Contents

Declaration of Academic Integrity **2**

1 Introduction **6**

2 Theoretical introduction to reinforcement learning **6**
- 2.1 Problem statement . . . 6
- 2.2 Resolve an MDP environment . . . 9
- 2.3 SARSA and Q-learning . . . 10
- 2.4 Deep Reinforcement Learning . . . 11
- 2.5 Policy-based RL . . . 12

3 Music Generation **14**
- 3.1 Melody Representation . . . 14
- 3.2 MIDI preprocessing . . . 16
- 3.3 Music models . . . 19
  - 3.3.1 Deep Learning Solutions . . . 19
  - 3.3.2 Reinforcement learning melody control techniques . . . 23

4 Extractive text summarization **25**
- 4.1 Introduction . . . 25
- 4.2 Sentence Encoder . . . 26
  - 4.2.1 Embedding and Convolution . . . 26
  - 4.2.2 Pooling . . . 26
  - 4.2.3 Dimension . . . 27
- 4.3 Document Encoder . . . 27
- 4.4 Sentence Extractor . . . 28
- 4.5 Reinforcement Learning . . . 29
  - 4.5.1 Difficulties and Discrepancies of the Problem . . . 29
  - 4.5.2 ROUGE Score . . . 29
  - 4.5.3 RL technique description . . . 30
- 4.6 Experiments . . . 31
  - 4.6.1 Selection of phrases . . . 31
  - 4.6.2 Experimental setup 1 and results . . . 32
  - 4.6.3 Experimental setup 2 and results . . . 34
  - 4.6.4 Experimental setup 3 and results . . . 34
- 4.7 Possible Applications . . . 35

5 Conclusion **36**

Appendices **39**

A Examples of MIDI preprocessing **39**

B Examples Deep LSTM **39**

C Examples VAE **39**

D Examples RL **39**

1 Introduction

In the last years the reinforcement learning approach has become extremely important as a deep learning extension for some problems as well as a standalone method for other problems. The range of it's application is very large. Invariant in all these cases is a **sequential decision making** principle - optimize next step of an agent in an environment w.r.t. to some goal function given an observation of the current state of the environment.

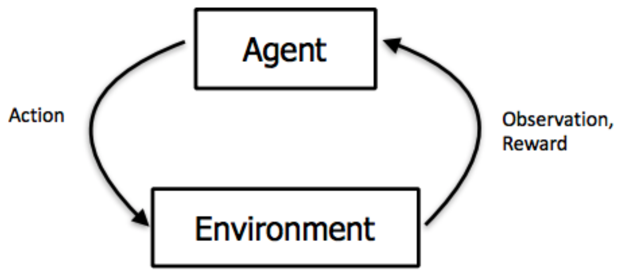
In this work two applications are mainly discussed: the **music generation problem** and **extractive text summarization**. The latter is important for the industrial purposes. We first give a brief introduction to the reinforcement learning method referring to [PS18] and [Sil15] .

2 Theoretical introduction to reinforcement learning

This introduction is based on materials from [Sil15] and [PS18].

2.1 Problem statement

Let there is an environment and an agent acting in this environment and taking decisions $a_t \in \mathcal{A}$ at every discrete time moment t ($t = 0, 1, 2 \ldots,$).



The state of the environment is described by state vector $s_t \in \mathcal{S}$. Let $\mathcal{A}$ and $\mathbb{S}$ be finite. We suppose that environment is a Markov Process:

$$\mathbb{P}(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \ldots) = \mathbb{P}(s_{t+1}|s_t, a_t).$$

At each time step the agent takes an action a_t, the state s_t changes to s_{t+1} depending on a_t(transition is not necessarily definitive) and the agent obtains the reward r_{t+1}. The agent aims to maximize the **total reward**. The tradeoff between short-term reward and long-term reward importance for the agent is described by parameter $\gamma \in [0,1]$ so that total reward is written as:

$$R = \sum_{t=0}^{\infty} \gamma^t r_{t+1}.$$

The action a_t is chosen by the agent according to the policy $\pi(\cdot|s_t, a_{t-1}, s_{t-1}, \dots)$. Policy is called **stationary** if it depends only on the current environment state: $\pi(\cdot|s_t)$. We will consider only stationary policies. Policy can be **deterministic** or **non-deterministic**. We define the return at time t as:

$$G_t := \sum_{k=0}^{\infty} r_{t+k+1}\gamma^k.$$

Given a stationary policy π we can define the **state-value** function:

$$v(s) := \mathbb{E}_\pi[G_t|S_t = s] \tag{1}$$

Since policy is stationary v(s) doesn't depend on t and the definition is correct. Formally agent optimizes his policy to maximize the function $v_\pi(s) \to \max_\pi$. The value function can be decomposed into two parts:

$$\begin{aligned} v_\pi(s) = \mathbb{E}_\pi[G_t|s_t = s] = \mathbb{E}_\pi[r_{t+1} + \gamma(r_{t+2} + \gamma r_{t+3} + \dots)|s_t = s] = \\ = \mathbb{E}_\pi\mathbb{E}_\pi\Big[[r_{t+1} + \gamma(r_{t+2} + \gamma r_{t+3} + \dots)|s_{t+1}]\Big|s_t = s\Big] = \mathbb{E}_\pi[r_{t+1} + \gamma v(s_{t+1})|s_t = s]. \end{aligned} \tag{2}$$

This is the **Bellman expectation equation**. If we know the **dynamics** $p(r_{t+1}, s_{t+1}|s_t, a_t)$ of the environment - the structure of our environment and the probabilities of transitions between the states, we can sometimes solve it and find the solution in the closed form depending on the policy π. But generally it's not the case: the *agent can learn about the environment and discover it only based on his experience.* Such an optimization problem is called the problem of **model-free control**. We define analogically the **action-state** function:

$$\begin{aligned} q_\pi(s,a) := \mathbb{E}_\pi[G_t|s_t = s, a_t = a] = \mathbb{E}_\pi\Big[\mathbb{E}_\pi[r_{t+1} + \gamma G_{t+1}|s_{t+1}]\Big|s_t = s, a_t = a] = \\ = \sum_{r,s'} p(r,s'|s,a)\Big[r + \gamma\mathbb{E}_\pi[G_{t+1}|s_{t+1} = s']\Big] = \sum_{r,s'} p(r,s'|s,a)\Big[r + \gamma v_\pi(s')\Big] \end{aligned} \tag{3}$$

Provided that

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{r,s'} p(r,s'|s,a)\Big[r + \gamma v_\pi(s')\Big], \tag{4}$$

we have:

$$v_\pi(s) = \sum_a \pi(a|s) q_\pi(s,a) \tag{5}$$

and

$$q_\pi(s,a) = \sum_{r,s'} p(r,s'|s,a)\Big[r + \gamma \sum_{a'} \pi(a|s') q_\pi(s',a')\Big] \tag{6}$$

We define the optimal state-value and action-value function as follows:

$$v_*(s) = \max_\pi v_\pi(s),$$

$$q_*(a, s) = \max_\pi q_\pi(a, s).$$

In order to ensure the correctness of this definition we now refer to the lectures by David Silver on Reinforcement Learning [Sil15] and formulate a theorem:

Theorem. *For any Markov Process(MDP):*

- *There exists an optimal policy π^* that is better than or equal to all other policies, $\pi^* \geq \pi, \forall \pi$.*
- *All optimal policies achieve the optimal value function, $v_{\pi^*}(s) = v_*(s)$.*
- *All optimal policies achieve the optimal action-value function, $q_{\pi^*}(a, s) = q_*(a, s)$.*
- *There is always a deterministic optimal policy for any MDP.*
- *An optimal policy can be found by maximizing over $q_*(a, s)$*

$$\pi^*(a|s) = \begin{cases} 1, & \textit{if } a = \arg\max_{a \in \mathcal{A}} q_*(a, s) \\ 0, & \textit{otherwise} \end{cases} \tag{7}$$

So if we know $q_*(s, a)$, we immediately have the optimal policy.

This theorem implies that the equation 4 for an optimal policy π^* can be rewritten as

$$v_*(s) = \max_a \sum_{r,s'} p(r, s'|s, a)[r + \gamma v_*(s')] = \max_a q_{\pi^*}(s, a) = \max_a q_*(s, a) \tag{8}$$

and the equation 6 in it's turn becomes

$$q_*(s, a) := q_{\pi^*}(s, a) = \sum_{r,s'} p(r, s'|s, a)\Big[r + \gamma \max_{a'} q_{\pi^*}(s', a')\Big] =$$

$$= \sum_{r,s'} p(r, s'|s, a)\Big[r + \gamma \max_{a'} q_*(s', a')\Big] \tag{9}$$

The latter is called **Bellman optimality equation**. We note that if we know $q_*(\cdot, \cdot)$ we can easily restore $\pi^*(\cdot|\cdot)$ by 7. But if we have only $v_*(\cdot)$ from 8 we need to know the dynamics $p(r, s'|s, a)$ too. This means that in practice the action-value function is more important than state-value.

In a model-free case it's impossible to solve the system of linear equations 9 exactly because we don't have it's dynamics. We will discuss possible approaches to this problem.

2.2 Resolve an MDP environment

In practice we do not really need to find an exact solution for 9. What we need is to obtain the optimal behaviour defined by $a_t := \pi^*(s) = \arg\max_{a'} q_*(s_t, a')$. In practice to achieve this goal we use iterative methods. For the **model-based setup** - the case when we know the dynamics p - there are so called **policy evaluation, policy iteration** and **value iteration methods** [PS18] [Sil15]. Actually within this work we will be in the case of the non-model-based setup, i.e. the dynamics p will be unknown. Below we will discuss the most well-known approach for this case.

The key idea is to approximate the expectation in 9. There are two main approaches to this problem:

- **Monte-Carlo** approach: sample trajectories containing (s, a), estimate $G(s, a)$ for each trajectory and average over them to get an estimate of $Q(s, a)$;
- **Temporal differences**: the idea is to exploit 9 and replace the expectation by a Monte-Carlo approximation over N samples $((s_{t+1})_i = s'_i, r_{t+1})_{i=1}^N$

$$\mathbb{E}_{r_{t+1}, s_{t+1}}[r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a')] \approx \frac{1}{N} \sum_i r_i + \gamma \max_{a'} Q(s'_i, a'),$$

or, which is better by running average:

$$Q(s, a) \leftarrow (1 - \alpha) Q(s, a) + \alpha(r_i + \gamma \max_{a'} Q(s_{t+1}, a')), \tag{10}$$

Which method is better to use in practice? Here is a table of comparison between two approaches:

Monte-Carlo	**Temporal Differences**
Needs full trajectory to learn	Learns from partial trajectory and infinite MDP
Less reliant on Markov property	Needs less experience to learn
Higher variance, no bias	Bias, but lower variance

The equation 10 presents indeed a one step of the action-value function evaluation. What about policy? How can we improve it? Let us define first the ϵ **- greedy policy** for an m-element action space $\mathcal{A}$:

$$\pi_{\epsilon-greedy}(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon, & \text{if } a = \arg\max_{a \in \mathcal{A}} Q_*(a, s) \\ \epsilon/m, & \text{otherwise} \end{cases}$$

We refer again to [Sil15] and formulate another theorem.

Theorem. *For any $\epsilon-$greedy policy π, the $\epsilon-$greedy policy π' with respect to Q_π is an improvement, $v_{\pi'}(s) \geq v_\pi(s)$*

Proof:

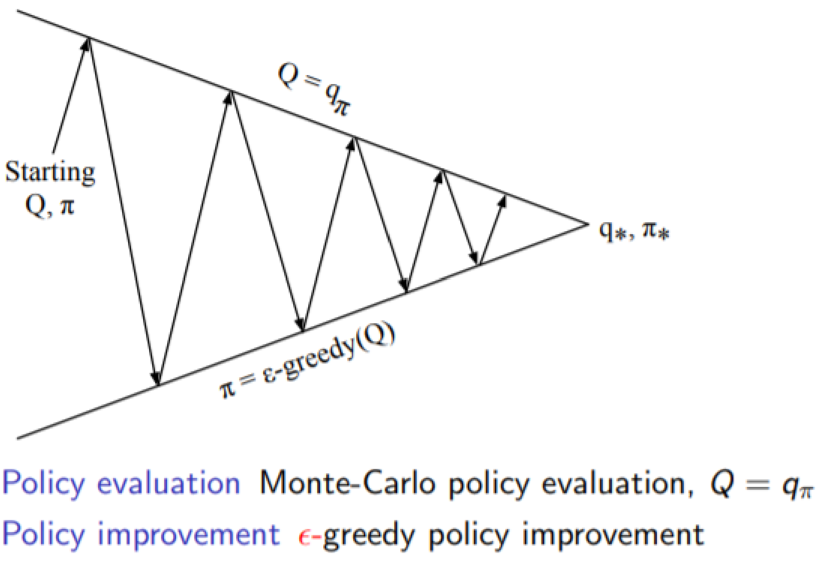
$$Q_\pi(s,\pi'(s)) = \sum_{a\in\mathcal{A}} \pi'(a|s)Q_\pi(s,a) = \frac{\epsilon}{m}\sum_{a\in\mathcal{A}} Q_\pi(s,a) + (1-\epsilon)max_{a\in\mathcal{A}}Q_\pi(s,a) \geq$$

$$\geq \frac{\epsilon}{m}\sum_{a\in\mathcal{A}} Q_\pi(s,a) + (1-\epsilon)\sum_{a\in\mathcal{A}} \frac{\pi(a|s)-\frac{\epsilon}{m}}{1-\epsilon}Q_\pi(s,a) =$$

$$= \sum_{a\in\mathcal{A}} \pi(a|s)Q_\pi(s,a) = v_\pi(s) \tag{11}$$

Both 10 and 11 let us formulate a general **principle**:

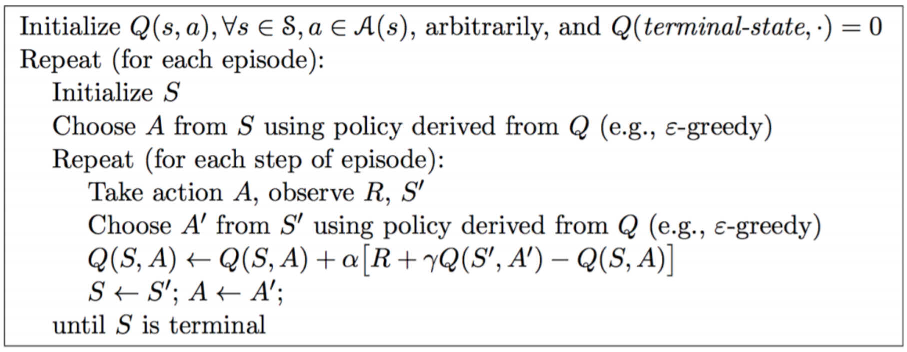
Since for a non-optimal policy this equation fails, we can find the optimal solution iterating over v and π as shown in the picture below([Sil15]).


Starting Q, π
Q = q_π
q*, π*
π = ε-greedy(Q)
Policy evaluation Monte-Carlo policy evaluation, Q = q_π
Policy improvement ε-greedy policy improvement

In the next section we will present algorithms SARSA and Q-learning concertizing this approach.

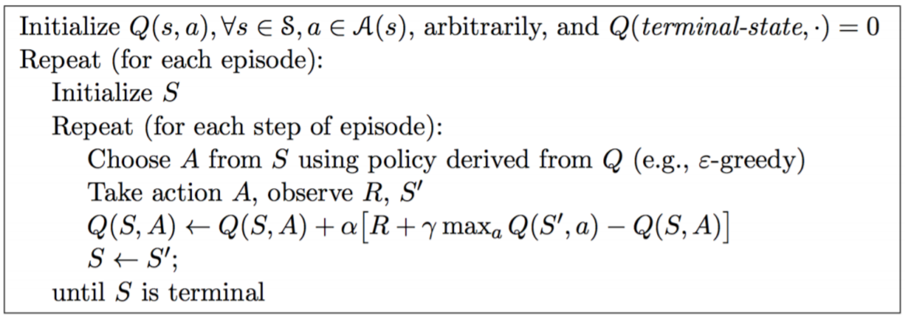
2.3 SARSA and Q-learning

SARSA stands for "state-action-reward-state-action". We recall that within the $\epsilon-$greedy policy the agent takes a random action with a small probability ϵ/m in order to discover the environment. We now present and compare the two algorithms. The difference between them is not that large but important. In SARSA both a and a' are sampled from the $\epsilon-$greedy policy, while in the Q-learning algorithm A is sampled from $\epsilon-$greedy policy but A' is sampled from the greedy policy. This subtle difference is extremely important. Generally SARSA outperforms Q-learning but the final choice depends also on the specificity of the problem.



```
Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize $S$
    Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    Repeat (for each step of episode):
        Take action $A$, observe $R$, $S'$
        Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
        $S \leftarrow S'$; $A \leftarrow A'$;
    until $S$ is terminal
```

SARSA



```
Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize $S$
    Repeat (for each step of episode):
        Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        Take action $A$, observe $R$, $S'$
        $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
        $S \leftarrow S'$;
    until $S$ is terminal
```

Q-learning

We note that the common term in both algorithms

$$R + \gamma Q(\cdot, A') - Q(S, A)$$

measures the discrepancy between the Bellman optimal equation and the state-of-art value and represents some properties of gradient(direction of evolution) in the gradient descent methods.

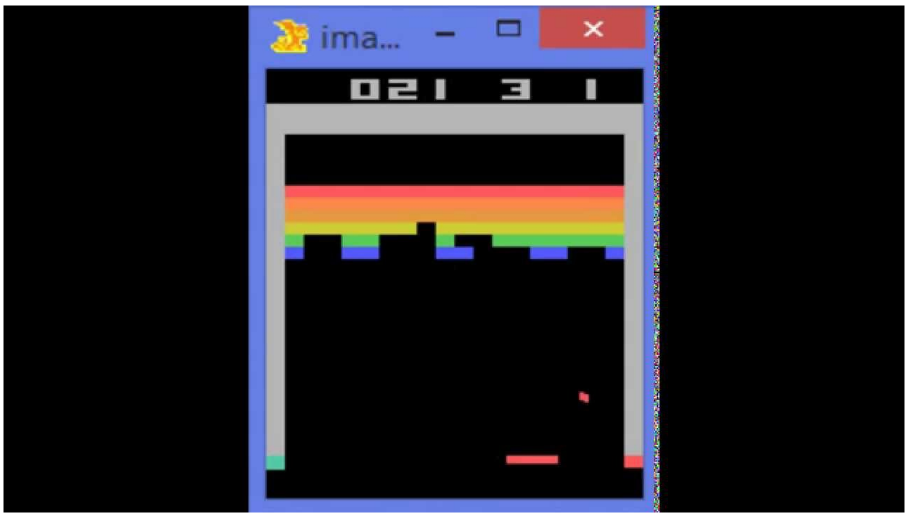
2.4 Deep Reinforcement Learning

Some problems and environments (e.g. video-games) have large state and action spaces and produce a lot of data with a large diversity. This makes sense to give a try to the Deep Learning methods. Usually the function Q is approximated by the output $\hat{Q}$ Neural Network(e.g. CNN for a video-game monitor, considered as the current state s).

For an input state s(e.g. the photo of the game monitor) the NN outputs the values $\hat{Q}(\cdot, s, \theta)$ for each possible action in the action space: $a \in \mathcal{A}$. θ stands for the parameters of the NN. The learning process is also iterative. Essentially the Q-learning algorithm is applied.

$$\begin{aligned} \mathcal{L}(\theta) &:= \mathbb{E}_\beta(r(a,s) + \gamma \max_{a'} \hat{Q}(a', s', \theta) - \hat{Q}(a, s, \theta))^2, \\ \Delta\theta &:= \alpha[(r(a,s) + \gamma \max_{a'} \hat{Q}(a', s', \theta) - \hat{Q}(a, s, \theta^-))]\nabla_\theta \hat{Q}(a, s, \theta), \end{aligned} \tag{12}$$

β is an exploration policy, e.g. $\epsilon -$ greedy policy,



θ^- is kept fixed during the gradient computation.

There are several advanced extensions of this method allowing to avoid some convergence issues [HGS15] [Mni+13].

2.5 Policy-based RL

There is another approach to reinforcement learning different with the described above. The key idea is to try to learn $\pi(a|s)$ directly using a parameterization

$$\pi(a|s) := \pi_\theta(a|s).$$

Now we need a way to learn the set of parameters θ. In the easiest case we have a **one-step** MDP: given the initial state s we need to decide which action to take to maximize the expectation of the reward $r(s,a)$:

$$\mathbb{E}_{\pi_\theta} r \to \max_\theta$$

What we need to do is to calculate the gradient of the expectation. Here we use so called log-derivative trick. Let S be continuous and p(s) be the density of state distribution.

$$\nabla_\theta \mathbb{E}_{\pi_\theta} r = \nabla_\theta \int_s p(s)ds \int_{\hat{a}} r(\hat{a},s)\pi_\theta(\hat{a}|s)d\hat{a} = \int_s p(s)ds \int_{\hat{a}} r(\hat{a},s)\nabla_\theta \pi_\theta(\hat{a}|s)d\hat{a} =$$
$$= \int_s p(s)ds \int_{\hat{a}} r(\hat{a},s)\frac{\nabla_\theta \pi_\theta(\hat{a}|s)}{p_\theta(\hat{a}|s)}\pi_\theta(\hat{a}|s)d\hat{a} =$$
$$= \int_s p(s)ds \int_{\hat{a}} r(\hat{a},s)\nabla_\theta \log \pi_\theta(\hat{a}|s)\pi_\theta(\hat{a}|s)d\hat{a} = \mathbb{E}_{\pi_\theta}[r\nabla_\theta \log \pi_\theta].$$

In practice the latter expectation is approximated by the actual value of the $r(s,a)\nabla_\theta \log \pi_\theta(a|s)$ during the Monte-Carlo simulations. And we use the gradient descent to optimize this function:

$$\theta := \theta - \alpha \nabla_\theta \mathbb{E}_{\pi_\theta} r,$$

$$\theta :\approx \theta - \alpha \cdot r(s,a)\nabla_\theta \log \pi_\theta(s,a)$$

This approach is called REINFORCE algorithm. It has also extensions for the case of **multi-step** MDPs. REINFORCE will play an important role in the second part of the work.

3 Music Generation

The music generation task is not a domain Société Générale is specially interested at. Despite that this part of work reflects the author's particular interest to the domain, it illustrates one of two most important ways of problem statement within the reinforcement learning paradigm and perfectly fits the goal of the global work.

The domain of Musical Generation is being largely investigated an developed. The nature of musical perception in humans is more difficult to formalize than for example the perception of images. Here the problem of effective musical representation arises. Authors attempt to solve globally two main problems:

1. find hidden musical structure and use it for generation;
2. create a better music models based on musical structure;

Here we list last advances:

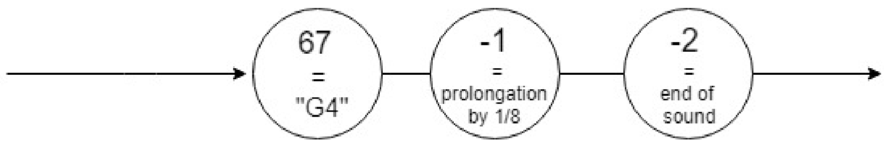
1. Musical structure
   - VAE embedding and GAN look very promising for the musical structure learning and classification [HUW17] [AL18] [TY17] [Rob+18]
   - Authors use the music theory knowledge to create a new image-based representation [CH18]
2. Musical models
   - Models based on RNN cells: LSTM or GRU [KY18]
   - Reinforcement learning techniques used to fine-tune model according to an appropriate metric [JGT16]

In this work we propose another approach to music representation and discover the behaviour of different algorithms on it.

3.1 Melody Representation

We consider music as a sequence of two types of intervals: **temporal intervals**(rhythm) and **frequency intervals**(pitches). Music is encoded as a sequence of note frequencies(from 0 to 127 corresponding to MIDI format convention) with some special events("-1" and "-2") allowing to prolongate the duration of the last note. For example in the picture below the sequence of three events encodes a one note "G4" with prolonged duration from $\frac{1}{8}$ to $\frac{1}{8} + \frac{1}{8} = \frac{1}{4}$.

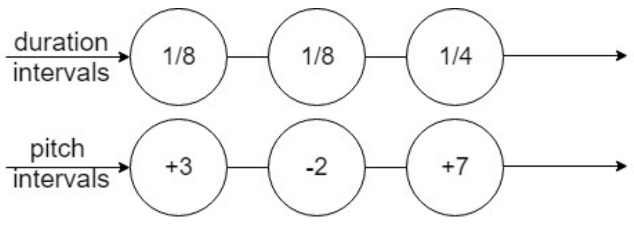
Although our experiments on the jazz music training set have shown that this kind representation is trainable and models can catch some **jazz patterns** with success and actually(as in [JGT16]) can achieve a good performance it suffers from several disadvantages:



Melody encoding

- This way to encode a melody returns too long results: to obtain a note with duration $\frac{1}{2}$ we have to repeat a ”-1” event 3 times; this needs more RAM resources to feed a sequence to a model and makes sequences longer and harder to learn the structure from;
- This representation doesn't allow generalizations: in musical theory it's well known that any sequence of notes may be transposed to another tonality. So if our model learns a particular sequence of notes in one tonality it's totally unable to reproduce it lower or higher;
- The way to encode pitch intervals together with rhythm intervals is too rigid: in reality a music model does not need a strict connection between these types of intervals: they may be treated separately allowing to our model a better freedom of creativity;
- The distribution of patterns is not uniform with respect to the frequency position: there are bass patterns which are played normally lower and solo guitar patterns - those are played usually higher. Because of that during the improvisation the number of appropriate patterns to continue the improvisation is limited by the absolute frequency of the last several notes of our melody;
- Generally a good music training set is not very easy to obtain and it evidently doesn't cover all the patterns for all the instruments and frequencies. From this point of view the usage of relative pitches(differences of pitches, e.g. $+3$ semitones or -7 semitones) could give us a richer training set.

The musical theory pays a lot of attention to the pitch intervals(”minor third”, ”major second”) as well as to the duration intervals - note values($\frac{1}{8}, \frac{1}{4}, \dots$) so it's more natural to consider them **both** as shown in the picture below.



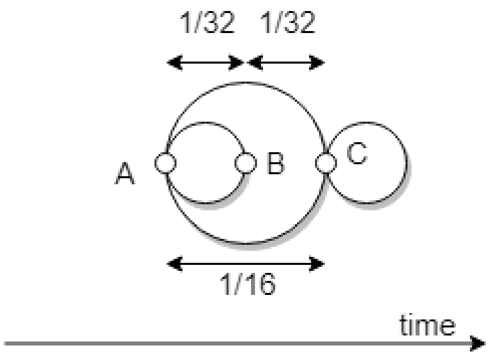
Melody representation

3.2 MIDI preprocessing

In this part we discuss the creation of a data set. In order to extract data we use **.midi** files. Files of this format contain the descriptions of all tracks in the song - one for each instrument in the song. We open .midi files using libraries **midiutils** and **python-midi**. Each note in midi format is characterized by it's **pitch**, **onset time**, **offset time**. The on- and offset times are expressed in **ticks** - an absolute time measure within a song. In this work we will not try to generate sequences of chords as in [CH18] so we need to extract from every instrument's track a monophonic melody. Two main problems arise here:

1. On the one hand we need to choose one note between simultaneously sounding notes while preserving the global melody perception on the other;
2. We need to define what "simultaneously sounding notes" means: in midi format all notes of a chord rarely begin on the same tick, very often the vary by several ticks. So we need to find a way to find all such chords - "clusters of notes" - and then choose a one note from each of them.

To our best knowledge there are not many works on this topic: a partial solution(to the problem 1) can be found in [OIA05a](**Skyline** algorithm) and [OIA05b] proposes some extensions to a multi-track task. Though both don't tackle the problem number 2, which is unavoidable if we want to get a mono-track with standardized rhythm intervals: $\frac{1}{16}$, $\frac{1}{8}$, $\frac{1}{4}$, $\frac{1}{2}$, and $\frac{1}{1}$. A quarter note($\frac{1}{4}$) expression in ticks can be extracted with tools provided by python-midi library for Python2. We propose a framework providing a solution to this problem. Our algorithm allows different extension providing different properties of the output. This approach(see Algorithm 1 below) is based on the DBSCAN clustering algorithm and Dijkstra algorithm for the shortest paths problem.



To the heuristic illustration

To understand this algorithm one may use the following heuristic. We will first make an assumption:

1. Let the maximal distance between two notes in the same cluster be $\frac{1}{32}$;

Algorithm 1: Framework: mono-track extraction

```
input  : A sequence of onsets s=$\langle s_1, s_2, \dots, s_n\rangle$, a sequence of pitches p =
         $\langle p_1 = p(s_1), p_2 = p(s_2), \dots, p_n = p(s_n)\rangle$, a function $d(\cdot,\cdot) : \mathbb{R} \times \mathbb{R} \to \mathbb{R}_+$,
output: sequence of temporal position of "chord"(in ticks, sequence of chosen pitches for
         each chord), sequence of chosen pitches for every chord
1  epsilon := $\frac{1}{32}$ note in number of ticks;
2  Apply DBSCAN(eps = epsilon, minSamples = 1) to s and attribute a cluster number
     $c(s_i) \geq 0$ to each element $s_i$;
3  directional weighted graph G := $\emptyset$ with weights $w(\cdot,\cdot)$;
4  K := number of clusters;
5  Calculate $m_0, m_2, \dots, m_{K-1}$ - minimas of onsets for each cluster in ticks;
6  j := 0;
7  while $j < K - 1$ do
8      for $\forall q : c(s_q) = j$ do
9          for $\forall r : c(s_r) = j + 1$ do
10             Add $(s_q, s_r)$ to G, $w(s_q, s_r) := d(p_q, p_r)$;
11         end
12     end
13 end
14 Create vertices $s_0$ and $s_{n+1}$;
15 for $\forall q : c(s_q) = 0$ do
16     Add $(s_0, s_q)$ to G, $w(s_0, s_q) := 0$;
17 end
18 for $\forall q : c(s_q) = K - 1$ do
19     Add $(s_q, s_{n+1})$ to G, $w(s_q, s_{n+1}) := 0$;
20 end
21 Apply Dijkstra algorithm to the graph (G, w), find an optimal path x =
     $\langle s_0, x_0, \dots, x_{K-1}, s_{n+1}\rangle$ from $s_0$ to $s_{n+1}$;
22 return pitches $p' = \langle p(x_0), \dots, p(x_{K-1})\rangle$, ticks $m = \langle m_0, \dots, m_{K-1}\rangle$
```

If there is a note A(see the picture above) which is the beginning point of the first cluster, let B be the end of this cluster(at $\frac{1}{32}-$ distance max from A). Then by definition of DBSCAN, the most-left point of the next cluster C will be at least on distance $\frac{1}{32}$ from C. In the worst case of single point in the first cluster($A = B$) the actual $\frac{1}{32}$(indeed it has to be slightly larger not to fall in the same cluster as A) distance between clusters will become $\frac{1}{16}$ after approximation. But given that the actual $\frac{1}{32}-$notes are extremely rare it's not a big problem. This argument is used as a **heuristic** to the framework.
We will list here some important technical details of a midi preprocessing:

- Each track of a melody is processed individually and separately from others;
- We extract the differences between two consequent beginnings of clusters($dc_i := m_{i+1} -$

m_i) and we find the most appropriate(closest) approximation of dc_i among $\frac{1}{16}, \frac{1}{8}, \frac{1}{4}, \frac{1}{2}$ and $\frac{1}{1}$);

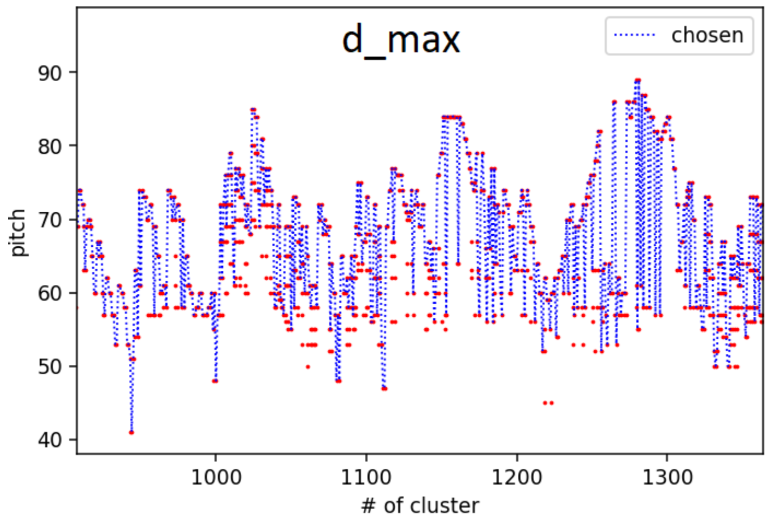
- We also extract the differences between every pair of **chosen(by algorithm)** consequent pitches ($dp_i := p_{i+1} - p_i$); we limit the pitches jumps by -2 and $+2$ octave to avoid extremely rare and particular cases: given that an octave interval contains 12 semitones we limit the jumps by $]-24, 24[$ semitones i.e. every pitch jump is encoded by a $23+23+1 = 47-$vector.

One of the most important parts of this framework is the choice of distance function $d(\cdot,\cdot)$. This choice can significantly affect the behavior of the algorithm and the character of the final result. In our experiments we've tried several types of function d, but the most important to mention are two following:

$$d_{rdist}(p_1, p_2) = \begin{cases} |p_2 - p_1|, & \text{if } 0 < |p_2 - p_1| < 12 \\ 12 + |p_2 - p_1| \ \% \ 12, & \text{if } |p_2 - p_1| > 12 \\ 10, & \text{if } |p_2 - p_1| = 0 \end{cases},$$

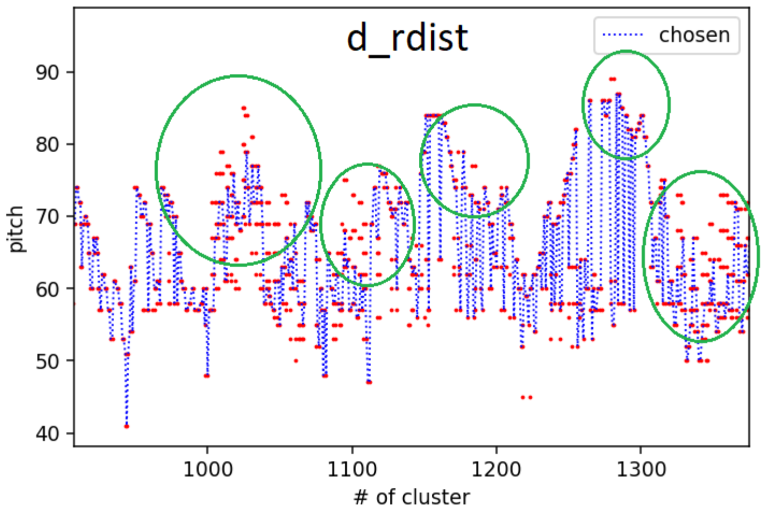
$$d_{max}(p_1, p_2) = 128 - p_2 > 0, \forall p_2$$

Evidently the d_{max} gives a **greedy** solution like Skyline algorithm(if we choose the highest note in each cluster, this choice will minimize the total cost of a path in the graph). And d_{rdist} controls the smoothness - minimizes the total sum of jumps. Below one can see the difference between the results corresponding to each of d_{max} and d_{rdist}.



Result of d_{max} application

Here are some advantages and disadvantages of these functions noticed during the experiments:



Result of d_{rdist} application

- The highest notes are more likely to contain the melody and d_{max} is better to catch it;
- d_{rdist} produces naturally a smoother sounding results;
- d_{rdist} presents a more "open-minded" approach: the bass notes also can be included in some cases if it lets minimize the total jumps sum;
- d_{rdist} penalizes the zero -jumps because it's unwanted behaviour for learning;

We noticed that probably because of the first point above, the d_{max} function is better to catch the jazz character of music. For our experiments we used d_{max}. The data was extracted from MIDI files scrapped from the web-site www.thejazzpage.de. Cf. the **Appendix A** for results of preprocessing.

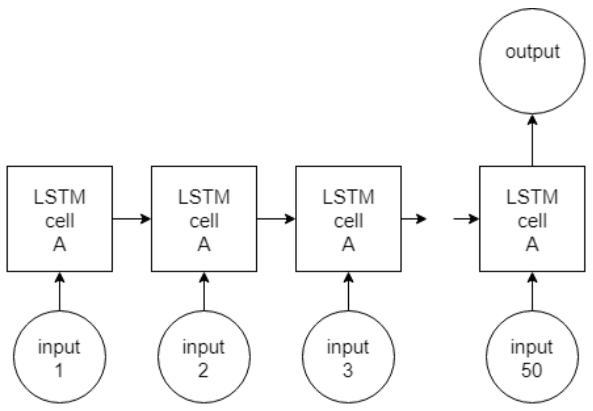
3.3 Music models

3.3.1 Deep Learning Solutions

Deep LSTM RNN First of all we will try to build an end-to-end Deep Learning approach. As in [JGT16] we will use a RNN model with architecture many-to-one(see below) and use a standard representation described in the first part of section **3.1**: events 0 and 1 correspond to "-2" and "-1" respectively, events $2 - 127$ correspond to the midi format standard pitches.

We generate a training set of 1.300.000 entries: X_{train} has the dimension $(1.300.000, 50, 128)$ and y_{train} - $(1.300.000, 1, 128)$: the model wil predict the 51th even py previous 50. The experiments with with a simple LSTM unit(one layer, 100 units in hidden state) showed up the following issues:

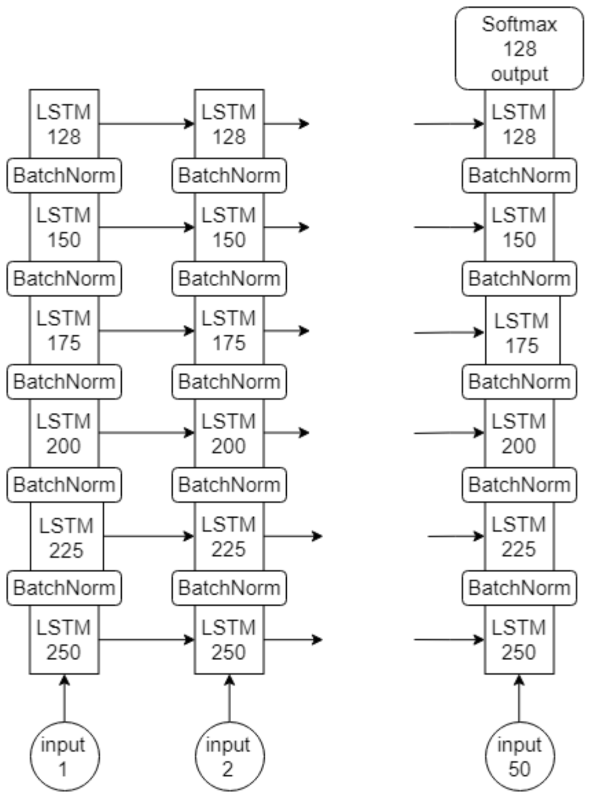
- The model suffers from repetitions of the same note(as in [JGT16]);



- There is a lack of structure in the melody;
- We cannot control the behaviour and characteristics of the composition;

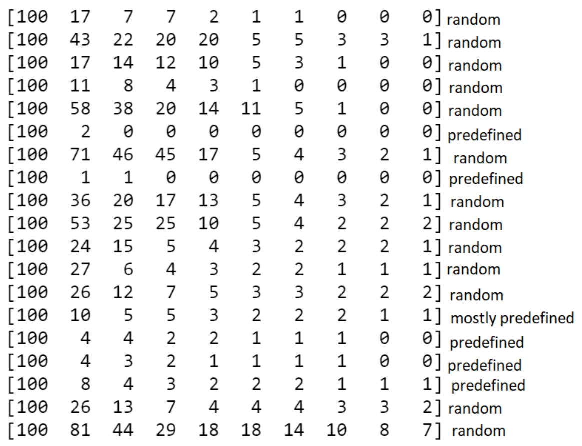
To **avoid multiple repetitions** of the previous note we assign the probability of the previous event(from 2 to 127 of course) with zero and renormalize the distribution.

After several experiments we came up with a following far more complicated architecture(see below) which after 60 epochs of training gives 0.72 of multiclass-crossentropy loss. Indeed we do not need so many training epochs: each number of epochs shows it's own melody character.



This architecture performs significantly better, the phrasing of improvisation is more or less conscious, there is a structure and characteristic jazz sounding.

Is the improvisation really randomized? We should verify if our network has not learned by heart all the dataset. In order to do that we express the probabilities of the most probable 10 notes in percentage of the maximum value among them(e.g. for vector of probabilities $p = [0.25, 0.25, 0.50]$ we will get $[100\%, 50\%, 50\%]$). The results below show that the generation is indeed random. Another way to check this is to generate different melodies with the same initial set of events: we need to initialize the first 50 events and then move our window step by step adding the notes. It turns out that from the same initial set of events we never obtain the same melody. This also proves the randomness of the generation.



```
[100  17   7   7   2   1   1   0   0   0] random
[100  43  22  20  20   5   5   3   3   1] random
[100  17  14  12  10   5   3   1   0   0] random
[100  11   8   4   3   1   0   0   0   0] random
[100  58  38  20  14  11   5   1   0   0] random
[100   2   0   0   0   0   0   0   0   0] predefined
[100  71  46  45  17   5   4   3   2   1] random
[100   1   1   0   0   0   0   0   0   0] predefined
[100  36  20  17  13   5   4   3   2   1] random
[100  53  25  25  10   5   4   2   2   2] random
[100  24  15   5   4   3   2   2   2   1] random
[100  27   6   4   3   2   2   1   1   1] random
[100  26  12   7   5   3   3   2   2   2] random
[100  10   5   5   3   2   2   2   1   1] mostly predefined
[100   4   4   2   2   1   1   1   0   0] predefined
[100   4   3   2   1   1   1   1   0   0] predefined
[100   8   4   3   2   2   2   1   1   1] predefined
[100  26  13   7   4   4   4   3   3   2] random
[100  81  44  29  18  18  14  10   8   7] random
```

Randomness of the generation Deep RNN

To avoid copying the experiments of [JGT16] and in order to investigate possible improvements we decided to separate pitch and rhythm intervals for further works as discussed in the section 3.1. Cf. **Appendix B** for examples.

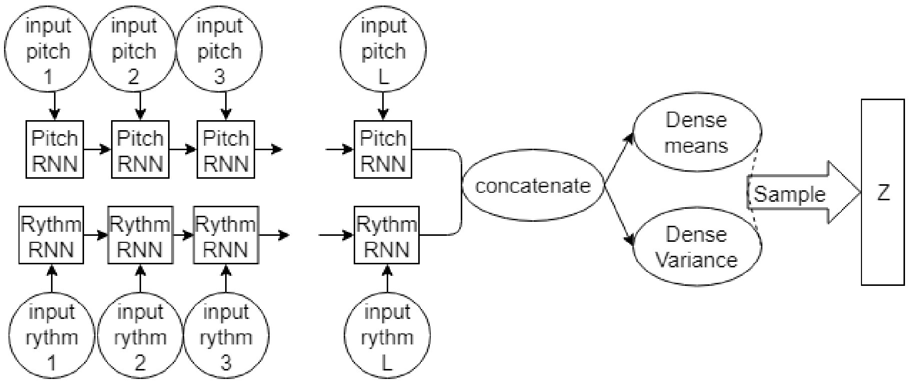
Variational Autoencoder The usage of VAE RNN is inspired by [Rob+18], [TY17] and [AL18], technical details of implementation are adapted from [Liu17] for our music representation. We had essentially two goals in this part of work:

- Create an architecture appropriate for the new way of music encoding(pitches and rhythms separetely);
- Make this system able to train and obtain some melodies;

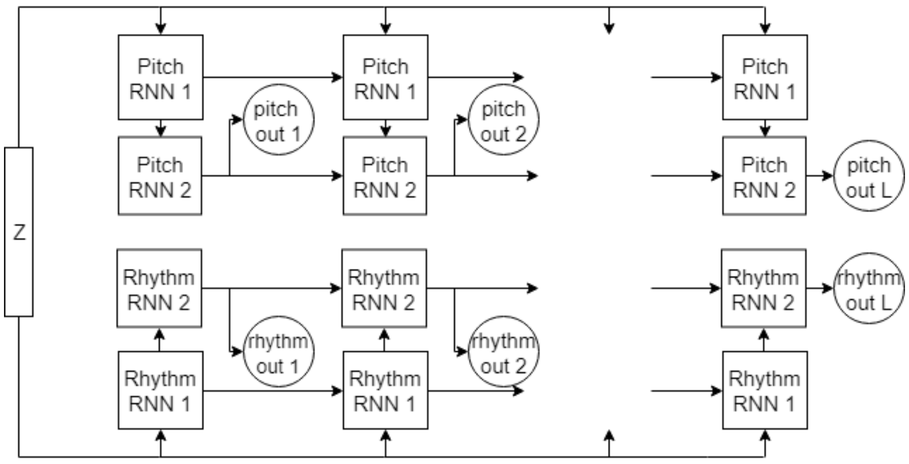
Variational autoencoder consists as usual of an **encoder** and a **decoder**. During training the categorical crossentropy loss is applied along with the Kullback–Leibler divergence(the loss is the sum of three parts: the KL-loss, cross-entropy for pitches and for rythms). Latent space has the dimension of 100 and we fit a standardized Gaussian distribution.

In different setups we vary the value of L: $L = 10, 15, 20$. For smaller values of L the training is faster and the behaviour of generation is more smooth and predictable. To generate music we do as follows:

1. We generate a multivariate Gaussian $\mathcal{N}(\vec{0}, \mathbb{I}_{100})$ vector Z, $i := 1$



Encoder Architecture



Decoder Architecture

2. We extract L pairs of rhythms and pitches:

$$M_i := \{(r_1, p_1), \ldots, (r_L, p_L)\}_i$$

3. For some value of *alpha* $\in [0, 1]$ we put

$$\tilde{Z} = (1 - \alpha) \cdot Z + \alpha \cdot \mathcal{N}(\vec{0}, \mathbb{I}_{100})$$

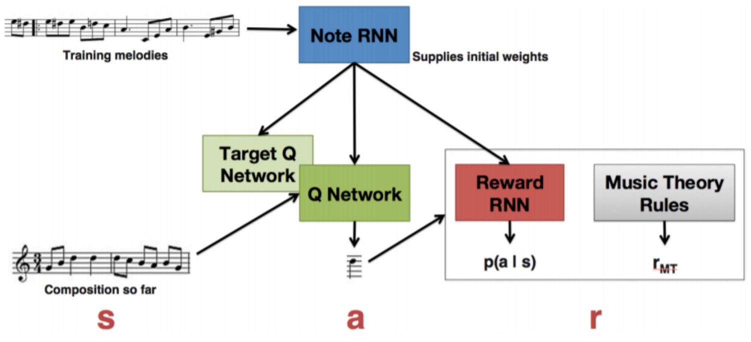
4. Put $Z := \tilde{Z}$ and go to the point $1(i := i + 1)$ to continue or stop and output $< M_1, M_2, \ldots, M_k >$;

This approach allows us to control *alpha*. On the one hand for a small value of alpha $\tilde{Z}$ and Z in the point 3 will not be very different and it's reasonable to expect that the two parts will be correlated. On the other hand, if α is closer to 1 that the characters of two consecutive pieces will be different.

Style mixture
This architecture allows a style mixture: we can learn the Autoencoder twice: for jazz music and for classical music and next to take a pitch cell from a Jazz dataset and Rhythm cell from a Classical dataset. Given that the latent space is expected to follow $\mathcal{N}(\vec{0}, \mathbb{I}_{100})$ in both cases we may proceed without any problem and get a mixture of two styles.
Analysis of results
This approach gives brighter and more nontrivial melodies. Moreover we can control the melody itself by controlling alpha. It's not excluded that for a lower dimension of Z it's possible to discover the geometry of the latent space and generate Z with a specific location for the desired melody behaviour. Sometimes the melody is not smooth enough and suffers from rythmic issues. For results cf. **Appendix C** .

3.3.2 Reinforcement learning melody control techniques

In this part we aim to control some properties of our generation. We follow the experience of [JGT16]. We will give a brief introduction to their approach. Here is the schema(from the original article [JGT16]):



DQN schema

Generally speaking we refer to the formula 12 for the Deep Q-Learning. Here we explain the setup of the experiment.

- First we create two models: pitch model and rhythm model. Each of the will be reinforced independently of the other. In both cases we call this model **Note RNN**(we use a simple LSTM unit with 100 hidden units and a softmax output);
- The Note RNN is used in three different ways:
  1. The weights of Note RNN are used to initialize the weights of **Target Q-Network**(except the ReLu Dense layer output);
  2. The weights of Note RNN are used to initialize the weights of **Q-Network**(except the ReLu Dense layer output);
  3. Note RNN is used to calculate a part of reward based on the initial model(reward is the sum of this one and so-called **music theory based** reward);

- The weights of Q-Network are continuously updated; they are called θ in 12;
- The weights of Target Q-Network follow the weights of Q-Network but much slower: they are updated within a moving average with parameter $\eta = 0.05$; these weights are called θ^- in 12;
- The $\epsilon-$ greedy parameter ϵ is continuously updated with exponential decay(multiplier equals to $0.998 - 0.9999$) depending on the number of epochs;

Design of reward

One of the most important principles in the design of reward is *"to give reward for WHAT is done but not HOW it's done"*. The agent should learn itself how to maximize it's utility. Here we list the musical generation properties that we have succeeded to control using different reward architectures:

Pitches

- Avoid "too long transitions"(more than one octave) in the generation;
- Give a priority to descending or ascending intervals(so that we can "draw music" connecting a descending and ascending parts);
- Avoid repetitions: penalize a "zero transition" and high autocorrelation with parameters 1 and 2;

Rythms

- Avoid exhaustive repetitions of the same pattern(with the help of autocorrelation with lag 1 and 2);
- Avoid the long intervals: $\frac{1}{1}$ for example;

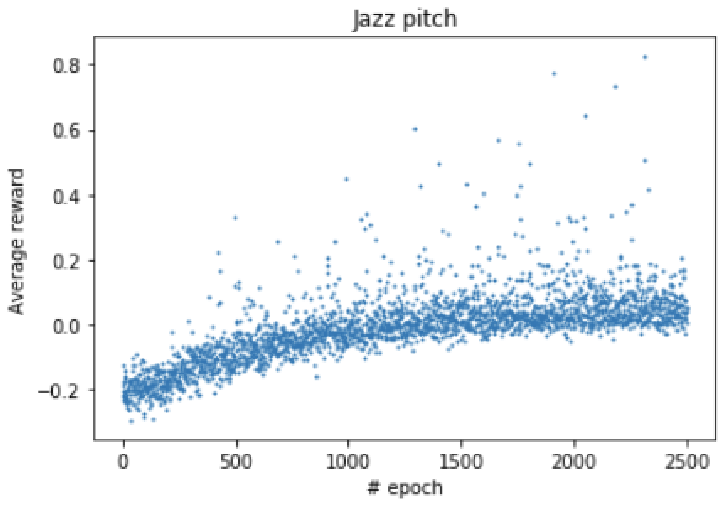
Example of pitch reinforcement reward architecture

Let Note RNN returns $p(a|s)$, where a stands for the action to take.

$$r(s,a) := \begin{cases} p(a|s) - 0.8, & \text{if } a = 0 \\ p(a|s), & \text{if } 0 < |a| < 12 \\ p(a|s) - 0.4 & |a| > 11 \end{cases}$$

Here is a typical Q-learning plot for the average reward per training epoch: it increases and stabilizes.

Cf. the **Appendix D** for some examples.



Reward evolution

4 Extractive text summarization

4.1 Introduction

This part of our work was based on the article [NCL18c] and the repository with the source code, data and other materials [NCL18a] dedicated to the extractive text summarization task. The subject of text summarization is important for industrial applications and effective treatment of huge volumes of text. More precisely, Société Générale is interested to develop a framework allowing to accelerate the treatment of texts and to extract the essential information from them. Here are some important aspects of this interest and use cases:

- Effectively analyze busyness-oriented editions(e.g. Bloomberg) and magazines and follow the main global trends in different domains;
- Shrink large domain-oriented busyness reports to a smaller more generic documents faster to read;
- Highlight the most important information in text.

The last version of the article [NCL18c] was published on the 16th of April 2018. Authors claim that their model outperforms the SoTA for both automatic and human evaluation.

Here we explicit what we've made on this project:

- Transfer the original code from an old version of Tensorflow (0.1) to a newer one($\geq$ 1.7) Tensorflow;
- Understand the model, the project and launch learning with appropriate parameters;

References

[OIA05a] G. Ozcan, C. Isikhan, and A. Alpkocak. "Melody extraction on MIDI music files". In: (2005). DOI: https://ieeexplore.ieee.org/document/1565863/.

[OIA05b] Giyasettin OZCAN, Cihan ISIKHAN, and Adil ALPKOCAK. "Melody Extraction on MIDI Music Files". In: (2005).

[Mni+13] Volodymyr Mnih et al. "Playing atari with deep reinforcement learning". In: (2013).

[HGS15] Hado Van Hasselt, Arthur Guez, and David Silver. "Deep reinforcement learning with double q-learning". In: (2015).

[Sil15] David Silver. "UCL Course on RL". In: (2015). DOI: http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html.

[JGT16] Natasha Jaques, Shixiang Gu, and Richard E. Turner. "Generating Music by Fine-Tuning Recurrent Neural Networks with Reinforcement Learning". In: (2016). DOI: https://static.googleusercontent.com/media/research.google.com/ru//pubs/archive/45871.pdf.

[HUW17] Jay A. Hennig, Akash Umakantha, and Ryan C. Williamson. "A Classifying Variational Autoencoder with Application to Polyphonic Music Generation". In: (2017). DOI: https://arxiv.org/pdf/1711.07050.pdf.

[Liu17] Jerry Liu. "Keras implementation of LSTM Variational Autoencoder". In: (2017). DOI: https://github.com/twairball/keras_lstm_vae/blob/master/lstm_vae/vae.py.

[TY17] Alexey Tikhonov and Ivan P. Yamshchikov. "Music Generation with Variational Recurrent Autoencoder Supported by History". In: (2017). DOI: https://arxiv.org/pdf/1705.05458.pdf.

[AL18] JMohammad Akbari and Jie Liang. "SEMI-RECURRENT CNN-BASED VAE-GAN FOR SEQUENTIAL DATA GENERATION". In: (2018). DOI: https://arxiv.org/pdf/1806.00509.pdf.

[CH18] Ching-Hua Chuan and Dorien Herremans. "Modeling Temporal Tonal Relations in Polyphonic Music Through Deep Networks with a Novel Image-Based Representation". In: (2018). DOI: http://dorienherremans.com/sites/default/files/preprint_lstm.pdf.

[KY18] Nikhil Kotecha and Paul Young. "Generating Music using an LSTM Network". In: (2018). DOI: https://arxiv.org/ftp/arxiv/papers/1804/1804.07300.pdf.

[NCL18a] Shashi Narayan, Shay B. Cohen, and Mirella Lapata. In: (2018). DOI: https://github.com/EdinburghNLP/Refresh.

[NCL18b] Shashi Narayan, Shay B. Cohen, and Mirella Lapata. In: (2018). DOI: https://nurture.ai/papers/ranking-sentences-for-extractive-summarization-with-reinforcement-learning/tldr.

[NCL18c] Shashi Narayan, Shay B. Cohen, and Mirella Lapata. “Ranking Sentences for Extractive Summarization with Reinforcement Learning”. In: (2018). DOI: https://arxiv.org/pdf/1802.08636.pdf.

[PS18] Alexander Panin and Pavel Svechnikov. “Practical Reinforcement Learning”. In: (2018). DOI: https://www.coursera.org/learn/practical-rl.

[Rob+18] Adam Roberts et al. “MusicVAE: Creating a palette for musical scores with machine learning”. In: (2018). DOI: https://magenta.tensorflow.org/music-vae.

Appendices

Here are some examples for different methods on music generation:
https://drive.google.com/file/d/1_-GNmNODdh2uuczWslBVEd03k_4e8tc1/view?usp=sharing

A Examples of MIDI preprocessing

We introduce the results of preprocessing of two songs: "Wave" by Antônio Carlos Jobim and "The second time around" by Jimmy Van Heusen. There are always **original** track and different sub-tracks each for one instrument. In both cases the most important is one with number 0 - it contains the main melody.

B Examples Deep LSTM

There are two Series in the folder. The second was made later and is more clear and smooth. We remind that in the case of Deep LSTM architecture we first initialize our generate with a short piece of melody(the end of this piece is indicated by a long high tick at about 4s from the beginning of a melody). Two files with the only difference in the name in the last number have the same initialization.

C Examples VAE

- **Series1:** these melodies are generated for different values of α, so some of them are looped(small α). These melodies look like more or less "conceptual music", they are recorded in a high tempo;
- **Series2:** these melodies are smoother and more moderate;
- **Mixture:** here we experiment taking for example a pitch from a jazz model and rhythm - from classic and vice verse;

D Examples RL

There are also two series of experiments. For the first we train on consecutive parts of melodies, and for the second - on the randomly placed parts of melodies.

- **Series1:** this one vary less: it's probably learned some specific gamma of notes;

- **Series2:** this part is richer in transitions

In general, both are very **smooth** compared to the previous methods. Probably there is a missing global melody - it's indeed a task of a global AI - for the future!

Mixture of styles

Here we experiment with taking pitch and rhythm from different sources: jazz or classic.