Semester Project Suggestion "Generating a drum rhythm"

Overview. The task is to train an ML system to generate a drum rhythm that a human listener finds … well, at least, "interesting".

This task can be tackled in various degrees of sophistication. Good human drummers generate amazingly complex drum patterns. Simple versions of the task are quite easily doable, whereas at the high quality end, even current advanced professional music modelling hits limits. This task invites to use recurrent neural networks, but other kinds of models can also be used.

The simplest versions would look something like this:

- The output of the trained system is an endless sequence of 1's and 0's. Segmenting this sequence in intervals of length 16 ("bars" in music terminology), a very regular beat (too boring) would set the output to 1 on positions 1 and 9 in each of the bars, and to 0 in all other positions. To make this minimally un-boring, in some bars the pattern should be different, for instance 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 instead of 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0. The challenge is to make the model switch between these two kinds of bars in a random way.

A high-end version would look like this:

- The output of the trained system has as many channels as the drummer has drums (bass drum, cymbals, …). The bars are segmented into fine intervals, say 128 bar slices, to allow for fine variations of timing. Again, zero values are almost always generated in each channel, interspersed by nonzero values between 0 and 1, with the size of these nonzero values indicating loudness. A richly structured output is generated, like from the solo of a good drummer. … this is likely impossibly difficult to achieve in a semester project … but you can try to go as far in this direction as you can manage.

Listen to the music. It would be a rather dry exercise to just produce such numerical output sequences. You should sonify your network's numerical output and play it. This can be done (and this is how everybody in the business does it) by transforming your NN output to the MIDI format (check out https://en.wikipedia.org/wiki/MIDI) and sending this to a MIDI player, of which there are plenty. MIDI is the absolute standard for professional music encoding and you should not do this project without importing from and exporting to MIDI.

Data. This is a machine learning task – you need training data. There are many online sources for music pieces in MIDI format. Choose your preferred genre and collect a number of MIDI recordings of pieces from that genre. Make sure that they are all in the same basic beat (4/4 would be the most common I would think). The more varied the example pieces in your collection, the more interesting output your NN can learn, but also the whole task becomes more challenging.

Data preprocessing. You cannot directly feed MIDI (transformed to CSV as a first step, check out https://www.fourmilab.ch/webtools/midicsv/) to a neural network or other kind of ML model. You have to think of a way to encode that into a numerical input sequence, presumably in vector format if you want to model more than a single drum.

Training and generation: principle. In what follows I only explain set-ups where loudness of drum hits is not encoded – a drum is hit or it is not hit in a given bar slice, which can be encoded in 0-1-values. This will result in generated rhythms which are monotonous and not super pleasant to listen to, so you might want to do better than that.

The basic idea is to train your network on a one-timestep prediction task. The training data consist then in one or several paired timeseries of the form $(\mathbf{u}(n), \mathbf{y}(n))_{n = 1, \ldots, N}$, one timeseries for each piece in your data collection, where $\mathbf{y}(n) = \mathbf{u}(n+1)$. Each vector $\mathbf{u}(n)$ is representing the drum hits in bar slice n and has as many components as there are drums that you model. Let us assume the dimension of $\mathbf{u}(n)$ is k.

If you choose an MLP for your network, the input at time n is a time window of h preceding bar slices, that is, it will be a vector of size kh obtained from concatenating $\mathbf{u}(n\text{-}h\text{+}1)$, …, $\mathbf{u}(n)$. If you choose a RNN architecture, the input at time n just $\mathbf{u}(n)$.

After training, when the input is a (new, test) sequence $\mathbf{u}(n)$, the output at time n will be a *hypothesis vector* $\hat{\mathbf{y}}(n)$ which, if the training went perfectly, should have *probability values* in the range [0,1] in each component. These probability values reflect the degree of "belief" of the network that in the respective drum channel, in the time slice n+1 the associated drum should be hit.

In order to use the trained network to *generate* drum patterns autonomously in a free-running fashion, the procedure is slightly different depending on whether you use MLPs or RNNs.

If you use MLPs, you need an initial input sequence $\mathbf{u}(1)$, …, $\mathbf{u}(h)$ that was not generated by the network to get the generation started. Take this cue sequence from one of your collected training pieces, it does not greatly matter which.

If you use an RNN, you would first "prime" the network by driving it with a short initial sequence taken from one of your training pieces, and start the autonomous generation thereafter.

In both cases, at some time n during the generation, the network will generate a hypothesis output $\hat{\mathbf{y}}(n)$ containing probability values in the ideal case. If the world isn't perfect (it isn't!), the output $\hat{\mathbf{y}}(n)$ may contain values outside [0,1] – you would then have to trim the outputs to this range in order to obtain a vector made of probability values.

At this point, you need to design and implement a decision-making algorithm which turns the hypothesis vector $\hat{\mathbf{y}}(n)$ into an actual beat pattern. There are three major design options:

- *Thresholding*: whenever $\hat{\mathbf{y}}(n)$ is larger than a predetermined threshold in some channel i, let the drum i beat in bar slice n+1.
- *Weighted random choice:* generate a beat in channel i with the probability that you find in the i-th component of $\hat{\mathbf{y}}(n)$.
- *Combined method*: you can mix the thresholding method (which is deterministic) with the random choice method by first transforming the probability values p which you find in $\hat{\mathbf{y}}(n)$ by a formula like $p \to \sigma(a(p-1/2))$, where σ is the logistic sigmoid and $a \geq 4$ is a scaling factor (the number 4 comes from the fact that the derivative of the logistic sigmoid at argument 0 is ¼). The greater a, the more strongly will the original p values pushed toward 0 or 1 depending on whether it was smaller or greater than ½. The use the transformed p values like in the weighted random choice method. If a is set to very large values, you will come close to the deterministic thresholding method (with a threshold of ½).

Once you have decided how to transform the hypothesis vector $\hat{\mathbf{y}}(n)$ to an actual drum output, you actually generate that output, and feed it back as next input $\mathbf{u}(n+1)$ to your network.

Judging musical quality. You will discover that there is no easy to define, natural mathematical way to quantitatively measure the musical quality of output "music" that your network generates. The ultimate quality assessment comes from your personal listening judgement. This ultimate judgement is something to enter into the game in your final discussion of the obtained results. However, human subjective judgement is too imprecise and time-consuming to be used during cross-validation (you will have to set up some cross-validation scheme during training to optimize the prediction qualities of your network). You need a suitable loss function both for training your network, and for assessing validation errors during cross-validation. Use a procedure called *logistic regression* (tutorial: https://machinelearningmastery.com/logistic-regression-tutorial-for-machine-learning/) which uses a specific kind of output units and loss function.

Deliverables: as specified in the "SemesterProjectInstructions.pdf" (report, link to code or code) plus one or two generated drum music samples (not too long, say 20 sec) in mpeg4 format, such that I can become emotionally touched (or shattered) by your results.

A 2014 **Bachelor thesis by Tomas Pllaha**, *Echo State Networks: Music Accompaniment by Prediction*, treats a topic that is similar in several ways to this project suggestion. If you take a look into it (file , included in this project suggestion package), you'll get at taste of the flavour of this kind of project.