**Ben-Gurion University of the Negev**

**Faculty of Engineering Sciences**

**Department of Software and Information systems Engineering**

**Deep Learning**

**Assignment 3**

**The purpose of the assignment**

Enabling students to experiment with building a recurrent neural net and using it on a real-world dataset. In addition to practical knowledge in the “how to” of building the network, an additional goal is introducing the students to the challenge of integrating different sources of information into a single framework.

**Submission instructions:**

1. The assignment due date: 11.7.24
2. The assignment is to be carried out using PyTorch.
3. Submission in **pairs** only. Only **one copy** of the assignment needs to be uploaded to Moodle.
4. Plagiarism of any kind (e.g., GitHub) is forbidden.
5. Submit your work using a notebook.
6. The entire project will be submitted as a single zip file, containing both the report (in PDF form only) and the code. It is the students’ responsibility to make sure that the file is valid (i.e. can be opened correctly).

**Introduction**

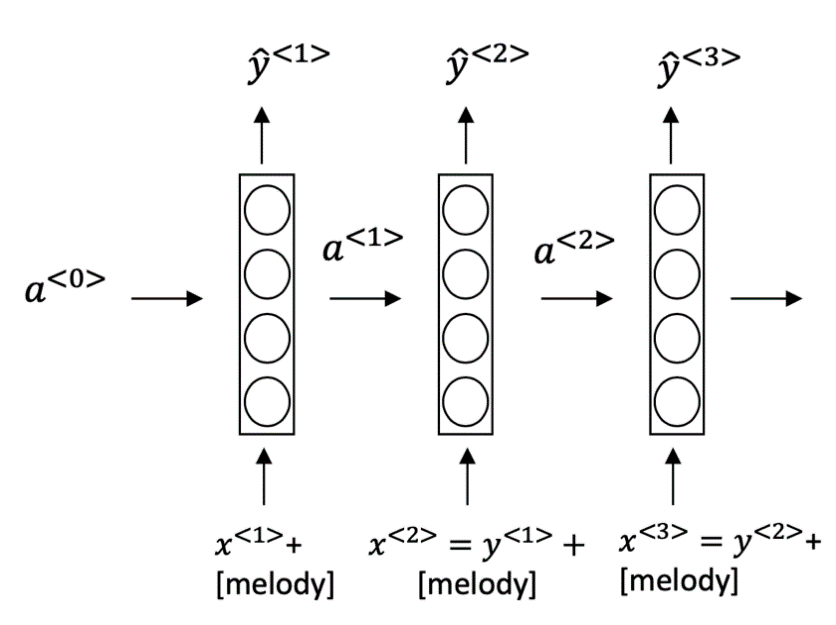
In class we covered the topic of automatic sentence completion/generation. In addition to the completion of “regular” sentences, this technique can also be applied to other domains such as lyrics and melodies generation (a nice example is [BachBot](https://soundcloud.com/bachbot)). In this task you will train a neural net to generate lyrics based on the provided melody.

During the training of the model you will have access both to the lyrics of a song and its melody. The melodies are stored in .mid (MIDI files) and contain various types of information – notes, the instruments used etc. You are encouraged to experiment with various methods to incorporate this information with the lyrics. During the test phase, you are required to automatically generate lyrics for a provided melody.

Please note that this assignment cannot be measured using objective (i.e., absolute) performance measures. Instead, we will be evaluating your approach to the solution, the implementation, and your analysis of your model’s performance.

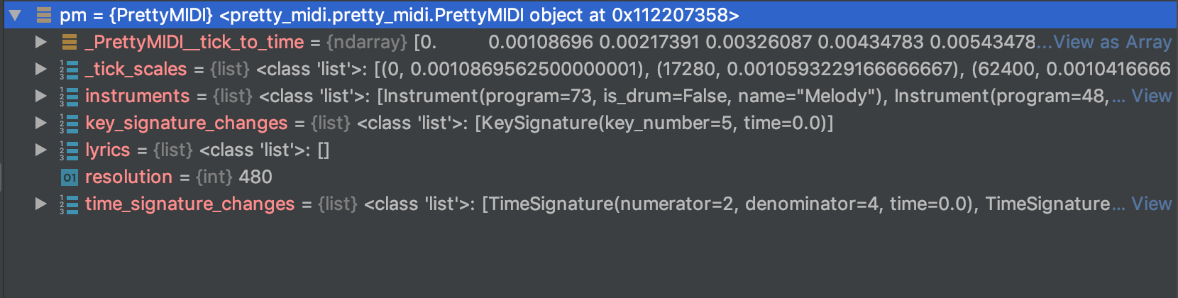
**Instructions**

1. Please download the following:
   1. A .zip file containing all the MIDI files of the participating songs
   2. the .csv file with all the lyrics of the participating songs (600 train and 5 test)
   3. [Pretty\_Midi](https://nbviewer.jupyter.org/github/craffel/pretty-midi/blob/master/Tutorial.ipynb), a python library for the analysis of MIDI files
2. Implement a recurrent neural net (LSTM or GRU) to carry out the task described in the introduction.
   1. During each step of the training phase, your architecture will receive as input one word of the lyrics. Words are to be represented using the Word2Vec representation that can be found online (300 entries per term, as learned in class).
   2. The task of the network is to predict the next word of the song’s lyrics. Please see Figure 1 for an illustration. You may use any loss function
   3. In addition to this textual information, you need to include information extracted from the MIDI file. The method for implementing this requirement is entirely up to your consideration. Figure 1 shows one of the more simplistic options – inserting the entire melody representation at each step.
      1. Using midi vectorization:
         1. Concatenate the midi vector
         2. Use deep learning layer to vector
         3. Use parralel deep learning model
   4. Note that your mechanism for selecting the next word should not be deterministic (i.e., always select the word with the highest probability) but rather be sampling-based. The likelihood of a term to be selected by the sampling should be proportional to its probability.
   5. You may add whatever additions you want to the architecture (e.g., regularization, attention, teacher forcing)
      1. I added droput, maybe add regularization
   6. You may create a validation set. The manner of splitting (and all related decisions) are up to you.
   7. You need to set “guidelines” (either fixed rules, preferably through the loss function) regarding the length of the generated text, the maximal number of words per line, etc. The goal is to make your lyrics look like those of an actual song.
3. The Pretty\_Midi package offers multiple options for analyzing .mid files. Figures 2-4 demonstrate the types of information that can be gathered.
4. You can add whatever other information you consider relevant to further improve the performance of your model.

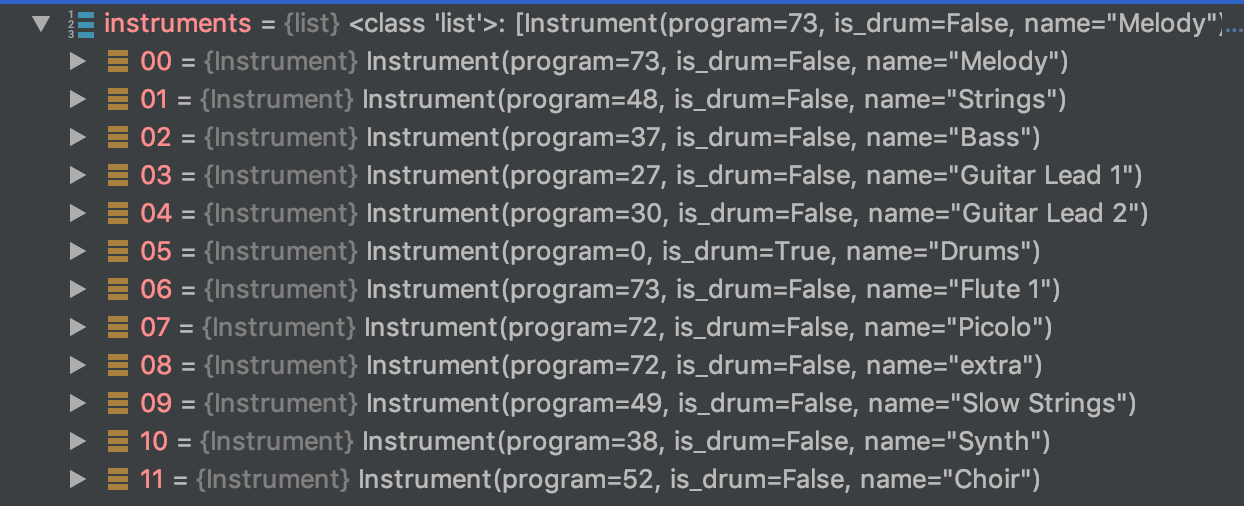


**Figure** **1:** an example of a simplistic way of combining the melody and lyrics

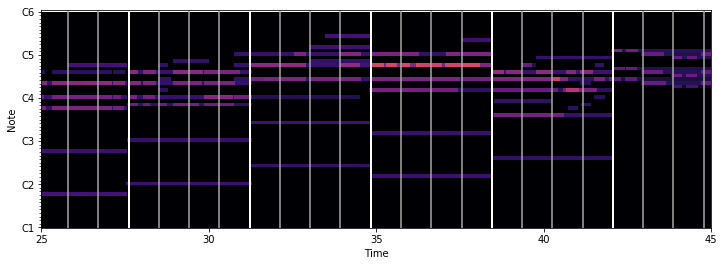
1. You are to evaluate two approaches for integrating the melody information into your model. The two approaches don’t have to be completely different (one can build upon the other, for example), but please refrain from making only miniature changes.



**Figure 2**: general .mid file information



**Figure 3**: Instrument information of the analyzed file



**Figure 4**: timing information of the analyzed file

1. Please include the following information in your report regarding the training phase:
   1. The chosen architecture of your model
   2. A clear description of your approach(s) for integrating the melody information together with the lyrics
   3. TensorBoard graphs showing the training and validation loss of your model.
2. Please include the following information in your report regarding the test phase:
   1. For each of the melodies in the test set, produce the outputs (lyrics) for each of the two architectural variants you developed. The input should be the melody and the initial word of the output lyrics. Include all generated lyrics in your submission.
   2. For each melody, repeat the process described above three times, with different words (the same words should be used for all melodies).
   3. Attempt to analyze the effect of the selection of the first word and/or melody on the generated lyrics.

Good Luck!

# Assignment 3 - lyrics generation using RNNs

## .Intorduction

* + - 1. Dataset analysis

Sinai Tbd add analysis of the dataset + lyrics

* + - 1. Melody analysis

Sinai Tbd add analysis of the melody (size,len etc)

## Methodology

### 2.1 Lyrics preprocessing

The preprocessing includes: 1) lowercase , 2) remove punctuation ,3) add end of sentence sign (‘eos’) 4) we keep ‘&’, it line break, so one of the prediction is line break.

### 2.2 text embedding

We used word2vec moudle to embeds the word to numeric vector, we used (As required) embedding entries per term. We train the moudle on the train dataset and get the contextual numeric representation.

### 2.3 input structure process

We preocess the lyrics in the next way: we use lstm mode, so the input should be a vector of size n, which in position is a differnet “time-line”, in the context of llm, the vector is a sentence in size n, and each position is diffrenent word. The output (target) is vector with the same size, while each position is the next word of the same position input vector.

We created ? train examples, each example contains x vector, wich contain ? words, and y vector, which contains “true predicted next word”

the code can be found in generate\_sequence function in tools.py

### 2.4 Melody procesing ( MIDI files)

We use 2 approches to handle the midi files;

#### 2.4.1 graph embedding

We used a graph embedding method, ealborating describe in Lisena paper [1] MIDI2vec uses graph embedding techniques to represent MIDI files, capturing tempo, time signature, programs, and notes. It optimizes node2vec for generating embeddings that predict musical genre and metadata. This method achieves high accuracy with Feed-Forward Neural Networks, enabling scalable and automated metadata tagging in symbolic music collections. We use the implemented github repositoy [2], with the next parameters:

For the computing edgelist:

1) -n =100 (Number of groups of simultaneous notes to be taken in account for each MIDI)

For the converting to vectors:

1. walk\_length = 5 (Length of walk per source)
2. –dimensions=50 (number of dimension)

The rest parameters are the deafult parameter ( added in the reference)

Finaly we got vector of 50 dimensions for each melody

#### 2.4.1 Sinai embedding

tbd

## 3. Model architecture

3.1 train vlaidation split:

We use 0.95 train and 0.05 validation. We expacting that the loss will decrease mainly in the train , and in the validation, It hards to beclieve that th loss reduces, becouase the complex task of predicting and generating lyrics. So in this case we think 5 percents will be more than enough.

3.2 model architecture

3.2.1 naïve cocncatenate word and midi vectors (Naïve)

We used lstm model with the next structure:

* + - 1. Concatenate word vector + midi vector
      2. Use LSTM configuration with ? hidden dimension, ? layers and dropout of 0.2
      3. Fully connected layer, with vocabulary size output

3.2.2 merge parralel layers of midid and word vector (Merge)

1. diffrencet single- layer for word vector and midi vector

2. concatenate those embedded layes

3. Use LSTM configuration with ? hidden dimension, ? layers and dropout of 0.2

4. Fully connected layer, with vocabulary size output

#chek bidirectional lstm

For each epoch we get output of the train and validation loss and printed lyrics (for make sense tracking)

### 3.3 generated lyrics rules

For the generated text we use the next pipeline ( combine determinissitc and stocashtice attributes):

Define the parameters: 1) max length – maximal lyrics length

Send the first word, and the chosen Integrated midi embedding

Send the first word to the LyricsGenerator model

Select the next word based on the model's output probabilities, meaning we do not simply choose the word with the highest probability. Instead, we make a probabilistic selection based on the entire distribution of probabilities provided by the model

We append the predicted word to the generated sentence

If the seq is bigger than 5, we choose the last 5 words, and repeat 2-5

The breaking point is one of the next conditions: 1) we get the maximum length 2) we predict ‘eos’

## 5 Model Optimization

We haave 2 model configurations (Naïve and merge) and 2 embedding methods ( midi by graph mode, and midi by sinai ,

We run each model with the next hyper parameter tuning:

{learning rate : [0.0001,0.001]:

Sequence length: [3,5],

Batch [ 36,128]

}

Overall 32 running ( 8 parmeter combination \* 4 model and embedding combinations), of courese if we woulf havemore time and computational power we would use bigger grid search, include dropout, epochs,number of layers, entities per word and midi file etc).

We used perpelexity metric to choose the best configuration (on the validation dataset). We extracted the 2 best model ( one oof each configuartion) to evalute the results

The experiment results found in X

## Result

Add plot of perpelixty vs epochs

Add plot of loss vs epochs

Use tensorboard

Add genrated text of each test model

## Reference

* + - 1. Lisena P., Meroño-Peñuela A. & Troncy R. MIDI2vec: Learning MIDI Embeddings for Reliable Prediction of Symbolic Music Metadata. In Semantic Web Journal, Special issue on Deep Learning for Knowledge Graphs, vol.13, no.3, pp. 357-377, IOSPress, 6 April 2022 <http://doi.org/10.3233/SW-210446>
      2. https://github.com/midi-ld/midi2vec?tab=readme-ov-file

**midi2vec deafult parameter:**

* -i, --input Input graph (edgelists) path. Default: .\edgelist;
* -o, --output Output file name. Default: embeddings.bin;
* --walk\_length Length of walk per source. Default: 10;
* --num\_walks Number of walks per source. Default: 40;
* -p Return hyper-parameter (as in node2vec). Default: 1;
* -q Inout hyper-parameter (as in node2vec). Default: 1;
* --dimensions Number of dimensions. Default: 100;
* --window-size Context size for optimization. Default: 5;
* --iter Number of epochs in word2vec. Default: 5;
* --workers Number of parallel workers. Default: 0 (full use);
* --exclude Edgelists to be excluded from the computation