GAN for music (lyrics) generation

陳冠元、 盧聖約

Dataset: [10]

- Piano-midi.de[1] : Source (124 files, 951 KB) or Piano-roll (7.1 MB)
- Nottingham[2] : Source (1037 files, 676.1 KB) or Piano-roll (23.2 MB)
- MuseData[3] : Source (783 files, 3.0 MB) or Piano-roll (30.1 MB)
- JSB Chorales : Source (382 files, 210 KB) or Piano-roll (2.0 MB)

Dataset information

Important: If you intend to run experiments and compare the accuracy of your model to our results, make sure to compute the expected frame-level accuracy correctly. For each time step, given the ground truth sequence history, compute the expectation over the output vector configurations (as defined by the conditional distribution of your probabilistic model) of the accuracy of that vector for that time step. The accuracy is computed as in Bay et al. (2009) and takes into account insertions, misses and replacement errors. Then report the average of that expectation across time steps. If the expectation is not tractable under your model as for the RNN-RBM/NADE, you can estimate it by sampling vector configurations and reporting the empirical mean of the accuracy. Increase the number of vector samples until the standard error of the averaged mean is satisfactory (usually 20-50 samples per time step for < 0.1% error). Note that the log-likelihood metric is much more meaningful than accuracy for polyphonic music generation and transcription, and I recommend basing your evaluation on it exclusively.

Below are the source files (MIDI) for the 4 datasets evaluated in the paper (split in *train*, *validation* and *test* sets). You can generate piano-rolls from the source files by transposing each sequence in C major or C minor and sampling frames every eighth note (quarter note for JSB chorales) following the beat information present in the MIDI file. Alternatively, pickled piano-rolls for use with the Python language are also provided.

Metric:

$$Bleu = BP \cdot \exp(\sum_{n=1}^{N} w_n \log p_n)$$

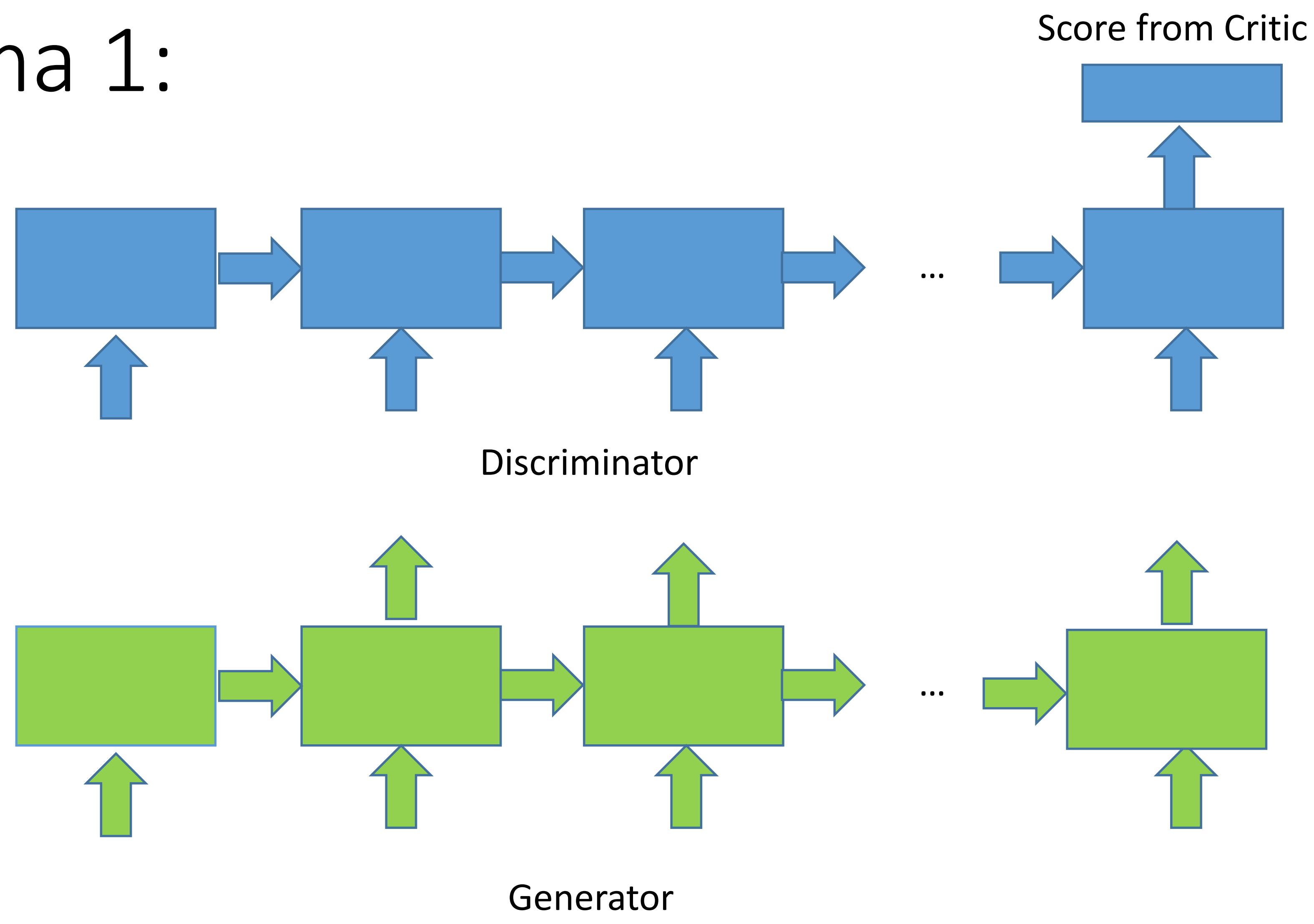
Discuss:

1. How to improve the performance of sequence-like work with adversarial training. [3, 4]

1. Analyze and compare the result with specific metrics.

2. The issues of hyper-parameters and concatenation method.

Shema 1:



$$J^{(D)} = -E_{x\sim p_{data}}[D(x)] + E_{z\sim p_z}[D(G(z))] \quad (1)$$

$$\theta_d = \theta_d - \eta RMSProp(\theta_d, g_{\theta_d}), \quad \theta_d \leftarrow clip(\theta_d, c, -c) \quad (2)$$

其中，θ_d 為 discriminator 的參數。另定義 a prior on input noise variables $p_z(z)$ (from a random normal distribution)，作為 generator function 初始的 input value。細部的 procedure 與 algorithm 之定義如下:

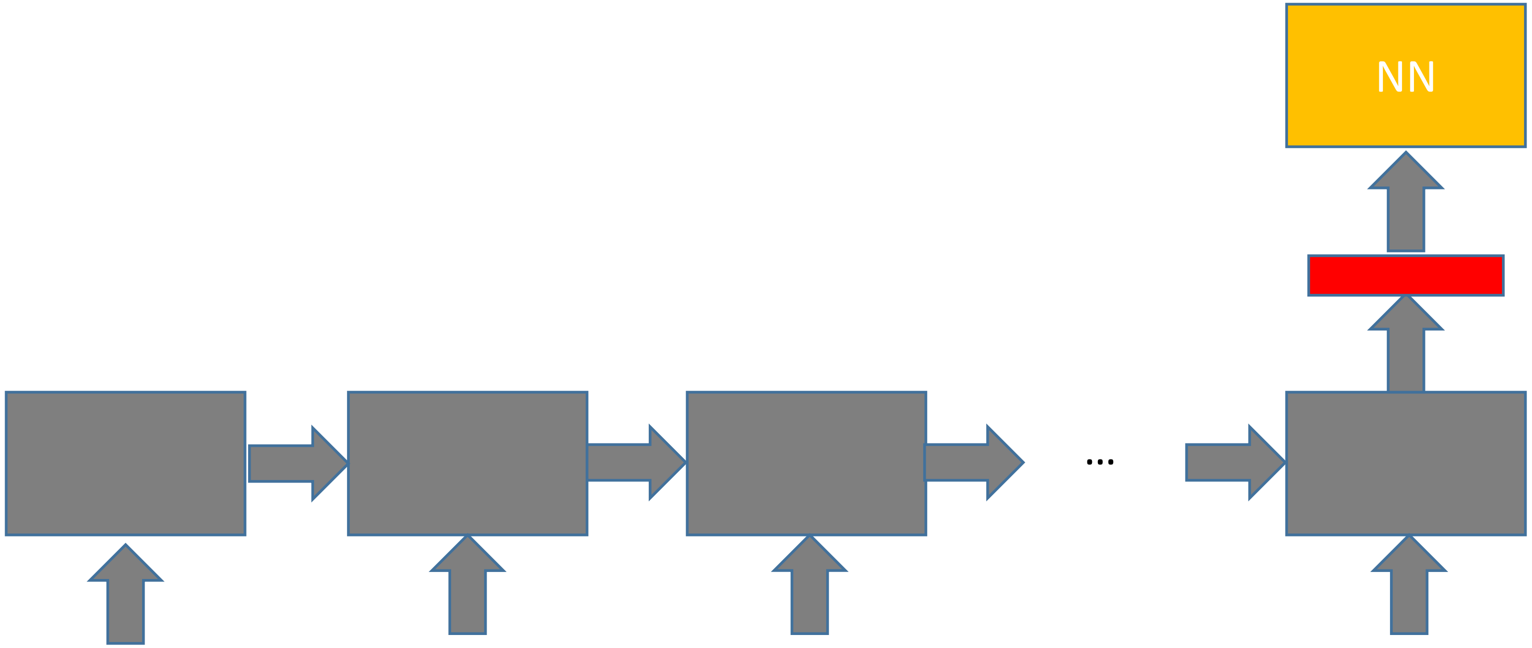
Algorithm Wasserstein GAN with RL

Require: generator G_{θ_g}; discriminator D_{θ_d}; a sequence dataset $S = \{X_{1:T}\}$; roll-out policy G_ϕ

Initialize G_{θ_g}, D_{θ_d} with random weights θ_g, θ_d;
$\phi \leftarrow \theta_g$
Generate negative samples using G_{θ_g} for training D_{θ_d}
repeat
    Pre-train G_{θ_g} using MLE on S
    for g-steps **do**
        Generate a sequence $Y_{1:T} = (y_1, \ldots, y_T) \sim G_{\theta_g}$
        for t in 1 : T **do**
            Compute $Q(a = y_t; s = Y_{1:t-1})$ by Eq. (4) in [2]
        end for
        Update generator parameters via policy gradient Eq. (8) in [2]
    end for
    for d-steps **do**
        Use current G_{θ_g} to generate negative examples and combine with given positive example S
        Train discriminator D_{θ_d} for 5 epochs by Eq. (1) and (2)
    end for
    $\phi \leftarrow \theta_g$
until D_{θ_d} converges

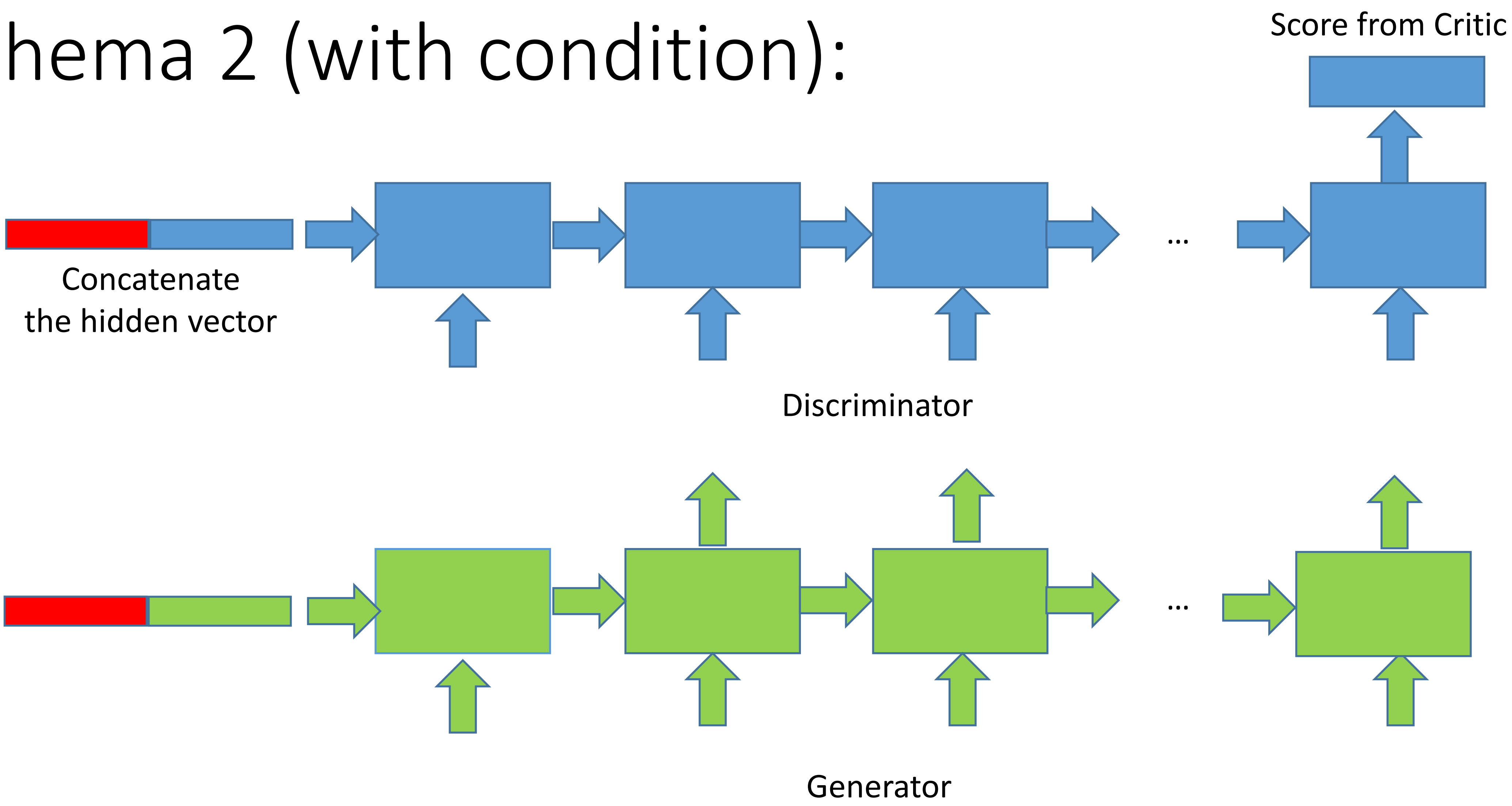
Shema 2 (with condition):

Result of classification (probability)
e.g. [Blues, Hip Hop, Jazz, …]



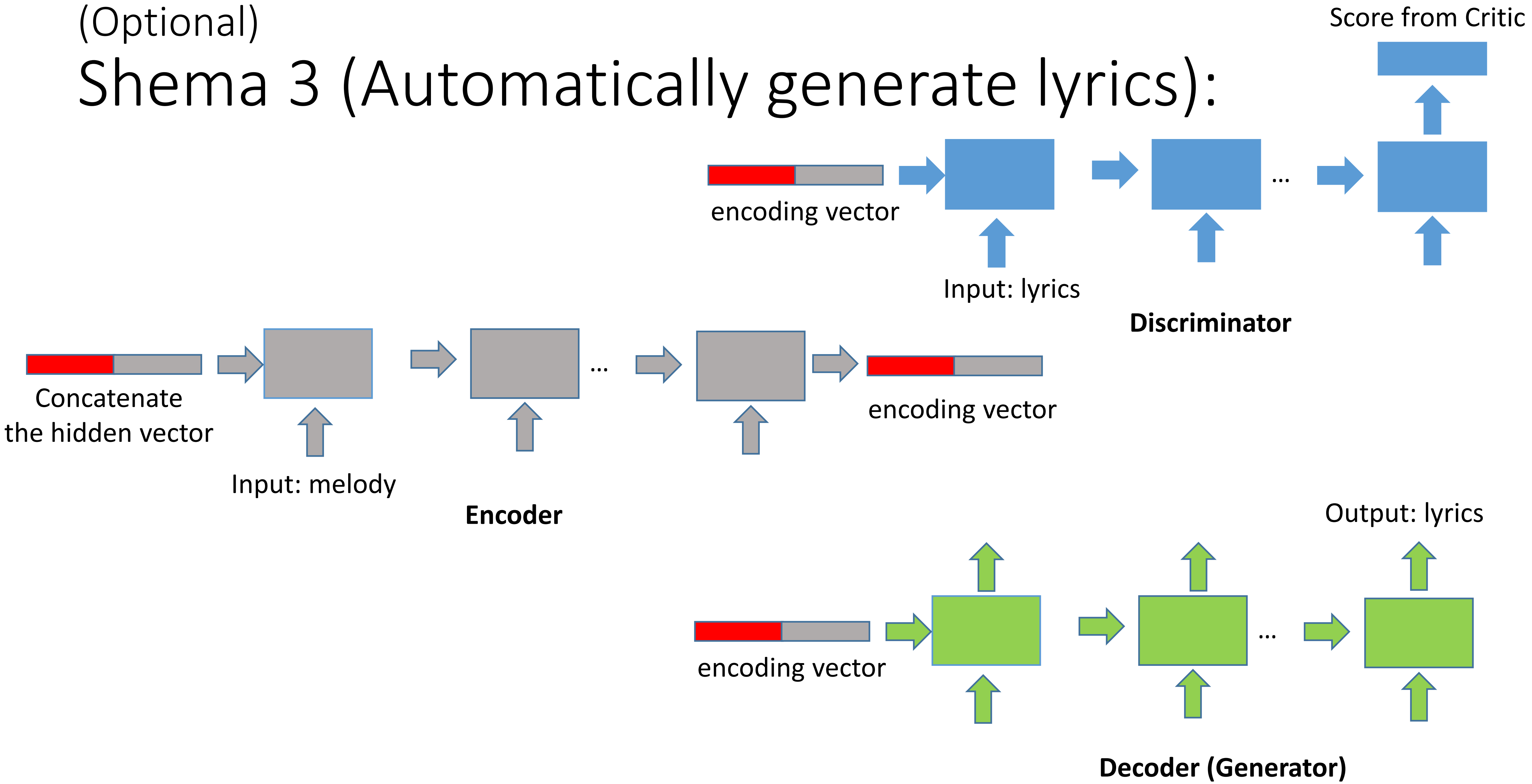
Classifier with the input (music)

Shema 2 (with condition):



(Optional)

Shema 3 (Automatically generate lyrics):



Experiments (Shema 1) (before 2017/12/12):

We referred to the [12, 13] to process the datasets [10] and used all midi files that in Piano-midi.de, Nottingham, MuseData, JSB Chorales source files, and ran the code "midi_to_statematrix.py" [13] to transform the midi files to the "statematrix".

And then we reshaped and extracted "statematrixs" to the shape(30, 156) for keeping all training data in the same length of time steps.

In this step, we found that the generator prone to generate nonsense tones. We changed the learning phase of discriminator and generator with 5:1 then become better.

Input and Output Details

My network is based on this architectural idea, but of course the actual implementation is a bit more complex. First, we have the input to the first time-axis layer at each time step: (the number in brackets is the number of elements in the input vector that correspond to each part)

- **Position** [1]: The MIDI note value of the current note. Used to get a vague idea of how high or low a given note is, to allow for differences (like the concept that lower notes are typically chords, upper notes are typically melody).
- **Pitchclass** [12]: Will be 1 at the position of the current note, starting at A for 0 and increasing by 1 per half-step, and 0 for all the others. Used to allow selection of more common chords (i.e. it's more common to have a C major chord than an E-flat major chord)
- **Previous Vicinity** [50]: Gives context for surrounding notes in the last timestep, one octave in each direction. The value at index 2(i+12) is 1 if the note at offset i from current note was played last timestep, and 0 if it was not. The value at 2(i+12) + 1 is 1 if that note was *articulated* last timestep, and 0 if it was not. (So if you play a note and hold it, first timestep has 1 in both, second has it only in first. If you repeat a note, second will have 1 both times.)
- **Previous Context** [12]: Value at index i will be the number of times any note x where (x-i-pitchclass) mod 12 was played last timestep. Thus if current note is C and there were 2 E's last timestep, the value at index 4 (since E is 4 half steps above C) would be 2.
- **Beat** [4]: Essentially a binary representation of position within the measure, assuming 4/4 time. With each row being one of the beat inputs, and each column being a time step, it basically just repeats the following pattern:

```
0101010101010101
0011001100110011
0000111100001111
0000000011111111
```

Demo: train from 1026 midi files. e.g.

Hyper parameters (fail):

Num of epoch == 100, Learning phase (D:G) == 1:1, Batch size: 64

Num of layer of rnn == 3, Dropout keep probability == 0.7,

hidden size of G: 200, hidden size of D: 200

Epoch 1 Epoch 50 Epoch 99 Epoch 100

Hyper parameters:

Num of epoch == 1000, Learning phase (D:G) == 5:1, , Batch size: 64

Num of layer of rnn == 3, Dropout keep probability == 0.7,

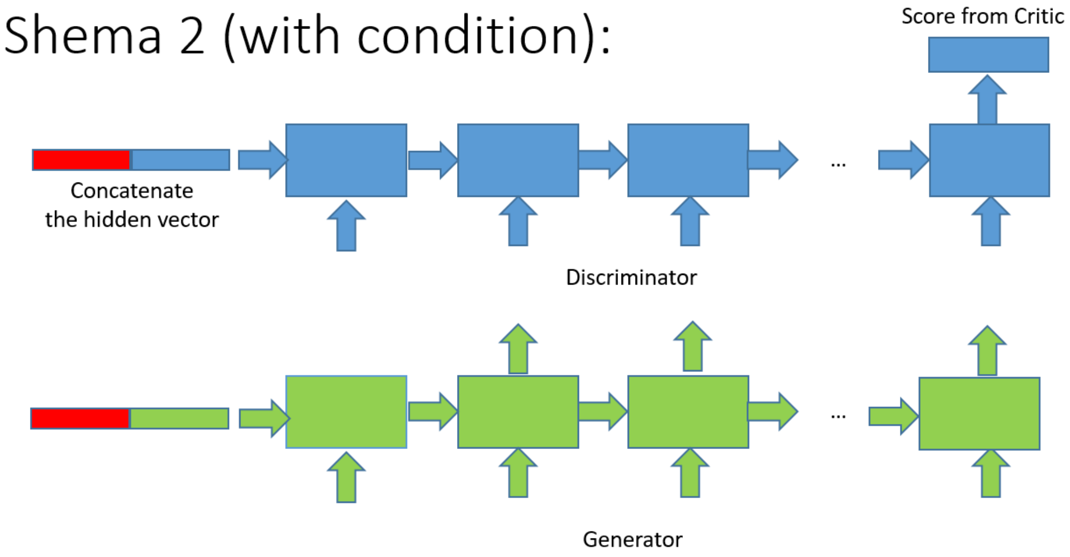
hidden size of G: 200, hidden size of D: 200

Epoch 1 Epoch 50 Epoch 99 Epoch 150

Keep training…

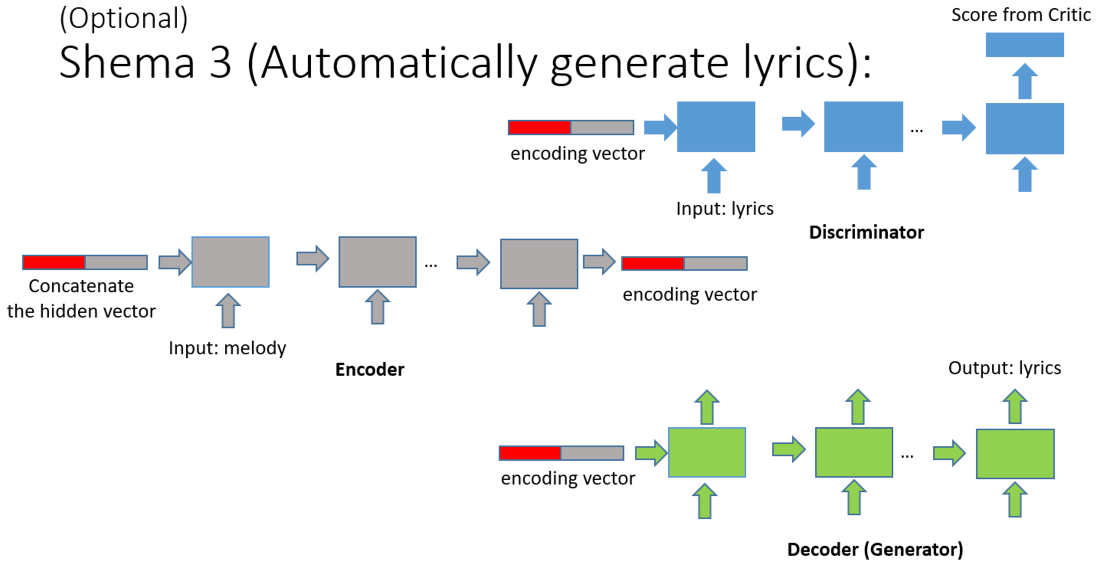
Next :

Shema 2 (with condition):



(Optional)

Shema 3 (Automatically generate lyrics):



THX

References:

[1] Lantao Yu, Weinan Zhang, Jun Wang, Yong Yu. SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient.

[2] Zhen Yang, Wei Chen, Feng Wang, Bo Xu. Improving Neural Machine Translation with Conditional Sequence Generative Adversarial Nets.

[3] Tong Che, Yanran Li, Ruixiang Zhang, R Devon Hjelm, Wenjie Li, Yangqiu Song, Yoshua Bengio. Maximum-Likelihood Augmented Discrete Generative Adversarial Networks.

[4] Anirudh Goyal, Nan Rosemary Ke, Alex Lamb, R Devon Hjelm, Chris Pal, Joelle Pineau, Yoshua Bengio. ACtuAL: Actor-Critic Under Adversarial Learning.

[5] Martin Arjovsky, Soumith Chintala, Léon Bottou. Wasserstein GAN.

[6] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio. Generative Adversarial Networks.

[7] https://github.com/LantaoYu/SeqGAN

[8] https://github.com/codekansas/seqgan-text-tensorflow

[9] https://github.com/ZiJianZhao/SeqGAN-PyTorch

[10] http://www-etud.iro.umontreal.ca/~boulanni/icml2012

[11] Nicolas Boulanger-Lewandowski, Yoshua Bengio and Pascal Vincent. Modeling Temporal Dependencies in High-Dimensional Sequences: Application to Polyphonic Music Generation and Transcription

[12] http://www.hexahedria.com/2015/08/03/composing-music-with-recurrent-neural-networks/

[13] https://github.com/hexahedria/biaxial-rnn-music-composition