TOWARDS END-TO-END POLYPHONIC MUSIC TRANSCRIPTION: TRANSFORMING MUSIC AUDIO DIRECTLY TO A SCORE

Ralf Gunter Correa Carvalho[♯], *Paris Smaragdis*[♯,♭]

[♯]University of Illinois at Urbana-Champaign, [♭]Adobe Research

ABSTRACT

We present a neural network model that learns to produce music scores directly from audio signals. Instead of employing commonplace processing steps, such as frequency transform front-ends, harmonicity and scale priors, or temporal pitch smoothing, we show that a neural network can learn such steps on its own when presented with the appropriate training data. We show how such a network can perform monophonic transcription with very high accuracy, and how it also generalizes well to transcribing polyphonic music.

Index Terms— Music transcription, deep learning, end-to-end systems, seq2seq

1. INTRODUCTION

Automatic music transcription has been a long standing problem in audio analysis. Since the earliest attempts more than years ago [1], we have seen a large variety of approaches employing a wide set of technical tools [2, 3, 4, 5, 6, 7, 8, 9, 10]

One potential point of contention in this pursuit, is the amount of user-guidance in the design of such systems. Quite often, algorithm designers make assumptions which, although sensible, are not necessarily optimal. For example, it is customary to use a short-time Fourier transform front-end (or equivalents), to assume certain pitch quantizations corresponding to musical scales, to expect harmonicity in sounds, to apply some smoothing in the resulting pitch estimates, etc. In this paper we consider the design of a system that is largely free from any such assumptions, and optimally learns all the necessary steps to perform transcription by being only presented with pairs of audio waveforms and their corresponding music scores (in text form). We show that doing so results in a system that automatically learns a harmonic-like front-end, and other common music transcription structures on its own. Because this is done in the framework of an adaptive system, these structures are adapt to being optimal for this task, thereby putting to end questions such as what is the optimal front-end for transcription, the setting of various parameters, etc.

This system is designed using a deep learning architecture. It is based on a seq2seq model [11] preceded by a convolutional layer [12] that acts as a front-end. The system is presented with windows of raw waveform data, and for each window it produces as an output a 1-hot encoding of a music score using the Lilipond language [13].

A key advantage of this approach is the explicit avoidance of forced intermediate parametric representations (such as frequency decompositions or piano roll representations), which can often complicate the translation to a musical score and can be the source of additional errors and ambiguity.

This work was supported by NSF grants #1319708 and #1453104

2. TRANSCRIPTION AS SEQUENCE TO SEQUENCE TRANSLATION

One of the great advantages of deep learning methods is the ability to easily design "end-to-end" systems. These are systems that do not use explicit intermediate representations (e.g. features, or explicit latent variables), and can instead accept unprocessed input data in the format that they are collected, and produce an output in the desired final form. In the context of music transcription that would imply receiving an audio waveform and directly producing a music score. In this section we will describe a network architecture that is able to do so.

2.1. Representing the data

Before continuing with the description of the proposed network, let us discuss how we will be representing the input and output data. The input audio data will be represented using one-second long real-valued sequences with 8192 samples, scaled between -1 and 1. For now, we will assume that each such window will correspond to one bar of music, and for the sake of simplicity we assume $\frac{4}{4}$ measures. To start with, we will consider only piano notes, but later on we will present results on additional instruments as well. The reason why we start with piano, is that it is an instrument that allows for a relatively strict 1-to-1 mapping between sound and score. Bowed and wind instruments can introduce pitch embellishments which are not represented in the score and thereby present an ambiguity in the training data. It is highly likely that with enough training data, the necessary invariances to these embellishments can be learned, but that's an issue that outside of the introductory scope of this paper.

For the output data, we want to obtain a direct representation of a score, as opposed to a set of pitch probabilities, or some intermediate representation that does not have a deterministic mapping to a score. In order to explicitly represent an output score we use the Lilypond music notation language [13], which is a text-based computer language that is used to generate music scores. The way that we will represent the output text will be using a *1-hot encoding*, i.e. a sequence of sparse vectors, of which each element corresponds to a unique character, and where the only non-zero element will be indicating which character is to be active at any point in time. For an brief illustration of the Lilypond language and the subsequent encoding see figure 1.

2.2. Network architecture

Having defined the format of the data we will use, we will now describe the structure of a deep network that is capable of accepting waveform inputs and producing a 1-hot text output representing a score. We split the representation into two parts; first the part that will be equivalent to a front-end, and then the part that is responsible

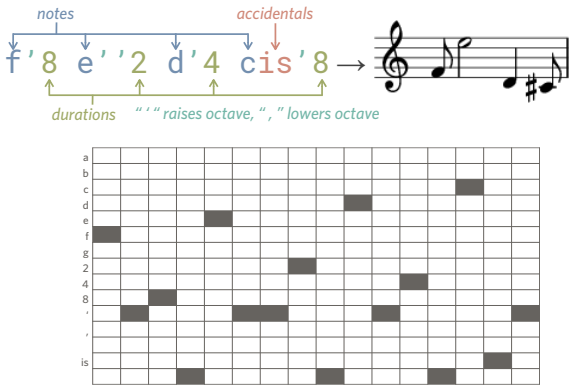

Figure 1: An example of the output representation. On the top right we see the score we wish to express, in the top left we see its description using the Lilypond language, and on the bottom we see the 1-hot encoding of that text description. Our approach produces the latter representation which deterministically maps to a score.

for translating the latent representation of the input sequence to a Lilypond text sequence.

2.2.1. *Adaptive front-end*

Traditionally, music transcription systems start with some form of a harmonic transform. This is a sensible type of front-end since the main feature that informs transcription is pitch, and pitch can be best detected with a frequency-type decomposition. We will start with a similar step, but we will not assume a particular type of transform. We will instead use an adaptive filterbank, or in the deep learning parlance, a convolutional layer. As is traditional, we also subsequently apply an activation function (a ReLU [14]), which acts as a kind of rectifier. We therefore start by transforming the input sequence using:

$$c[f,t] = \max\left(\sum_{k}^{L} x[t-k]q[f,k], 0\right), f \in \{1,\ldots,F\} \quad (1)$$

where $x[t]$ is the input sequence as described above, and $q[f,k]$ is a set of F filters of length L. In this step we effectively convolve the input with this set of filters and keep the F rectified outputs $c[f,t]$. Unlike traditional transcription systems though, we will not specify the filters $q[f,t]$ ourselves, we will instead adapt them in order to optimally perform transcriptions tasks. In this work we use 128 filterbanks of size 256 samples.

2.2.2. *A sequence to sequence model*

Having expanded the input waveform to a multidimensional sequence we now turn to the problem of translating this sequence to a sequence of one of 1-hot vectors that represent the desired Lilypond score. Before we address this process we make one more observation, the input sequence is of length 8192 samples but the output sequence will only contain a handful of characters. Therefore we can safely downsample the output of the filterbank in order to reduce the computational burden of this network. We will do so using another common deep learning construct, the maxpooling process

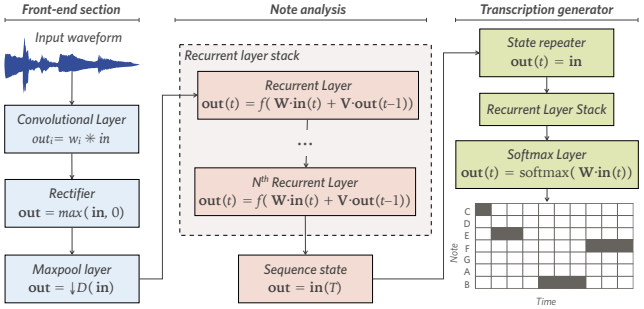

Figure 2: Flow graph of the proposed network architecture. The leftmost components represent the front-end processing, the middle components result in a state vector that represents the necessary information in the entire input waveform, and the rightmost components use that vector as a seed to synthesize the text of a score description language.

[15]. In our case, this is defined as:

$$c_m[f,t/P] = max(c[f,t],\ldots,c[f,t+P]) \quad (2)$$

Where P represents the amount of downsampling that this operation performs. For the experiments in this paper we used $P = 64$, resulting in a filterbank output with 128 time steps.

The subsequent sequence to sequence translation is not necessarily straightforward. We note that although the resulting maxpooled sequence is of fixed length, the desired output is one of variable length, meaning that we cannot use straightforward transforms to map one onto the other. Keeping with the deep learning theme, we will use the seq2seq model [11], which is able to find maps between arbitrary length sequences. The model is defined using two parts, the *encoder* and the *decoder*. The encoder is a set of Recurrent Neural Net (RNN) layers defined as:

$$\mathbf{e}_i[t] = \tanh\left(\mathbf{W}_i^{(e)} \cdot \mathbf{e}_{i-1}[t] + \mathbf{V}_i^{(e)} \cdot \mathbf{e}_i[t-1] + \mathbf{b}_i^{(e)}\right) \quad (3)$$

Where $i \in \{1,\ldots,E\}$, and $\mathbf{e}_0[t] = [c_m[1,t],\ldots,c_m[F,t]]^\top$. The parameters of this model correspond to the variables $\mathbf{W}_i^{(e)}$, $\mathbf{V}_i^{(e)}$ and $\mathbf{b}_i^{(e)}$. The (e) superscript denotes that these parameters belong to the encoder stage. The number of layers E is user-defined, and for our experiments we used $E = 2$. The input to the first layer is simply the multidimensional sequence that the maxpooling operation produces. From the final layer of the encoder we only keep the output vector at the last time step $\mathbf{e}_E[T]$. The creation of this vector is influenced by all the preceding time steps (due to the recurrent nature of the layers before it), and it can be seen as a vector that summarizes the input data.

Likewise, the decoder stage is defined as a sequence of RNNs as well:

$$\mathbf{d}_i[t] = \tanh\left(\mathbf{W}_i^{(d)} \cdot \mathbf{d}_{i-1}[t] + \mathbf{V}_i^{(d)} \cdot \mathbf{d}_i[t-1] + \mathbf{b}_i^{(d)}\right) \quad (4)$$

Where $i \in \{1,\ldots,D\}$, and $\mathbf{d}_0[t] = \mathbf{e}_E[T]$, i.e. the input is simply a repetition of the last output of the encoder stage. For the experiments in this paper we used $D = 1$. At the end of this set of layers we are presented with a new sequence $\mathbf{d}_D[t]$, which can be arbitrarily long depending on how many times we repeatedly feed $\mathbf{e}_E[T]$ as an input (something that we will specify below).

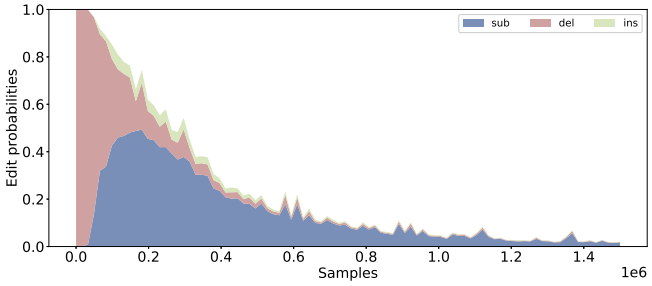

Figure 3: Training profile for a single-octave range experiment. This plot shows the probability of a substitution, insertion, or deletion of a note over adapting to increasingly more training examples.

In order to obtain the desired 1-hot representation we add one more processing step. We use a simple softmax layer on the output of the decoder stage, i.e.:

$$\mathbf{z}[t] = \text{softmax}\left(\mathbf{W} \cdot \mathbf{d}_D[t] + \mathbf{b}\right) \tag{5}$$

Where $\text{softmax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$ and $\mathbf{W}$ and $\mathbf{b}$ are the layer parameters. The softmax operation ensures that the output vectors will sum to 1, hopefully containing only one non-zero element as the desired targets do. Once the sequence of output vectors indicates a sequence of characters that implies the end of the Lilypond score (either a "}" character, or in the case of polypohonic music the sequence ">>}"), we stop feeding new inputs and halt the process. The entire network is illustrated in figure 2.

It is important to note at this point that there is no constraint that guarantees that the output will result in a valid Lilypond statement. This model will produce a sequence of characters, which might compile (or not). We do not take any extra steps to post-process the score in order to correct the syntax. In practice we obtain good enough convergence that the majority of outputs are well-formed score descriptions.

2.2.3. Training details

Given that we specified this entire process using neural network primitives, we can simply provide a set of training data composed out of waveforms and their corresponding 1-hot score representations, and thus estimate the optimal values for all the layer parameters. For training this network, we use the RMSProp algorithm [16] and also apply 25% dropout at every layer [17]. In order to avoid numerical saturation issues we also add a batch normalization layer [18] after the front-end section. As is common, instead of using RNN layers as defined above, we use LSTM layers [19] which are more amenable to efficient gradient-style training, whilst providing the conceptual processing of RNNs.

The training data was synthetically generated. We generated random melodies for which we computed both their score (as a 1-hot vector of Lilypond statements), and their corresponding waveform using a MIDI synthesizer. For evaluation we kept a set of 1024 melodies which were never used in the training data. The smallest durations used in the melodies were eighth notes. The range of notes, and the sound generation parameters were dependent on the experiment, and are described separately below.

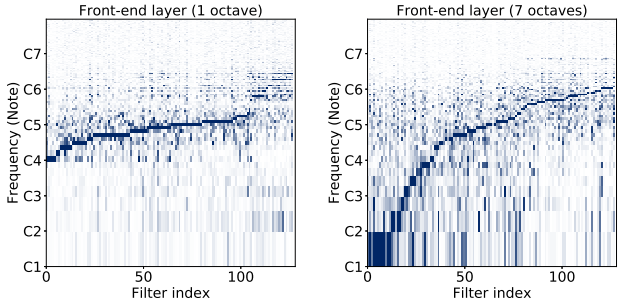

Figure 4: Front-end filter comparison between 1- and 7-octave training sets. In both plots we show the magnitude Fourier transform of each filter, ordered by dominant frequency. It is clear that both cases learn pitch-detecting filters, but the filters for the 7-octave training data (on the right) span a wider range than the 1-octave case (left).

3. EXPERIMENTS

We will now show some of the results we obtained, a small-range simple monophonic example, a large-range monophonic example, a polyphonic example, and examples that stress the ability of this algorithm to generalize. To evaluate how close the predicted score was to the real score we will present a breakdown of the Levenshtein edit distance between the two scores. We will report the probabilities of the network's prediction to substitute $P(sub)$, delete $P(del)$, or insert $P(ins)$ a note as compared to the ground truth note sequence.

3.1. Monophonic music

In order to show how this model can learn melodies, and what features it clings on we will start with a simple example in which the input contains monophonic melodies with a one-octave range, using only eighth, quarter and half notes. Using these parameters we note that the number of possible melodies is 729,601,488. Upon training on only 1,500,000 samples we see that we can achieve near-perfect transcription performance on unseen data. This is an important point to make since it highlights that despite the extremely large space of possibilities this network rapidly learns to generalize and represent musical structure. The summary of the training process on this data is shown in figure 3. Note that, initially, the network being unable to make any meaningful predictions, or to produce a syntactically correct score, exhibits a high probability of deleting notes. As the model parameters adapt, we see that it starts trading deletions with substitution mistakes, which over time are also minimized until we get good performance. Insertions are generally a rare mistake with this system.

We then repeat the same experiment with seven octaves in order to see how well this model can deal with larger note ranges. The space of melodies in this case balloons to 2,690,655,004,956,240. Training on only 1,280,000 samples, we obtain the final validation set edit probabilities of $P(sub) = 0.006$, $P(ins) = 0.001$, $P(del) = 0$, which once again imply near flawless performance. To get some sense of how this model makes mistakes, we show the note confusion matrix in figure 5. A surprising feature of this algorithm is that we don't see prominent octave confusion mistakes. An interesting comparison with the preceding example takes place in the front-end filters that are learned. In figure 4 we show the

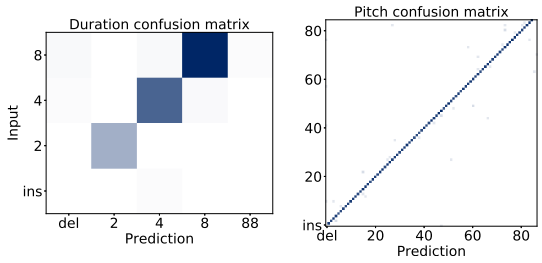

Figure 5: Confusion matrices for a single-voice monophonic piano dataset covering seven octaves. The ins and del entries denote insertions and deletions respectively. Note that there are very few pitch errors, and no consistent error patterns. Likewise we see that there are very few note duration estimation errors (an interesting error is that rarely the 8 is repeated forming a syntactically nonsensical duration of 88)

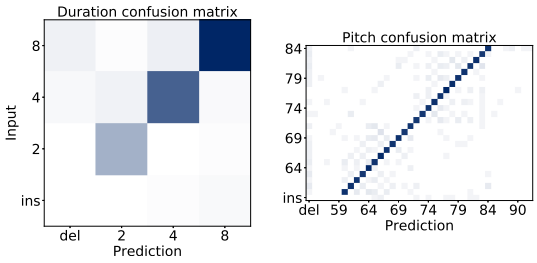

Figure 6: Confusion matrices for a polyphonic transcription example with two overlapping melodies defined over a range of two octaves. The left plots shows the duration confusions, and the right plot shows the pitch confusions.

learned filters from the one-octave example, vs. the 7-octave example. Note that in each case the dominant frequency elements of these filters predominantly lie in the range in which the training data fundamental frequencies are. Whereas the one-octave filters mostly exhibit energy in the range between C4 and C5, we see that for the 7-octave data this range is expanded. Additionally, as a stress test we added various sources of variance in the data to examine the ability of this model to deal with more uncertainty. We did so by randomly detuning the notes in the audio stream (less than half a semitone), and for each training sample we add random amounts of reverberation, and chorusing effects, as well as training on multiple piano types. We also added 50% dropout to the input, meaning that a random half of the input samples were set to zero. Compared to the 1-octave example above, the probability of edits now becomes $P(sub) = 0.024$, $P(ins) = 0.002$, $P(del) = 0.002$, i.e., slightly worse, but qualitatively the same and still very accurate. We also trained this system on melodies of different input lengths – 8192 to 10240 samples – obtaining edit probabilities of $P(sub) = 0.008$, $P(ins) = 0.001$, $P(del) = 0$, which shows the model is fully capable of handling musical performances at varying speeds. By dynamically changing the amount of maxpooling in order to produce filterbank outputs of the same length, we were also able to train on sequences of 8192 samples, and correctly recognize sequences of arbitrary sample lengths.

3.2. Polyphonic music

Of course, monophonic melodies are not a particularly difficult challenge, and a question of significant interest is whether this model can perform just as well in the case of polyphonic inputs.

In order to test for that, we setup an experiment with two simultaneous voices. This case necessitates a different format in the output. To define two voices using the Lilypond specification, the output sequence has to be comprised of two consecutive sequences, each containing the notes of each voice [13]. This makes for an interesting sequence to sequence mapping problem. Previously, there was some correspondence between the timing of the audio data and the score (early notes appear at the early parts of both the sequences). In the polyphonic case, because in we need to outline the two melodies consecutively, this direct temporal mapping isn't present. A note in the beginning of the input might influence the beginning of the output, or its middle. Thankfully, this is an issue to be resolved by the training process and does not necessitate any actions from our part, but creates a more challenging problem that sometimes necessitates larger layer sizes.

There is an additional complication when using polyphonic inputs. If we use the same instrument sound for both voices, we would introduce identifiability problems since the same input waveform could map to multiple Lilypond descriptions. For example, if we swap notes between the two voice melodies the score would stay the same but the Lilypond text file would be different. This issue can be resolved by appropriately redefining our cost function, but that is outside the scope of this paper. For now, in order to demonstrate the ability to parse polyphonic music we will use a different instrument for each voice, so that the network can use timbre to map the appropriate note to the correct voice. This will allow us to also test the ability of this method to deal with non-piano sounds. Both voices will use notes in the range of C4 to C6 and we will use a piano and a human chorus sound as the two instrument types.

The results of this experiment are shown in figure 6. We can see that they are slightly worse, but qualitatively equivalent to the previous experiments, albeit achieving convergence takes more iterations than before. The resulting network resulted in edit probabilities of $P(sub) = 0.031$, $P(del) = 0.004$, and $P(ins) = 0.001$.

4. CONCLUSIONS

In this paper we presented an approach to directly estimate a music score from a raw waveform. In doing so we presented an algorithm that appears to be quite resilient to many nuisances and does not suffer from common shortcomings of transcription systems. Additionally, this approach removes the need for intermediate representations, which can often constraint or degrade performance if they are poorly chosen. Finally, deploying this algorithm involves a minimal amount of fine-tuning, or codifying human intuition (often a source of problems!) which makes it a very attractive alternative.

We see this work as a stepping stone towards fully functional end-to-end transcription systems. The model that we proposed is still a proof of concept. It can only transcribe a single bar at a time, it has not been tested on varying time signatures, does not know about the intricacies of key signatures, how to add expressive information in scored, etc. However, we feel that these issues can we be easily addressed with further modifications and expanded training data. We believe that that such an approach can lead to significant performance advantages, in addition to offering an alternative approach to the long-standing established methods.

5. REFERENCES

[1] J. Moorer, "On the segmentation and analysis of continuous musical sound," *PhD Dissertation, Stanford University, CCRMA report STAN-M-3*, 1975.

[2] A. Klapuri and M. Davy, Eds., *Signal Processing Methods for Music Transcription*. New York: Springer-Verlag, 2006.

[3] J. Salamon, E. Gomez, D. Ellis, and G. Richard, "Melody extraction from polyphonic music signals: Approaches, applications, and challenges," *IEEE Signal Processing Magazine*, vol. 31, no. 2, pp. 118–134, mar 2014.

[4] I. Ari, U. Simsekli, A. Cemgil, and L. Akarun, "Large scale polyphonic music transcription using randomized matrix decompositions," in *Proceedings of the 20th European Signal Processing Conference (EUSIPCO)*, Aug 2012, pp. 2020–2024.

[5] J. P. Bello, L. Daudet, and M. B. Sandler, "Automatic piano transcription using frequency and time-domain information," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 14, no. 6, pp. 2242–2251, Nov. 2006.

[6] E. Benetos and S. Dixon, "A shift-invariant latent variable model for automatic music transcription," *Computer Music Journal*, vol. 36, no. 4, pp. 81–94, Winter 2012.

[7] S. Böck and M. Schedl, "Polyphonic piano note transcription with recurrent neural networks," in *IEEE International Conference on Audio, Speech, and Signal Processing*, Mar. 2012, pp. 121–124.

[8] B. Fuentes, R. Badeau, and G. Richard, "Harmonic adaptive latent component analysis of audio and application to music transcription," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 21, no. 9, pp. 1854–1866, Sept. 2013.

[9] G. Grindlay and D. Ellis, "Transcribing multi-instrument polyphonic music with hierarchical eigeninstruments," *IEEE Journal of Selected Topics in Signal Processing*, vol. 5, no. 6, pp. 1159–1169, Oct. 2011.

[10] A. Cogliati and Z. Duan, "Piano music transcription with fast convolutional sparse coding," in *Proc. IEEE International Workshop on Machine Learning for Signal Processing (MLSP)*, 2015.

[11] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[12] Y. LeCun, Y. Bengio, *et al.*, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.

[13] H.-W. Nienhuys and J. Nieuwenhuizen, "Lilypond, a system for automated music engraving," in *Proceedings of the XIV Colloquium on Musical Informatics (XIV CIM 2003)*, vol. 1, 2003.

[14] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[15] Y.-L. Boureau, J. Ponce, and Y. LeCun, "A theoretical analysis of feature pooling in visual recognition," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 111–118.

[16] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, 2012.

[17] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting." *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[18] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[19] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," *Neural computation*, vol. 12, no. 10, pp. 2451–2471, 2000.