DL LAB 2: An application with RNN

Senne Deproost and Diego Saby

November 2019

1 Introduction

In the field of machine learning we try create computational models to find solutions for a wide set of tasks. One of the more popular problems to solve are the classification of instances and function approximation with regression. Solutions like *Support Vector Machines* and *Decision Trees* have already proven their worth in the past. Techniques from the deep learning (DL) sub-field of machine learning have shown their usability when tackling on data with high dimensionality. Deep learning's ability to scale better with these kinds of data makes it useful in many application domains like machine control and decision support systems. Variations in the model's architecture could insure better performance on a domain specific task.

One DL architecture we'll focus on in this lab report is the *Recurrent Neural Network* architecture. Whereas classical *Feedforward Neural Network* (FNN) only allows the passage of activation through the neuron layers, a RNN network will remember the neuron activations from a previous step in time. This is implemented with the inclusion of memory nodes that will hold on to previous encountered activation values. An advantage of this kind of networks is the usage of *Temporal Sequences*, allowing the model to train on data like text sentences, music recordings and streams of data.

Automated content creation using DL models has known a significant rise in popularity over the past years. With *Generative Adverserial Networks* (GAN's), we can use two competing networks to train a generator for images (Goodfellow et al., 2014). In this architecture, one network, the has to mimic given image data input distribution and another adversarial network that has to estimate the loss of the predicted regression with given training data. This prediction can then be used to optimize the generative network. Other types of models like *transformers*, recently used in OpenAI's GPT-2 model (Radford et al., 2018), can generate human-like text paragraphs with the potential of passing the famous Turing test. With a rise in the exploration of the generative content field, researches already used RNN based models to generate content like images (Gregor, Danihelka, Graves, Rezende, & Wierstra, 2015), voice synthesis (van den Oord et al., 2016; Van Den Oord, Kalchbrenner, & Kavukcuoglu, 2016) and music (van den Oord et al., 2016).

Music generation can be seen as a series of notes. We can try to predict the next note given the previous series of notes. For this kind of problem RNNs are useful.

1.1 Problem

In this report, we will focus on three types of models in the task of creating music from MIDI files. The data will consists of fragments of Chopin's Nocturne music composition in the MIDI file format with 30 available notes available on the tone ladder. The music consists of only one music instrument: the piano.

Apart from regular RNNs, we will experiment on *Long Short-Term Memory* (LSTM)(Hochreiter & Schmidhuber, 1997) and *Gated Recurrent Unit* (GRU) models (Cho, van Merrienboer, Bahdanau, & Bengio, 2015).

1.2 LSTM

An LSTM is a form of RNN which solves the problem of the large decay in loss function gradient over long periods of time. Specialized memory cells are used to recall on several past activations for longer periods of time. In addition, the network contains gates to control the information flow to and from memory and when it can be returned to its initial status.

1.3 GRU

GRU are similar in construction to LSTM that they have gates to control the specialized memory cells. However, these models are simplified containing less gates and dont make the distinction between memory cells and normal ones (Chung, Gulcehre, Cho, & Bengio, 2014).

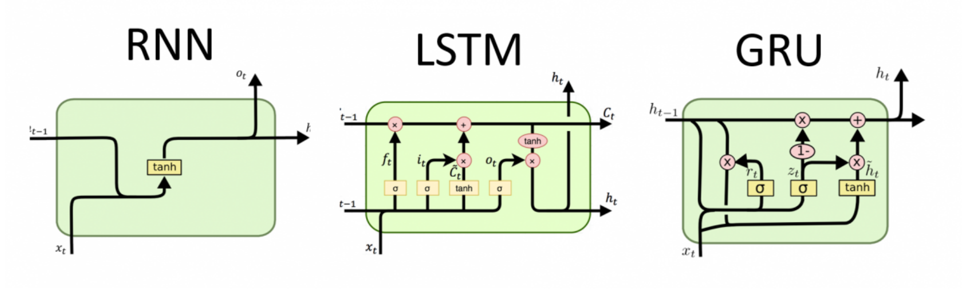

Figure 1: An overview of the three kinds of models used in the experiments. Left we have a simple RNN node containing a simple *tanh* activation function. In the middle the internal gates of the LSTM node are shown. The right contains the simplified layout of a GRU network node.

2 Preprocess the data

One of the first questions we were opposed to was how to encode music in order to train the Neural Network. Our source files are midi files. They are an useful way to encode the music where we can extract the notes and their duration from the file.
To simplify the problem we decided to abstract the notes duration and just use the notes pitch.

2.1 Enumerating the notes

Our first attempt was done inspired by some code found on the internet to produce video games music. **add reference**
To extract the notes the notes from midi file we used a library called *Music21*. It provided us with the notes names or chord names that were played. From there we have a sequence of notes names. Then we just assign the notes a number from 1 to the length of the vocabulary (notes and chord names). For the prediction we used a vector of all the notes and a with 1 for the next note that follows the sequence and 0 for the others.
After several experiments using a as the last layer a Fully connected NN with a *softmax* activation function and taking the note with the biggest probability to be the next one as the prediction of the next note, the results were disappointing No matter what was the input it always yielded the same note. After changing the RNN architecture, the learning rate, the

optimization function, it changed the note that was predicted, but always was a sequence of the same note.

2.2 Notes as Integers

Our second approach was to translate directly a note pitch into an integer. The lower the pitch the bigger the integer. This could be done thanks to another library called *mido*, and was inspired from a project on github. **add reference**
And for the prediction we used a linear regression as the last layer of the RNN architecture. Since the notes are just integers and not floating numbers the result was truncated in order to transform it later into a note.
This approach made more sense. Not only the notes have an actual order, but there is also the fact that we could even produce notes that were never seen before. Our first attempt with this approach was surprisingly good. After testing, not only the sequence of notes were different of each other, but the beginning of the *song* produced was not bad to hear. But most of the *song* produced afterwards one could ask itself if it's not a sequence of random notes. It sounded like a child smashing random keys on a piano.

3 Experiments

In the experiments, we mainly focus on the three different kind of models as described before. We first vary the numbers of node in the hidden layers of the network. Afterwards, we change the number of layers in the network. We train with a fixed batch size of 64 during a 200 epoch session. The results will be discussed based upon the loss function of the model and the qualitative analysis of the generated music fragments. We fixed the window lag of notes to 30 notes, to predict the next one, and we did not put any skipping to generate the data training.

3.1 Vanilla RNN

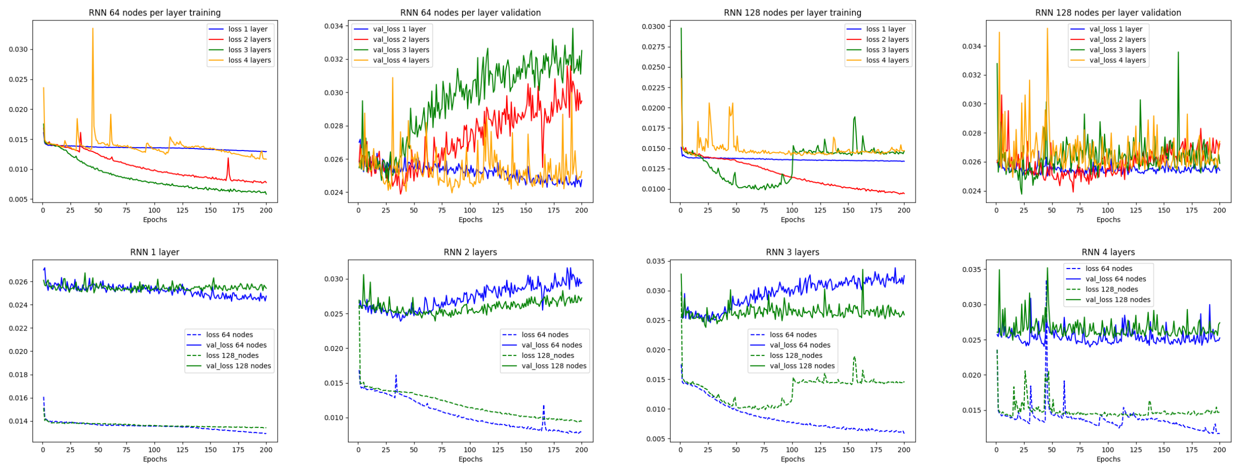

Figure 2: Results from RNN models

If we have a look at the resulting plots of *figure 2*, we see large fluctuation of the loss function when validating the models. The general trend of the loss for a 64 node per layer model is a drop around 0.025 between 25 and 50 epochs. The model with 3 hidden layers starts to gain the most lost the quickest and the simplest one with one layer continues to drop to a loss of around 0.024. The validation of the 128 nodes variant fluxes the most and tends to average around 0.026 for any amount of layers.
The RNN model with 3 layers shows the most divergent loss functions between the 64 and 128 nodes variant. For the 4 layer variants we can observe a huge spike in validation loss when training loss spikes. This does not immediately indicates a correlation between both functions rather it could be similar impactful, yet different entries that generated a big spike in both training an validation in the proximity of that epoch.

3.2 LSTM

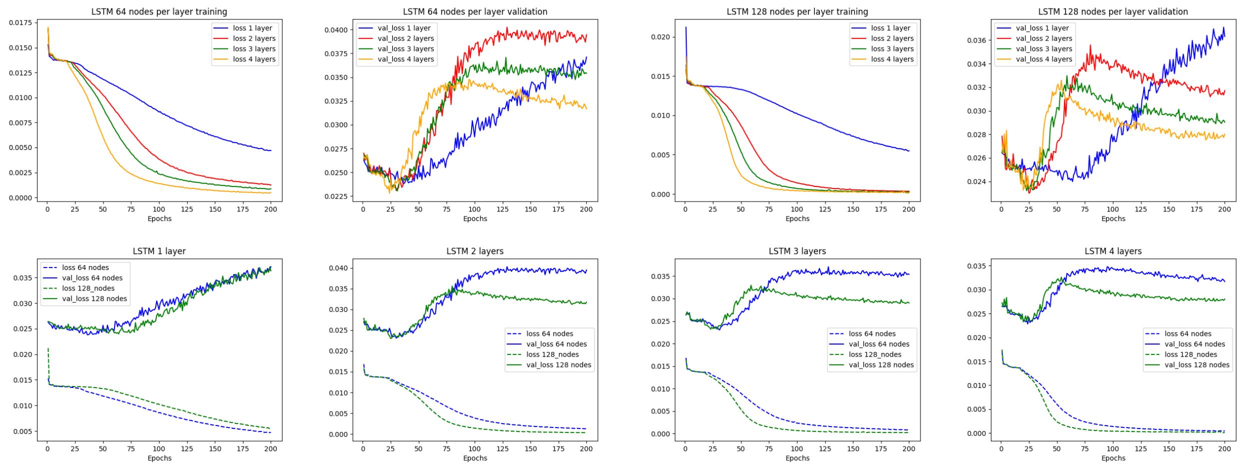

Figure 3: Results from LSTM models

In *figure 3* we can observe a similar trend for the loss functions of different models and in general less sporadic fluctuations then the RNN ones . The loss starts with a significant drop to a local minimum before gradually rising again. At different epochs we then observe a ceiling for the function and a consecutive decrease. This trend however is not noticeable for the simple 1 layered model as it keeps increasing in loss, even after the experiment's limit of 200 epochs.
For a 64 nodes model, we can conclude overfitting will occur after the local minimum of the validation loss function at around +30 epochs for more then one layer models and around +45 epochs for the single layered one. For the 128 nodes variant we observe a significant increase in epochs needed for overfitting the 1 layer model compared to the other ones. While they have their loss minimum at 25 epochs, the single layer reach it at 75 epochs. As for the training part of the loss functions, we observe much smoother ones then those of the trained RNN models.

3.3 GRU

Last we run experiments with the GRU variant of RNN's.

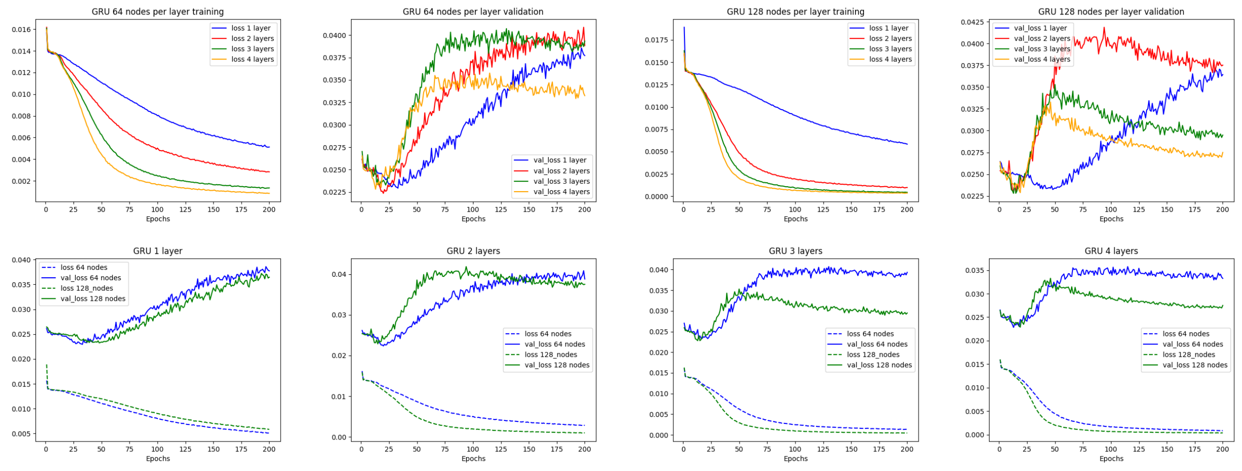

Figure 4: Results from GRU models

4 Evaluation

4.1 With plots of the loss function

We cannot evaluate the quality of the music generated with those plots. In one hand we cannot really validate the training data with a validation data, since the validation data are two different compositions and it is *impossible* that the network predicts the sequences of a new song base solely . It is normal that the the validation and training loss error grow apart in the plots.
But in the other hand, since we are training with a particular theme: Chopin's Nocturnes, we do not want them to grow too much apart, we want to generalize the theme and the network learns the *style* of Chopin's Nocturnes.

With the exception of Vanilla RNN, the validation loss error tends to grow and then reach a point were it *stabilizes* or even in some cases it decreases.
Those parts could be explained as when the model learns the training music a little, therefore the validation music differs from the training model and the error grows. And when the validation loss error starts to decrease we could think that the network learned better the sequences of notes that go good together so the error in validation decreases.

The GRU model with 4 layers and 128 hidden units is a good example of this **figure** 4.

For the Vanilla RNN just watching the plots of the loss function, one could say that the network barely learns something with models with 64 hidden units per layer, and not at all with the models with 128 hidden units per layer.
But what does the music produced sounds like?

4.2 Comparing the music

5 Conclusion

RNNs can generate music. Given a good training set the network could learn sequence of notes that are good to hear and maybe avoid sequences that do not sound good. But how *creative* could it get?
One of the main difficulties with this approach is that evaluating music is really subjective. There is not a metric that can objectively evaluate how good a sequence of notes is.
One of our main problems that we can see is how bad the accuracy was when comparing the training set with a validation set. This is not surprising, since, even though the compositor was the same and the style was similar (we only worked on the Chopin's Nocturnes), the songs in the validation set were different and new from the training set.

Talk more about the results, were they good? what happened? What was the best architecture

For further work on this we could see the effect of changing the *windowing* for the sequence of notes in the training set. And to produce more realistic music add the tempo to the pitch to produce real notes.
We do believe that some interesting music can be generated with RNNs but never as good and creative as humans.

References

Cho, K., van Merrienboer, B., Bahdanau, D., & Bengio, Y. (2015). On the Properties of Neural Machine Translation: Encoder–Decoder

Approaches. In *Proceedings of ssst-8, eighth workshop on syntax, semantics and structure in statistical translation* (pp. 103–111). doi: 10.3115/v1/w14-4012

Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. Retrieved from `http://arxiv.org/abs/1412.3555`

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... Bengio, Y. (2014). Generative Adversarial Nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems 27* (pp. 2672–2680). Curran Associates, Inc. Retrieved from `http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf`

Gregor, K., Danihelka, I., Graves, A., Rezende, D. J., & Wierstra, D. (2015). DRAW: A recurrent neural network for image generation. In *32nd international conference on machine learning, icml 2015* (Vol. 2, pp. 1462–1471).

Hochreiter, S., & Schmidhuber, J. (1997, nov). Long Short-Term Memory. *Neural Comput.*, *9*(8), 1735–1780. Retrieved from `http://dx.doi.org/10.1162/neco.1997.9.8.1735` doi: 10.1162/neco.1997.9.8.1735

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2018). Language Models are Unsupervised Multitask Learners.

Van Den Oord, A., Kalchbrenner, N., & Kavukcuoglu, K. (2016). Pixel recurrent neural networks. In *33rd international conference on machine learning, icml 2016* (Vol. 4, pp. 2611–2620).

van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., ... Kavukcuoglu, K. (2016). *WaveNet: A Generative Model for Raw Audio* (Tech. Rep.). Retrieved from `http://arxiv.org/abs/1609.03499`