Music Transcription Using Recurrent Neural Networks

Sara Hahner

March 29, 2019

MusicNet is a dataset of labeled classical music [THK17]. The authors of [THFK18] used a translation-invariant feed-forward neural network to recognize successfully the music notes played in a sequence of music from MusicNet. The translation invariance in the freuqency domain works very well for music transcription, since in classical music the pitch differences of the played notes depend highly on harmonic rules [Hin40]. But not only the notes played at the same time depend on harmonic rules, also the melody over time. Because of those strong temporal dependences, the neural network will be expanded to a recurrent neural network.

1 The Dataset: MusicNet

MusicNet was introduced in 2017 as the first publicly available collection of labeled classical music [THK17]. It consists of 330 music recordings of solo and chamber music pieces. The labels based on digital MIDI scores consist of the note (or notes) and the instrument by which it is played at a specific time. The dataset contains $\approx$ 34 hours of labels classical music recordings.

1.1 Dataset Creation

For creating the labels an algorithm based on Dynamic Time Warping was applied to align the given audio of the performance x and the score y. The applied method minimizes a cost function C defined on the performance space between the audio from the MIDI file and a mapping Φ of the score to the performance space.

$$C(x, y) = ||\Psi(x) - \Phi(y)||$$

$\Psi = \text{Id}$ because the cost function C is defined on the performance space. The mapping Φ corresponds to generating an audio based on the scores. The audios are filtered so that only the low frequencies remain and represented in the form of log-spectrograms ([OS01, TE03, SRS03]).

The alignment which minimizes the defined cost function C can be calculated by Dynamic Time Warping and might be readjusted by hand. The authors of [THK17] estimate the labeling error rate in the MusicNet data base to be approximately 4%.

An example of a sound sample (1a) and the aligned music score (1b) is illustrated.

1.2 Data Augmentation

When training a neural network, the size of the training set is a very important factor for success. That is why in [THFK18] data augmentation is applied to every minibatch using two different methods to randomly stretch or shrink the input audio:

1. **Pitch-shift in the frequency domain**: Each data point in a minibatch is randomly shifted by an integral number of semitones (generally between 0 and ± 5 semitones) in order to reinforce the architectural structure of the translation-invariant network.

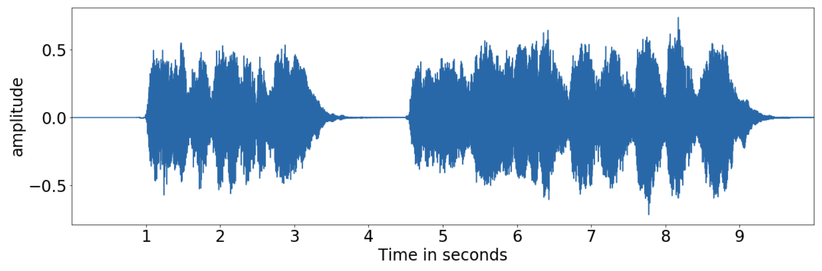
   A pitch shift of i.e. $+5$ semitones corresponds to a multiplication in the frequency domain by $2^{5/12} = 1.33$. The realization of the shift in the time domain corresponds to playing the audio $2^{5/12}$ times faster.

   A $+5$ semitone pitch-shift in comparison to the original score is illustrated in figure 1c.

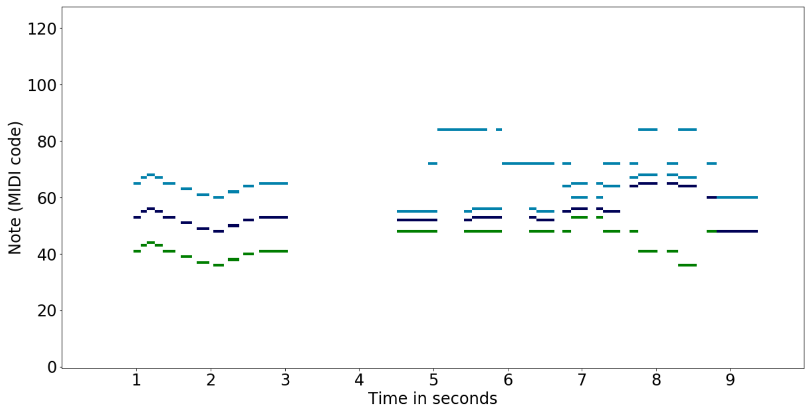
2. **Jittering**: Apply a continuous shift to each data point (generally between -0.1 and $+0.1$ semitones) that is not noticeable by human hearing. This makes the models more robust to tuning variation between recordings and at the same time works for regularization.

1.3 Normalization

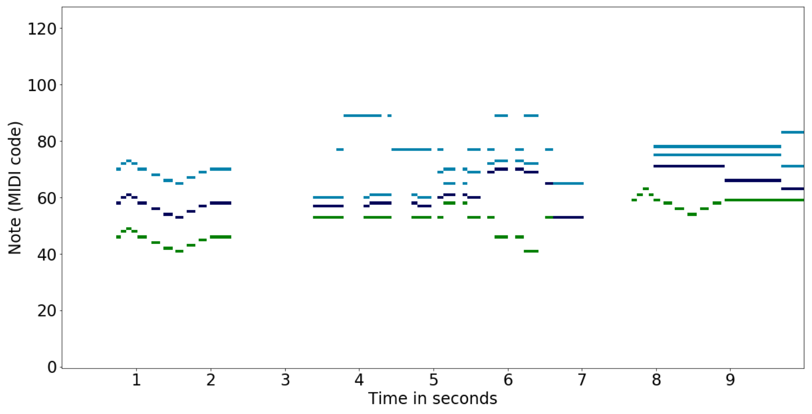
The amplitude of the music segments, which are given into the neural network, is the volume of the music. The author preprocessed each window by normalizing the norm of segment to one, $x \mapsto x/||x||_2$. This "can be interpreted physically as normalizing the audible volume of each frame" ([THFK18]). The mean of the training data set has been calculated and is approximately zero (-3.5×10^{-7}).



(a) Amplitudes of the original recording



(b) Aligned music scores



(c) Music scores after applying a 5 semitone pitch shift

Figure 1: First 10 seconds of String Quartet No 11 in F minor from Beethoven, first movement Allegro con brio (recording-ID 2494)

2 The Task: Music Transcription

Having the labels assigned to the sample set of performances the learning problem considered in [THK17] and the subsequent paper [THFK18] is the identification of notes in a segment of an audio. This multi-label classification problem is defined as follows: To every segment $x \in \mathcal{X}$ of an audio is assigned a binary label vector $y \in \mathcal{H} = \{0, 1\}^{128}$. Every dimension corresponds to a frequency value of a note and $y_n = 1$ if and only if the note n is present at the midpoint of x. This defines the feature map $f : \mathcal{X} \to \mathcal{H}$ and a multivariant linear regression to predict the label $f(x)$ minimizing square loss is trained.

2.1 The input

For the task of learning music, the window size, that is the length of the segment of music considered at one time, is a very sensitive parameter. Too short segments lead to losing relevant information and too long segments lead to a loss of temporal resolution. The advantage of a neural network is that the first layer weights the parts of the analyzed segment. A large window size for the input can be chosen and automatically the outer weights should have lower values, because the model learns that this information is irrelevant.

In [THK17] and [THFK18] the best results were obtained with a window size of 16,384 samples (when using sample rate 44,100 Hz this corresponds to a music sequence of ≈ 0.37 seconds).

In this work the extended test set as defined in [THFK18] has always been utilized for training and testing.

2.2 Neural Networks as Solutions

The learning algorithms utilized by the authors in [THK17] to learn the labels are

- a mutli-layer perceptron in form of a two-layer network with features $f_i(x) = \log(1 + max(0, \mathbf{w}_i^{\mathrm{T}} x))$,
- (Log-)spectograms, and
- a Convolutional Network that regresses against features of a collection of shifted segments x_l near to the original segment x.

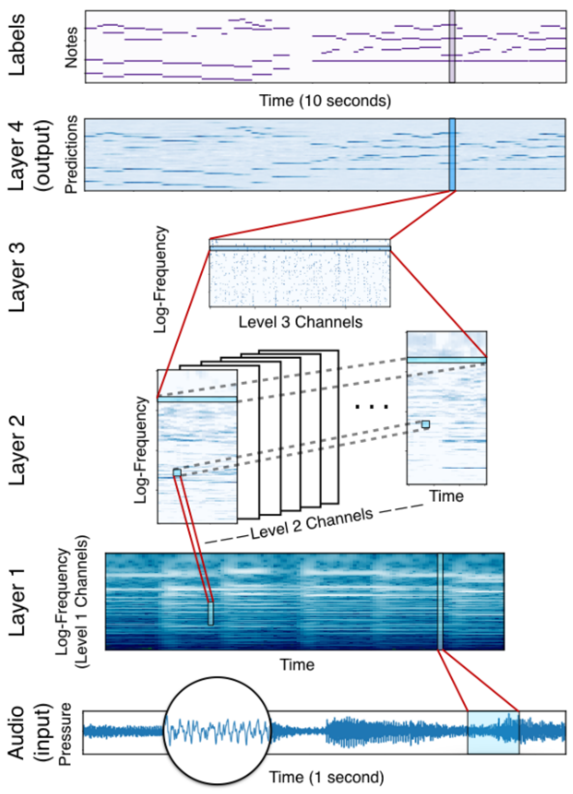


Figure 2: The translation-invariant network for note classification. Figure from [THFK18]

With a Convolutional Network the authors of [THK17] obtained the best results with an average precision of 67.7%.

In the subsequent paper [THFK18] a three-layer translation-invariant network was developed which reaches an average precision of 79.9% (table 1). Also, the table lists the results the authors of [THFK18] obtained, when applying the popular commercial software Melodny [Cel] on the MusicNet dataset. Their developed algorithm outperforms the commercial solution.

The three-layer translation-invariant network ([THFK18]) is illustrated in figure 2 and is explained in detail since the recurrent cell will be incorporated to a very similar version of the neural network.

1. The first Layer receives an input (segment) with the window length 16,384 (sample rate: 44,100 Hz).

   (a) A strided convolution with a 4,096-sample receptive field and a 512 sample stride is computed over the time dimension. Since every segment has 16,384 samples there are 25 regions of size 4,096.

   (b) For each region of the convolution a filterbank representation is computed. As a filter a log-spaced filterbank is chosen (512 sine and cosine filters with logarithmically spaced frequencies from 50 Hz to 6,000 Hz) because of the human ear's logarithmic perception of frequency. It is going to be applied to all the regions and the output can be interpreted as a frequency-ordered spectrogram.

2. The second layer consists of learned translation-invariant filters. A convolution is applied along the log-frequency axis opening up a third channel dimension, see figure 2. Since the output of layer one is frequency-ordered, this layer can learn patterns that are invariant to translations in frequency, for example triad chords. This pattern is preserved under linear translations in log-frequency space.

3. The third layer is a convolution again along the log-frequency axis. Filters of height 1 are used that fully connect along the time and channel axes of Layer 2.

4. The output layer predicts the labels by linear classification.

2.3 Training and Output

To the output of the neural network is applied a threshold to obtain the labels in $\{0, 1\}$, which the authors of [THK17] and [THFK18] have chosen to be 0.4.

As the Optimizer the authors have used the Momentum Algorithm with momentum=0.95. During learning they applied no regularization, because it impaired the performance and, additionally, when learning on the full MusicNet dataset, they did not observe overfitting. Also the final network weights are a moving average of the training weights with a decay factor of 2×10^{-4} ([THK17], [THFK18]).

Algorithm	Avg. Prec.	Acc.	Err.	Reference
Melodyne	57.9%	.395	.744	[Cel], [THFK18]
Translation-invariant	79.9%	.599	.423	[THFK18]
Translation-invariant	78.1%	.583	.427	Replica
Translation-invariant	78.5%	.589	.424	Regularization
Translation-invariant	76.9%	.566	.452	Sigmoid
Translation-invariant	71.1%	.512	.512	Sigmoid-Reg.
Bidirectional RNN	64.0%	.433	.589	Best RNN

Table 1: Test results from the authors of [THFK18] and my replicas. For the average Precision scikit-learn version 0.19.1 was used. Accuracy and Error are computed by mireval [RMH$^+$14] assuming a global prediction threshold of 0.4.

3 Replica of the Three-Layer Translation-Invariant Network

The three-layer translation-invariant network has been implemented using exactly the same architectures in the three convolutional layers and the output layer. I have not used the pretrained weights from the implementation from the authors of [THFK18] but initialized all the weights with the glorot initializer. The authors of [THK17] state to have applied a factor of .001 to all the initialized weights, which I have not implemented. The lerning rate at the beginning is .00001, the decay rate after 10,000 iterations is 0.95.

Utilizing the same training and test data set the results of [THFK18] have been reproduced obtaining similar results on precision (see tables 1 and 2) after 300,000 iterations of training on a batch of size 150. Additionally, figure 6 visualizes the output scores for a sample from the test set and colors the false alarm and false negative results. The chosen learning rate is illustrated in figure 7, as well as the development of the precision, mean squared error and the norm of the weights.

3.1 Regularization

The data augmentation steps explained in section 1.2 already have a strong regularization effect on the weights.

Nevertheless, since I observed an increasing norm of the weights at the output layer (see figure 7c), I applied L^2 parameter norm penalty to all of the trained weights (see table 2). The regularization term

$$\frac{1}{2}\sum_i \|w^{(i)}\|_2^2 \ ,$$

Regularization β	Output Function	Avg. Prec.	Acc.	Err.	MSE	Reference
0	id	78.1%	.583	.427	53.2	Replica
0.01	id	78.5%	.589	.424	52.7	Regularization
0	sigmoid	76.9%	.566	.452	52.1	Sigmoid
0.01	sigmoid	71.1%	.512	.512	60.0	Sigmoid-Reg.

Table 2: Test results of different training modes and output functions of the three-layer translation-invariant network, including the mean squared error. Calculation of statistics as described in table 1.

with $w^{(i)}$ being the trainable weights from layer i, of the neural network, is weighted by some factor $\beta \in [0, \infty)$ and added to the mean squared error of the output before the training, see chapter 7.1.1 from [GBC16].

The test results (see table 2) are very similar, but the norm of the weights is now increasing (figure 9).

3.2 Activation Function of Output Layer

The targets of all the samples are binary 128-dimensional label vectors $y \in \mathcal{H} = \{0,1\}^{128}$. The authors of [THFK18] used as the activation function of the output layer the identity, which implies that the outputs of the neural network not necessarily are in the interval $[0,1]$.

When looking at the outputs of the test set when applying the learned translation-invariant neural network, 46.9% of the output values are not in $[0,1]$, but more than 99.7% of the output values are in $[-0.1,1]$. Figure 8 illustrates the distribution of the output values.

Outliers have a strong influence on the mean squared error, which is minimized during training, although they might be classified correctly after applying the threshold during the labelling. That's why the sigmoid function instead of the identity as output function was tested.

A version without regularization and another with regularization (L^2 parameter norm penalty) were trained for 300,000 iterations. The learning rate at the beginning had to be changed to 0.005, the decay rate after 10,000 iterations stays 0.95. The results are listed in the table 2.

When not applying a regularization the mean squared error of the results of the test set is smaller, which could be because the output is now limited to $[0,1]$. Nevertheless, the average precision is slightly lower than in the original version.

When applying regularization to the loss, the average precision and accuracy are more than 5% lower.

4 Recurrent Neural Network

The harmony of more than one note sounding at the same time is being perceived as part of the translation-invariant convolution along the log-frequency axis in layer two of the network from [THFK18].

Nevertheless, harmony does not only influence the combination of notes played at one time. Also, the difference to the following pitch (or pitches) and the development of a melody are strongly effected by harmonic rules. That means the notes played at a specific moment in a harmonic music piece depend very much on the notes that have been played right before or will be played right after.

At first, big intervals of one instrument are not very common. Actually ,it is quite probably that the same note is played more than once or for a long time. Also some intervals are more typical (i.e. a minor or major third and quint) than other intervals (i.e. a Tritone), because to most human ears they sound harmonic ([Hin40]). Finally, in a longer time frame, a melodic passage, typical chord sequences can be observed.

When looking at the typical pitch differences in the MusicNet dataset in between two sequences of 0.37 seconds (see figure 3 and table 3), it can be observed, that the statements from above apply. The most common pitch difference is 0, which corresponds to playing the same note again. The second most common interval is the octave, followed by the harmonic intervals. Pitch differences above 60 semitones appear with a relative frequency below 0.001%.

Because of the strong temporal dependence of music notes from a harmonic music piece a recurrent neural network, which can incorporate information from nearby timesteps (chapter 10.3, [GBC16]), will be tested on the MusicNet dataset.

4.1 The Arquitecture

A bidirectional recurrent neural network is chosen, since the notes played before as well as after the analyzed sequence have an influence and the music transcription is done offline. Figure 5 illustrates how the information from both the past and the future inputs to the output $o^{(t)}$ at timestep t .

To be able to train the recurrent neural network while maintaining a window size of approximately 0.37 seconds the whole music net dataset has been downsampled from 44,100 Hz to 11,025 Hz. That leads to a reduction of size in all layers of the neural network. The reduction, as illustrated in figure 4, has been chosen.

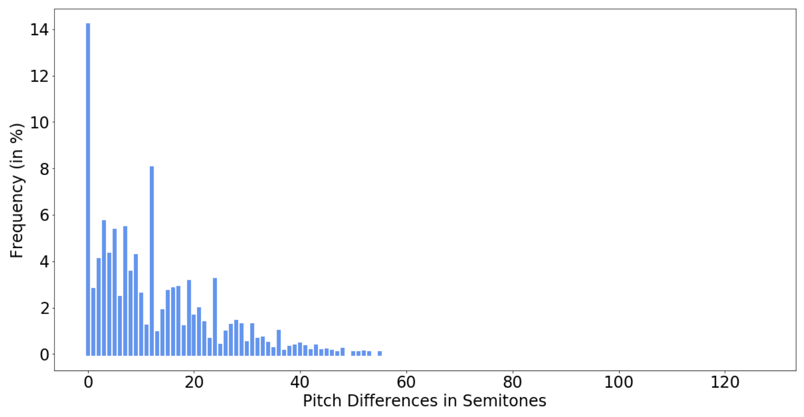
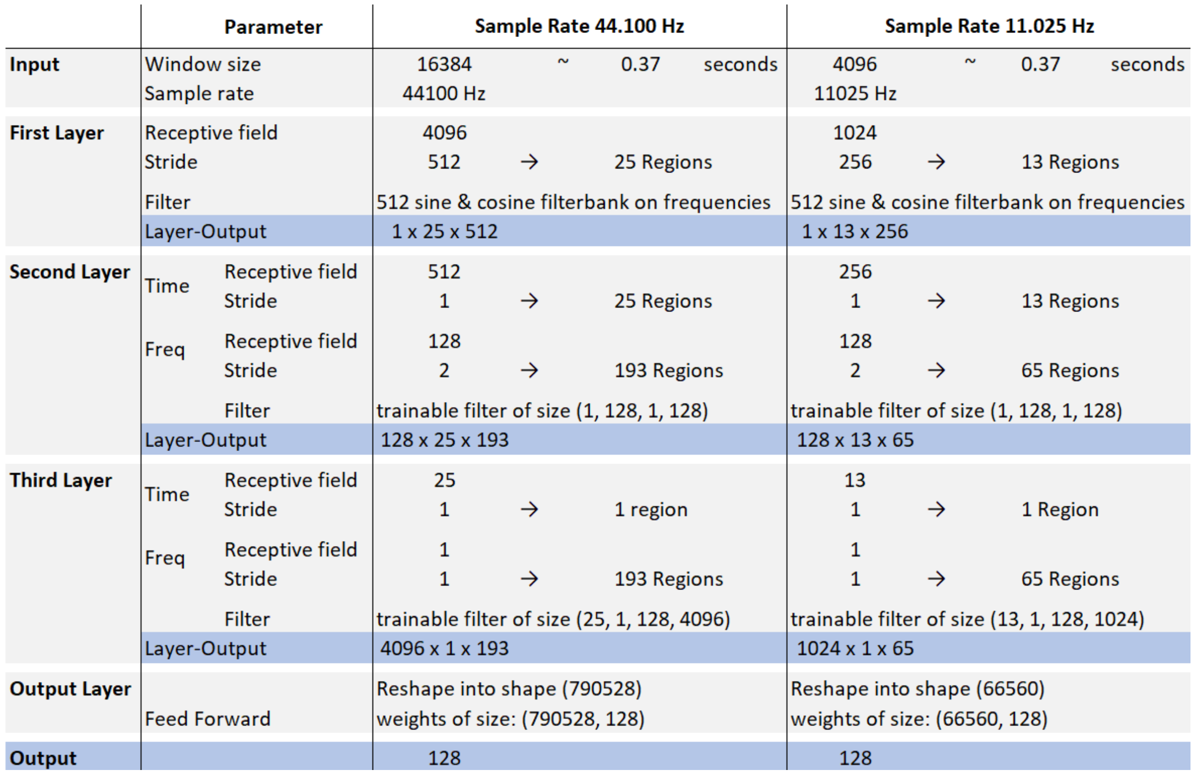


Figure 3: Relative Frequency of pitch differences after 0.37 seconds in MusicNet.

Semitones	Chord	Relative Frequency of Interval
0	Perfect unison	14.16%
12	Perfect octave	8.0%
3	Minor third	5.68%
7	Perfect fifth	5.41%
5	Perfect fourth	5.32%
4	Major third	4.28%
9	Major sixth	4.23%
2	Major second	4.06%
8	Minor sixth	3.49%
24	Double octave	3.18%
19		3.11%
17		2.84%
16		2.79%
1	Minor second	2.77%
15		2.67%
10	Minor seventh	2.56%
6	Tritone	2.41%
21		1.93%
14		1.83%
20		1.62%
28		1.39%
22		1.31%
29		1.23%
31		1.23%
27		1.22%
11	Major seventh	1.19%
18		1.16%

Table 3: Relative Frequency of most common pitch differences after 0.37 seconds in MusicNet.



		Parameter	Sample Rate 44.100 Hz	Sample Rate 11.025 Hz
Input		Window size	16384 ~ 0.37 seconds	4096 ~ 0.37 seconds
		Sample rate	44100 Hz	11025 Hz
First Layer		Receptive field	4096	1024
		Stride	512 → 25 Regions	256 → 13 Regions
		Filter	512 sine & cosine filterbank on frequencies	512 sine & cosine filterbank on frequencies
		Layer-Output	1 x 25 x 512	1 x 13 x 256
Second Layer	Time	Receptive field	512	256
		Stride	1 → 25 Regions	1 → 13 Regions
	Freq	Receptive field	128	128
		Stride	2 → 193 Regions	2 → 65 Regions
		Filter	trainable filter of size (1, 128, 1, 128)	trainable filter of size (1, 128, 1, 128)
		Layer-Output	128 x 25 x 193	128 x 13 x 65
Third Layer	Time	Receptive field	25	13
		Stride	1 → 1 region	1 → 1 Region
	Freq	Receptive field	1	1
		Stride	1 → 193 Regions	1 → 65 Regions
		Filter	trainable filter of size (25, 1, 128, 4096)	trainable filter of size (13, 1, 128, 1024)
		Layer-Output	4096 x 1 x 193	1024 x 1 x 65
Output Layer		Feed Forward	Reshape into shape (790528) weights of size: (790528, 128)	Reshape into shape (66560) weights of size: (66560, 128)
Output			128	128

Figure 4: Illustration of the chosen size reductions in all layers of the neural network when applying downsampling from 44,100 Hz to 11,025 Hz.

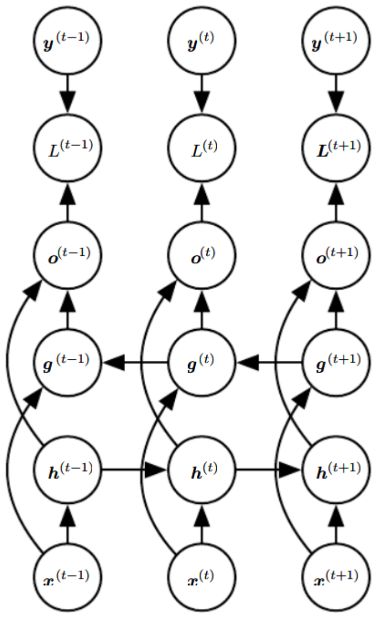


Figure 5: Arquitecture of the utilized bidirectional recurrent neural network: Given input $x^{(t)}$, the value $o^{(t)}$ of the output unit is calculated considering the information from timestep $t-1$, namely $h^{(t-1)}$, and the information from the next timestep $g^{(t-1)}$. The loss $L^{(t)}$ is calculated comparing $o^{(t)}$ to the target $y^{(t)}$. Figure from [GBC16]

A Long-Short-Term-Memory cell (implementation: tf.nn.rnn_cell.LSTMCell) will be integrated after the three-layer translation-invariant neural network and we allow the RNN to incorporate $(s_{time} - 1)/2$ timesteps before and $(s_{time} - 1)/2$ timesteps after the analyzed sequence. m is the number hidden units of the LSTM-cell. The dynamic version of bidirectional recurrent neural network (tf.nn.bidirectional_dynamic_rnn) has been chosen.

4.1.1 Normalization

The author normalized the norm of every segment to 1, as explained in section 1.3. This cannot directly be applied when looking at 3 or 5 windows at one time. The solutions I have chosen is normalizing the norm of every window by itself to one.

5 Results of the RNN-Network

The runtime of the training of the recurrent neural network is a lot higher than the runtime of the feed-forward network. Also when training the complete recurrent network (using Momentum Optimizer or Adam Optimizer) the algorithm has only found the local minimum projecting all the labels to 0. Therefore, the feed-forward network before the LSTM-cell has been pretrained. Different combinations of the parameters characterizing the LSTM-Cell have been chosen and the results are summarized in table 4.

5.1 Pretrain Feed-Forward Network

To speed up the training of the recurrent neural network, the feed-forward network was pretrained on the 11,025Hz Dataset for 150,000 iterations. Starting with a learning rate of .00001, every 10,000 iterations steps a learning rate decay of 0.95 was applied. The resulting average precision of 76.1% is comparable to the results from the 44,100Hz dataset but the runtime for training 1000 iterations reduces from 500 seconds to 75.

5.2 Training the LSTM-Cell

The results of training of the LSTM-Cell depend highly on the chosen hyperparameters. Not only the hyperparameters number of units m and timesteps s_{time} of the LSTM-cell were evaluated, but also two different optimizers were chosen and regularization was tried. The learning rate needed to be higher

Timesteps s_{time}	Units m	Opt.	Avg. Prec. Train	Test	Acc.	Err.	Runtime for 1000 iter.
1	-	MomOpt	79.8%	76.1%	.559	.463	75 sec
3	128	MomOpt	58.9%	57.6%	.384	.655	135 sec
3	128	AdamOpt	58.3%	58.0%	.390	.640	138 sec
3	256	MomOpt	59.2%	57.7%	.393	.636	140 sec
9	1024	MomOpt	67.8%	60.6%	.401	.626	235 sec
9	1024	AdamOpt	70.3%	64.0%	.433	.589	360 sec
9	1024	AdamOpt*	70.2%	63.7%	.436	.592	350 sec
15	2048	AdamOpt*	78.7%	63.9%	.433	.589	700 sec

Table 4: Results for different variable choices for the recurrent neural network. (* indicates the implementation of L^2 parameter norm penalty with $\beta_{reg} = 0.01$)

than when training the feed-forward network. In the beginning it was set to .001.

The results are shown in table 4. It can be observed, that the average precision when training the recurrent neural network is lower than the average precision of the feed-forward network.

The best results were obtained when setting the number of timesteps s_{time} to 9 and the number of units m to 1024. The Optimizer AdamOpt gave better results than the Momentum Optimizer. The results highly tend to overfitting. When increasing s_{time} and m even more, only the traning precision rises. The model capacity of the neural network is so high, that it memorizes the small dataset. Also the application of regularization cannot prevent that.

6 Conclusions

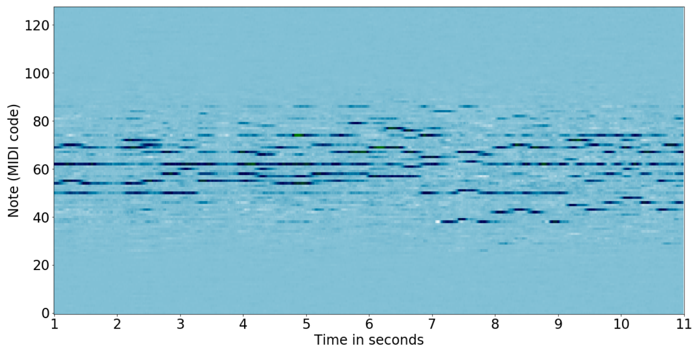
As visualized in figure 6 the translation invariant feed-forward neural network can predict very well the pitches of the different music notes but struggles with the recognition of the notes' beginning and ending. Especially when the same note is being repeated twice the neural network does not separate the different notes.

The chosen recurrent neural network fails to solve the task of recognition of the rhythm. The bidirectional recurrent neural network detects the pitches right, but generally recognizes less notes than the feed-forward network (figure 11). Especially when insecurities occur in nearby windows, they reinforce each other and nothing is detected. Additionally, when increasing the model capacity only the training precision rises, the model overfits.

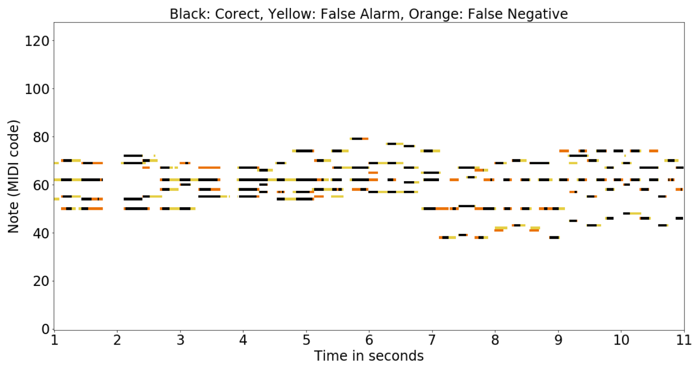
In conclusion, the feed-forward translation invariant neural network can outperform commercial solutions [Cel] by more than 20% of average precision. 78% of the music notes are approximately predicted correctly (see table 1 for a comparison of the results). Also the neural network works well and a lot faster on music recordings with smaller sample rate. Sadly, it struggles with the recognition of the rhythmical elements. Therefore, it should be tested to include a method to track the beat. Recurrent networks could solve this task, but given their high model capacity and the respectively small size of the dataset MusicNet, a simpler method possibly based on time series analysis could be tested.

7 Plots and Graphics of the Results

Figure 6, 7 and 8 show the results of my replica from the authors translation-invariant neural network ([THFK18]). Figure 9 illustrates the influence of regularization on the norm of the weights when utilizing the feed-forward version. Finally, figure 10 gives some statistics to the results and the training of the bidirectional recurrent neural network and figure 11 visualizes the predicted scores.

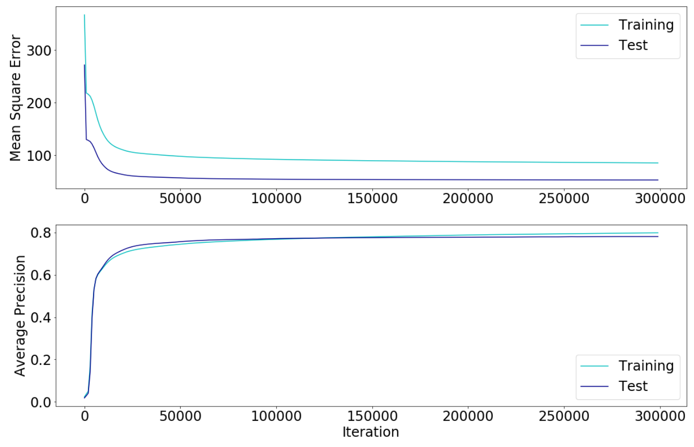


(a) Visualization of the output of the replica of the translation-invariant neural network.

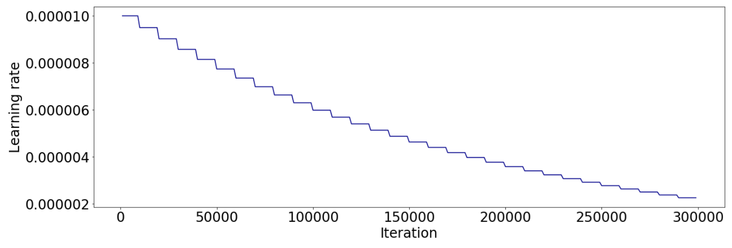


(b) Comparison of the original scores and the prediction of the neural network after applying a threshold of 0.4.

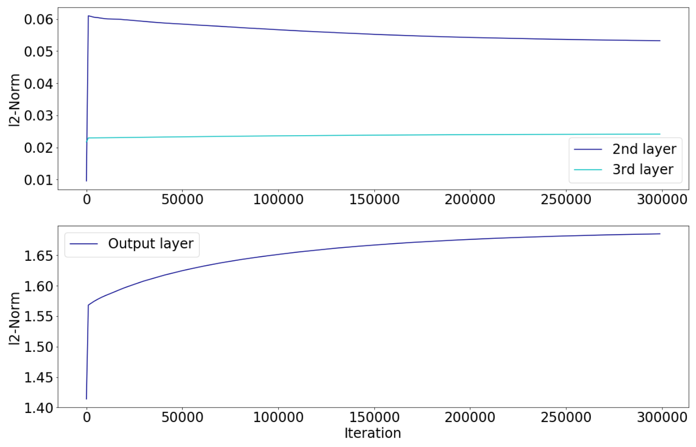
Figure 6: Translation-Invariant Neural Network: Original and prediction of the scores in the seconds 1 to 11 of the recording from third movement (Scherzo) of the Violin Sonata No. 10 in G major from Beethoven (Recording ID: 2628)



(a) Average Precision and Mean Squared Error.



(b) Learning Rate



(c) Norm of the weights of second, third and output layer.

Figure 7: Statistics from the training of my replica of the three-layer translation-invariant network.

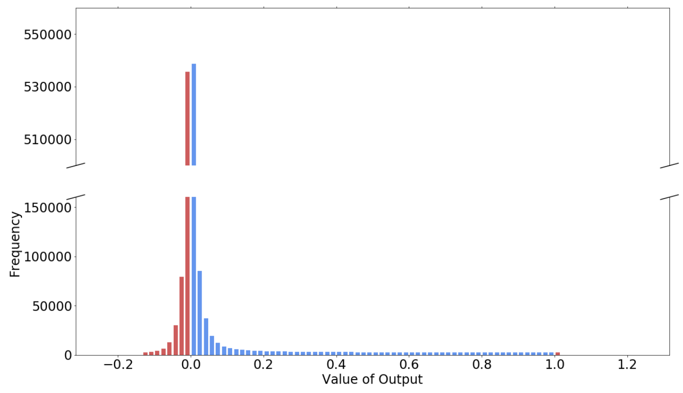


Figure 8: Histogram showing the distribution of the output values when applying my replica of the translation-invariant neural network to test set.

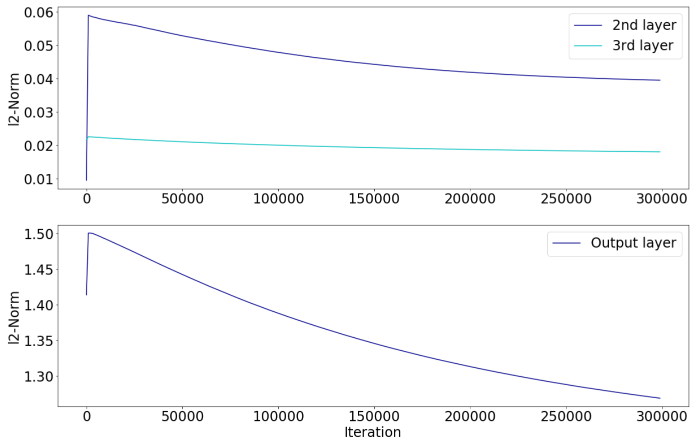
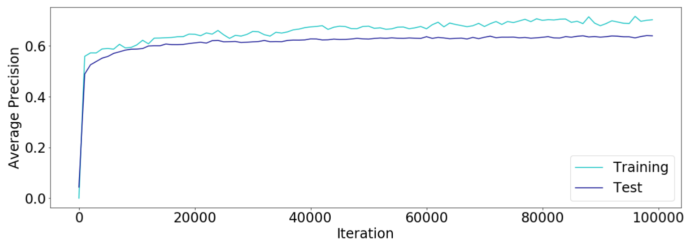
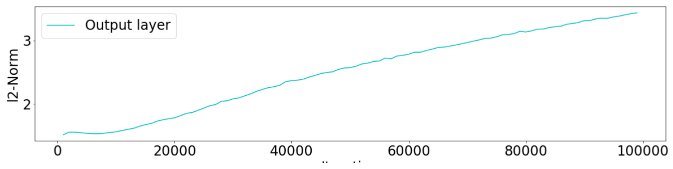


Figure 9: Translation-Invariant Neural Network. Norm of the weights of second, third and output layer when applying L^2 parameter norm penalty.



(a) Average Precision



(b) Norm of the weights of the output layer

Figure 10: Test statistics from the training of the translation-invariant neural network in the bidirectional recurrent version. $s_{time} = 9$ and $m = 1024$, trained without L^2 parameter norm penalty.

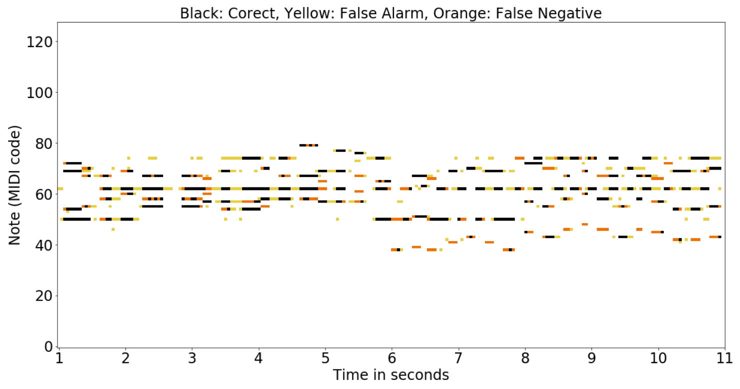


Figure 11: Recurrent Neuronal Network: Comparison of original/predicted score for recording No. 2628

References

[Cel] Celemony. Melodyne. `http://www.celemony.com/en/melodyne/what-is-melodyne`.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[Hin40] P. Hindemith. *Unterweisung im Tonsatz*. Number Bd. 1 in Edition Schott. B. Schott's Söhne, 1940.

[OS01] Nicola Orio and Diemo Schwarz. Alignment of monophonic and polyphonic music to a score. In *ICMC*. Michigan Publishing, 2001.

[RMH$^+$14] Colin Raffel, Brian Mcfee, Eric J. Humphrey, Justin Salamon, Oriol Nieto, Dawen Liang, Daniel P. W. Ellis, C Colin Raffel, Brian Mcfee, and Eric J. Humphrey. mir_eval: a transparent implementation of common mir metrics. In *In Proceedings of the 15th International Society for Music Information Retrieval Conference, ISMIR*, 2014.

[SRS03] Ferréol Soulez, Xavier Rodet, and Diemo Schwarz. Improving polyphonic and poly-instrumental music to score alignment. In *International Conference on Music Information Retrieval (ISMIR)*, page 6, Baltimore, United States, October 2003. 6pp.

[TE03] Robert J. Turetsky and Daniel P. W. Ellis. Ground-truth transcriptions of real music from force-aligned midi syntheses. In *ISMIR*, 2003.

[THFK18] John Thickstun, Zaid Harchaoui, Dean P. Foster, and Sham M. Kakade. Invariances and data augmentation for supervised music transcription. In *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2018.

[THK17] John Thickstun, Zaid Harchaoui, and Sham M. Kakade. Learning features of music from scratch. In *International Conference on Learning Representations (ICLR)*, 2017.