Deep Learning

Recurrent Neural Network

Music Dataset

Olga Valls Murcia

Introduction

The aim of this Laboratory is to configure and train a Recurrent Neural Network to be able to predict, for one song, the next chord to be played.
Given the code and datasets for the Guided Laboratory, the goal is to use another kind of dataset that follows the same idea as the "*Task 1: Time Series Regression (Air Quality Prediction)*" using a music dataset where, instead of weather stations, air quality particles and hour samples, songs, notes/chords and time samples are given.

Dataset

The data set used consists on 47 chorales of Bach, extracted from the original data set that can be found in the following URL: https://archive.ics.uci.edu/ml/datasets/Bach+Chorales

It has the following 17 attributes:
1. Choral ID: corresponding to the file names from (Bach Central)[[Web Link]].
2. Event number: index (starting from 1) of the event inside the chorale.
3-14. Pitch classes: YES/NO depending on whether a given pitch is present.
Pitch classes/attribute correspondence is as follows:
15. Bass: Pitch class of the bass note
16. Meter: integers from 1 to 5. Lower numbers denote less accented events,
higher numbers denote more accented events.
17. Chord label: Chord resonating during the given event.

Distribution of training and test sets

For the present work, only 47 chorales were selected, those with a minimum number 70 events, and all of them were cut at this same number. Thus the training set is balanced, and no chorale has more or less events than the others to avoid that the tendency of one song can have more weight over the rest of the songs.

Another option has been considered in order to have larger sequences of songs, which consists on padding silences at the end of the shortest songs. This has been discarded as the system would learn that is coherent to have a subset of silences in a song, which is not.

Issues on the size of the dataset

The final size of the dataset is a big issue to consider that will affect the experiments done, as the values of the loss function will be high and the R2 low. However, that has been used to analyse how far can we get on setting the hyperparameters of the network and be able to experiment how to get to the better results.

Pre-processing the dataset

One script has been created to process the original dataset and crop each chorale to 70 events (*create_dataset.py*) which creates an npz file (*bach_dataset.npz*) with the 3D matrix of (47, 70, 15) size. From the original 17 attributes, the name of the song and the sequence inside each song is discarded.

Splitting of the dataset

46 of the 47 has been used to train the Recurrent Neural Network, and one choral to test whether it can predict the next chord in the sequence.
To increase the number of samples and help the Network learn, a windowing has been applied. For each original sample, a number of splitted samples are provided, given the number of windows:

$$\text{windows} = \text{columns} - \text{lag} - \text{ahead} + 1$$

where lag is the size of the window, and ahead is the number of events to predict. The window moves ahead positions in each step and provides a new sample of width windows.

Configuration of the Recurrent Neural Network

The problem at hand is a regression problem, as it tries to predict the value of an attribute for a given test set, specifically, the chord. Therefore, the loss function used to compute the error function is Mean Squared Error (MSE), which is the sum of squared distances between our target variable and predicted values.

We want to train the neural network so that it learns how the training set behaves, and extrapolate that behaviour to the test set. On the other hand we are not interested in converging fast to overfitting as it would mean that the networks learns fast the characteristics of the training set, but it’s probably unable to generalise well and make a good prediction given the test set.

Number of Epochs

The value of loss function for the training set decreases in time, as the network is more capable to learn from the training set. That is also the behaviour for the test set, until a number of epochs is reached, when the value for the loss function changes its direction and the predictions start to give worse results, as if the network unlearned.

Below, figure 1 shows this effect, where the test loss curve starts to follow the training loss curve, and at epoch 100 changes its behaviour drifting apart from the training loss curve.

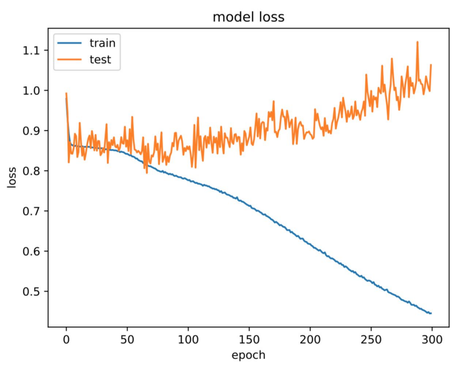


Figure 1: epochs

Batch size

The correct batch size to use depends on the problem and the number of samples. It is true that a small number will help training the network faster because it updates the weights after each propagation. If we use all samples, as batch size, during propagation we would make only one update for the network's parameter.

On the other hand, the smaller the batch we use, the less accurate will be the estimation of the gradient. Thus, we have to experiment on our network, which is the best number of batch size given our dataset size and characteristics. In our case, a batch size of 50 is the one that gives the best results.

Activation function

The default activation function is tanh (bound to range (-1, 1)), although experimenting with relu (its range is [0, inf), which means it can blow up the activation), we got better results with, for one RNN layer, and this is the activation we are using for the next experiments with different hyperparameter configurations.

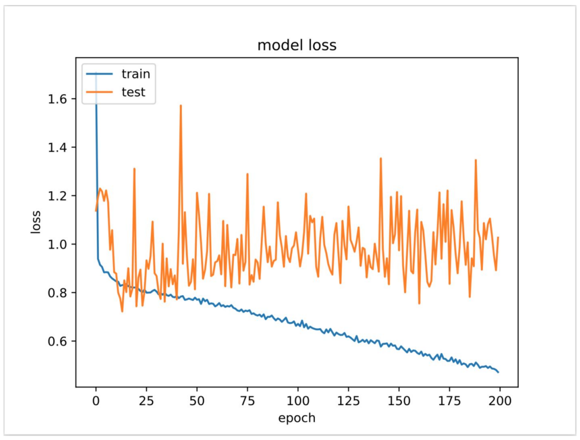
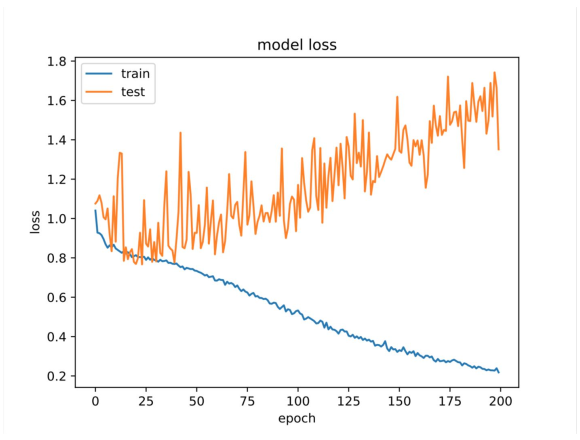
Regularization Techniques

Batch Normalization and Dropout help us get better results, as the first one lowers the weights and the second one drops a percentage of the neurons.
In the experiments we have run, normalizing after each layer has improved a little bit our results. That’s not the case for Dropout which removes the positive effect of BatchNormalization() function.

Number of layers and neurons

Increasing the number of neurons in each layer helps the network learn faster the training set, but leads to divergence between the training loss function and the test loss function (*Figure 2*). While decreasing them converges with the training function but the results are worse, as the values for the loss functions for the training set as well as for the test set are too high (*Figure 3*).

When the number of layers and neurons are very different in range, we see those problems of divergence in the plots. We need to find an equilibrium of number of layers and neurons so that the R2 result is high and the convergence of the training and test loss functions are similar.

 *Figure 2: 2 layers and 32 neurons*	 *Figure 3: 2 layers and 64 neurons*

Below (*figure 4*) the best combination of number of layers and neurons found so far, is shown:

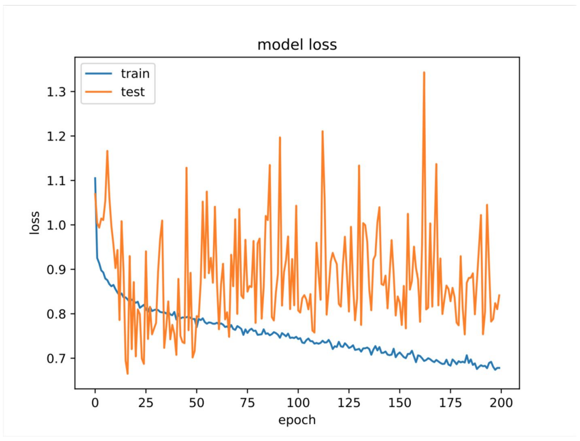


Figure 4: 2 layers and 16 neurons, adam.

Optimizer

Given the best configuration so far, nadam and rmsprop optimizers have been tested. As seen in figures 5 and 6, each optimizer seems to behave better with one activation function. For out dataset, relu works best with nadam (*figure 5*), and tanh with rmsprop (*figure 6*).

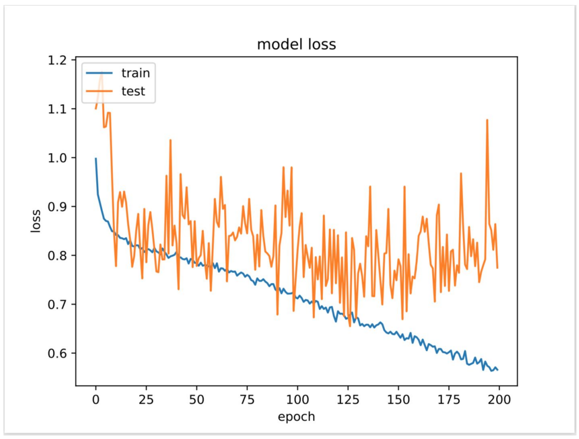


Figure 5: 2 layers, 16 neurons, relu and nadam.

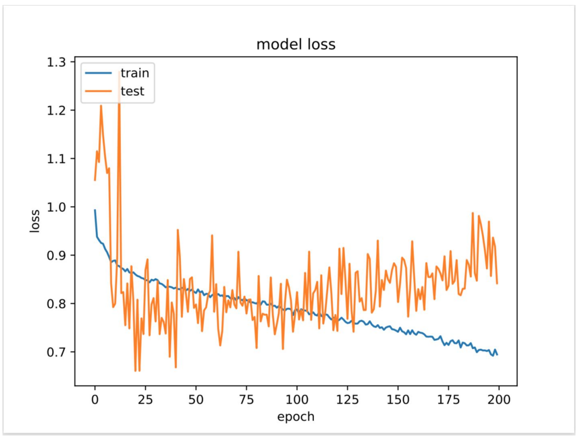


Figure 6: 2 layers, 16 neurons, tanh and rmsprop.

GRU

In our case, using GRU gives a very bad result, as the test loss function stabilizes after few epochs in high values, as shown in figure 7.

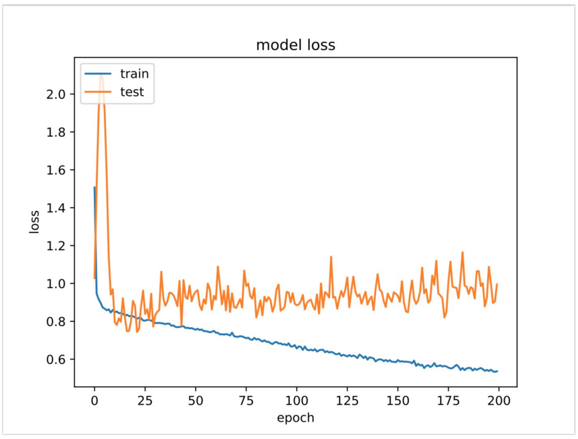


Figure 7: 2 layers, 16 neurons, GRU.

Lag

As mentioned in page 1 under the section "*Issues on the size of the dataset*", the dataset we are working with has few samples, and we use the size of the lag to create more samples. The default lag we have been using for all the experiments is 6, with 1 ahead, which will be the label for each of the samples.

The lower the lag is, the more samples of sequences will be created. The drawback is that if the lag is very low, that means that the sequences for each sample are very short and the combinations of sequences are very simple.

On the other hand, if the lag is high, we add complexity to the sequences that the network learn, which will make it robust, but the number of samples is reduced. As our case has already few samples, reducing the number of samples of combinations of sequences that we can create is not a good strategy to follow.

As shown below, in figures 8 and 9, a lag of 3 (*figure 8*) produces bad results and the values of the loss functions for train and test follow the same direction, although the values are high. For lag 9 (*figure 9*) the values for the train loss function decrease getting to low values, but the test loss function diverges from that.

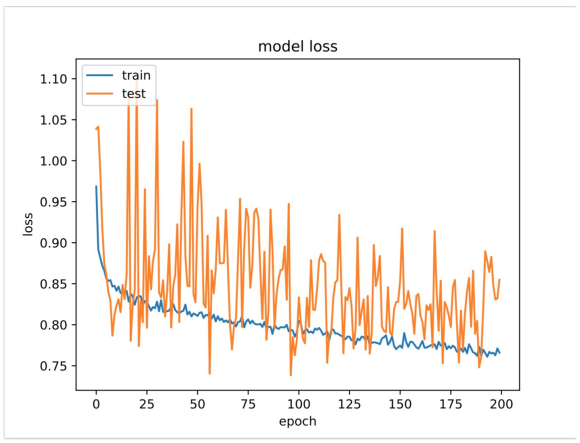


Figure 8: 2 layers, 16 neurons, relu, nadam, lag=3

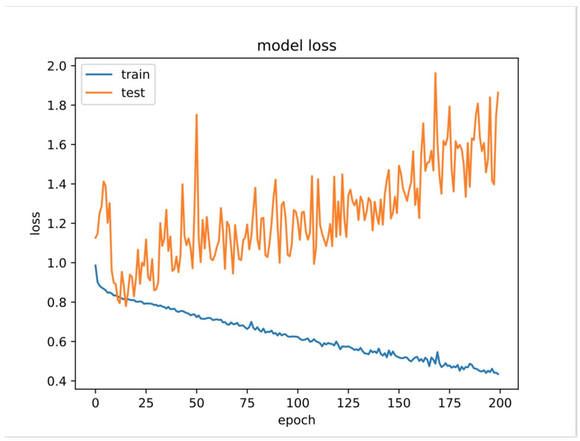


Figure 9: 2 layers, 16 neurons, relu, nadam, lag=9

Final Results

As mentioned before, the music dataset we are working with has 47 samples of 70 events. Once applied the windowing it becomes a dataset of 2944 samples of 6 events for training (plus the label) and 64 samples of 6 events for test (plus the label).

The best hyperparameter configuration (*figures 10 and 11*) found is the following:

2 layers
16 neurons
LSTM
Relu activation
Nadam optimizer
Batch size 50
Epochs 200

R2Test = 0.14544

Figure 10: Best configuration found

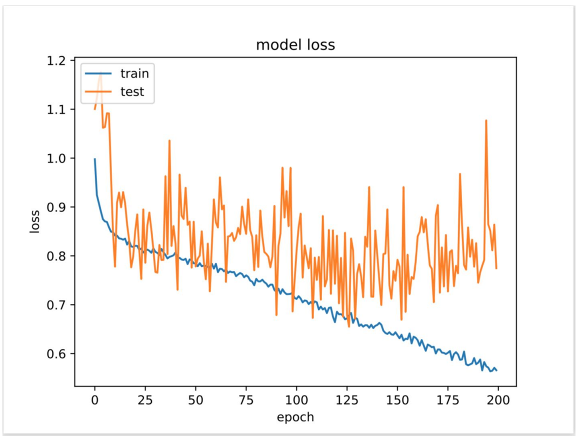


Figure 11: Plots for the best configuration found

Conclusions

It has been proven that a much more bigger dataset is needed to configure the hyper-parameters and be able to make a coherent prediction.

Future Work

The most important lesson learned with this work is that it's worth spend time analyzing and choosing a proper dataset with a huge amount of data with the higher variance possible so that the network can be properly set and capable to learn. Given the time spent preparing the data to feed the network it would be a good inversion of time.

Git Repository

In https://github.com/ovalls/mai_dl you can find the code for the RNN along with some scripts:

- `datasets/bach.csv`: original csv dataset of bach chorales.
- `create_dataset.py`: script to convert the original csv dataset into npz file.
- `datasets/bach_dataset.npz`: npz file of 47 songs x 70 events of time x 17 attributes per song.
- `music.py`: code for the configuration of the Recurrrent Neural Network.
- `config.json`: best hyperparameter configuration found.