Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 159 (2019) 117–124

Procedia Computer Science

www.elsevier.com/locate/procedia

23rd International Conference on Knowledge-Based and Intelligent Information & Engineering Systems

Bach 2.0 - generating classical music using recurrent neural networks

Alexandru-Ion Marinescu[a]

[a]Babeş-Bolyai University, Faculty of Mathematics and Computer Sciences, 400084, Cluj-Napoca, România

Abstract

The main incentive of this paper is to approach the sensitive subject of classical music synthesis in the form of musical scores by providing an analysis of different Recurrent Neural Network architectures. We will be discussing in a side-by-side comparison two of the most common neural network layers, namely Long-Short Term Memory and Gated Recurrent Unit, respectively, and study the effect of altering the global architecture meta-parameters, such as number of hidden neurons, layer count and number of epochs on the categorical accuracy and loss. A case study is performed on musical pieces composed by Johann Sebastian Bach and a method for estimating the repetition stride in a given musical piece is introduced. This is identified as the primary factor in optimizing the input length that must be fed during the training process.

Keywords: classical music; music synthesis; RNN; GRU; LSTM; FFT

1. Introduction

It is our opinion that music, in general, and especially classical music can be thought of as a form of language designed to convey one's feelings when words alone are not enough. It has several traits which we wish to mention here: pitch, melody, harmony, rhythm, texture, timbre, expression and form.

Consequently, this means that there must be equivalents to words, so the challenge is finding a bijection between a grammatical construct and a musical construct. Fortunately there exists the musical notation system and digital counterparts for encoding it in plain text file formats which can be easily parsed and manipulated. One such format is the ABC music file notation, which is detailed in [1]. The main inconvenient of this file format is that it does not cover the full spectrum of musical constructs found in musical sheets, but on the other hand it brings the advantage that there

* Corresponding author. Tel.: +40-729-788154 ; fax: +40-264-591906.
E-mail address: amarinescu@cs.ubbcluj.ro

10.1016/j.procs.2019.09.166

```
|Note|Dur:8th|Pos:-6|Opts:Stem=Down,Beam=First
|Note|Dur:8th|Pos:-9|Opts:Stem=Down,Beam=End
|Note|Dur:8th|Pos:-13|Opts:Stem=Down,Beam=First
|Note|Dur:16th|Pos:-4|Opts:Stem=Down,Beam
|Note|Dur:16th|Pos:-3|Opts:Stem=Down,Beam=End
|Note|Dur:16th|Pos:-2|Opts:Stem=Down,Beam=First
|Note|Dur:16th|Pos:-3|Opts:Stem=Down,Beam
|Note|Dur:16th|Pos:-4|Opts:Stem=Down,Beam
|Note|Dur:16th|Pos:-5|Opts:Stem=Down,Beam=End
|Bar
```

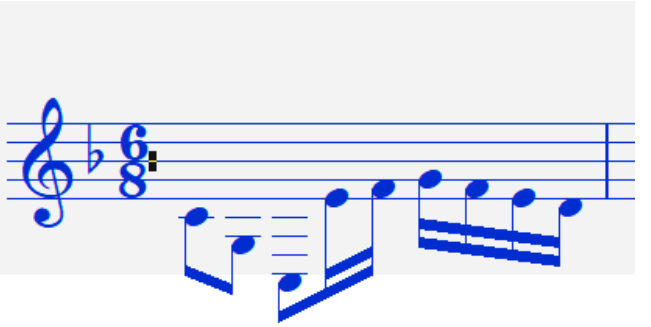

Fig. 1. Example of NWCTXT input and corresponding musical score.

exist extensive data sets which we could use for benchmarking http://abc.sourceforge.net/NMD/. Again, our aim is to try and analyze the style of a certain composer, not a mash-up of different artists and forms. An interesting alternative is the Music XML notation https://www.musicxml.com/ which is becoming increasingly popular and has integration with most professional composition software, but proves difficult to integrate in our test setup. In the end, our choice was Note Worthy's https://noteworthycomposer.com/ NWCTXT file format (Figure 1), where each musical construct (note, rest, chord, bar) is identified by a human-readable string encoding its attributes (duration, pitch and optional specifiers), terminated by the new line character. This, in turn enables us to simplify the parsing of the input and focus on establishing the methodology.

Recent literature emphasizes the success of RNNs at generating musical pieces, but the focus is on LSTM type layers [3, 1, 4, 9, 8, 5]. Very few attempts exist at using GRUs and this is what drove us to perform our benchmark. The scientific literature on the subject of music generation is scarce, hinting to the fact that this branch of artificial intelligence is still in its infancy. [3] focuses on an in-depth analysis of the performance of LSTM type recurrent neural networks in blues genre music generation with two major experiments, first attempting to learn chords and then trying to learn both chords and the melody itself. [1] makes use of the ABC musical notation and compares character-level RNNs with sequence-to-sequence models, while also hinting at the possibility of exploiting the capabilities of generative adversarial networks (or GANs). [4] employs deep recurrent neural networks in an attempt to generate chorales, based on compositions by Johann Sebastian Bach and provides an outline of their musical structure. The authors of [9] propose a new type of recurrent neural network, namely tied-parallel neural networks, which is able to yield polyphonic music (multiple "voices" playing different tunes simultaneously). [8] reviews existing artificial intelligence techniques aimed at music generation which were considered state-of-the-art at the moment of writing, providing valuable insight into this particular research domain. Finally, [5] takes a different approach than ours, tackling music synthesis directly in the MIDI binary file format, whilst simultaneously analyzing the capabilities of RNNs (bi-directional GRU and LSTM).

Concluding this discussion, we now have the building blocks for training a Recurrent Neural Network (RNN), whichever flavor is preferred, as it has originally been designed for Natural Language Processing (NLP). The following sections first introduce the reader to a brief scientific background, continuing with our proposed architecture backed up by experimental results, and finally closing with an answer as to whether it is feasible to employ a RNN for classical music synthesis in the style of J. S. Bach and hint as to which architecture performs best out of the experimental trials.

2. Theoretical background

A recurrent neural network (RNN) is a class of artificial neural network where connections between nodes form a directed graph along a sequence. This allows it to exhibit temporal dynamic behavior for a time sequence. Unlike feedforward neural networks, RNNs can use their internal state (memory) to process sequences of inputs. In machine learning, the vanishing gradient problem is a difficulty found in training artificial neural networks with gradient-based learning methods and backpropagation. In such methods, each of the neural network's weights receives an update proportional to the partial derivative of the error function with respect to the current weight in each iteration of training. The problem is that in some cases, the gradient will be vanishingly small, effectively preventing the weight

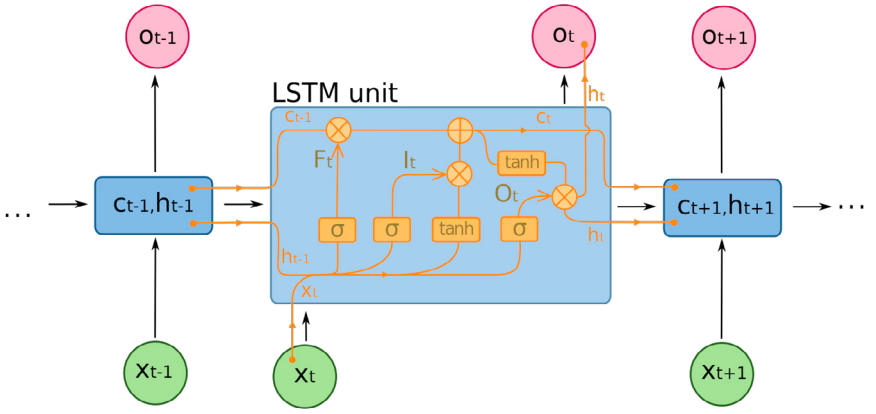

Fig. 2. LSTM unit courtesy of François Deloche from Wikipedia.

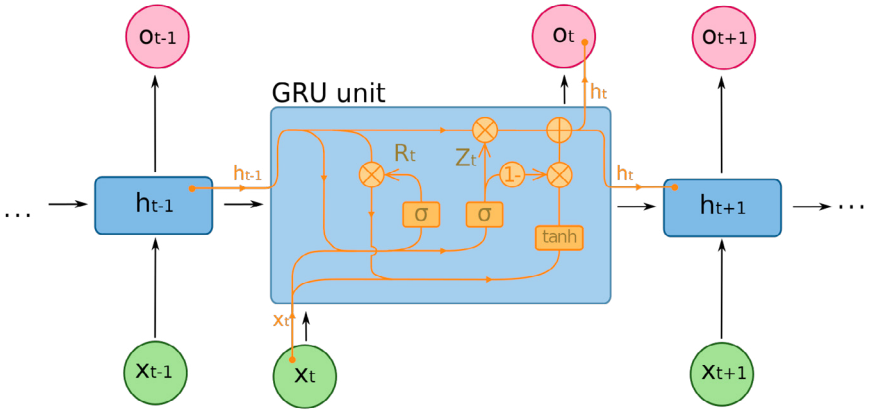

Fig. 3. GRU unit courtesy of François Deloche from Wikipedia.

from changing its value. In the worst case, this may completely stop the neural network from further training. As one example of the problem cause, traditional activation functions such as the hyperbolic tangent function have gradients in the range (0, 1), and backpropagation computes gradients by the chain rule.

Long short-term memory (LSTM — Figure 2) is a deep learning system that avoids the vanishing gradient problem [7]. LSTM is normally augmented by recurrent gates called ”forget” gates. LSTM prevents backpropagated errors from vanishing or exploding. Instead, errors can flow backwards through unlimited numbers of virtual layers.

Gated recurrent units (GRUs — Figure 3) are a gating mechanism in recurrent neural networks introduced in 2014 [10]. They have fewer parameters than LSTM, as they lack an output gate. This in turn implies that they can be trained faster and have a lower memory footprint.

3. Proposed architecture

The core method has been implemented in the Python programming language using the Keras framework, on top of a TensorFlow backend. Originally, we used 3 GRU layers stacked one on top of each other, with a hidden neuron count of 512, but afterwards we decided to experiment also on LSTM layers and varied the hidden neuron count and the number of layers for the purpose of gaining a better understanding on which performs best. The corresponding initialization file where the parameter dictionary is defined has the following content:

- *number of steps = 16*

Fig. 4. The musical note that corresponds to *|Note |Dur:8th |Pos:-2 |Opts:Stem=Down*.

- *batch size = 32*
- *hidden size = 512*
- *number of epochs = 1000*
- *temperature = 10*

Next we will discuss the data that we used for our benchmarks. It comprises 35 manually transposed to NoteWorthy J. S. Bach works, with an average length of 50 musical measures. They have been normalized to a measure length of 6/8, in the B minor scale, with a tempo of 100. The files contain all musical units (note, rest, chord and bar) supported by the NWCTXT file format. The musical pieces comprise sonnatas and partitas written by the famous composer. We have explicitly stated this information since all data are stripped of this header, which is prepended when we predict how a sequence should continue to preserve format integrity.

In order to obtain the training and validation files, the database is traversed sequentially, the header information is removed and the file is split, without random shuffling of entries, into a 80% training — 20% validation proportion, which are then merged. At the end of this preprocessing step, we obtain a training file with a length of 18677 musical units and a corresponding validation file with a length of 4686 musical units. We chose the methodology of per-file splitting instead of per-whole dataset mainly due to the fact that we wanted to keep a proportion of each musical piece since each is unique in structure and this is exactly what we want the RNN to identify.

A key aspect worth mentioning here is that a musical unit may have a varying number of corresponding words inside the vocabulary, but this is generally ≤ 4. For example, a Bar is represented by a single word, a Rest, by 2, while a Note or a Chord, by 3 or 4 words. Finally, at the end of this process, the RNN builds a vocabulary of 578 distinct words, the vast majority being optional modifiers attached to notes, which are treated as a whole instead of being divided into components.

To further detail the musical note to vocabulary word mapping, let us take for example the following note definition (Figure 4) in NWCTXT format: **|Note |Dur:8th |Pos:-2 |Opts:Stem=Down**. It yields 4 words:

- *Note*, which tells us that we are dealing with a musical note. Valid values are: *Note*, *Rest*, *Chord* and *Bar*
- *Dur:8th*, which specifies the note duration to be one eighth
- *Pos:-2*, which means that the note is a G. Here we can also have a # (sharp) note modifier — *Pos:#-2*
- *Opts:Stem=Down*, which hints that the stem of the note should be oriented downwards on the musical score. Multiple options can be specified here, such as beam start/end, but we will treat all optional modifiers as a whole word

In order to actually build the vocabulary, all input data is parsed and all resulting words are stored in a dictionary, which is sorted in decreasing order by frequency of occurrence in the training data set. In the end, the input fed to the RNN contains the replacement of each word with its corresponding index in the vocabulary.

4. Experimental results

We have performed experiments with both GRU and LSTM layers, with layer depth ranging from 1 to 3 in order to examine which performs best (Figure 6). We have also tried varying the number of hidden neurons for each layer, more specifically 256, 512 and 1024 respectively. In addition to this we have also altered the number of steps the RNN keeps track of, 8, 16 and 32. The number of steps can be determined in an algorithmic manner [2] by performing a FFT analysis on the input thus computing the distance between successive peaks on the spectrogram (Figure 5). This gives us an overview of the repetition factor for the musical composition. In order to actually predict a musical piece,

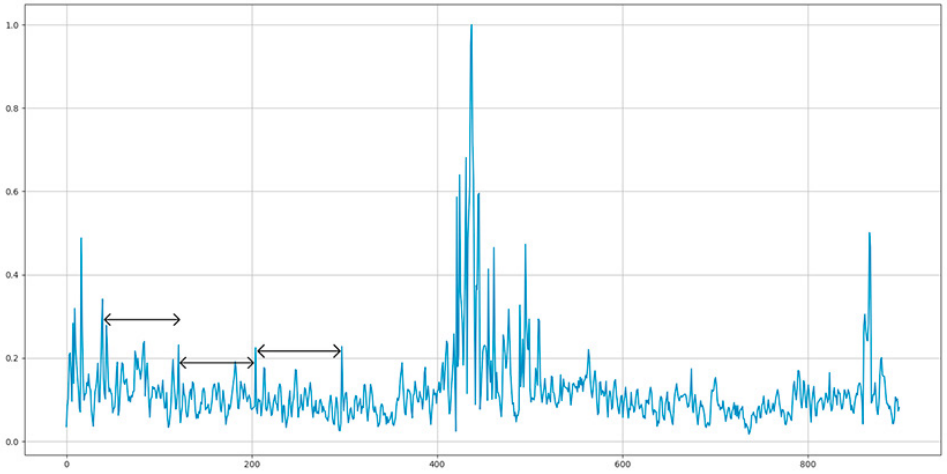

Fig. 5. An empirical way of determining the repetition stride for use as RNN number of steps.

the RNN is primed with a sequence of notes taken from the beginning of a musical piece, and successively generates notes based on a queuing mechanism. We avoid generating repetitive sequence by means of a temperature mechanism: at each prediction there is a chance that we will not choose the best word, but instead take the second or third and so on. This probability decreases proportional to the probability of the word, as given by the SoftMax function [6]. The significance of each experimental run illustrated in Figures 7 and 8, respectively is detailed below:

- **Run 1:** *number of steps = 16, batch size = 32, hidden size = 512, number of epochs = 100*
- **Run 2:** *number of steps = 16, batch size = 32, hidden size = 512, number of epochs = 1000*
- **Run 3:** *number of steps = 16, batch size = 32, hidden size = 256, number of epochs = 100*
- **Run 4:** *number of steps = 16, batch size = 32, hidden size = 1024, number of epochs = 100*
- **Run 5:** *number of steps = 8, batch size = 32, hidden size = 512, number of epochs = 100*
- **Run 6:** *number of steps = 32, batch size = 32, hidden size = 512, number of epochs = 100*

We deduce from Figure 6 that the best training and validation accuracy is attained by the 3-layer gated-recurrent unit neural network, with the parameters listed in Section 3. Contrary to what is stated in literature [11], the gated recurrent unit, at least for our test setup manages to outperform similarly layered LSTM architectures. But the most striking behavior is obtained for the loss function (namely categorical cross-entropy), with a much sharper drop for GRU than LSTM confirming the fact that GRUs are more suitable for training on small-scale data sets. Figures 7 and 8 illustrate the effects of varying different network meta-parameters, with respect to a fixed 3-layer architecture.

5. Conclusions

As a consequence of our experiments we have determined that a 3-layer GRU is a very promising candidate for classical music synthesis, being robust with respect to relatively small data sets. It has been trained for 100 epochs, albeit judging by the behavior of the validation loss we can safely stop at 50 epochs. Interestingly, this is achieved with a number of time steps equal to 32, suggesting that there is room for improvement if we increase this even further. The GRU outperforms the LSTM, reaching a training accuracy of 95% and a validation accuracy well above 75%. One can find sample output and corresponding musical sheet at the following URL: `https://amarinescu.ro/bach/`.

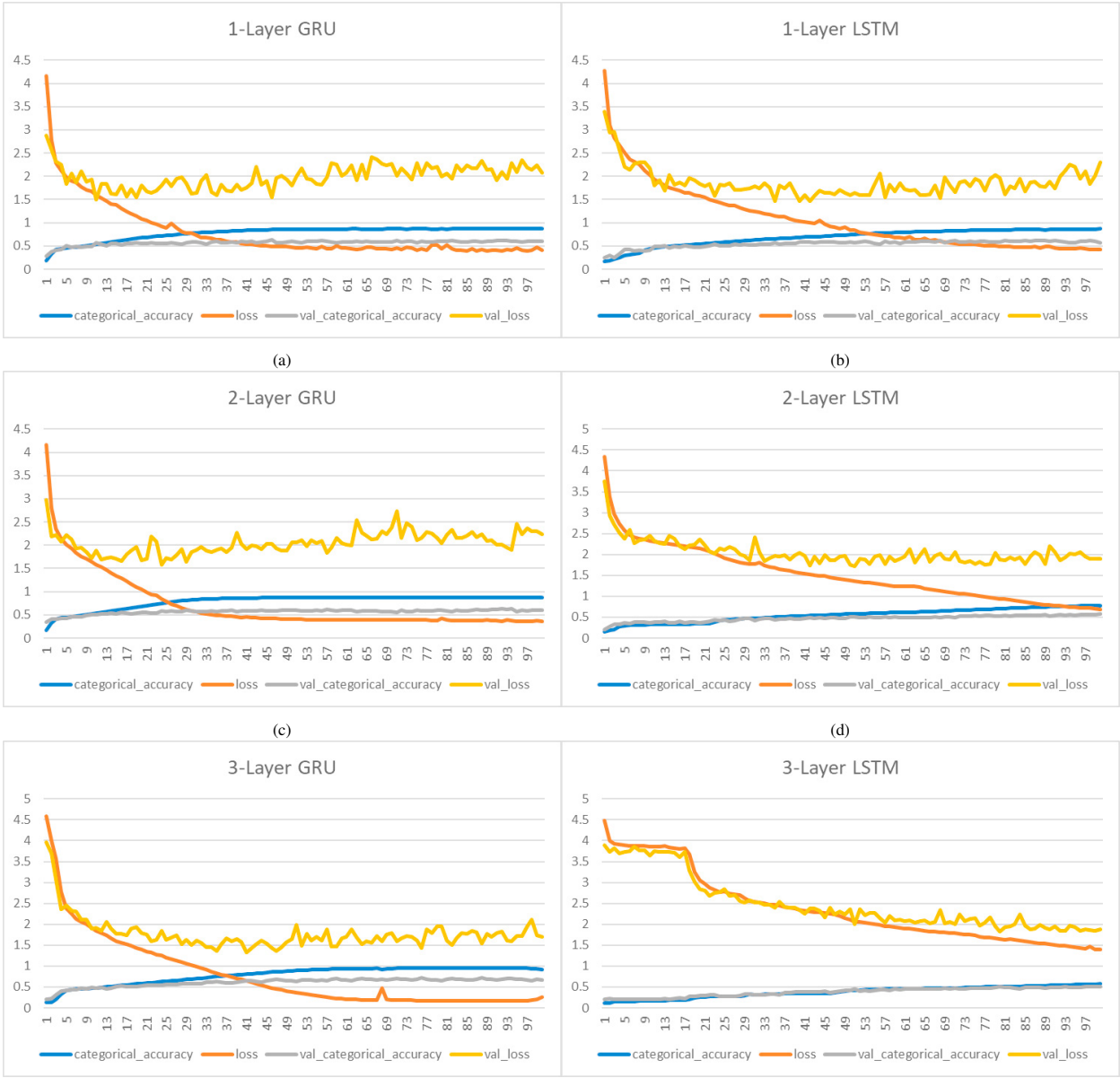

Fig. 6. (a) 1-Layer GRU. (b) 1-Layer LSTM. (c) 2-Layer GRU. (d) 2-Layer LSTM. (e) 3-Layer GRU. (f) 3-Layer LSTM.

As far as future research is concerned, we shall further refine the number of time steps the RNN keeps track of based on a FFT analysis of the input, enabling an automatic detection mechanism for this hyper-parameter. Furthermore, we wish to extend our database with more compositions and perform a cleaner output post-processing since, at the current time, the placement of bars in the output piece is inconsistent with the time measure.

References

[1] Agarwala, N., Inoue, Y., Sly, A., 2017. Music Composition using Recurrent Neural Networks.
[2] Bracewell, R., 1978. The Fourier Transform and its Applications. Second ed., McGraw-Hill Kogakusha, Ltd., Tokyo.
[3] Eck, D., 2007. A First Look at Music Composition using LSTM Recurrent Neural Networks.

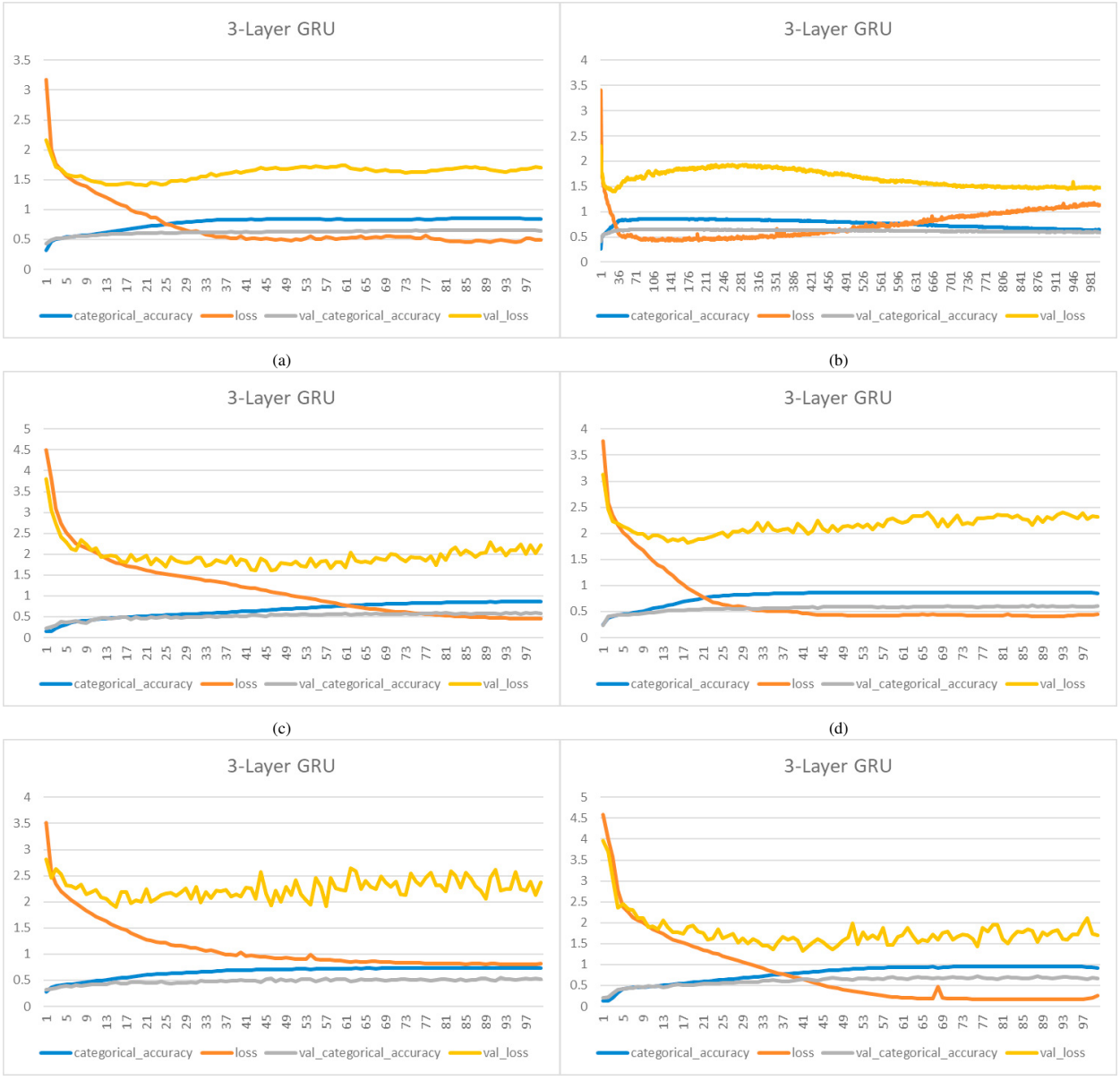

Fig. 7. (a) Run 1. (b) Run 2. (c) Run 3. (d) Run 4. (e) Run 5. (f) Run 6.

[4] Gaëtan Hadjeres and François Pachet, 2017. DeepBach: a Steerable Model for Bach chorales generation, in: ICML.
[5] Hilscher, M., Shahroudi, N., 2018. Music Generation from MIDI datasets.
[6] Hinton, G.E., Vinyals, O., Dean, J., 2015. Distilling the Knowledge in a Neural Network. CoRR abs/1503.02531.
[7] Hochreiter, S., Schmidhuber, J., 1997. Long Short-Term Memory. Neural Computation 9, 1735–1780.
[8] Jean-Pierre Briot and François Pachet, 2018. Music Generation by Deep Learning - Challenges and Directions. CoRR abs/1712.04371.
[9] Johnson, D.D., 2017. Generating Polyphonic Music Using Tied Parallel Networks, in: EvoMUSART.
[10] Kyunghyun Cho and Bart van Merrienboer and Çaglar Gülçehre and Fethi Bougares and Holger Schwenk and Yoshua Bengio, 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. CoRR abs/1406.1078.
[11] Nayebi, A., Vitelli, M., 2015. GRUV : Algorithmic Music Generation using Recurrent Neural Networks.

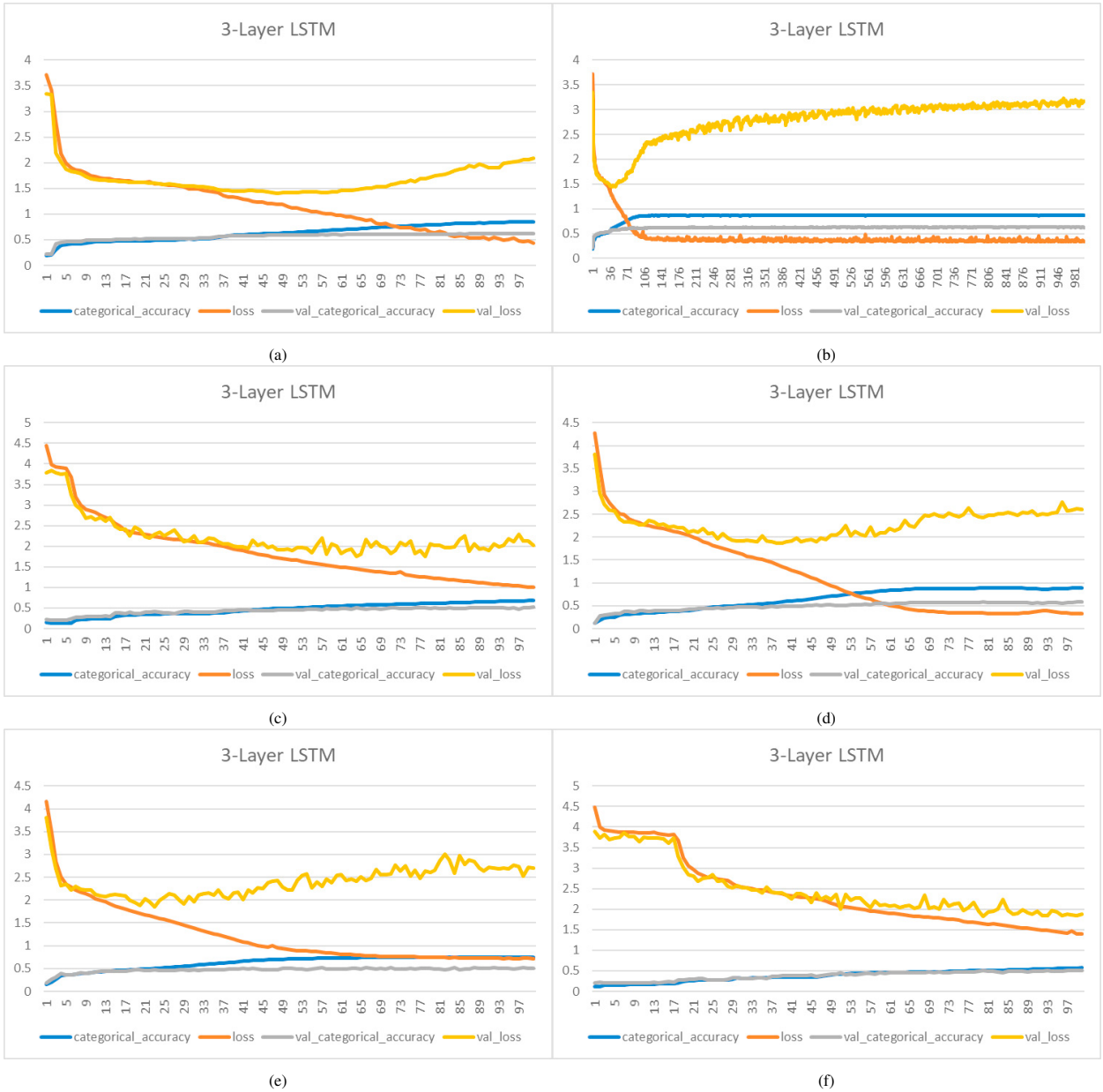

Fig. 8. (a) Run 1. (b) Run 2. (c) Run 3. (d) Run 4. (e) Run 5. (f) Run 6.