Generating lyrics using recurrent neural networks

Aaron van Geffen Johanna de Vos Willem de Wit

June 19, 2017

Abstract

We compared different configurations of recurrent neural networks with long short-term memory cells for generating song lyrics. Three hyperparameters were investigated: the number of layers, the number of nodes per layer, and the drop-out rate. Model goodness was evaluated in terms of the categorical cross-entropy achieved during training. The optimal configuration was used to generate lyrics in the genres of country, metal and pop. This configuration was found to be three layers, 512 nodes, and 20% drop-out rate.

1 Introduction

The goal of statistical language modelling is to define a probability distribution over a sequence of words (Zhai & Massung, 2016, p. 50). This makes statistical language models (SLMs) of central importance to any natural language application involving prediction, for example automatic speech recognition, machine translation, and text generation. Perhaps the best-known SLM is the *N*-gram, but most progress in recent years can be attributed to artificial neural networks (ANNs).

Specifically, recurrent neural networks (RNNs) are at the heart of this progress. Because their output is not only fed forward, as it is in traditional ANNs, but can be looped back onto itself, RNNs have a kind of 'memory' that traditional ANNs lack. This is crucial for SLMs, because the next word in a sentence usually depends on the words that came before it. The superiority of RNNs over *N*-gram models was demonstrated by Mikolov, Karafiát, Burget, Černocký, and Khudanpur (2010), whose best mixture of three RNN models achieved a reduction of 50% in perplexity as compared to a state-of-the-art 5-gram back-off model.

However, conventional RNNs are difficult to train, and their memory capacity only suffices for short-term dependencies (Bengio, Simard, & Frasconi, 1994). For example, consider this sentence: "We visited Berlin, the capital of Germany". Here, to predict "Germany", it is essential to still remember "Berlin". Should the distance between "Berlin" and "Germany" increase, however, then it becomes more difficult for conventional RNNs to make the right prediction. A crucial innovation in this respect was the invention of long short-term memory (LSTM) networks (Hochreiter & Schmidhuber, 1997): a specific type of RNN in which each node consists of four interacting layers. Within a node, three gates regulate which information is let through, and which is forgotten. Sundermeyer, Schlüter, and Ney (2012) showed that LSTM networks yielded a perplexity about 8% lower than conventional RNNs.

Despite being state-of-the-art, text generated by LSTM RNNs can currently still fairly easily be distinguished from human-generated text, usually because the computer-generated texts are not very coherent. Thus, the quest to construct an SLM that has the same language capacity as humans do, is still ongoing. Greff, Srivastava, Koutník, Steunebrink, and Schmidhuber (2015) compared a 'vanilla' LSTM architecture (as originally proposed in Graves and Schmidhuber (2005)) to eight possible modifications (for example, leaving out one of the gates). They found that none of the modifications significantly improved the vanilla architecture in speech recognition, handwriting recognition, and polyphonic music modelling tasks.

In addition to a model's architecture, another domain of potential improvement are a model's hyperparameters. These include the number of hidden layers and the number of nodes within each layer. Sak, Senior, and Beaufays (2014) explored different numbers of hidden layers (1, 2, 5, 7) for LSTM RNN models, while keeping the total number of parameters constant at 13 million. The best performance, in terms of word error rate in speech recognition, was found for the model with five hidden layers. For this model, increasing the number of nodes per layer from 440 to 840 (now for a total of 37 million parameters) did not affect performance.

The aim of the current study is to build upon the work of Sak et al. (2014), and explore various hyperparameter settings in order to build the best-possible SLM, this time for text generation rather than speech recognition. One example illustrating the usefulness of good models for text generation comes from Turner, Sripada, Reiter, and Davy (2006), who constructed such a system to automatically produce pollen forecasts for hay-fever sufferers in Scotland. In this study, we will train our models on song lyrics in three genres: country, metal and pop.

Model goodness will be operationalised in terms of the categorical cross-entropy that the model achieves during training (objective), and the quality of a lyric generated by this model (subjective). Based on Greff et al. (2015), we will be using Graves and Schmidhuber (2005)'s vanilla LSTM RNNs. The following three hyperparameters will be explored:

1. Number of layers (1, 2, 3)
   Using deeper RNNs, in other words RNNs with more hidden layers, has been found to benefit the generalisation of training to test data (Sak et al., 2014), although it may make training harder and convergence slower. To this end, we will compare networks with 1, 2 and 3 hidden layers.
2. Number of nodes (256, 512)
   While Sak et al. (2014) did not find an effect of increasing the number of nodes per hidden layer from 440 to 840, Greff et al. (2015) detected with a random search that this hyperparameter explained 10% of the variance in their speech recognition task (although it was only 2% in handwriting recognition). This suggests that the number of nodes is worth exploring further. We will compare hidden layers with 256 and 512 nodes.
3. Drop-out rate (20%, 50%)
   Like the number of layers, another hyperparameter concerning the generalisation from training to test data is the drop-out rate. At the presentation of each training case, a pre-defined percentage of randomly selected nodes is dropped from the model, thus preventing "complex co-adaptations" between several neurons (Hinton, Srivastava, Krizhevsky, Sutskever, & Salakhutdinov, 2012, p. 1). Hinton et al. (2012) found that a drop-out rate of 50% resulted in big improvements on a variety of speech and image recognition tasks in feedforward ANNs. We will explore the effects of drop-out rates of 20% and 50%.

2 Method

2.1 Obtaining data

In order to train a neural network for text generation, an abundant corpus of (human-generated) textual data is required. When we set out on this project, we had initially hoped to use a promising Kaggle dataset, which offered 380,000+ lyrics scraped from the MetroLyrics database.[1]

Regrettably, on closer inspection we found the data had been overly sanitised. For example, only one line break remained after each sentence, effectively removing the lyrics' verse structure entirely. We also found it lacking the genre information we were hoping to use to split our training set. It contained a fair share of non-English lyrics as well.

Therefore, we opted not to use this dataset. Instead, we chose to scrape the MetroLyrics website ourselves. Fortunately, the Python script used to generate the Kaggle corpus, a simple web-scraper, was open sourced on GitHub.[2] Hence, we used it as a starting point.

The lyrics were scraped by first downloading the alphabetical artist indices. After distilling all available artists from these indices, the artist pages themselves were downloaded. These pages contained links to the artists' songs, as well as their years of publication, and a genre. The lyrics were downloaded into separate files, structured hierarchically by their artist and year. In total, we ended up with 636,620 non-empty lyrics.

2.2 Sanitising the data

As reported in the previous section, the Kaggle dataset had already been curated to some extent. The data we downloaded ourselves, however, still required a thorough filtering. It included not just a plethora of genres and artists, but also a fair number of lyrics in different languages, including Korean, Japanese, Chinese and Greek.

In order to generate cohesive lyrics in English, we aimed to filter out all lyrics in non-English languages. This was done in three stages.

First, we filtered the scraped lyrics by language using the `langdetect`[3] Python package. This package will classify a given input string on 55 different languages, yielding the end-result as the ISO 639-1 code of the language (e.g. 'en' for English).

The first phase narrowed our dataset down to 563,407 lyrics. Unfortunately, however, the package did not handle particular edge-cases. Most notably, certain Asian pop cultures (e.g. Japanese and Korean) use English words in-between those of their own languages. Curiously, this was enough to mark the entire lyric as English – even though the majority of the lyric was written in non-Latin characters.

To combat this, we introduced a second, more rigorous stage, intended to filter all lyrics that were not written in the Latin alphabet. In it, we loaded lyrics one-by-one, discarding those whose alphabet consisted of $>5\%$ non-ASCII characters.

The resulting filtered lyrics were mostly English lyrics with Latin letters. However, after generating a character frequency count from the resulting dataset, we found out a substantial number of characters were control characters, punctuation marks and numbers.

[1] `https://www.kaggle.com/gyani95/380000-lyrics-from-metrolyrics`

[2] `https://github.com/h4ck3rk3y/lyrics_substance`

[3] `https://pypi.python.org/pypi/langdetect`

In the final phase, we converted the lyrics to lower case, and stripped the aforementioned control characters, punctuation characters and numbers, reducing the vocabulary size considerably by about 50%.

2.3 Generating the training sets

The output of an RNN is always a function of its input. Therefore, we chose to split up the lyrics based on the genre information that was contained in the scraped data, as mentioned earlier. More precisely, every lyric has exactly one genre descriptor, which made this splitting a trivial task. By using only lyrics from one genre, and comparing it to different genres, we hoped to see a certain pattern specific to that genre. To make these differences as clear as possible three genres were chosen that, instinctively, are far apart: country, metal and pop.

This process further reduced the number of lyrics usable for each training set (country: 37,879 lyrics, metal: 42,694 lyrics and pop: 80,465 lyrics). These numbers were still sufficient for our purposes.

The three sets were sampled to make them balanced. The initial sets contained 10,000 lyrics each, but this resulted in overly long training times and memory allocation difficulties. Given these problems, new sets were generated with 1,000 lyrics each, showing much more manageable training times. Given the length of lyrics, this seemed sufficient, and still produced training files of approximately 1MB.

As the training sets were simply concatenated filtered lyrics, tokens were inserted to signal the start (<s>) and end (<e>) of each lyric. This gives the network an indication of typical lyric length.

We should point out that the reduction in vocabulary size, mentioned in section 2.2, by accident resulted in slightly different vocabularies for each of the three different genres, as the reduction was achieved by means of blacklisting certain characters.

2.4 Fixed hyperparameters

To train each of the models, we sequenced their respective training set using the *sliding window* principle. The first sequence would consist of the first 100 characters of the dataset. The second sequence would see the 'window' shift by one character, yielding a sequence from the second to the 101st character, et cetera. Thus, for a dataset of $N > 100$ characters, this results in $N - 100$ *sequence patterns.*

The models were trained in batches, each of them using a maximum *batch size* of 128 of the aforementioned sequences. To optimise the learning rate of the model training process, we employed the ADAM optimiser (Kingma & Ba, 2014), as implemented in the Keras framework, which provides adaptive learning rates. Evaluating each of the epochs was done by using *categorical cross-entropy* as the loss function.

2.5 Manipulated hyperparameters

As mentioned in section 1 on page 1, we explored three hyperparameters: the number of layers, the number of nodes per layer, and the drop-out rate. As an unintended side-effect of the order in which we did training and preprocessing, some models had already been trained before the vocabulary size was reduced (see table 1 on the following page). We retrained these networks using the reduced vocabulary.

The number of layers was increased from one hidden layer to three hidden layers throughout the testing phase. The hidden layers contained either 256 or 512 nodes (this was always the same for all layers in a network). Due to time constraints, this manipulation was only applied to the three-layer networks. Lastly, the drop-our rate was varied between 20% and 50%. In order to test the specific impact of each hyperparameter, only one hyperparameter was varied while all others were kept stable.

2.6 Training the neural network

We used restricted the comparison between the different RNN configurations to the country dataset. Networks for metal and pop were only trained in the configuration that was found to work best on the country data. For training, we opted to use the *Keras* framework, a high-level neural network API for Python, using the *TensorFlow* back end.

Training time is often a hurdle when working with neural networks. Given the time-constrained nature of the project, we had to invest in performance and reduce the training time as much as possible. The Keras framework makes it possible to run the neural network on either the CPU or the GPU. Running the code on a GPU is generally considered superior for non-preprocessing, as training time is drastically reduced when compared to CPU-based training (Bahrampour, Ramakrishnan, Schott, & Shah, 2015; Hadjis, Zhang, Mitliagkas, Iter, & Ré, 2016). We found this to be true; training on a GPU was around eight times as fast as training the identical network on a CPU. In order to have the best training times we made sure to use the GPU where possible. The only occasion we were unable to use the GPU, was during the initial training on the DCC computing cluster.

One other important factor which is of major influence on the training time, is the number of epochs used during the training of the network. On the one hand, while fewer epochs would be faster, the network might not learn enough and therefore its output might be of lesser quality, potentially even nonsensical. On the other hand, having a very large number of epochs was not feasible given the time constraints, but also due to the fact that overfitting might become a real possibility. In the end, we settled for 50 epochs, which resulted in interesting output, while constraining the training time to a relatively low 30 hours per network.

All of the training was done on three machine configurations. For the sake of completeness, we give a brief overview of their specifications below.

Willem's pc has an Intel Core i5-4460 CPU @ 3.20Ghz, GeForce GTX970 (4GB RAM) and 8GB RAM. Calculations were GPU-backed.

George's pc has an Intel Core i5-7600K CPU @ 3.80GHz, NVIDIA TITAN X (12GB RAM) and 32GB RAM. Calculations were GPU-backed.

The DCC cluster has variable specifications,[4] depending on the node used. However, all the calculations were done using high-end Xeon processors with at least 32GB of RAM available. Hence, calculations were CPU-backed.

3 Results

3.1 Configurations

#Voc	#Lrs	#LSTMs	Drop	Loss	#Epochs
61	1	256	0.2	1.8550	50
61	1	256	0.2	1.7953	50
30	1	256	0.2	1.6705	48
30	1	256	0.5	1.8440	50
30	2	256	0.2	1.3485	50
30	2	256	0.5	1.4751	50
30	3	256	0.2	1.2742	25
30	3	256	0.2	1.2142	50
30	3	256	0.5	1.4122	50
30	3	512	0.2	1.2320	9
30	3	512	0.2	1.5844	50
30	3	512	0.2	**1.1621**	50
30	3	512	0.5	1.3231	50

Table 1: Experimental findings for the country data set, yielding best configuration.

Table 1 shows the best loss obtained (out of 50 epochs, unless otherwise specified) for each RNN configuration. The most important results are also visualised in figure 1.

As can be seen from table 1, some configurations were trained multiple times. This had various causes, for example switching from CPU to GPU (# 61 vocab), not having been able to finish 50 epochs, and the finding that one network achieved its best loss already in its second epoch (1.5844). Retraining this last configuration led to a best loss of 1.1621, obtained in epoch 19. We think that the process which had yielded 1.5844 may have been interrupted by another user on the same machine.

[4]`http://wiki.science.ru.nl/DCC/Advanced_Cluster_Structure`

Now looking at the results, he reduction of the vocabulary size from 61 to 30 resulted in a smaller loss, as expected. Comparing the best loss for the different number of layers shows a *decrease* in loss when *increasing* the number of layers from one to two to three. The change from 256 nodes to 512 nodes also lowered the best loss. These findings show that an increase in network size results in lower loss for training with 50 epochs. Increasing the last hyper-parameter, drop-out rate, from 20% to 50% increased the best loss.

3.2 Loss over time

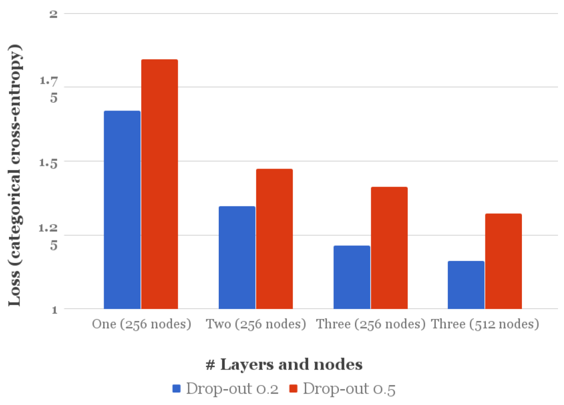

Figure 1: Experimental results for the country dataset, corresponding with table 1.

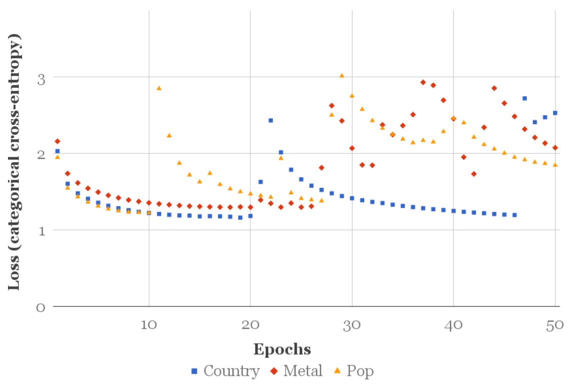

Figure 2: Loss over time for the final models for each genre.

Having found the best set-up for our network (three layers, 512 nodes, 20% drop-out), networks for metal and pop were also trained in this set-up. Figure 2 shows the development of loss for the three genres over the 50 epochs.

The results for the three different datasets have a similar trend shape, in that initially loss is dropping steadily, suddenly spiking in similar fashion, followed by another steady decrease and floating off at the end. Moreover, all three configurations achieved their lowest losses in the first half of their training, with the best loss for country, metal and pop occurring in epochs 19, 20 and 11, respectively.

3.3 Example lyrics

Example lyrics for each combination of hyperparameters that we tried (all in the country genre), can be found in Appendix A. For metal and pop, lyrics can be found for the optimal configuration only (as established on the country dataset). Again, see Appendix A for examples.

4 Discussion

In this study, we investigated three hyperparameter settings for generating lyrics using an RNN with LSTM cells. Training these RNNs on the 1000-lyric country dataset, we varied one hyperparameter at a time, to arrive at the different configurations summarised in table 1 on the previous page. The goodness of the resulting models was evaluated in terms of the loss they achieved during training, and the quality of the lyrics they generated.

Regarding the number of layers, it was found that the loss during training was lowest (thus, best) in the networks with three hidden layers (followed by two, then by one). With regard to the lyrics produced, the one-layer model produced lyrics that were visibly worse in terms of spelling and syntax. While spelling was good in the models both with two and three layers, the latter produced lyrics with better syntax (compare the three-layered output "`i was the one that i can see`" to the two-layered output "`i was to the sing`"). Thus, our findings that having more layers leads to increased generalisability are in line with Sak et al. (2014). It is important to note, though, that Sak et al. (2014) kept the total number of parameters constant whereas our total number of parameters increased when adding an extra layer.

The comparison between 256 and 512 nodes per layer was performed for the three-layered model only, due to limited computational resources and time. As can be seen in table 1 on the preceding page, the number of nodes had much less impact on loss than the number of layers. Seeing that Greff et al. (2015) found that number of nodes explained only 2-10% of the variance in language modelling tasks, this outcome is not very surprising. While the generated lyrics all show a lot of repetition, the loop seems to have a wider scope (spanning two lines of text) for the models with more hidden nodes.

The effect of drop-out rate is immediate visible in terms of loss achieved during training, but visual inspection revealed no notable patterns for the two drop-out rates. It would be interesting to let models with different drop-out rates for a longer period of time, as those models which randomly omit 50% rather than 20% of the nodes during training can be expected to need more training time. An interaction could become visible between the drop-out rate and the number of epochs, in which models with 50% drop-out rates might overtake (or come closer to) the 20% models as time goes on. This could be a possible explanation for the beneficial effect of a 50% drop-out rate as found by Hinton et al. (2012).

In general, the lyrics produced by our best models already are of a reasonable quality. The lyrics are seeded with white space (including white lines) in the right places, and the repetition is reminding of verses. To improve the underlying SLMs further, we recommend that future studies systematically examine the hyperparameter of learning rate, as it was identified by Greff et al. (2015) to account for a large proportion of variance (at least two-thirds) on their test set performance.

The hyperparameter exploration, as described above, was performed on the country dataset only. The configuration that achieved the lowest loss during training (three layers, 512 nodes, 20% drop-out) was then used to train two models on the pop and metal lyrics datasets. As can be seen from Appendix A, the metal lyrics are different in tone from the country lyrics. The rather gloomy nature of the printed example lyric can be seen from the repeating use of negation: "`never`", "`wont`", "`cant`" (apostrophes are lacking because they were not part of the vocabulary). The quality of the pop lyric is much worse compared to the other lyrics we have seen, containing spelling errors, virtually no syntax, and a repetition of the meaningless phrase "`the shings`". One possible explanation might be that the vocabulary size of the pop training set was higher than that of the other training sets (36 for pop, 34 for metal, 30 for country). This was a mistake that we discovered too late. As a result, more output labels had to be predicted for pop, which might have impacted the accuracy.

We also question whether the optimiser distributed the learning rate adequately to progressively reduce the loss over time. Rather than seeing a progressive decline in loss, we observe several spikes in the second half of the training process, as can be clearly seen in figure 2 on the previous page. However, we would expect a declining trend over time, albeit with potential minor fluctuations. For a follow-up study, we suggest the use of different optimisers or different loss functions to explore their influence on the development of loss over time.

Another limitation of the current study (resulting from lack of time) that could also be improved upon in future studies, was that the only objective measure of model goodness came from the loss achieved during training. We judged the generalisability of our models based on visual inspection of the lyrics they generated, but more objective and/or systematic methods are conceivable. However, the reader should keep in mind that in the case of lyric writing, there is no real test set: it is a creative process that cannot be defined in terms of (in)correctness. Nonetheless, the quality of the generated lyrics could be rated using other measures, such as the correctness of spelling and syntax. Coherence and aesthetic quality could be judged by human readers in a rating study. It would be interesting to see how well these measures correlate with the loss achieved during training.

In conclusion, our study confirmed that RNNs with LSTM cells are promising SLMs. An exploration of hyperparameters revealed that their settings (especially the number of layers) can have a large impact on the loss achieved during training, and the quality of the generated lyrics. Given more time and computational resources, it will surely be possible to improve these models even further, bringing us closer to the day when the profession of creative writer is no longer exclusively the domain of humans.

5 Author attribution

From the project's inception, all of us were involved in the concept development. Below, we will give a quick overview of what we specialised in individually.

Aaron wrote and ran the Python scripts to scrape and process the lyrics, as well the LSTM model's Python class. Ran training instances on both the DCC cluster and George's pc. Processed the results in a spreadsheet. Wrote part of the methods and results sections. Did the final presentation.

Johanna did the literature research. Prepared the use of the DCC cluster. Trained a few networks. Gave the final presentation. Wrote the introduction, discussion, reference list, and Appendix A.

Willem trained a fair share of the networks. Gave project introductory presentation. Wrote scripts for data sanitisation. Wrote part of the method and results sections.

6 Source code

To reproduce our results, please find our source code on GitHub: `https://github.com/AaronVanGeffen/LyricsGenerator`.

References

Bahrampour, S., Ramakrishnan, N., Schott, L., & Shah, M. (2015). Comparative study of deep learning software frameworks. *arXiv preprint arXiv:1511.06435*.

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, *5*(2), 157–166.

Graves, A., & Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, *18*(5), 602–610.

Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., & Schmidhuber, J. (2015). LSTM: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems.*

Hadjis, S., Zhang, C., Mitliagkas, I., Iter, D., & Ré, C. (2016). Omnivore: An optimizer for multi-device deep learning on cpus and gpus. *arXiv preprint arXiv:1606.04487*.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 1–18.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780.

Kingma, D., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980.*

Mikolov, T., Karafiát, M., Burget, L., Černocký, J., & Khudanpur, S. (2010). Recurrent neural network based language model. In *Interspeech* (Vol. 2, pp. 1045–1048).

Sak, H., Senior, A., & Beaufays, F. (2014). Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *Interspeech 2014* (pp. 338–342).

Sundermeyer, M., Schlüter, R., & Ney, H. (2012). LSTM neural networks for language modeling. In *Interspeech 2012* (pp. 194–197).

Turner, R., Sripada, S., Reiter, E., & Davy, I. P. (2006). Generating spatio-temporal descriptions in pollen forecasts. In *Proceedings of the Eleventh Conference of the European Chapter of the Association for Computational Linguistics* (pp. 163–166).

Zhai, C., & Massung, S. (2016). *Text data management and analysis: A practical introduction to information retrieval and text mining.* New York, NY: Morgan & Claypool.

Appendices

A Lyrics

A.1 Country

This is the seed for all country lyrics (taken from Danielle Peck's "Findin' a Good Man"):

```
if you hear me girls raise your hand ↩
lets have a toast ↩
lets have a toast ↩
heres to findin a good man ↩
```

Layers: 1, Nodes: 256, Drop-out: 20%

```
nh hold and love in the bar ↩
↩
nh i cone to the soon would be there ↩
↩
i was hone to the bas ↩
↩
nh i cone to the soon wht io the wiy ↩
↩
i was always the one that i do ↩
↩
io the simes of the some ↩
i was always t
```

Layers: 1, Nodes: 256, Drop-out: 50%

```
now the way that i could tee the say ↩
ie i could to took her to the same oo the bar ↩
i was toiether the saie the saie the saie the saie
the saie the saie the saie the saie the saie the saie
the saie th
```

Layers: 2, Nodes: 256, Drop-out: 20%

```
when the sun goes and the song ↩
i was a touch of the same ↩
↩
i was a touch of the same ↩
↩
i was a touch of the same ↩
↩
i was a touch of the same ↩
↩
i was a touch of the same ↩
↩
i was a touch of the same ↩
↩
i was a
```

Layers: 2, Nodes: 256, Drop-out: 50%

```
when i said theres a little bit i was to the sing to
the sing to the sing ↩
i was the one that i was to the side ↩
of the sun was the way that i was to the side ↩
of the sing that i was to the sing to see t
```

Layers: 3, Nodes: 256, Drop-out: 20%

```
i can see the way you want me to be alone ↩
↩
i can see the way you want me to be ↩
↩
i can see the way you want me to be ↩
↩
i can see the way you want me to be ↩
↩
i can see the way you want me to be ↩
↩
i can se
```

Layers: 3, Nodes: 256, Drop-out: 50%

```
i was the one that i can see ↩
↩
i was the one that i can see ↩
↩
i was the one that i can see ↩
↩
i was the one that i can see ↩
↩
i was the one that i can see ↩
↩
i was the one that i can see ↩
↩
i was the one that i
```

Layers: 3, Nodes: 512, Drop-out: 20%

```
i was always been there to the store ↩
↩
i was the way that i was the way that i was the
way that i was the same ↩
↩
i was alone to the store ↩
↩
i was the way that i was the way that i am the
way i want to b
```

Layers: 3, Nodes: 512, Drop-out: 50%

```
i wanna see the same old song the stars and the
stars ↩
```

```
and i was a star and the stars and the stars ↩
and i was a star and the stars and the stars ↩
and i was a star and the stars and the stars ↩
and i was
```

A.2 Metal

This is the seed for the metal lyric (taken from Mastodon's "The Czar"):

```
the henchmen are gathered and waiting ↩
dont stay run away ↩
your role as usurper is found out ↩
dont stay ↩
```

Layers: 3, Nodes: 512, Drop-out: 20%

```
run away ↩
↩
i will never go ↩
↩
i want to be a different standay ↩
↩
i wont be a single throne ↩
you cant see the truth ↩
↩
i will never see the world ↩
the stars are the ones who seems shall be found
↩
↩
i will never
```

A.3 Pop

This is the seed for the pop lyric (taken from Owl City's "Dental Care"):

```
when hygienists ↩
leave on long vacations ↩
thats when dentists scream and ↩
lose their patience ↩
↩
talking
```

Layers: 3, Nodes: 512, Drop-out: 20%

```
you iow the sr the shings the shings the shings
the shings the shings the shings the shings
the shings the shings the shings the shings
the shings the shings the shings the shings the
shings the shing
```