(deltatime);
      Msg{msgNum,3} = formatTime(seconds);

      if (midimeta==0)
      Msg{msgNum,4} = 'meta';
      else
      Msg{msgNum,4} = '';
      end

      [name,dataStr] = getMsgInfo(midimeta, type, data);
      Msg{msgNum,5} = name;
      Msg{msgNum,6} = dataStr;
    end

  end %% end track.

  %% any note-on that are not turned off?
  nleft = sum(Notes(:,8)==-1);
  if (nleft > 0)
    %warning(sprintf('%d notes needed to be turned off at end of track.',
nleft));
    Notes(Notes(:,8) == -1, 6) = seconds;
  end

  if (outputFormat ~= 0)
    printTrackInfo(Msg,tracknum,fid);
  end

end

% make this an option!!!
% - I'm not sure why it's needed...
% remove start silence:
first_t = min(Notes(:,5));
Notes(:,5) = Notes(:,5) - first_t;
Notes(:,6) = Notes(:,6) - first_t;

% sort Notes by start time:
[junk,ord] = sort(Notes(:,5));
Notes = Notes(ord,:);


if (fid ~= -1)
  fclose(fid);
end
```

```
function printTrackInfo(Msg,tracknum,fid)


% make cols same length instead of just using \t
for i=1:size(Msg,2)
  maxLen(i)=0;
  for j=1:size(Msg,1)
    if (length(Msg{j,i})>maxLen(i))
      maxLen(i) = length(Msg{j,i});
    end
  end
end


s='';
s=[s sprintf('--------------------------------------------------\n')];
s=[s sprintf('Track %d\n',tracknum)];
s=[s sprintf('--------------------------------------------------\n')];

if (fid == -1)
  disp(s)
else
  fprintf(fid,'%s',s);
end


for i=1:size(Msg,1)
  line='';
  for j=1:size(Msg,2)
    sp = repmat(' ',1,5+maxLen(j)-length(Msg{i,j}));
    m = Msg{i,j};
    m = m(:)';  % ensure column vector
%    line = [line Msg{i,j} sp];
    line = [line m sp];
  end

  if (fid == -1)
    disp(line)
  else
    fprintf(fid,'%s\n',line);
  end

end



function s=formatTime(seconds)

minutes = floor(seconds/60);
secs = seconds - 60*minutes;
```

```
s = sprintf('%d:%2.3f',minutes,secs);



function [name,dataStr]=getMsgInfo(midimeta, type, data);

% meta events:
if (midimeta==0)
  if     (type==0);  name = 'Sequence Number';          len=2;  dataStr =
num2str(data);
  elseif (type==1);  name = 'Text Events';              len=-1; dataStr =
char(data);
  elseif (type==2);  name = 'Copyright Notice';         len=-1; dataStr =
char(data);
  elseif (type==3);  name = 'Sequence/Track Name';      len=-1; dataStr =
char(data);
  elseif (type==4);  name = 'Instrument Name';          len=-1; dataStr =
char(data);
  elseif (type==5);  name = 'Lyric';                    len=-1; dataStr =
char(data);
  elseif (type==6);  name = 'Marker';                   len=-1; dataStr =
char(data);
  elseif (type==7);  name = 'Cue Point';                len=-1; dataStr =
char(data);
  elseif (type==32); name = 'MIDI Channel Prefix';      len=1;  dataStr =
num2str(data);
  elseif (type==47); name = 'End of Track';             len=0;  dataStr =
'';
  elseif (type==81); name = 'Set Tempo';                len=3;
    val = data(1)*16^4+data(2)*16^2+data(3); dataStr = ['microsec per quarter
note: ' num2str(val)];
  elseif (type==84); name = 'SMPTE Offset';             len=5;
    dataStr = ['[hh;mm;ss;fr;ff]=' mat2str(data)];
  elseif (type==88); name = 'Time Signature';           len=4;
    dataStr = [num2str(data(1)) '/' num2str(data(2)) ', clock ticks and
notated 32nd notes=' num2str(data(3)) '/' num2str(data(4))];
  elseif (type==89); name = 'Key Signature';            len=2;
    % num sharps/flats (flats negative)
    % but data(1) is unsigned 8-bit
    if (data(1)<=7)
       %   0   1   2    3    4   5    6    7
      ss={'C','G','D', 'A', 'E','B',  'F#', 'C#'};
      dataStr = ss{data(1)+1};
    elseif (data(1)>=249)
       %    1   2    3    4   5    6    7
       %   255   ...                 249
       ss={'F','Bb','Eb','Ab','Db','Gb','Cb'};
       dataStr = ss{255-data(1)+1};
    else
       dataStr = '?';
    end
    if (data(2)==0)
      dataStr = [dataStr ' Major'];
    else
      dataStr = [dataStr ' Minor'];
    end
  elseif (type==89); name = 'Sequencer-Specific Meta-event';   len=-1;
    dataStr = char(data);
    % !! last two conflict...

  else
    name = ['UNKNOWN META EVENT: ' num2str(type)]; dataStr = num2str(data);
  end

% meta 0x21 = MIDI port number, length 1 (? perhaps)
else

  % channel voice messages:
  %  (from event byte with chan removed, eg 0x8n -> 0x80 = 128 for
  %  note off)
  if     (type==128);  name = 'Note off';                   len=2; dataStr =
['nn=' num2str(data(1)) '  vel=' num2str(data(2))];
  elseif (type==144);  name = 'Note on';                    len=2; dataStr =
['nn=' num2str(data(1)) '  vel=' num2str(data(2))];
  elseif (type==160); name = 'Polyphonic Key Pressure';   len=2; dataStr =
['nn=' num2str(data(1)) '  vel=' num2str(data(2))];
  elseif (type==176); name = 'Controller Change';          len=2; dataStr =
['ctrl=' controllers(data(1)) '  value=' num2str(data(2))];
  elseif (type==192); name = 'Program Change';             len=1; dataStr =
['instr=' num2str(data)];
  elseif (type==208); name = 'Channel Key Pressure';       len=1; dataStr =
['vel=' num2str(data)];
  elseif (type==224); name = 'Pitch Bend';                 len=2;
    val = data(1)+data(2)*256;
    val = base2dec('2000',16) - val;
    dataStr = ['change=' num2str(val) '?'];

  % channel mode messages:
  %  ... unsure about data for these... (do some have a data byte and
  %  others not?)
  %
  % 0xC1 .. 0xC8
  elseif (type==193);  name = 'All Sounds Off';             dataStr =
num2str(data);
  elseif (type==194);  name = 'Reset All Controllers';      dataStr =
num2str(data);
  elseif (type==195); name = 'Local Control';              dataStr =
num2str(data);
  elseif (type==196); name = 'All Notes Off';              dataStr =
num2str(data);
  elseif (type==197); name = 'Omni Mode Off';              dataStr =
num2str(data);
  elseif (type==198); name = 'Omni Mode On';               dataStr =
num2str(data);
  elseif (type==199); name = 'Mono Mode On';               dataStr =
num2str(data);
  elseif (type==200); name = 'Poly Mode On';               dataStr =
num2str(data);

    % sysex, F0->F7
  elseif (type==240); name = 'Sysex 0xF0';                dataStr =
num2str(data);
```

```
  elseif (type==241); name = 'Sysex 0xF1';            dataStr =
num2str(data);
  elseif (type==242); name = 'Sysex 0xF2';            dataStr =
num2str(data);
  elseif (type==243); name = 'Sysex 0xF3';            dataStr =
num2str(data);
  elseif (type==244); name = 'Sysex 0xF4';            dataStr =
num2str(data);
  elseif (type==245); name = 'Sysex 0xF5';            dataStr =
num2str(data);
  elseif (type==246); name = 'Sysex 0xF6';            dataStr =
num2str(data);
  elseif (type==247); name = 'Sysex 0xF7';            dataStr =
num2str(data);

    % realtime
    % (i think have no data..?)
  elseif (type==248); name = 'Real-time 0xF8 - Timing clock';
dataStr = num2str(data);
  elseif (type==249); name = 'Real-time 0xF9';            dataStr =
num2str(data);
  elseif (type==250); name = 'Real-time 0xFA - Start a sequence';
dataStr = num2str(data);
  elseif (type==251); name = 'Real-time 0xFB - Continue a sequence';
dataStr = num2str(data);
  elseif (type==252); name = 'Real-time 0xFC - Stop a sequence';
dataStr = num2str(data);
  elseif (type==253); name = 'Real-time 0xFD';            dataStr =
num2str(data);
  elseif (type==254); name = 'Real-time 0xFE';            dataStr =
num2str(data);
  elseif (type==255); name = 'Real-time 0xFF';            dataStr =
num2str(data);


  else
    name = ['UNKNOWN MIDI EVENT: ' num2str(type)]; dataStr = num2str(data);
  end


end

function s=controllers(n)
if (n==1); s='Mod Wheel';
elseif (n==2); s='Breath Controllery';
elseif (n==4); s='Foot Controller';
elseif (n==5); s='Portamento Time';
elseif (n==6); s='Data Entry MSB';
elseif (n==7); s='Volume';
elseif (n==8); s='Balance';
elseif (n==10); s='Pan';
elseif (n==11); s='Expression Controller';
elseif (n==16); s='General Purpose 1';
elseif (n==17); s='General Purpose 2';
elseif (n==18); s='General Purpose 3';
elseif (n==19); s='General Purpose 4';
elseif (n==64); s='Sustain';
elseif (n==65); s='Portamento';
elseif (n==66); s='Sustenuto';
elseif (n==67); s='Soft Pedal';
elseif (n==69); s='Hold 2';
elseif (n==80); s='General Purpose 5';
elseif (n==81); s='Temp Change (General Purpose 6)';
elseif (n==82); s='General Purpose 7';
elseif (n==83); s='General Purpose 8';
elseif (n==91); s='Ext Effects Depth';
elseif (n==92); s='Tremelo Depthy';
elseif (n==93); s='Chorus Depth';
elseif (n==94); s='Detune Depth (Celeste Depth)';
elseif (n==95); s='Phaser Depth';
elseif (n==96); s='Data Increment (Data Entry +1)';
elseif (n==97); s='Data Decrement (Data Entry -1)';
elseif (n==98); s='Non-Registered Param LSB';
elseif (n==99); s='Non-Registered Param MSB';
elseif (n==100); s='Registered Param LSB';
elseif (n==101); s='Registered Param MSB';
else
  s='UNKNOWN CONTROLLER';
end

%Channel mode message values
%Reset All Controllers  79    121   Val ??
%Local Control     7A    122   Val 0 = off, 7F (127) = on
%All Notes Off     7B    123   Val must be 0
%Omni Mode Off     7C    124   Val must be 0
%Omni Mode On      7D    125   Val must be 0
%Mono Mode On      7E    126   Val = # of channels, or 0 if # channels equals
# voices in receiver
%Poly Mode On      7F    127   Val must be 0
```

ReadMIDI()

```
function midi = readmidi(filename, rawbytes)
% midi = readmidi(filename, rawbytes)
% midi = readmidi(filename)
%
% Read MIDI file and store in a Matlab structure
% (use midiInfo.m to see structure detail)
%
% Inputs:
%  filename - input MIDI file
%  rawbytes - 0 or 1: Include raw bytes in structure
%             This info is redundant, but can be
%             useful for debugging. default=0
%

% Copyright (c) 2009 Ken Schutte
% more info at: http://www.kenschutte.com/midi


if (nargin<2)
  rawbytes=0;
end

fid = fopen(filename);
[A count] = fread(fid,'uint8');
fclose(fid);

if (rawbytes) midi.rawbytes_all = A; end

% realtime events: status: [F8, FF].  no data bytes
%clock, undefined, start, continue, stop, undefined, active
%sensing, systerm reset

% file consists of "header chunk" and "track chunks"
%   4B  'MThd' (header) or 'MTrk' (track)
%   4B  32-bit unsigned int = number of bytes in chunk, not
%       counting these first 8


% HEADER CHUNK --------------------------------------------------------
% 4B 'Mthd'
% 4B length
% 2B file format
%    0=single track, 1=multitrack synchronous, 2=multitrack asynchronous
%    Synchronous formats start all tracks at the same time, while
asynchronous formats can start and end any track at any time during the
score.
% 2B track cout (must be 1 for format 0)
% 2B num delta-time ticks per quarter note
%

if ~isequal(A(1:4)',[77 84 104 100])  % double('MThd')
    error('File does not begin with header ID (MThd)');
end
```

```
header_len = decode_int(A(5:8));
if (header_len == 6)
else
    error('Header length != 6 bytes.');
end

format = decode_int(A(9:10));
if (format==0 || format==1 || format==2)
     midi.format = format;
else
    error('Format does not equal 0,1,or 2');
end

num_tracks = decode_int(A(11:12));
if (format==0 && num_tracks~=1)
    error('File is format 0, but num_tracks != 1');
end

time_unit = decode_int(A(13:14));
if (bitand(time_unit,2^15)==0)
  midi.ticks_per_quarter_note = time_unit;
else
  error('Header: SMPTE time format found - not currently supported');
end

if (rawbytes)
  midi.rawbytes_header = A(1:14);
end

% end header parse ----------------------------------------------------




% BREAK INTO SEPARATE TRACKS ------------------------------------------
% midi.track(1).data = [byte byte byte ...];
% midi.track(2).date = ...
% ...
%
% Track Chunks----------
% 4B 'MTrk'
% 4B length (after first 8B)
%
ctr = 15;
for i=1:num_tracks

  if ~isequal(A(ctr:ctr+3)',[77 84 114 107])  % double('MTrk')
    error(['Track ' num2str(i) ' does not begin with track ID=MTrk']);
  end
  ctr = ctr+4;

  track_len = decode_int(A(ctr:ctr+3));
  ctr = ctr+4;
  % have track.rawbytes hold initial 8B also...
  track_rawbytes{i} = A((ctr-8):(ctr+track_len-1));

  if (rawbytes)
    midi.track(i).rawbytes_header = A(ctr-8:ctr-1);
  end

  ctr = ctr+track_len;
end
% ----------------------------------------------------------------------






% Events:
%  - meta events: start with 'FF'
%  - MIDI events: all others

% MIDI events:
%  optional command byte + 0,1,or 2 bytes of parameters
%  "running mode": command byte omitted.
%
% all midi command bytes have MSB=1
% all data for inside midi command have value <= 127 (ie MSB=0)
% -> so can determine running mode
%
% meta events' data may have any values (meta events have to set
% len)
%



% 'Fn' MIDI commands:
%  no chan. control the entire system
%F8 Timing Clock
%FA start a sequence
%FB continue a sequence
%FC stop a sequence

% Meta events:
%  1B 0xFF
%  1B event type
%  1B length of additional data
%  ?? additional data
%


% "channel mode messages"
% have same code as "control change": 0xBn
%  but uses reserved controller numbers 120-127
%


%Midi events consist of an optional command byte
% followed by zero, one or two bytes of parameters.
% In running mode, the command can be omitted, in
% which case the last MIDI command specified is
% assumed.  The first bit of a command byte is 1,
% while the first bit of a parameter is always 0.
%   In addition, the last 4 bits of a command
%   indicate the channel to which the event should
%   be sent; therefore, there are 6 possible
%   commands (really 7, but we will discuss the x'Fn'
%   commands later) that can be specified.  They are:


% parse tracks -----------------------------------------
for i=1:num_tracks

  track = track_rawbytes{i};

  if (rawbytes); midi.track(i).rawbytes = track; end

  msgCtr = 1;
  ctr=9;  % first 8B were MTrk and length
  while (ctr < length(track_rawbytes{i}))

    clear currMsg;
    currMsg.used_running_mode = 0;
    % note:
    %  .used_running_mode is necessary only to
    %  be able to reconstruct a file _exactly_ from
    %  the 'midi' structure.  this is helpful for
    %  debugging since write(read(filename)) can be
    %  tested for exact replication...
    %

    ctr_start_msg = ctr;

    [deltatime,ctr] = decode_var_length(track, ctr);

    % ?
    %if (rawbytes)
    %  currMsg.rawbytes_deltatime = track(ctr_start_msg:ctr-1);
    %end

    % deltaime must be 1-4 bytes long.
    % could check here...


    % CHECK FOR META EVENTS ------------------------
    % 'FF'
    if track(ctr)==255

      type = track(ctr+1);

      ctr = ctr+2;

      % get variable length 'length' field
      [len,ctr] = decode_var_length(track, ctr);

      % note: some meta events have pre-determined lengths...
      %  we could try verifiying they are correct here.

      thedata = track(ctr:ctr+len-1);
      chan = [];

      ctr = ctr + len;

      midimeta = 0;

    else
      midimeta = 1;
      % MIDI EVENT ---------------------------




      % check for running mode:
      if (track(ctr)<128)

      % make it re-do last command:
      %ctr = ctr - 1;
      %track(ctr) = last_byte;
      currMsg.used_running_mode = 1;

      B = last_byte;
      nB = track(ctr); % ?

      else

      B  = track(ctr);
      nB = track(ctr+1);

      ctr = ctr + 1;

      end

      % nibbles:
      %B  = track(ctr);
      %nB = track(ctr+1);


      Hn = bitshift(B,-4);
      Ln = bitand(B,15);

      chan = [];

      msg_type = midi_msg_type(B,nB);

      % DEBUG:
      if (i==2)
        if (msgCtr==1)
          disp(msg_type);
        end
      end


      switch msg_type
```

```
     case 'channel_mode'

    % UNSURE: if all channel mode messages have 2 data byes (?)
    type = bitshift(Hn,4) + (nB-120+1);
    thedata = track(ctr:ctr+1);
    chan = Ln;

    ctr = ctr + 2;

    % ---- channel voice messages:
     case 'channel_voice'

    type = bitshift(Hn,4);
    len = channel_voice_msg_len(type); % var length data:
    thedata = track(ctr:ctr+len-1);
    chan = Ln;

    % DEBUG:
    if (i==2)
      if (msgCtr==1)
        disp([999  Hn type])
      end
    end

    ctr = ctr + len;

     case 'sysex'

    % UNSURE: do sysex events (F0-F7) have
    %  variable length 'length' field?

    [len,ctr] = decode_var_length(track, ctr);

    type = B;
    thedata = track(ctr:ctr+len-1);
    chan = [];

    ctr = ctr + len;

     case 'sys_realtime'

    % UNSURE: I think these are all just one byte
    type = B;
    thedata = [];
    chan = [];

    end

    last_byte = Ln + bitshift(Hn,4);

end % end midi event 'if'


currMsg.deltatime = deltatime;
currMsg.midimeta = midimeta;
currMsg.type = type;
```

```
    currMsg.data = thedata;
    currMsg.chan = chan;

    if (rawbytes)
      currMsg.rawbytes = track(ctr_start_msg:ctr-1);
    end

    midi.track(i).messages(msgCtr) = currMsg;
    msgCtr = msgCtr + 1;


  end % end loop over rawbytes
end % end loop over tracks

function val=decode_int(A)

val = 0;
for i=1:length(A)
  val = val + bitshift(A(length(A)-i+1), 8*(i-1));
end


function len=channel_voice_msg_len(type)

if     (type==128); len=2;
elseif (type==144); len=2;
elseif (type==160); len=2;
elseif (type==176); len=2;
elseif (type==192); len=1;
elseif (type==208); len=1;
elseif (type==224); len=2;
else
  disp(type); error('bad channel voice message type');
end


%
% decode variable length field (often deltatime)
%
%  return value and new position of pointer into 'bytes'
%
function [val,ptr] = decode_var_length(bytes, ptr)

keepgoing=1;
val = 0;
while (keepgoing)
  % check MSB:
  %  if MSB=1, then delta-time continues into next byte...
  if(~bitand(bytes(ptr),128))
    keepgoing=0;
  end
  % keep appending last 7 bits from each byte in the deltatime:
  val = val*128 + rem(bytes(ptr), 128);
  ptr=ptr+1;
end
```

```
%
% Read first 2 bytes of msg and
%  determine the type
%  (most require only 1st byte)
%
% str is one of:
%  'channel_mode'
%  'channel_voice'
%  'sysex'
%  'sys_realtime'
%
function str=midi_msg_type(B,nB)

Hn = bitshift(B,-4);
Ln = bitand(B,7);

% ---- channel mode messages:
%if (Hn==11 && nB>=120 && nB<=127)
if (Hn==11 && nB>=122 && nB<=127)
  str = 'channel_mode';

  % ---- channel voice messages:
elseif (Hn>=8 && Hn<=14)
  str = 'channel_voice';

  %  ---- sysex events:
elseif (Hn==15 && Ln>=0 && Ln<=7)
  str = 'sysex';

  % system real-time messages
elseif (Hn==15 && Ln>=8 && Ln<=15)
  % UNSURE: how can you tell between 0xFF system real-time
  %   message and 0xFF meta event?
  %   (now, it will always be processed by meta)
  str = 'sys_realtime';

else
  % don't think it can get here...
  error('bad midi message');
end
```

WRITE-MIDI()

```
function rawbytes=writemidi(midi,filename,do_run_mode)
% rawbytes=writemidi(midi,filename,do_run_mode)
%
% writes to a midi file
%
% midi is a structure like that created by readmidi.m
%
% do_run_mode: flag - use running mode when possible.
%    if given, will override the msg.used_running_mode
%    default==0.  (1 may not work...)
%
% TODO: use note-on for note-off... (for other function...)
%

% Copyright (c) 2009 Ken Schutte
% more info at: http://www.kenschutte.com/midi


%if (nargin<3)
do_run_mode = 0;
%end


% do each track:
Ntracks = length(midi.track);

for i=1:Ntracks

  databytes_track{i} = [];

  for j=1:length(midi.track(i).messages)

    msg = midi.track(i).messages(j);

    msg_bytes = encode_var_length(msg.deltatime);

    if (msg.midimeta==1)

      % check for doing running mode
      run_mode = 0;
      run_mode = msg.used_running_mode;

      % should check that prev msg has same type to allow run
      % mode...


      %      if (j>1 && do_run_mode && msg.type == midi.track(i).messages(j-
1).type)
%     run_mode = 1;
%      end


msg_bytes = [msg_bytes; encode_midi_msg(msg, run_mode)];
```

```
    else

      msg_bytes = [msg_bytes; encode_meta_msg(msg)];

    end

%    disp(msg_bytes')

%if (msg_bytes ~= msg.rawbytes)
%  error('rawbytes mismatch');
%end

    databytes_track{i} = [databytes_track{i}; msg_bytes];

  end
end


% HEADER
% double('MThd') = [77 84 104 100]
rawbytes = [77 84 104 100 ...
            0 0 0 6 ...
            encode_int(midi.format,2) ...
            encode_int(Ntracks,2) ...
            encode_int(midi.ticks_per_quarter_note,2) ...
           ]';

% TRACK_CHUCKS
for i=1:Ntracks
  a = length(databytes_track{i});
  % double('MTrk') = [77 84 114 107]
  tmp = [77 84 114 107 ...
       encode_int(length(databytes_track{i}),4) ...
       databytes_track{i}']';
  rawbytes(end+1:end+length(tmp)) = tmp;
end


% write to file
fid = fopen(filename,'w');
%fwrite(fid,rawbytes,'char');
fwrite(fid,rawbytes,'uint8');
fclose(fid);

% return a _column_ vector
function A=encode_int(val,Nbytes)

for i=1:Nbytes
  A(i) = bitand(bitshift(val, -8*(Nbytes-i)), 255);
end


function bytes=encode_var_length(val)

% What should be done for fractional deltatime values?
% Need to do this round() before anything else, including
%  that first check for val<128 (or results in bug for some fractional
values).
% Probably should do rounding elsewhere and require
% this function to take an integer.
val = round(val);

if val<128 % covers 99% cases!
    bytes = val;
    return
end
binStr = dec2base(round(val),2);
Nbytes = ceil(length(binStr)/7);
binStr = ['00000000' binStr];
bytes = [];
for i=1:Nbytes
  if (i==1)
    lastbit = '0';
  else
    lastbit = '1';
  end
  B = bin2dec([lastbit binStr(end-i*7+1:end-(i-1)*7)]);
  bytes = [B; bytes];
end


function bytes=encode_midi_msg(msg, run_mode)

bytes = [];

if (run_mode ~= 1)
  bytes = msg.type;
  % channel:
  bytes = bytes + msg.chan;  % lower nibble should be chan
end

bytes = [bytes; msg.data];

function bytes=encode_meta_msg(msg)

bytes = 255;
bytes = [bytes; msg.type];
bytes = [bytes; encode_var_length(length(msg.data))];
bytes = [bytes; msg.data];
```