Programming Assignment 4:
Generating Music with LSTM Networks

CSE 154: Deep Learning

Fall 2019

Instructions

Due Sunday, December 1^{st}:

1. Start on this assignment NOW! If you have any questions or uncertainties about the assignment instructions, please ask about them as soon as possible (preferably on Piazza, so everyone can benefit). We want to minimize any possible confusion about this assignment and have tried very hard to make it understandable and easy for you to follow.

2. Please hand in your assignment via Gradescope. We prefer a report written using LaTeXin NIPS format for each assignment. You are free to choose an alternate method (Word, etc.) if you want, but we still prefer NIPS format.

3. You should submit your code on Gradescope along with your report. For your own purposes, keep your code clean with explanatory comments, as it may be reused in the future.

4. We expect you to use Pytorch, a Deep Learning library, for the tasks in this assignment.

5. Please work in teams of size 2-3 (no more than 3). In extraordinary circumstances (e.g., you have a highly contagious disease and are afraid of infecting your teammate), we will allow you to do it on your own. Please discuss your circumstances with your TA, who will then present your case to me.

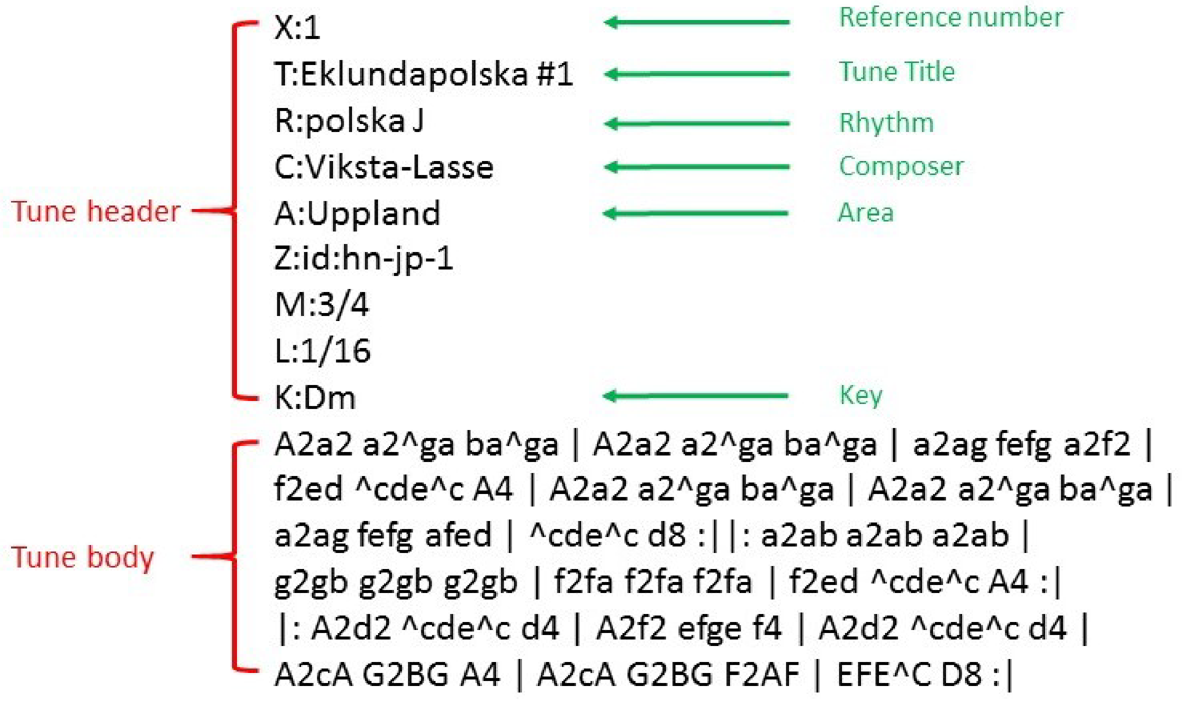
Character level LSTM for music generation (40 points)

In this assignment we will explore the power of Recurrent Neural Networks to deal with data that has temporal structure. In this problem, we will generate music in abc format. You will train an LSTM model using characters extracted from a music dataset provided to you and then run the network in generative mode to "compose" music.

Problem

1. **Getting familiar with the data** In this part, we are going to see how we can convert music from ABC notation to a playable format(.midi in this case) online. Go to the website mandolintab.net/abcconverter.php. Copy the text from sample-music.txt file (found under Data.zip,which contains music in ABC notation) and hit Submit. Download the tune in midi format and play it on your computer.

   You will be generating the music in a similar format as the sample file, which is ABC format. A sample ABC file is shown in Fig 1.



```
X:1
T:Eklundapolska #1
R:polska J
C:Viksta-Lasse
A:Uppland
Z:id:hn-jp-1
M:3/4
L:1/16
K:Dm
A2a2 a2^ga ba^ga | A2a2 a2^ga ba^ga | a2ag fefg a2f2 |
f2ed ^cde^c A4 | A2a2 a2^ga ba^ga | A2a2 a2^ga ba^ga |
a2ag fefg afed | ^cde^c d8 :| |: a2ab a2ab a2ab |
g2gb g2gb g2gb | f2fa f2fa f2fa | f2ed ^cde^c A4 :|
|: A2d2 ^cde^c d4 | A2f2 efge f4 | A2d2 ^cde^c d4 |
A2cA G2BG A4 | A2cA G2BG F2AF | EFE^C D8 :|
```

Reference number
Tune Title
Rhythm
Composer
Area
Key
Tune header
Tune body

Figure 1: A music file in ABC notation.

2. **Read in data**. Read in the data from the train.txt (found under Data.zip) file. This file contains multiple tunes in ABC format, with each tune delineated by <start> and <end> tags. Create a one-hot encoding of the unique characters in the text data and create data structures to efficiently get the index of a character in the encoding and vice-versa.

3. **Train a network** First, train an LSTM network to learn the structure of an ABC notation music file through **"Teacher Forcing"**. Teacher forcing works by using the teaching signal from the training dataset at the current time step, $target(t)$, as input in the next time step $x(t+1) = target(t)$, rather than the output $y(t)$ generated by the network.

   To start, we will be using BPTT(100): That means we will only backprop 100 time steps. The way we will do this is to use a minibatch size of 1, and that one example will be of length 100. Now, the *next* minibatch will be from the same piece of music, etc., until you reach the end. PyTorch should let you have a smaller chunk length in the last minibatch, which is very likely to not be of length 100 (the end of the piece). Repeat this for all music pieces in the training set.

Now, there is one issue. In order for the network not to "lose its place" in the piece over these mini-batches, you will need to copy the hidden unit state from the last time step in the previous minibatch (*from the same song*) into the next minibatch. When you start a new song, the hidden layer state should start at all 0's.

Note: In case of recurrent networks, "batch size" and "chunk length" are two distinct hyperparameters. Here, the batch size is 1, and the chunk length is 100.

- Divide each piece of music into chunks of length 100 characters. Obviously, as noted above, the last set of characters is very unlikely to be of length 100. That's fine.
- Pass the first chunk into your LSTM as input and pass the same chunk (left-shifted by 1) as the target. Backpropagate through time and update the weights of the RNN.
- Give the network the next chunk from the song, but carry forward the the last hidden state of the previous chunk to maintain continuity. Continue the process till you reach the last chunk (end of a piece) and then repeat with a different piece of music.
- Within an epoch, you can randomize the order of music samples. But don't randomize the order of chunks within a sample, so that the continuity is not disrupted.

Implementation Guidelines

- Start out with one hidden layer of LSTM units. Two hidden layers may work better, or they may overfit. You are welcome to try different configurations, but at least do the task with one hidden layer. You should use the available Pytorch modules to build the LSTM. Try using around 100 neurons in your hidden layer, but then try to optimize that number with the validation set.
- You should use a softmax output and cross entropy loss.
- Training, validation and test splits have been provided to you. Use the validation set for tuning your hyper-parameters (number of hidden units, optimizer learning rate, chunk-length). Report the negative log likelihood over the validation and test set. To get this metric, pass your validation/test data in the same manner as your training data, chunk by chunk carrying forward the last hidden state and average the cross-entropy error over all chunks. The only difference here is that you would have learning turned off.
- Evaluate your model on the training and validation set after every epoch. Include the training and validation loss curves in your report.
- Write a function to generate music using the approach described below. It is a good idea to print out the generated music after every few epochs (complete passes through the dataset) to see if the generated music is improving and makes sense.

4. **Generate music** In the generation stage, the idea is to "let the network run on its own", predicting the next character, and then use the network's prediction to obtain the next input character. There are at least two ways to obtain the next character: One is to take the maximum output. This is generally a bad idea, and leads to unmusical output. Also, it is deterministic. The right way is to flip an n-sided coin, assuming n outputs, based on the probability distribution at the softmax layer. You can make the network more or less deterministic by adjusting the Temperature parameter T in the softmax, but start with 1. What's interesting about this is you should get a different piece of music each time from the same network and the same priming (you can "prime" the network by giving it some initial characters from a song).

   Save the sequence to a file, and then use http://mandolintab.net/abcconverter.php to convert back to midi format, and play what is generated. At first, the network will probably predict garbage, which won't be translatable, but after training, you should get reasonable output.

   Note that you will want to re-seed the random number generator between generation runs.

   You can also "prime" the network by giving it part of an ABC file, and then let it generate the rest. It would be fun to compare what the network generates to what the original tune was.

5. **Try different temperature values** Experiment with different temperatures during generation. Is there a range that works better than 1?

For your report, provide:

(a) Generate 6 sample music pieces, two at $T = 1$, two at $T = 2$, and two at $T = 0.5$. They should be of reasonable length. Pick ones that sound the best to you for your report. Provide their ABC notation and music representation generated from http://mandolintab.net/abcconverter.php (if the ABC files are syntactically correct). Also upload the tunes in midi format on gradescope. Discuss your results. Also report all your hyperparameters.*(15 points)*

A sample output would look like the example shown in Figure 2. Note that this music was one of the best examples generated by our character level rnn in 2 hours; your tune can be shorter than this. Figure 3 shows the music in Figure 2 in standard musical notation.

```
X:44
T:Farscrisue FB cF2 d2Az|B3d fd :|
w: Laka Gan vout B'a
M:3/4
L:1/8
K:Bb
B>G | ABcB | G2cB MBABA | A4c/B/c/B/ cc | BA/G/ BB | G2B2 | cdcA | FABAA | B2z2 | B4z2 | B2A cBA | FAG:|
F2e|B3 GA BB B/G/B | G2B | A F/G/G/A/ B4
M:2/4
L:1/4
K:C
(AGA G2z2 | BBc2 c2A2|B2B2 B2c2 | d2ef (fce)f3g2e| [M:6/8]:
F2A G2B|c2c g2c a4|f2c2|cdc FGB|cBF BAA | B2F2 A2B2|c2z3 ebee|B2c2 bonastot eaut Fupteesafs2 sennc
e,>c/d/2|BA cc | c2G2|d ef/f/ | d2cBcB | B4 |]
```

Figure 2: Generated Music

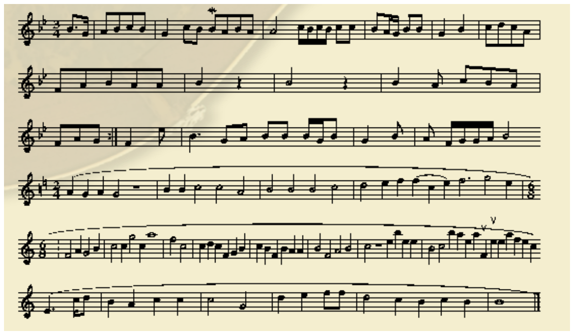


Figure 3: Music from Figure 2 in standard music notation.

(b) Provide a plot of your training loss and validation loss vs number of epochs. Discuss your findings. Report the results of your best model on the test set. *(5 points)*

(c) Try changing the number of neurons in your hidden layer for at least 3 different numbers, for ex. 50, 75 and 150. For other hyperparameters, use the best values from part b). Now, again plot your training loss and validation loss vs number of epochs on data. What do you observe? Discuss your findings. *(5 points)*

(d) Instead of the temperature-based random sampling, use the maximum at each time-step. Using the same initial seed (i.e. priming with the same sequence), generate one music sample through temperature based sampling with T = 0.7 and one sample through choosing the maximum output. Qualitatively compare and comment on the generation quality for the 2 sampling techniques. *(5 points)*

(e) Replace the LSTM module with a vanilla RNN (the hidden layer can be tanh, or whatever works best for you). Compare the training/validation loss curves for the 2 implementations (LSTM and vanilla RNN) for the same set of hyper-parameters (100 hidden units, same optimizer, same learning rate). Does it make better music? Discuss your findings. *(5 points)*

(f) **Feature Evaluation** - For one of your generated music samples, do forward propagation through the network and save the activation of each hidden neuron for each of the characters. Plot each of these activations as a heatmap and report the heatmap for at least 1 neuron whose activation pattern you can interpret as signaling some feature of the music.*(5 points).*

An example of one of these heatmaps is given in Figure 4. It shows some generated text from our trained network and shows how one hidden neuron behaves for each of the characters in the sequence. In this case, the neuron had low activation for the body of the music and high activation for header, which shows that it is able to recognize header of music in ABC format. *(5 points)*

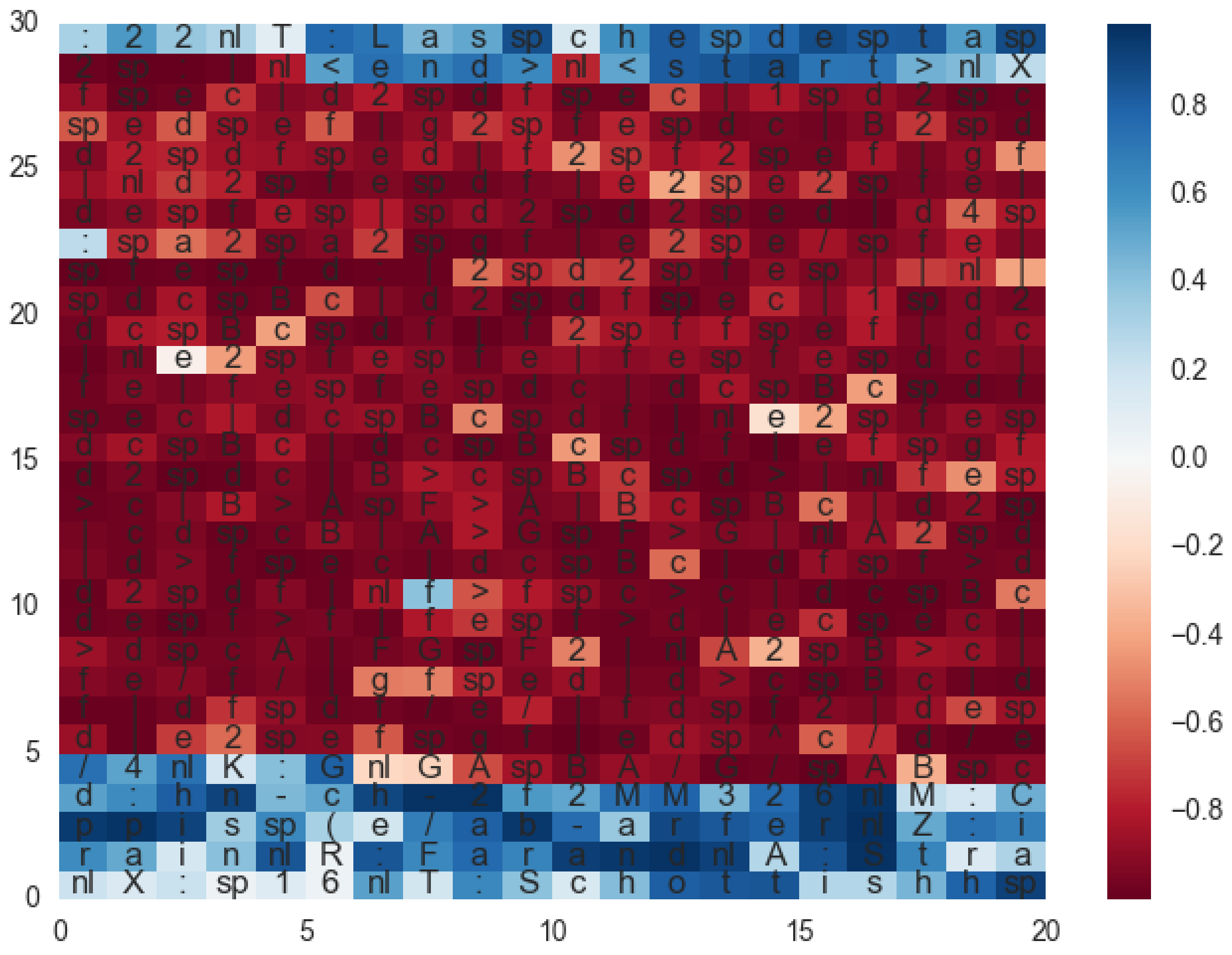


Figure 4: Heatmap of a hidden neuron: The x and y axes denote the order in which it should be read, i.e., it starts at the bottom, from left to right, in order. It shows that this particular neuron fires for the header of the music sample, then turns off for the body and fires again for the header of the next sample.

Note- The above heatmap has been generated using an LSTM network that was trained for a whole day. So your heatmap might not be that effective. But a heatmap giving some kind of insight would be good enough for this task. Also note that this "heatmap" is backwards, with hot colors representing low values. Yours should be the other way around.