ReadMe - AIDL UPC

"MIDI Music Group" by Ferran Carrascosa, Oscar Casanovas, Alessandro Pintaudi and Martí Pomés. Advisor: Carlos Segura. July 2019.

Github link: https://github.com/alepintaudi/music-generation.git

Task Definition

This project presents the implementation of a basic MIDI compositions music generation system while we learn about the implementation of 3 Neural Network models (LSTM Seq2One with Keras, LSTM Seq2One with PyTorch, LSTM Seq2Seq with PyTorch). We also provide an overview of the current commercial and research projects on Music Generation systems using Deep Learning as well as other simpler but very interesting individual projects developed recently.

Introduction

Over the past 4 years, we have seen impressive progress in the field of generative music and Artificial Intelligence thanks to the developments made on Deep Learning technologies.

The goal of this Postgraduate project is to get hands-on experience on building our own Model that is trained with a collection of MIDI files and then is modified to generate new short composition snippets that are evaluated on their degree of musicality and closeness to passing a Turing test.

It is important to note that none of the four project participants had previous experience implementing Deep Learning models and that all the knowledge has been acquired during the past 5 months over the UPC School - Artificial Intelligence with Deep Learning Postgraduate Degree and also through online resources and publications. We also want to thank our supervisor Carlos Segura Perales for his dedication, availability and valuable insights provided over the past 2 months.

Statement of the Problem

Recent projects such as Bachbot (Feynman Liang), Coconet (Google Magenta), Music Transformer (Google Magenta) and MuseNet (OpenAI) among others, have demonstrated technology capable of achieving musical composition results able to pass the Turing test on certain cases and music genres/instrumentation.

At the same time, several private companies have been working on research and commercial applications of similar systems for certain "mainstream" use cases such as described in the Benchmark Section.

During the latest AI Music news boom, many journalists predict that soon we will see more and more of these generative systems deployed commercially for several real world applications, such as royalty-free, low-cost, quickly personalized and on-demand custom-tailored music.

These use cases could have a strong “Product/market fit” for new automatic music composing systems able to provide professional quality music at a very affordable rate for media companies, freelancers, individuals, music enthusiasts as well as general music listeners.

Benchmarks

For reference, we believe it is also important to mention the following Deep Learning Music Generation projects that show the output quality of current State of the art commercial and research applications:

- Google Magenta (CocoNet and Music Transformer).
- MuseNet (OpenAI): A deep neural network that can generate 4-minute musical compositions with 10 different instruments, and can combine styles from country to Mozart to the Beatles.
- Spotify (Spotify Creator Technology Research Lab): The lab focuses on making tools to help artists in their creative process.
- Jukedeck: Generating Royalty Free Music for Youtube videos and other applications.
- AI.music: Several Products related to generating tailored versions of songs for each user.
- AIVA: AIVA, the Artificial Intelligence music composer that creates original & personalized music for your projects.
- AMPER: Amper is an AI music composition company that develops tools for content creators of all kinds.
- ALYSIA: ALYSIA allows everyone to create original songs in under 5 minutes. Including lyrics and melodies. Get ready to sing karaoke on your own original music.
- Mubert: Generative Channels. Each generative channel is based on a fixed number of tags which algorithm uses to create endless streams.
- Endel: Personalized audio environments that help you focus and relax.
- IBM Watson Beat: The Watson Beat code employs two methods of machine learning to assemble its compositions; reinforcement learning that using the tenets of modern western music theory to create reward functions, and a Deep Belief Network (DBN) trained on a simple input melody to create a rich complex melody layer.
- Microsoft Research: Music Generation with Azure Machine Learning.
- Sony Flow Machines: Flow Machines is a research and deployment project aimed at providing augmented creativity for music artists.

Inspiration

We started our research looking at the BachBot Project, which inspired us to carry on with our first experiment. The BachBot Project aims at creating an LSTM based Model "which can generate and harmonize Bach chorales in a way that's indistinguishable from Bach's own work".

BachBot Project:

http://bachbot.com/

BachBot Paper:

http://www.mlmi.eng.cam.ac.uk/foswiki/pub/Main/ClassOf2016/Feynman_Liang_8224771_assignsubmission_file_LiangFe

BachBot Github:

https://github.com/feynmanliang/bachbot

Hypothesis

Since Natural Language Processing Deep Learning Methods are used to model and generate written language, they can also be used to model, reproduce and generate Musical Language.

Experiments

Here you'll find the various experiments that led us to the final model of music generation (4th Experiment). At each step we explain the issues we detected and how we tried to find a solution to move forward and get a step closer to the target solution previously defined: generate a model capable of being trained with music and eventually generate a novel midi file output with similar music style.

1st Experiment

Motivation

The way we approached the problem was trying to reproduce what has been done in classical NLP models, in which the text input is segmented in small sequences forming a dictionary, used to predict the following sequence of words.

Thus we decided to convert MIDI files into text to then pass them through our first developed network, written in Keras.

We based our first network on an existing model, which tackled the problem in the same way (converting of MIDI files into text and then using an LSTM to predict new notes), training it with a dataset of "Classical" piano music MIDI compositions from the Final Fantasy videogame soundtrack.

Here the Github repository and dataset that inspired our first experiment:

https://github.com/Skuldur/Classical-Piano-Composer/blob/master/predict.py
https://github.com/Skuldur/Classical-Piano-Composer/tree/master/midi_songs

As mentioned above the first step is to transform the MIDI files into text. Music21 Python Library (https://web.mit.edu/music21/) was used in order to convert the raw midi files into a stream of musical events (like words in a sentence). Our first approach into modelling the musical language was to map all the events of the input dataset (notes and chords) into a dictionary. Each note would corresponded to a combination of letters and numbers. It is important to note that for this first experiment we discarded the notes duration values, since we wanted our network to only focus on the pitch component of the music, which should be an easier problem to tackle than predicting pitch+duration of each note. Also important to note that this notation forces the model to include the chords events as a individual "word" of our vocabulary.

e.g.

E3
E-6
E6
E-5
A5
E5

Code to generate the vocabulary of notes and chords:

```
def get_notes(n=-1):
""" Get all the notes and chords from the midi files in the ./midi_songs directory """
notes = []
```

```
for i,file in enumerate(glob.glob("/content/Classical-Piano-Composer/midi_songs/*.mid")):

    if n==i:
        break

    midi = converter.parse(file)

    print("Parsing %s" % file)

    notes_to_parse = None

    try: # file has instrument parts
        s2 = instrument.partitionByInstrument(midi)
        notes_to_parse = s2.parts[0].recurse()
    except: # file has notes in a flat structure
        notes_to_parse = midi.flat.notes

    for element in notes_to_parse:
        if isinstance(element, note.Note):
            notes.append(str(element.pitch))
        elif isinstance(element, chord.Chord):
            notes.append('.'.join(str(n) for n in element.normalOrder))

with open('/content/Classical-Piano-Composer/data/notes', 'wb') as filepath:
    pickle.dump(notes, filepath)

return notes
```

The Model

In this initial experiment the model presented two layers of LSTM with Dropout and one last fully connected layer to generate the final vocabulary.

At the end we apply an activation function (Softmax) and combine it with a Cross-Entropy Loss function. The model also optimises the results using an RMSprop optimizer.

```
def create_network(network_input, n_vocab):
model = Sequential()
model.add(LSTM(
512,
input_shape=(network_input.shape[1], network_input.shape[2]),
return_sequences=True
))
model.add(Dropout(0.3))
model.add(LSTM(512, return_sequences=True))
model.add(Dropout(0.3))
model.add(LSTM(512))
model.add(Dense(256))
model.add(Dropout(0.3))
model.add(Dense(n_vocab))
model.add(Activation('softmax'))
model.compile(loss='categorical_crossentropy', optimizer='rmsprop')
```

Training

Results of the trained model with the classical piano compositions for 95 EPOCHs:

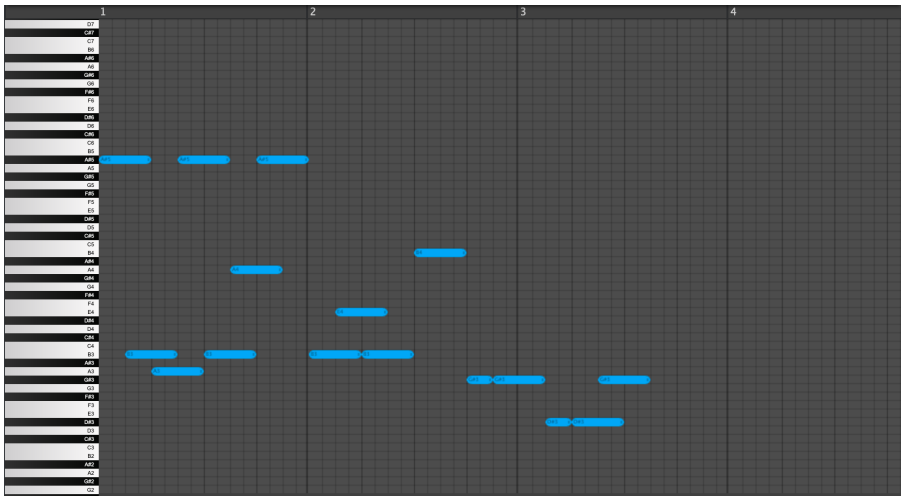
```
0 0.007250308990478516 1.0
1 0.026912927627563477 1.0
2 0.00890620518475771 1.0
3 0.06554309278726578 1.0
4 0.3337826728820801 0.6666666865348816
5 0.8159777522087097 0.6666666865348816
```

[...]

```
87 0.0015482902526855469 1.0
88 0.0032432873267680407 1.0
89 0.0057816109620034695 1.0
90 0.0038841168861836195 1.0
91 0.012463927268981934 1.0
92 0.006290217395871878 1.0
93 0.013199646957218647 1.0
94 0.003600984811782837 1.0
```

Audio example of the output

Example of one midi file musical output we got from the above model, trying to predict the continuation of an input short-sequence from the dataset:



https://s3-us-west-2.amazonaws.com/secure.notion-static.com/54d9a704-6b3e-4588-8221-1b29e7586a95/test_output(1)(1).mp3

Results and Conclusions of the 1st Experiment

Since this model is trained without note duration values, all the notes at the input and output have the same fixed step duration. This limitation helps this small network to focus only on learning about notes sequence composition (pitch) but it still does not perform amazingly well harmonically. The first part of this example sounds quite random even if it has an interesting pattern, and the ending part of this example sounds much more natural (closer to the performance of a professional music composer). The strange melodic part of the beginning of this sequence as well as the fixed note duration steps would make it difficult to pass any kind of musical composition Turing test.

2nd Experiment

Motivation

The biggest challenge with the second experiment has been transforming the code from the first experiment from Keras to PyTorch.

In order to achieve this we run a research to discover similar models within the music field that had been developed in Pytorch.

We particularly focused on a Network used for style transfer in which we specifically appreciated the way they encoded the note information from the MIDI files Dataset.

Here is an extract from the paper:

Firstly, we quantize each MIDI file to align to the particular time interval thereby eliminating imprecisions of the performer. Secondly, we encode the input MIDI file into a T ×P matrix where T is the number of time steps in the song and P is the number of pitches in the instrument (Example, for piano with 88 keys we have 88 pitches). Further, in each value of the matrix we encode the information regarding note using a 2-D vector i.e., [1 1] note articulated, [0 1] note sustained and [0 0] note off.

(**Source** - ToneNet : A Musical Style Transfer https://towardsdatascience.com/tonenet-a-musical-style-transfer-c0a18903c910)

Since we had a similar dataset (piano composer dataset, same as 1st experiment) and a similar problem to solve, we decided to adopt the 88 piano keys to define the length of our vector of notes.

In this experiment we also introduced the duration value for each note, using a "sampling frequency" that takes a snapshot of 88 piano keys for all the sampling time steps of the composition. We understood that having only one vector of notes active/inactive would have not allowed us to create an homogeneous composition.

Thus we passed from a 0 / 1 activation for each one of the 88 notes to one based on couples of binary numbers where:

(1,0) = the note starts

(0,1) = the note continues

(0,0) = the note is not played

The Model

The model chosen was an LSTM based on a sequence to one generator, in which each sequence of notes only generated one single note as output.

```
class NextNoteModel(nn.Module):
def init(self, input_dim, rnn_dim=512, rnn_layers=2):
super(NextNoteModel, self).init()
self.rnn = nn.LSTM( input_size=input_dim, hidden_size=rnn_dim, num_layers=rnn_layers, batch_first=True, dropout=0.2)
self.classifier = nn.Sequential(
nn.Linear(rnn_dim, 256),
nn.ReLU(),
nn.Dropout(0.5),
nn.Linear(256, input_dim)
)
```

At this point we decided to introduce Cross-Entropy Loss into our model in order to measure the error between computed outputs and the desired target outputs of the training data.

```
self.loss_function = nn.CrossEntropyLoss()
```

```
def forward(self, x):
```

```
output, (hn, cn) = rnn(input, (h0, c0))
```

```
output, (hn, cn) = self.rnn(x)
  return self.classifier(output[:,-1,:]) #no hace falta la softmax
```

```
def loss(self, x,y):
y_pred = y.argmax(dim=1)
return self.loss_function(x,y_pred)
```

```
def accuracy(self, x, y):
x_pred = x.argmax(dim=1)
y_pred = y.argmax(dim=1)
return (x_pred == y_pred).float().mean()
```

Training

Once we started training our network we noticed that while the training loss was going down, the validation loss initially decreased to eventually end up higher than it was at the beginning.

Accuracy increased during training but remained quite low during validation, tending to be close to zero:

```
0 3.900423049926758 4.751437187194824 0.1875 0.0
1 3.7298624515533447 4.751437187194824 0.125 0.0
2 3.70011830329895 4.751437187194824 0.125 0.0
3 3.7687315940856934 4.751437187194824 0.0625 0.0
4 3.727893590927124 4.751437187194824 0.0 0.0
5 3.28770112991333 4.751437187194824 0.0625 0.0
```

[...]

```
50 1.5549570322036743 5.31388521194458 0.625 0.2857142984867096
51 1.3869068622589111 5.31388521194458 0.6875 0.2857142984867096
52 1.51986825466156 5.31388521194458 0.625 0.2857142984867096
53 1.9102293252944946 5.31388521194458 0.4375 0.2857142984867096
54 1.8994156122207642 5.31388521194458 0.4375 0.2857142984867096
55 1.4201409816741943 5.31388521194458 0.5625 0.2857142984867096
```

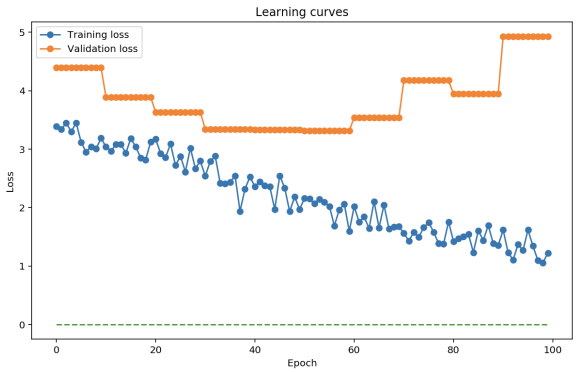
[...]

```
96 0.916029691696167 6.139454364776611 0.6875 0.0
97 1.110654354095459 6.139454364776611 0.6875 0.0
98 0.9400495886802673 6.139454364776611 0.6875 0.0
99 1.3639230728149414 6.139454364776611 0.5625 0.0
```

Learning Curves

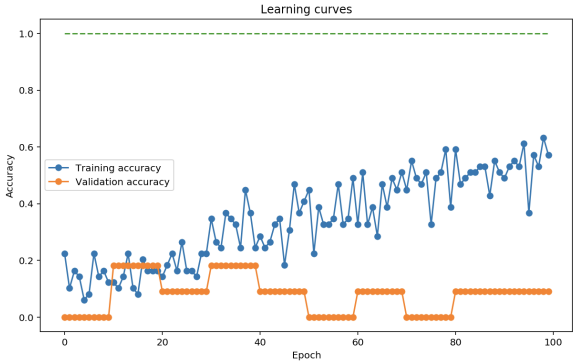
Loss

Despite the poor results obtained in terms of both loss and accuracy, we observed that during the training and validation steps, our network overfitted.



Accuracy

A similar behaviour can be observed for the learning curves of accuracy, where from an initiation convergence the two curves end up going far from each other.

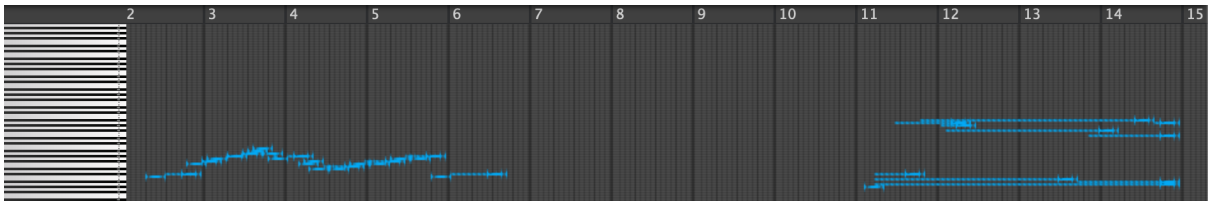


Testing

Results

Name	Loss	Accuracy
Mean	3.7440576871236164	0.1114343412220478
Standard Deviation	0.2657922414456095	0.03785291538125063

Audio example of the output



https://s3-us-west-2.amazonaws.com/secure.notion-static.com/cff1052a-9dd8-4511-ab3a-26ea66a73738/test_output_(1)_(1).mp3

Results and Conclusions of the 2nd Experiment

After several tests with this model it was apparent that the network almost always predicts the same note. Thus instead of generating an harmonic composition the model creates a very flat and random piece of music, far from being close to a realistic musical composition similar to the ones from the training dataset. The reason behind this behaviour can be deduce when looking at the task more closely. By subdividing the time step into such a small fraction of a quarter note (divided by twelve), and just predicting one time step, it is probable that for most of the times, the notes on last time step are just identical to the previous time steps. Given also the tendency of an LSTM without attention to lose track of long time correlations, it is clear why holding the same note could be considered a good solution by our system.

3rd Experiment

Motivation

In this third experiment we tried to solve the issues found during the second experiment. The biggest problem found previously is the monotony of the generated output sequence (same note repeated over and over), we decided to try to solve it moving towards a more standard Sequence to Sequence network.

Here the decoder was finally generating a sequence of the same length as the one sent to the encoder.

We have also increased the time step (from 1/12 to 1/4 of a quarter note) in order to avoid the problems mentioned above (monotony due to the LSTM being unable to look too far back). This makes our system less capable to understand fast/detailed note durations but should allow the network to focus more on the pitch (notes) and harmony, improving the variety and complexity of the notes generated.

The Model

Encoder

The encoder is made of 2 LSTM layers having 512 neurons each with a dropout layer in between them.

We then have a decoder with a first fully connected layer that reduces dimensionality to 256, and then an activation function and dropout before the final fully connected layer that reduces the dimensionality to the input size of the piano roll.

Cross-Entropy Loss function is finally applied to the results.

```
class Seq2Seq(nn.Module):
  def __init__(self, input_dim, rnn_dim=512, rnn_layers=2):
      super(Seq2Seq, self).__init__()
      self.encoder = nn.LSTM( input_size=input_dim, hidden_size=rnn_dim, num_layers=rnn_layers, batch_first=True, dropout=0.2)
      self.decoder = nn.LSTM( input_size=input_dim, hidden_size=rnn_dim, num_layers=rnn_layers, batch_first=True, dropout=0.2)
      self.classifier = nn.Sequential(
          nn.Linear(rnn_dim, 256),
          nn.ReLU(),
          nn.Dropout(0.5),
          nn.Linear(256, input_dim)
      )
      self.loss_function = nn.CrossEntropyLoss() #combina logsoftmax y NLLLoss
```

Results are then normalised using Softmax.

```
self.norm = nn.Softmax()
```

Decoder

The decoder used in here is providing what we missed in the previous experiment: an output sequence of the same length as the one at the input of the encoder. In this way the network is able to generate a sequence made of logical sounds.

```
def forward(self, x,y):
#           output, (hn, cn) = rnn(input, (h0, c0))
      output, (hn, cn) = self.encoder(x)
      #y = torch.cat([torch.zeros( (y.shape[0],1,y.shape[2]),).to(device) , y] , dim=1)

      output, (hn, cn) = self.decoder(y, (hn,cn)) #el y tiene que tener padding de cero, por ejemplo para marcar el inicio de s

      shape = output.shape

      x=output.unsqueeze(2)

      #x= output.view(output.shape[0],output.shape[1]*output.shape[2])

      x = self.classifier(x) #no hace falta la softmax

      x = x.view(shape[0],shape[1],-1)

      return x
```

Training

Results of Loss and Accuracy for both Training and Validation for 100 EPOCHs:

```
0 5.035482406616211 5.037669658660889 0.23246753215789795 0.6515151858329773
1 3.8860442638397217 5.037669658660889 0.27207791805267334 0.6515151858329773
2 1.9315378665924072 5.037669658660889 0.7129870057106018 0.6515151858329773
3 1.4045366048812866 5.037669658660889 0.798701286315918 0.6515151858329773
4 1.2527923583984375 5.037669658660889 0.8009740114212036 0.6515151858329773
5 1.1773496866226196 5.037669658660889 0.807467520236969 0.6515151858329773
```

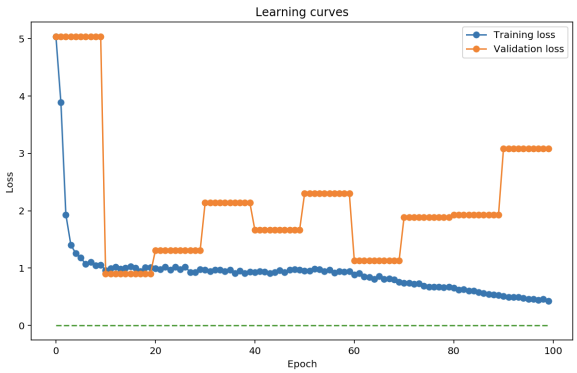
[...]

```
95 0.46134012937545776 3.0863828659057617 0.8600649237632751 0.7196969985961914
96 0.4543679654598236 3.0863828659057617 0.857467532157898 0.7196969985961914
97 0.4416855573654175 3.0863828659057617 0.8639610409736633 0.7196969985961914
98 0.4552253186702728 3.0863828659057617 0.8620129823684692 0.7196969985961914
99 0.42390283942222595 3.0863828659057617 0.8652597069740295 0.7196969985961914
```

Learning Curves

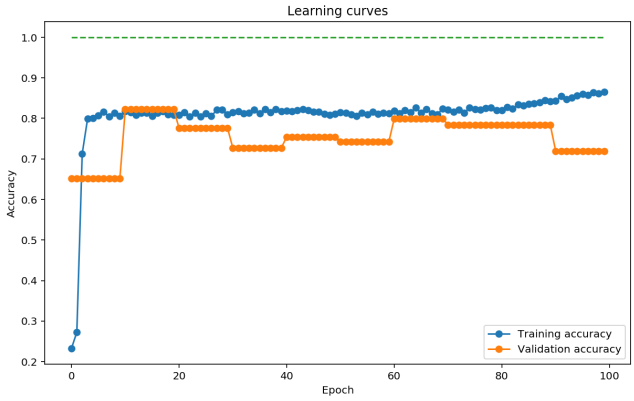
Loss

The loss suddenly decreases to then increase and end up overfitting our model



Accuracy

The two curves are immediately getting close, to then separated at the end ending with an accuracy around 70%



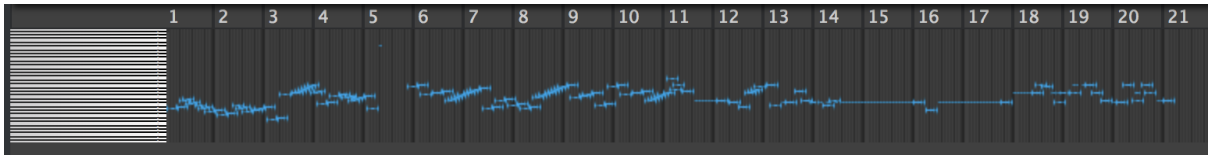
After training the system with training and validation dataset and using the resulted weights, here below the mean and standard deviation of both Loss and Accuracy during the testing phase.

Testing

Results

Method	Loss	Accuracy
Mean	1.4351666087195987	0.8488375544548035
Standard Deviation	0.09373127984156844	0.008533305530918942
Untitled		

Audio example of the output



https://s3-us-west-2.amazonaws.com/secure.notion-static.com/665a6290-7ea5-4ea4-8a0d-c077454d3afb/test_seq2seq_(2).mp3

Results and Conclusions of the 3rd Experiment

Contrary to the previous experiment here the loss function is able to detect sequences and therefore the output shows more variation, although it is still an issue to be improved on further experiments.

Another point to be mentioned is that the network is now able to generate a sequence of audible sounds, the decoder is generating each new sequence based on real values instead of the previously generated output. This means that, in this model, the decoder behaves with what we could call an implicit full teacher forcing. During training, the whole target sequence is used at the entry of the decoder instead of using the prediction at *n* to predict the output at *n+1*.

Another detected issue at this point was that we kept Cross-Entropy Loss (one single class/note as a target) since our 1st Experiment while trying to generate polyphonic music. This was correct for the 1st Experiment, since there we were using one-hot encoding even for chords (multi-note events). On the second and third experiments we moved away from one-hot encoding into 176 long vector (the Piano Roll) with multiple possible notes encoded in a single time step. Our mistake was to keep a Loss Function for a multi-class problem into a multi-label problem.

4th Experiment - Final Version

Motivation

In this fourth experiment we tried to improve the previous model, starting by solving its main problem of generating only monophonic music and generating polyphonic music at the output.

The Architecture

To make a recap. We keep the basis of our model mentioned in experiment 3: an encoder with 2 Layers LSTM (hidden dimension of 512) and a Decoder with a 2 Layer LSTM (hidden dimension of 512) and a Classifier based on 2 Layers of fully connected (intermidiate dimension of 256). All of them with dropout implemented.

In this model we also introduced a new hyper parameter, brought by the BCE Loss function that expects a threshold value to be checked against the output when applying the Sigmoid function.

The Loss Function

In order to achieve our main goal the Loss function had to be modified. To better reproduce the multi label nature of our task, a Binary Cross-Entropy Loss was implemented. This loss function enabled us to parse more than one line of instrument at the time and so ending up generating polyphonic sounds.

BCE with logits was chosen (instead of a standard BCE), as this loss combines a Sigmoid layer (which was not implemented in our previous model) and the BCELoss in one single class. *This version*, according to PyTorch's main documentation, *is more numerically stable than using a plain Sigmoid followed by a BCELoss as, by combining the operations into one layer, we take advantage of the log-sum-exp trick for numerical stability.*

```
self.loss_function = nn.BCEWithLogitsLoss()
```

Focal Loss

At this point we faced a problem, in which our loss was rapidly going below 0, due to the nature of the text, in which most of the times, 86-87 notes of the 88 available were easily guessed as not being played and having a value of 0.

We learned that in image classification, to solve tasks of object detection, some models (e.g. YOLO) had been using a type of loss that allowed them to weight each loss taking into account the several possible objects to be detected.

As our model presented a similar problem, we decided to introduce Focal Loss within our network.

```
def focal_loss(self, x, y, alpha = 0.5, gamma = 2.0):
'''Focal loss.
Args:
x: (tensor) sized [batch_size, n_forecast, n_classes(or n_levels)].
```

```
y: (tensor) sized like x.
Return:
(tensor) focal loss.
'''
    x = x.view(-1,x.shape[2])
    y = y.view(-1,y.shape[2])

    t = y.float()

    p = x.sigmoid().detach()
    pt = p*t + (1-p)*(1-t)         # pt = p if t > 0 else 1-p
    w = alpha*t + (1-alpha)*(1-t)  # w = alpha if t > 0 else 1-alpha
    w = w * (1-pt).pow(gamma)

    return F.binary_cross_entropy_with_logits(x, t, w, reduction='sum')
```

Teacher Forcing

On top of that we decided to introduce **Teacher Forcing** (initially set up with a threshold of 0.5) in order to improve model skill and stability at training time.

> Models that have recurrent connections from their outputs leading back into the model may be trained with teacher forcing.

— Page 372, Deep Learning, 2016.

In this way we were able to quickly and efficiently train our model that was using the ground truth from the prior time step as input.

```
def forward(self, x,y,teacher_forcing_ratio = 0.5):
    output, (hn, cn) = self.encoder(x)

    seq_len = y.shape[1]
    outputs = torch.zeros(y.shape[0], seq_len, self.input_dim).to(device)
    input = y[:,0,:].view(y.shape[0],1,y.shape[2])
    for t in range(1, seq_len):
        output, (hn, cn) = self.decoder(input, (hn, cn))
        teacher_force = random.random() < teacher_forcing_ratio
        shape = output.shape
        x=output.unsqueeze(2)
        x = self.classifier(x)
        x = x.view(shape[0],shape[1],-1)
        output = (x > self.thr).float()
        input = (y[:,t,:].view(y.shape[0],1,y.shape[2]) if teacher_force else output.view(y.shape[0],1,y.shape[2]))
        outputs[:,t,:] = x.view(y.shape[0],-1)
  return outputs
```

New evaluation metrics

The last change we made to our system was to change the metrics on how we evaluate the training. Although this doesn't help on the results, it helps us better understand how the system evolves during training. In that regard we have stopped using accuracy as a metric since it quickly reaches over 90% (most of the piano roll states at each step are silenced in both target and prediction) and, therefore, it doesn't orient us at all. Instead, we've started using recall and precision (see graph bellow) which better match the task and clearly show more meaningful and understandable results. Another value that is worth monitoring while training is the "density" of the notes (how many notes would be held during the sequence divided by the whole sequence piano roll). This metric allows us to discard any system that, even while lowering the loss, doesn't manage to generate high enough levels of density, since our focus is on generation.

```
def recall(self,x,y):
      x_pred = (x > self.thr).long() #where thr is 0 by default
      return torch.mul(x_pred.float(),y).float().sum()/y.sum()

  def precision(self,x,y):
      x_pred = (x > self.thr).long() #where thr is 0 by default
      return torch.mul(x_pred.float(),y).float().sum()/x_pred.float().sum()
```

Training

Results of Loss, Recall, Precission and Note density (a good value tends to be above 0.025, from training data)

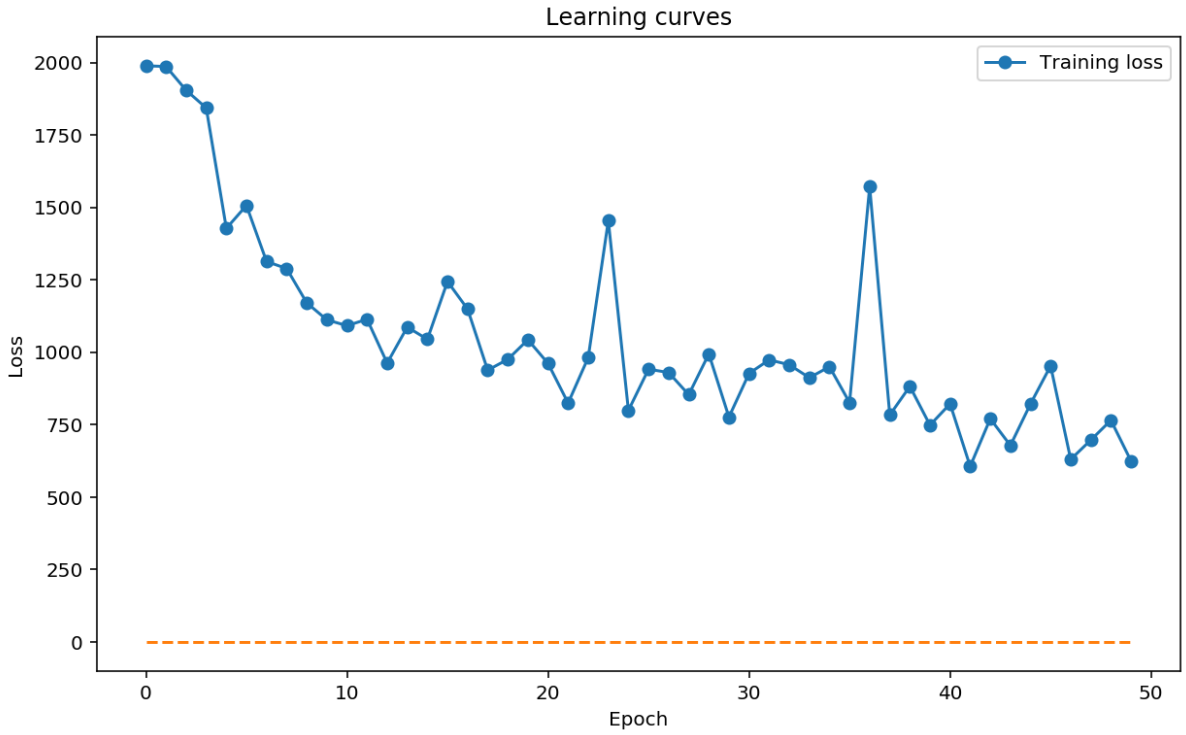
```
0 3153.242187 0.000173 0.5 7.3982e-06
1 3458.074707 0.0 nan 0.
2 3163.663574 0.0 nan 0.
3 3250.581298 0.006364 0.629032 0.0002
4 3162.822753 0.010844 0.270833 0.0009
```

[...]

```
45 2207.135742 0.316916 0.349118 0.0193
46 2128.198730 0.342842 0.367018 0.0200
47 2415.378906 0.362711 0.390139 0.0233
48 2270.105224 0.408077 0.384059 0.0261
49 2513.393066 0.378319 0.401861 0.0254
50 2423.801757 0.315098 0.373057 0.0200
```

Learning Curves

Loss



Recall and Precision

Results and Overall Conclusion of the current state

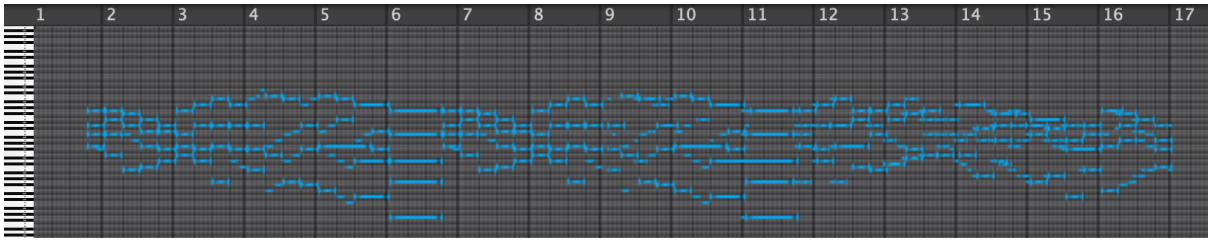
We can see that the results have improve substantially (even if only for the fact that now our system is capable of generating some notes above the threshold). This could be due to the larger dataset and training time that had been used (we clearly can appreciate overfitting when comparing results). On the examples above, Train represents the music generated while initializing the hidden states of the decoder with the hidden states generated by the encoder with a train example (first half of the song is the training example). Test represents the same but with a test example as a "initialiser of the hidden states". Predict 1 and 2 represent what happens when we just used a decoder with the hidden states initialized at zero or a normal distribution, respectably. Those two last examples show that the system is still far from generating the music fully independent as the sequence tend to converge rapidly into silence.

That being said, some extra improvements where introduced during training, that showed an improvement of the, at least, training results:

- Modification of the alpha parameter of the focal loss (from 0.5 to 0.75) in order to weight more the well played notes instead of the well played silences. This reduces precision of the note selection but makes the system more aggressive and increases the "density" of the piano roll.
- Decreasing the Teacher forcing with the Epochs during training. The system is capable of adapt better to the scenario of 0 teacher forcing (full predicting) if the teacher forcing gets diminished with time.
- Variable learning rate in order to escape the local minima of the full silence score. We believe that the system has a local minima in a full silence score due to the important amount of notes that are not playing at any given time. By giving an initially larger learning rate (10^2), the system seems to be able to escape that minima more easly at the beginning of training.

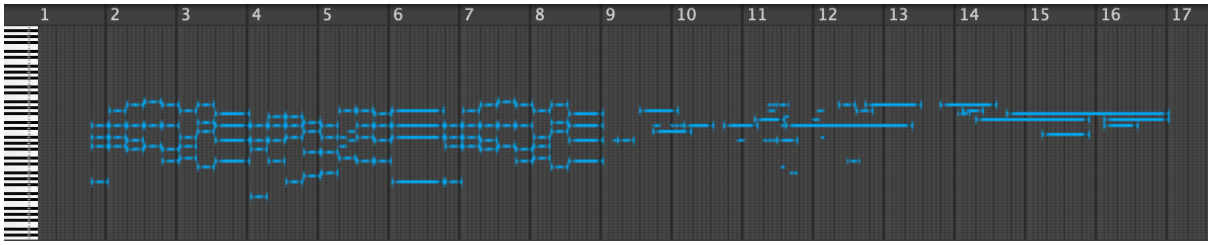
Audio examples of the Final output

Train (input sequence until second 14, prediction after second 14)



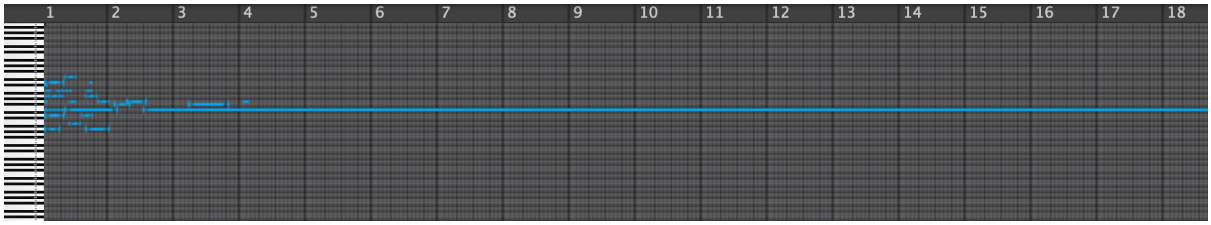
https://s3-us-west-2.amazonaws.com/secure.notion-static.com/b568ef5b-972d-4084-a65f-ea6a2ebea012/train_seq2seq.mp3

Test (input sequence until second 15, prediction after second 15)



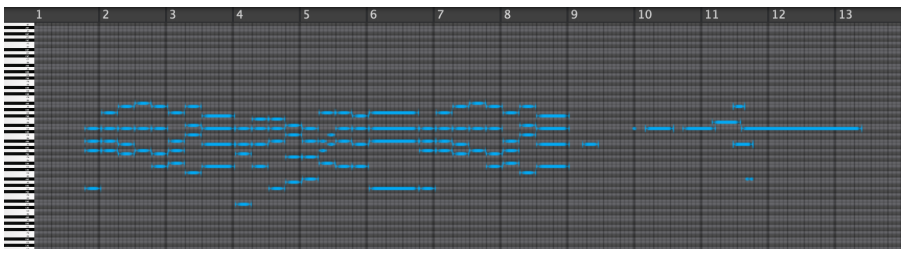
https://s3-us-west-2.amazonaws.com/secure.notion-static.com/38890420-8d9f-4a78-bfdb-706d871095f8/test_seq2seq.mp3

Predict #1 - Zeros



https://s3-us-west-2.amazonaws.com/secure.notion-static.com/eea05306-764f-4b70-aae5-0dce740936c5/predictseq2seq_zeros.mp3

Predict #2 - Rand



https://s3-us-west-2.amazonaws.com/secure.notion-static.com/3d16fdf3-b42c-4800-be0a-845b45457e70/predictseq2seq_rand.mp3

Further Work

What we could do to improve our latest model (4th Experiment):

- Stop using numerical methods to evaluate our Network's performance:
  We should start sending online Surveys of the Musical qualities of our output results. This way we could measure if it is very far or close from passing the Turing test, on which Datasets does our model perform better, etc.
- Train with more and better Datasets:
  The difficulty of this task depends heavily on the type and amount of training data. We should better explore how does our model perform with easier tasks (for example, training with hundreds of monophonic flute midi files). We should also train with bigger Datasets and different kinds of classical/modern composers.
- Improve Network Architecture and complexity of the Model, Training Hardware, Review the results of a higher number of epochs for training (for example 200, 500, etc.)

Other more ambitious paths to expand our work:

- Add Attention to our current Model
- Explore Transformer Models and develop one
- Explore the possibility of doing other tasks such as Musical Style transfer (Pokemon songs in the style of Bach, etc.) ...
- Explore the possibility of creating a model capable of generating music for several instruments at the same time.

Bibliography and Related Works

- Bachbot by F. Liang:
  https://ismir2017.smcnus.org/wp-content/uploads/2017/10/156_Paper.pdf
- ToneNet - A Musical Style TransferGo to the profile by Suraj Jayakumar:
  https://towardsdatascience.com/tonenet-a-musical-style-transfer-c0a18903c910
- How to Generate Music using a LSTM Neural Network in Keras by Sigurður Skúli:
  https://towardsdatascience.com/how-to-generate-music-using-a-lstm-neural-network-in-keras-68786834d4c5
- CocoNet - Counterpoint by Convolution by CZA Huang:
  https://ismir2017.smcnus.org/wp-content/uploads/2017/10/187_Paper.pdf
- Music Transformer: Generating Music with Long-Term Structure by CZA Huang: https://openreview.net/pdf?id=rJe4ShAcF7
- DeepBach - a Steerable Model for Bach Chorales Generation by Gaëtan Hadjeres, François Pachet, Frank Nielsen:
  https://arxiv.org/abs/1612.01010
- Clara: A Neural Net Music Generator by Christine Payne:
  christinemcleavey.com/clara-a-neural-net-music-generator/
- MuseNet, a deep neural network that can generate 4-minute musical compositions with 10 different instruments, and can combine styles from country to Mozart to the Beatles, by Christine Payne:
  https://openai.com/blog/musenet/