Lyrics Recognition with Sequence Based Learning

Jesper Brunnström
brunns@kth.se

Gustav Pettersson
gupett@kth.se

Abstract

An end to end neural network approach to automatic transcription of lyrics has been implemented and evaluated. Transcription and neural network training are performed sequence by sequence by utilizing CTC learning. Training was done on both larger datasets with speech data, and smaller datasets with sung a cappella vocals. To aid in the decoding of predictions from the neural network to text an N-gram language model was created from a large set of song lyrics. The model did not manage to learn a solid enough understanding of sung vocals to generalize to unseen data.

1 Introduction

1.1 Context

Speech recognition is seeing more and more real-world usage, due to a large increase in performance in recent years. However, automatic recognition of vocals in music is a largely unexplored domain compared to speech recognition. Automatic lyric recognition is generally a harder problem, due to a significantly lower signal to noise ratio, a larger variety in inflection, and a preference for musicality over textual clarity, to mention a few things.

1.2 Previous Work

1.2.1 End-to-end Speech Recognition

A sharp increase in methods using almost exclusively neural networks as classifiers has occurred in recent years. Successful attempts at capturing the sequential nature of speech in the neural network training itself have been made.

Graves et al. [2006] presented the first method for end to end learning with recurrent neural networks which does not need pre-segmented training data. The method is called *Connectionist Temporal Classification*, although most known under its abbreviation CTC. The method is evaluated on the TIMIT data set where two different CTC models are compared, achieving LER (label error rate) of 30.51% and 31.47%. As a benchmark, Context-dependent HMM achieved a LER of 35.21%. In order to allow for the inclusion of a language model in CTC learning, Hannun et al. [2014a] developed a beam search algorithm, which is used to decode the results given by a recurrent neural network. The report presented a character error rate improvement from 10%, when using a simple greedy algorithm, to 5.7% when using a beam search algorithm together with a bi-gram language model.

A speech recognition system in large scale, DeepSpeech, was created by Hannun et al. [2014b] using CTC as a loss function. There are many factors contributing to their good results, such as large amounts of data and a well considered, complex network architecture. The benefits of a CTC based system is that it allows a focus on those aspects, although it demands a large amount of data.

Another significant advance in sequence classification was made by Sutskever et al. [2014], who introduced *sequence to sequence learning*. A sequence to sequence architecture consists of an encoder neural network translating an input sequence of irregular length to an intermediate vector of fixed

length. The intermediate vector is then translated with another neural network, the decoder, into a an output sequence of irregular length. This approach has shown especially good results for tasks like machine translation, where there is an advantage to being able to keep the encoder the same while changing the decoder, or vice versa.

The sequence to sequence method was improved by Bahdanau et al. [2014], by adding a module named attention. The principal improvement of the earlier model being that instead of encoding the sequences to a vector of fixed size, the sequence is encoded to another sequence of vectors, of which a portion is used. This type of architecture has been shown to give great results as used by Chiu et al. [2017], which deploys a variation on this.

Using the CTC-based approach or the encoded-attention-decoder approach comes with different advantages and disadvantages. A prominent shortcoming of CTC-based learning is that it assumes conditional independence between audio frames. An important weakness in the encoder-attention-decoder approach is a high sensitivity to noise, due to induced poor alignment in the attention module. The two methods have been combined by Das et al. [2018], to create a CTC-based approach with attention, producing lower error rates than either approach separately.

1.2.2 Speech Recognition in Music

The amount of work concerning ASR for music is limited. There is a larger focus towards simpler tasks such as automatically aligning vocal and acoustic data when both are given. A common method for these problems are forced alignment (viterbi alignment), as used by Fujihara and Goto [2008] and Kan et al. [2008]. For automatic speech recognition without sequence methods, forced alignment can be very useful to automatically convert word or sentence level annotations to phoneme level.

Mesaros and Virtanen [2010] attempts to recognize words and phonemes in music. They use a phoneme level HMM together with a N-gram word and phoneme level language model. Their algorithm is also constructed to separate the vocal data from the acoustic data, an approach also taken by Hughes and Kristjansson [2012] for ASR in music. They used linearly adopted phonetic speech data for training, due to the lack of phonetic singing data. The best model presented achieves a word error rate of four in five.

2 Method

2.1 Problem Description

The goal is to automatically transcribe the lyrics of the vocals, given only the audio from the recorded song. As it is a very difficult problem, the target has been to achieve a reasonable level of performance on sung a cappella vocals before testing on vocals with music.

2.2 CTC loss

Most machine learning models, and especially deep learning models, needs large amounts of labeled data for training in order to achieve good results. For speech data it is a very time consuming task to transcribe a label to every input window. There exists a large amount of audio data with the words transcribed for a longer audio sequence. In order to be able to train on this loosely aligned data Connectionist Temporal Classification (CTC) loss is used. The input needed to compute a CTC loss is a probability distribution over the characters for each input window. As multiple input windows can represent the same character, CTC collapses all repeated characters to a single character. CTC also introduces a blank character, so that when a sequence is collapsed, predictions can still be made of several identical character in a row, by having the blank character separating the characters.

As an example, the sequences `hheel-llloo` and `h-ellll-lo` both collapse to `hello` while `hhhheeellllooo` and `hhh-eelo` collapse to `helo`, since there is no blank character separating the `l`'s

CTC loss is the log likelihood of the probability for correctly classifying according to the rules for collapsing sequences. The probability of correctly classifying the label l is defined as

$$\mathcal{P}(l|seq) = \sum_{C:\kappa(C)=l} \prod_{t=1}^{T} P(c_t|seq)$$

where κ is the function for collapsing sequences described above

2.3 CTC beam search

Decoding a CTC output can be done in a simple way by computing argmax for the probability distribution over the characters for each time frame, and collapsing the decoded sequence as described in section 2.2. This is however a simplification and does not give the most probable label. Finding the most probable label requires summing all the possible sequences, and summing the probabilities for all the sequences which collapses to the same label. However, the number of possible paths grow exponentially with the number of input frames, meaning that a sequence of only 100 input windows and 29 output characters yields 28^{100} possible sequences.

CTC with beam search is an approach for dealing with the computational complexity while at the same time attempting to find the probability of the most likely label given the CTC sequence output.

Beam search iterates over each time step output window and calculates the most probable labels given the most probable labels for the last time step, and the probability distribution over the characters in the current time step. The amount of labels to consider from the last time step is determined by the width of the beam search. As an example, if the beam width is ten, the the ten most probable labels in the last time step are evaluated in the next time step.

In the final time step the label with greatest probability is picked as the most probable label, given the entire output sequence. Notice that this is an approximation as many possible labels are disregarded in each time step.

In order to improve the predictions of the beam search, a language model can be used to evaluate the probability of the current label sequence for a time step. The language evaluation is performed every time a label sequence ends with a space, as that is the end of a word. When the most probable label sequences are evaluated for each time step, it now becomes important to normalize the predictions with the respect to the number of words in a label sequence.

2.4 Long Short-term Memory

In order to model sequential patterns in data, a recurrent neural network can be used. An especially successful version of recurrent neural networks is the Long Short-term Memory (LSTM), as proposed by Hochreiter and Schmidhuber [1997]. The improved LSTM by Gers et al. [1999] is commonly used today, consisting of a memory cell, input, output and a forget gate. LSTM networks has the advantage over simpler recurrent networks to be able to learn patterns over a longer time without suffering from vanishing or exploding gradients.

2.5 Language Model

Language models are used to predict the probability of a sequence of words or predict the probability of a new word given a sequences history of words. This is useful when doing speech to text translation. The models are operating in noisy environments, so the sequence translation of the audio data is bound to be erroneous, but since some sequences are more likely than others the incorrect sequences can be found with the help of a language model.

There are several different ways of constructing language models using neural networks, and LSTMs has shown great results in recent years, such as the model created by Sundermeyer et al. [2012]. Constructing probabilities directly from N-grams is an older and less time consuming way of creating a language model. An N-grams is just a sequence of N words. Predicting the probability of a word given a history of N-1 (predicting last word in a N-gram) words can be modeled by the chain rule of probabilities.

Since the chain rule is computationally inefficient and it is impractical to estimate the individual probabilities, the Markov assumption can be applied, which assumes that the next observation in

a sequence is solely dependent on the near history of the sequence. As an example, in a tri-gram language model the next word is only dependent on the previous two words. In general the problem has now been simplified to:

$$\mathcal{P}(w_n|w_{n-N+1}^{n-1}) = \frac{\mathcal{C}(w_{n-N+1}^{n-1}w_n)}{\mathcal{C}(w_{n-N+1}^{n-1})}$$

The individual probabilities of a N-gram are estimated from large amounts of text of the type for which the language model will be used. The number of individual tri-grams in a model trained on a large data set grows quickly, and in order to decrease the amount of tri-grams the training data can be cleaned to erase all special characters, stop words, digits and have capital letters converted to lower case.

3 Experiments

The raw audio data was split into frames of 25ms width, with a 10ms shift between each frame. 13 MFCC coefficients were calculated, which then was used as the neural network input. In one minibatch, all input sequences were zero padded to the length of the longest sequence. This allows the recurrent network to unroll only the amount needed. The benefit of this is that the same network can be used for both short and long sequences efficiently, as long as they are not in the same minibatch, which is desired for this project.

In an attempt to improve the predictions of a trained network, a beam search algorithm was implemented, which used a language model. The beam search was a straight forward implementation of an ordinary beam search, with the exception that every time a space character was found in the sequence the language model was invoked on the sequence. The language model feature could also be turned off, for when the beam search was used to predict single words. The language model was based on tri-grams. All tri-grams which occurred more than ten times were stored in the language model. In total the language model was trained on GyanendraMishra [2017], a corpus made up by more than 380 000 different song lyrics. The reason for creating a language model from scratch was that the only language models encountered were trained on ordinary language and not on lyrics data, which was deemed to be a great enough reason to create one from scratch solely trained on song lyrics. The language model was used at inference, and not when training.

The main metric used to evaluate accuracy was edit distance (levenshtein distance). The edit distance for the whole predicted sequence compared to the true transcription was computed, and normalized by the length of the true sequence. This gives an equal weight to a long sequence as a short sequence. The values are then averaged for all predictions in the dataset that was tested.

3.1 Datasets

A comprehensive dataset for music with time-annotated lyrics is not available, although larger databases for speech recognition is available to the public. Using speech for the principal part of training, but also smaller a cappella datasets to adapt the model was decided to be a reasonable approach.

The speech data used is 100 hours of read English speech from the LibriSpeech corpus, made by Panayotov et al. [2015]. The audio is extracted from audio books, coming from the LibriVox project. Annotations are made on the sentence level, giving sequences that are mostly 20 to 30 seconds long.

Two smaller datasets are used with sung vocals, taken from a range of popular songs. The two sets are *mauchs* dataset, created by Mauch et al. [2012], and *hansens* dataset which was created by Hansen [2012]. The two datasets combined contains about 30 minutes of songs with both a cappella vocals and vocals with music. They also contain an hour of songs with only music and vocals. Annotations for all songs are made at the word level.

In order to reduce the risk for overfitting, various changes and transformations were applied randomly to the sung vocals datasets. The transformations were applied randomly, giving many different combinations. The parameters of the transformations themselves were randomized within manually set bounds, such as the amount of gain change, and the scaling factor when resampling. Impulse

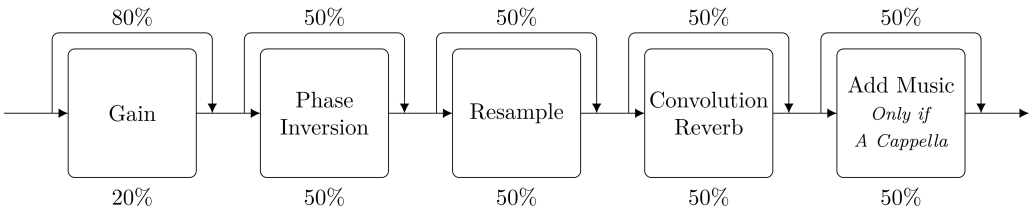

Figure 1: Block diagram showing transformations and the probabilities with which they are applied to the sung vocal dataset.

responses used to generate the reverb effect was taken from Samplicity [2010]. The instrumental music added came from a private dataset.

3.2 Neural Network Architecture

A bidirectional recurrent neural network primarily built with LSTM layers was used. The base of the network is four LSTM layers with 150 nodes in each layer, using the hyperbolic tangent function as activation function. Following is a single fully connected layer with as many outputs as possible labels, with a softmax activation to get a probability distribution as output. All LSTM layers use recurrent dropout when training, as described in Semeniuta et al. [2016], with a dropout probability of 30%. Recurrent batch normalization as shown in Cooijmans et al. [2016] was also used for all LSTM layers. Gradients are clipped by their norm, to avoid single large weight updates destabilizing learning.

Much larger networks were tested, with equal or worse results, at the cost of significantly larger computational time.

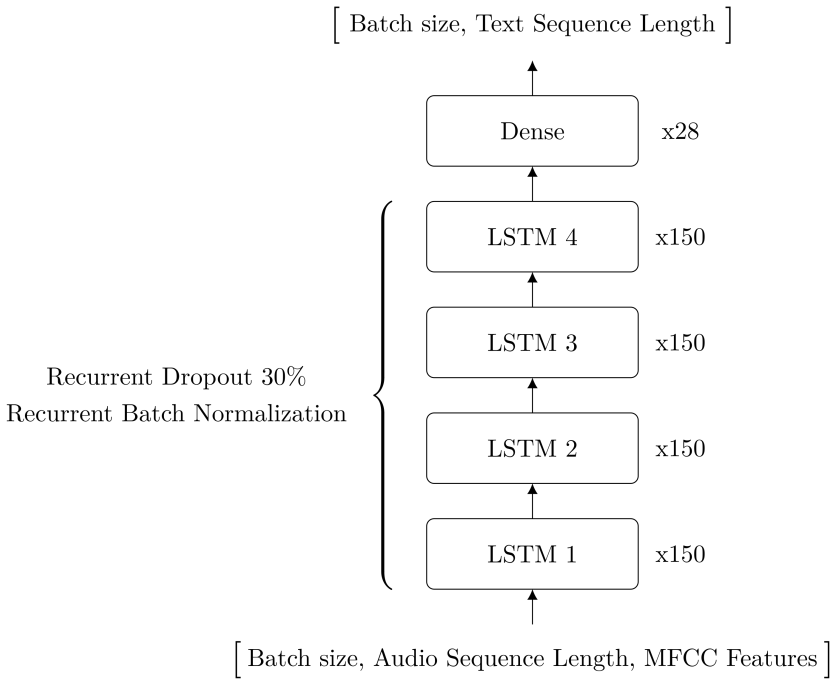

Figure 2: Architecture of the used bidirectional recurrent neural network, with input and output vectors. The number of neurons in each layer is for each direction, forward and backward.

4 Results

Training was performed on the spoken word dataset until the loss stopped decreasing. Both training and test data were simultaneously swapped for the sung vocals dataset. Both CTC loss and error rate as measured with edit distance is shown in Figure 4.

When training is begun with an untrained network, the predictions quickly move towards only predicting blank labels. Since the edit distance is normalized according to the length of the true label, this is why the error rate is exactly 1 for a large part of the training. This behaviour is observed also when training is started on the sung vocals dataset instead.

When the dataset is changed from the spoken word dataset to the sung vocals dataset, the CTC loss decreases fast, from around 500 to near 10. This is because the sequences in the new dataset are much shorter, and should not be seen as a sign of dramatic improvement.

After training on the sung vocals dataset the error rate falls to around 0.75 for both training and test sets. No further improvements are made with continued training.

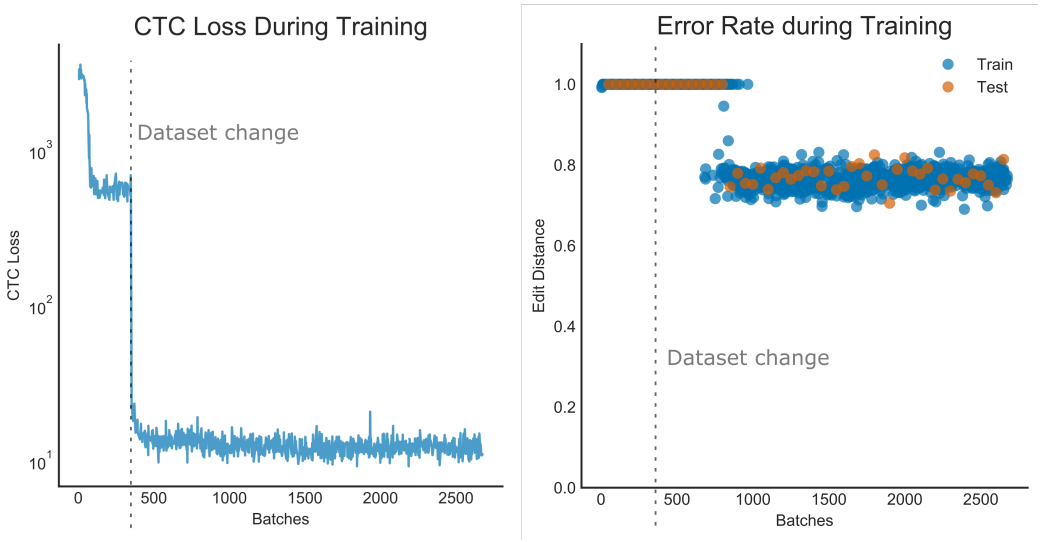

Figure 3: Results from training. To the left, CTC loss is shown. To the right, edit distance on the test set is shown. At *Dataset change*, both training and test set are changed from the spoken word set to the sung vocals dataset.

If training was performed on the sung vocals dataset without the random transformations, the neural network quickly overfitted. Predictions made after this kind of training is shown in Table 1 and Table 2. For the training set, predictions are very close to the true labels. When shown audio from the test set, the predictions appear to be similar to real words, but completely unrelated to the audio it is shown. This is because it is simply regurgitating memorized examples from the training set.

Table 1: Examples of predicted words on the sung vocals test set, using a greedy beam search without language model and the CTC beam search using no language model. The top row are true labels, and the two bottom rows are the predictions (beam search on the last row).

car	color	me	your	color	darling	i	know	who	you	are	come
is	blac	blcn	maen	blc	black	i	akck	wn	black	alks	ie
ilsk	black	blackn	maken	lack	black	iae	alkck	ween	black	alks	ive

Table 2: Examples of predicted words on the sung vocals training set, using a greedy beam search without language model and the CTC beam search using no language model. The top row are true labels, and the two bottom rows are the predictions (beam search on the last row).

make	been	they	i	no	said	to	tried	go	yes	rehab	me
ma	ben	they	i	no	said	to	trid	go	yes	rehab	me
makek	baeen	they	ive	no	saidd	io	tried	ton	yes	brehab	ee

5 Discussion

The main shortcoming of the sung vocal dataset is that it is small, which makes good generalization difficult. It is clear that the random transformations are needed, since the network overfits well before learning generalizable patterns. Although the variation in the audio is increased by applying transformations, the labels remained unchanged, so while it helps, it is not a substitute for a larger dataset.

The speech dataset has many hours of audio data which is good for generalization, but is difficult because of the long sequence length. For the chosen frame shift length, most sequences are around 2000 frames long. This makes the neural network very difficult to train.

With both datasets, the decrease of the CTC loss stagnated quickly. However, the accuracy improvement observed during testing happened during a period where the CTC loss appeared near stable. This suggests that it could happen in a similar way for the spoken word dataset as well. Yet, no such improvement was was observed, even after training for long periods of time.

When trained fully, the predictions from the neural network is not accurate enough to utilize the built language model. The possible benefits from using it for this project is therefore not explored thoroughly.

References

Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 369–376, New York, NY, USA, 2006. ACM. ISBN 1-59593-383-2. doi: 10.1145/1143844.1143891. URL `http://doi.acm.org/10.1145/1143844.1143891`.

Awni Y Hannun, Andrew L Maas, Daniel Jurafsky, and Andrew Y Ng. First-pass large vocabulary continuous speech recognition using bi-directional recurrent dnns. *arXiv preprint arXiv:1408.2873*, 2014a.

Awni Y. Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, and Andrew Y. Ng. Deep speech: Scaling up end-to-end speech recognition. *CoRR*, abs/1412.5567, 2014b. URL `http://arxiv.org/abs/1412.5567`.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc., 2014. URL `http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf`.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014. URL `http://arxiv.org/abs/1409.0473`.

Chung-Cheng Chiu, Tara N. Sainath, Yonghui Wu, Rohit Prabhavalkar, Patrick Nguyen, Zhifeng Chen, Anjuli Kannan, Ron J. Weiss, Kanishka Rao, Katya Gonina, Navdeep Jaitly, Bo Li, Jan Chorowski, and Michiel Bacchiani. State-of-the-art speech recognition with sequence-to-sequence models. *CoRR*, abs/1712.01769, 2017. URL `http://arxiv.org/abs/1712.01769`.

Amit Das, Jinyu Li, Rui Zhao, and Yifan Gong. Advancing connectionist temporal classification with attention modeling. *CoRR*, abs/1803.05563, 2018. URL http://arxiv.org/abs/1803.05563.

H. Fujihara and M. Goto. Three techniques for improving automatic synchronization between music and lyrics: Fricative detection, filler model, and novel feature vectors for vocal activity detection. In *2008 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 69–72, March 2008. doi: 10.1109/ICASSP.2008.4517548.

M. Y. Kan, Y. Wang, D. Iskandar, T. L. Nwe, and A. Shenoy. Lyrically: Automatic synchronization of textual lyrics to acoustic music signals. *IEEE Transactions on Audio, Speech, and Language Processing*, 16(2):338–349, Feb 2008. ISSN 1558-7916. doi: 10.1109/TASL.2007.911559.

Annamaria Mesaros and Tuomas Virtanen. Automatic recognition of lyrics in singing. *EURASIP Journal on Audio, Speech, and Music Processing*, 2010(1):546047, Feb 2010. ISSN 1687-4722. doi: 10.1155/2010/546047. URL https://doi.org/10.1155/2010/546047.

T. Hughes and T. Kristjansson. Music models for music-speech separation. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4917–4920, March 2012. doi: 10.1109/ICASSP.2012.6289022.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. 9:1735–80, 12 1997.

Felix A. Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural Computation*, 12:2451–2471, 1999.

Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. Lstm neural networks for language modeling. In *INTERSPEECH*, 2012.

GyanendraMishra. 380,000+ lyrics from metrolyrics. https://www.kaggle.com/gyani95/380000-lyrics-from-metrolyrics, 2017.

V. Panayotov, G. Chen, D. Povey, and S. Khudanpur. Librispeech: An asr corpus based on public domain audio books. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5206–5210, April 2015. doi: 10.1109/ICASSP.2015.7178964.

M. Mauch, H. Fujihara, and M. Goto. Integrating additional chord information into hmm-based lyrics-to-audio alignment. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(1): 200–210, Jan 2012. ISSN 1558-7916. doi: 10.1109/TASL.2011.2159595.

Jens Hansen. Recognition of phonemes in a-cappella recordings using temporal patterns and mel frequency cepstral coefficients. 2012.

Samplicity. Samplicity's bricasti m7 impulse response library, 2010. URL http://www.samplicity.com/bricasti-m7-impulse-responses/.

Stanislau Semeniuta, Aliaksei Severyn, and Erhardt Barth. Recurrent dropout without memory loss. *CoRR*, abs/1603.05118, 2016. URL http://arxiv.org/abs/1603.05118.

Tim Cooijmans, Nicolas Ballas, César Laurent, and Aaron C. Courville. Recurrent batch normalization. *CoRR*, abs/1603.09025, 2016. URL http://arxiv.org/abs/1603.09025.

Response on feedback:

Feedback: Perhaps missing in "compare different methods" since you present no baseline comparison. The presented method is very complex, how does a simpler model do for this task?
Response: In related work we present results from work done on similar tasks. But since the over all result of our network is so bad we see no point in comparing it directly with a baseline.

Feedback: You could explain better why the music ARS is more complex.
Response: Done in introduction

Feedback: Why would you create own language model, if much more powerful language N-grams from huge datasets are readily available?
Response: Added a part in 2.5 discussing the subject.

Feedback: The oldest reference is from 2010, but the field is much older. Did you not look at older nominal work? It would be good to go trough the limitations of older methods.
Response: We did not look for any older methods, does not have anything to do with the year the reports were released but rather the methods described in works.

Feedback: For example, it is not exactly clear why the beam search was chosen over the other approaches explored earlier, and that the beam search was also not mentioned at all in the section under 'Previous Works'.
Response: We added beam search to previous work, explaining the positive impact of it.