RNN-based Learning Framework for Music Generation

Prianka Bose

Machine Learning and Optimization Workshop

Department of Mathematical Sciences
NJIT

November 10, 2022

Overview

- What is RNN?
- Drawbacks: Vanishing Gradient Problem
- Methods to overcome the drawbacks
- Explaining the LSTM architechture
- Music Generation using LSTM
- Results on Tabla Compositions

What is RNN?

- Recurrent Neural Network (RNN) is a type of ANN that uses sequential data apply the same operation to every member of a sequence because the outcome depends on earlier calculations, whereas in classic neural networks, outputs are independent of earlier calculations.
- RNN is good at learning temporal patterns but lacks global coherence due to lack of long term memory.
- RNN's may leave out important information from the previous steps.
- During back propagation, RNNs suffer from the vanishing gradient problem.

Vanishing Gradient Problem

- The sigmoid function is one of the most popular activations functions used in neural networks for activating the output layers in binary classification problems.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

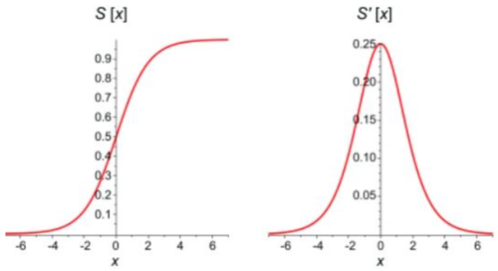


Figure: Left: Sigmoid Function, Right: First derivative of sigmoid function

Vanishing Gradient Problem

- A back propagation method minimizes the loss function by adjusting the weights and biases of the neural network.
- The gradient of the loss function is calculated with respect to each weight in the network.

$$W_{new} = W_{old} - \alpha * \frac{\partial \mathrm{E}}{\partial W_{old}} \tag{2}$$

  where E is the loss function, α is the learning rate and W_i correspond to the old and the new updated weight.
- The gradient of the loss function can be represented as a product of gradients of all the activation functions of the nodes with respect to their weights.

Drawbacks

- From Fig. 1 the derivative of the sigmoid function reaches a maximum of 0.25. When there are more layers in the network, the value of the product of derivative decreases until at some point the partial derivative of the loss function approaches a value close to zero, and the partial derivative vanishes.
- In RNNs, layers that get a small gradient update stops learning.
- Because these layers don't learn, RNN's can forget what it has seen in longer sequences, thus having a short-term memory.

Methods to overcome this issue

- Long Short Term Memory (LSTMs) and Gated recurrent units (GRUs) were created as the solution to short-term memory. They have internal mechanisms called gates that can regulate the flow of information.
- These gates can learn which data in a sequence is important to keep or throw away.
- Each LSTM memory cell maintains a hidden state vector and a cell state vector.
- The constant error backpropagation within memory cells results in LSTM's ability to bridge very long time lags to overcome the problem discussed above.
- LSTM memory cells stores information from previous timestep, forgets irrelevant information from the past and selectively updates the cell state.

LSTM architecture

$$
\begin{aligned}
f^{(t)} &= \sigma(W^f[h^{(t-1)}, x^{(t)}] + b^f) \\
i^{(t)} &= \sigma(W^i[h^{(t-1)}, x^{(t)}] + b^i) \\
\bar{C}^{(t)} &= tanh(W^c[h^{(t-1)}, x^{(t)}] + b^c) \\
C^{(t)} &= f^{(t)} C^{(t-1)} + i^{(t)} \bar{C}^{(t)} \\
o^{(t)} &= \sigma(W^o[h^{(t-1)}, x^{(t)}] + b^o) \\
h^{(t)} &= tanh(C^{(t)}) \times o^{(t)}
\end{aligned}
$$

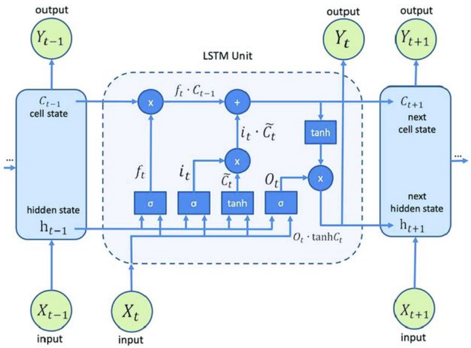


Figure: LSTM framework.

$$\hat{y}_i = softmax(Wh^{(t)} + b) \tag{3}$$

LSTM gating mechanism

- The "forget gate", a sigmoid layer, decides what data is to be discarded from the cell state. The input from the previous hidden layer h_{t-1} and the current input at time t, x_t, is examined which gives an output between 0 and 1 which is given by,

$$f_t = \sigma(W_f.[h_{t-1}, x_t] + b_f) \tag{4}$$

- To choose what additional data will be added to the cell state and stored is done in two parts. Initially, the "input gate layer," a sigmoid layer, selects the values that will be updated. The state is then updated with a vector of new candidate values, $\widetilde{C_t}$, which is created by a tanh layer. Merging the two and updating the state is done as follows,

$$i_t = \sigma(W_i.[h_{t-1}, x_t] + b_i) \tag{5}$$

$$\widetilde{C_t} = \tanh(W_C.[h_{t-1}, x_t] + b_C) \tag{6}$$

LSTM Gating Mechanism

- The previous cell state, C_{t-1}, is then updated with the new cell state, C_t is done by forgetting the irrelevant information and adding the new candidate values, scaled by the amount modified in each state value.

$$C_t = f_t * C_{t-1} + i_t * \widetilde{C_t} \tag{7}$$

- This output will be filtered and depends on the state of the cell. A sigmoid layer is executed first that determines which components of the cell state to output. Next, in order to output only the decided portions, the cell state is multiplied by the output of the sigmoid gate after passing the cell state through tanh which pushes the value between -1 and 1.

$$o_t = \sigma(W_o.[h_{t-1}, x_t] + b_o) \tag{8}$$

$$h(t) = o_t * \tanh(C_t) \tag{9}$$

Differences between GRU and LSTM

- GRU has two gates- Update and Reset whereas LSTM has three gates.
- GRU does not possess any internal memory, they don't have an output gate that is present in LSTM.
- In LSTM the input gate and target gate are coupled by an update gate and in GRU reset gate is applied directly to the previous hidden state. In LSTM the responsibility of reset gate is taken by the two gates i.e., input and target.
- GRU uses less training parameter and therefore uses less memory and executes faster than LSTM whereas LSTM is more accurate on a larger dataset.

Music Generation using LSTM

- **Data:** Obtain any set of MIDI files consisting of a single instrument.

  Step 1: The first step to implementing the neural network is to examine the data we will be working with.

  Step 2: Split the data in Notes and Chords. Notes will have information about the pitch and onset of a sound and Chords are essentially a container for a set of notes that are played at the same time.

  Step 3: After examining the data, we determine that the features to be used are the notes and chords as the input and output of the LSTM network.

Music Generation using LSTM

- **Preparing Data for the Network:** Append all the notes and chords of every sound file from the dataset and this forms a sequential list which can create the sequences that will serve as the input for our network.

  Step 1: Create a mapping function to map from string-based categorical data to integer-based numerical data.

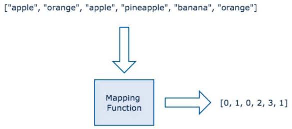

["apple", "orange", "apple", "pineapple", "banana", "orange"]
Mapping Function
[0, 1, 0, 2, 3, 1]

Figure: Categorical to Numerical Mapping Function

  Step 2: We have to create input sequences for the network and their respective outputs. The output for each input sequence will be the first note or chord that comes after the sequence of notes in the input sequence in our list of notes.

Music Generation using LSTM

- **Model Description:** There are four types of layer:

  LSTM layers is a Recurrent Neural Net layer that takes a sequence as an input and can return either sequences or a matrix.

  Dropout layers are a regularisation technique that consists of setting a fraction of input units to 0 at each update during the training to prevent overfitting.

  Dense layers or fully connected layers is a fully connected neural network layer where each input node is connected to each output node.

  The Activation layer determines what activation function our neural network will use to calculate the output of a node.

Music Generation using LSTM

- Calculate Loss:
  - The model is learning a multi-class classification, and categorical cross entropy loss is the appropriate loss function for this kind of issue.

$$\text{Loss} = - \sum_{i=1}^{\#\text{ of observations}} y_i . \log \hat{y}_i \tag{10}$$

which is an excellent indicator of how two discrete probability distributions are distinct from one another. The negative sign makes sure that when two distributions approach one another, the loss decreases.

Music Generation using LSTM

- **Music Generation:**

  Step 1: Once training is finished, the neural network is used to generate music by loading the weights that we saved during the training section into the model.

  Step 2: Pick a random index in the list of of note sequences as our starting point. This allows us to rerun the generation code without changing anything and get different results every time.

  Step 3: Create a mapping function to decode the output of the network. This function will map from numerical data to categorical data (from integers to notes).

  Step 4: To determine the most likely prediction from the output from the network, we extract the index of the highest value. The value at index X in the output array correspond to the probability that X is the next note.

Music Generation using LSTM

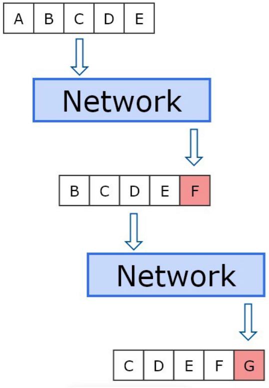


Figure: Input Output Sequence Flow of the Network.

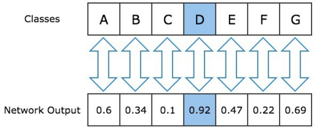
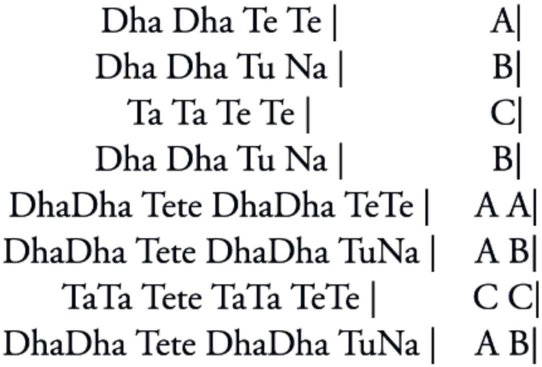


Figure: Probability of each note.

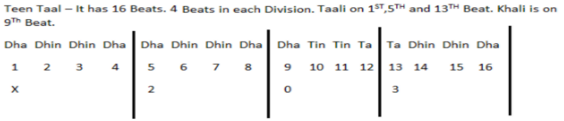
Some written tabla composition



Dha Dha Te Te |	A|
Dha Dha Tu Na |	B|
Ta Ta Te Te |	C|
Dha Dha Tu Na |	B|
DhaDha Tete DhaDha TeTe |	A A|
DhaDha Tete DhaDha TuNa |	A B|
TaTa Tete TaTa TeTe |	C C|
DhaDha Tete DhaDha TuNa |	A B|

Figure: Written Tabla Compositions.

Results on Tabla Compositions



Teen Taal – It has 16 Beats. 4 Beats in each Division. Taali on 1ST,5TH and 13TH Beat. Khali is on 9Th Beat.

Dha	Dhin	Dhin	Dha	Dha	Dhin	Dhin	Dha	Dha	Tin	Tin	Ta	Ta	Dhin	Dhin	Dha
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
X				2				0				3			

Figure: Standard Teental musical and metrical structure.

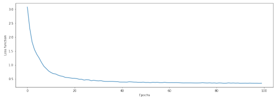


Figure: Loss function.

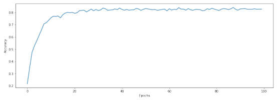
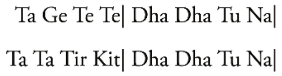


Figure: Accuracy.



Ta Ge Te Te| Dha Dha Tu Na|

Ta Ta Tir Kit| Dha Dha Tu Na|

Figure: New Teentaal composition generated by the model.

Thank you for listening!