Some cool title

Aaron A. Gauthier, Avijeet Kartikay and Pedro Uria Rodriguez

Machine Learning II

The George Washington University

Some cool title

Introduction

Something. Hey

BlaBla. Ho

Data

Mm

Encoding

Mmm

Network Architecture

Overview. As we have mentioned, we used an LSTM to generate music. This was also the reason we encoded the music as a sequence of notes, including the hold dimension. The network consists of an LTSM with an arbitrary number of neurons as the Hidden layer, followed by a fully connected layer with the same number of neurons as input dimensions as the Output Layer. We need this exact number of neurons because we want to predict the next note after a sequence of input notes, thus the output dimensions must be equal to the input dimensions. These will be 89 when generating melodies, and 178 when generating melodies and harmonies at the same time.

Output Layer. Because we are using a multi-many-hot encoding approach, we actually have a *multi-label* classification problem. That is, each output vector can belong to more than one class, as we have the hold dimension and also more than one note can be played at the same time. Then, we cannot use the regular *softmax* transfer function on the Output Layer, because this would only classify the output as a single note, as it models the joint probability of each dimension of the output multi-many-hot encoded vector. Indeed, this would result in only one dimension having a probability close to 1, while the rest are close to 0. Instead, we use a sigmoid function, which will model the probability of each dimension of our output vector individually, thus allowing for multiple probabilities close to 1, indicating that multiple notes and/or notes of larger duration are being generated.

Performance Index. The loss function used is then also different than *Categorical Cross-Entropy*. We are using *Binary Cross-Entropy*, because we are treating each dimension individually, so we only have two classes for each of them (0 or 1) but our output vector can have many class labels at the same time (many 1s).

Training

On a single song. After designing the architecture, we had to think about the training process. First, we wrote our code to train the network on only one note, in order to evaluate its ability to learn the pattern within the sequence of notes in such song. To do so, we choose a sequence of notes to input our network of arbitrary length, and then we compute the loss between the predicted output and the real one, that is, the next sequence of notes. So if we input the sequence $[0, 1, 2, 3]$, where each integer represents a note and its temporal position, then we compute the loss between the output and $[1, 2, 3, 4]$. We also wrote another approach where the loss is only computed between [4] and the last note of the output sequence. After computing this loss, we go backwards TODO: SPECIFY METHOD and update the weights and biases. Then we do the same thing with $[1, 2, 3, 4]$ as input and $[2, 3, 4, 5]$ as target, inputing also the hidden and cell states from the last iteration to the LSTM. We repeat this process over many epochs. We also added a condition on the code that decreased the learning rate by half when the loss does not decrease for 10 epochs, as we found this to be a somewhat common occurrence.

On many songs. After making sure our network was able to reproduce a single song with a high degree of accuracy, which was done by using a generator function that will be explained later, we expanded our code to work on many songs. Our first approach was .... TODO ....

Our definitive approach consists on the following: We simply stack all the encoded notes of all the songs on the training data together, sequentially. We end up with a huge matrix of sequences of notes, which does not distinguish where one song ends and where another song begins. Then we just input this matrix in the same way we did on our approach for only one song. Because looping over the whole songs takes a long time when we use a lot of them, we add the option of looping over only the first n sequences of each song. This way the network will not see the full songs, but will still learn some patterns, possibly more than enough, given that usually, the songs contain repeating patterns along their body of notes. Because we are also using many songs, our network will still see many different patterns.

Generator

Once we have trained our LSTM on some data, the final step is to use this network to generate new music. In order to do so, we used two slightly different approaches. The first one consists on ....

The second approach only takes a single note [1] as initial input. Then, we go forward and get a new note [2] as output. This note is actually a vector of probabilities, so we convert it to our multi-many-hot encoded vector of 1s and 0s by selecting either all the dimensions with probability greater than an arbitrary threshold, or if all the dimensions have probabilities less than this threshold, we select the

dimension with the highest probability. In the case of generating melody + harmony, we do this separately for each hand. We also add another threshold to include the hold when the probability of such hold is close to the highest probability. After deciding on the encoded note, we add this note to the sequence of notes we will be using as input, so now we have $[1, 2]$, and go forward again to generate, in this case, a sequence of two notes $[2', 3]$. This time we only take the last generated note [3] and add it to the next input sequence $[1, 2, 3]$. We do this again and again, until we reach a sequence length equal to the one our network had been trained with. When we reach this length, say we have $[1, 2, 3, ..., i]$, we use this sequence as input, but when we get our next generated note $[j]$, we remove the first note from our input sequence and add this new latest note, so on the next iteration we will input $[2, 3, ..., i, j]$. We repeat this until the number of wanted generated steps has been reached. Note that each output note (last note of each output sequence) is being added to a list of generated notes even when being removed from our input sequence at some point, and that we can keep going on forever and generate songs of any duration. We also remove the first generated notes until the constant sequence length has been reached, because these notes will not be representative of a network which was trained with a larger length of sequences.

TODO: TALK ABOUT GETTING STUCK ON A PARTICULAR NOTE AND A PARTICULAR SEQUENCE, ALTHOUGH THE CONDITIONS TO AVOID THIS ARE DEFAULT TO FALSE. ALSO ABOUT THE REMOVING RESTS STUFF.

Experiments and Results

Hyperparameters. Uhuh

Best Results. Blabla

Conclusions

Mmmm