Short Story Popularity Prediction using Neural Networks with Time Series-Based Circular Dependencies

Author: Karl-Johan Alm

Supervisors:
Prof. Hiroaki Saito (Keio)
Prof. Johan Boye (KTH)

Examinator:
Prof. Erik Fransén (KTH)

A thesis submitted in fulfillment of the requirements for the degree of Master of Computer Science in the Saito Lab Department of Information and Computer Science

August 5, 2016
Declaration of Authorship

I, Karl-Johan ALM, declare that this thesis titled, “Short Story Popularity Prediction using Neural Networks with Time Series-Based Circular Dependencies” and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

[Signature]

Date:

[Date]
Abstract

Science and Technology
Department of Information and Computer Science

Master of Computer Science

Short Story Popularity Prediction using Neural Networks with Time Series-Based Circular Dependencies

by Karl-Johan Alm

The act of judging the quality (or popularity) of a short story is a highly complex and subjective matter, dependent on numerous external factors such as the reader audience, the environment in which the short story is presented, as well as the undulation of trends. Even by looking exclusively at the short story content, ignoring all external factors, there is no good architecture available in Machine Learning to analyze very long sequences, such as a several hundred thousand word story, without resorting to approximations where only a small part (e.g. 40 words) are processed at a time.

This thesis introduces and evaluates an extension to neural networks called a memory module, a persistent representation of sequential data, as well as several adjustments to the training process necessary to facilitate the training of the memory module.

The memory module is a persistent representation of the input as a whole, and is continuously updated as the network iterates over each input sample.

The accuracy of the memory module-enabled model is compared to traditional neural network models performing the same task using the same input, such as recursive neural networks.

While the results are still in their infant stages, requiring more and deeper analysis beyond the scope of this thesis, the memory module shows great promise, beating the traditional approaches by roughly 20% in certain cases.

Keywords: machine learning, rating, nlp, natural language processing, persistency
Acknowledgements

I would like to thank my advisors Prof. Hiroaki Saito of the Department of Information and Computer Science at Keio University and Prof. Johan Boye at the School of Computer Science and Communications at KTH.

Firstly, Prof. Saito for his unwavering support. He gave me free reign to focus on what I considered important and gave invaluable advice throughout the process whenever he deemed it necessary.

Secondly, Prof. Boye for his invaluable feedback, in particular on the thesis content, but also on the research proposal and process. Despite the barrier of residing on different continents, his support felt close and readily available at all times.

I would also like to thank T.I.M.E. Association for providing me with the opportunity to enter the Double Degree Programme with KTH and Keio University.

Finally, I would like to extend my gratitude to my family and to my wife for their continuous support and encouragements throughout these years. I couldn’t have done it without you. Thank you.
Contents

Declaration of Authorship i
Abstract ii
Acknowledgements iii

1 Introduction 1
  1.1 Popularity as a quantifiable concept ................................ 1
  1.2 Time series data ......................................................... 3
  1.3 Research problem ...................................................... 4
  1.4 Purpose ................................................................. 4
  1.5 Goal ........................................................................... 5
  1.6 Method ....................................................................... 5
  1.7 Delimitations ............................................................ 5
  1.8 Ethics ......................................................................... 6
  1.9 Thesis structure ........................................................ 6

2 Related Work 8

3 Machine learning 10
  3.1 Deep and shallow neural networks ........................................ 10
  3.2 Over- and underfitting .................................................... 11
  3.3 Training and validation .................................................... 12
  3.4 Convolutional neural networks .......................................... 12
  3.5 Recurrent neural networks .............................................. 13
  3.6 Activation functions ...................................................... 13
    3.6.1 Sigmoid ($\sigma$) ...................................................... 14
    3.6.2 Hyperbolic tangent function (tanh) .............................. 14
    3.6.3 Rectifier .............................................................. 14
    3.6.4 Softmax .............................................................. 14
  3.7 F-measures .................................................................... 14
  3.8 Miscellaneous techniques .............................................. 15
    3.8.1 Dropout ............................................................... 15
    3.8.2 One-hot vectors ...................................................... 16
    3.8.3 Word vectors ......................................................... 17

4 Method 19
  4.1 Preparatory work ......................................................... 20
    4.1.1 Tokenization ......................................................... 20
4.1.2 Word vectorization ........................................ 20
4.1.3 Rating-to-short story mapping .......................... 20
4.1.4 Threshold determination .................................. 20

4.2 Sampling data ................................................. 21

4.3 Memory module implementation ............................ 21

5 Short stories ....................................................... 23
5.1 Selection ......................................................... 23
5.2 Vectorization ..................................................... 23
5.3 Rating .......................................................... 25
5.4 Sequencing ....................................................... 26

6 Memory module .................................................. 27
6.1 Memory module .................................................. 27
6.2 Muted softmax ................................................... 27
6.3 The waves approach .......................................... 30
6.4 Memory module implementation ............................ 32
6.5 LSTM and memory module .................................... 33

7 Results ............................................................. 35
7.1 Measuring accuracy .......................................... 35
7.2 Model selection ................................................ 36
7.3 Word vectors vs one-hot vectors ............................ 37
7.4 Softmax nodes vs real output ............................... 37
7.5 Genre-specific vs general-purpose .......................... 38
7.6 End, average, and sum predictors .......................... 38
7.7 Complete results .............................................. 41

8 Analysis ............................................................. 42
8.1 Model performance .......................................... 42
8.1.1 Meta properties ............................................. 42
8.1.2 LSTM performance ....................................... 43
8.2 Memory module applicability ............................... 43
8.2.1 Batch training .............................................. 43
8.2.2 Training fluctuations ..................................... 44
8.2.3 Dormant nodes ............................................. 44
8.2.4 Performance ............................................... 44
8.3 Biases and trends ............................................. 45
8.3.1 The last_updated value .................................. 45

9 Open questions .................................................. 46
9.1 LSTM .......................................................... 46
9.2 Memory module fluctuations in training .................... 46
9.3 Convolutional neural networks with processing layers .... 47
9.4 Representation of output .................................... 47
# Conclusions

Bibliography

<table>
<thead>
<tr>
<th>A</th>
<th>Sample stories</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A.1</td>
<td>Sample 1</td>
<td>53</td>
</tr>
<tr>
<td>A.2</td>
<td>Sample 2</td>
<td>54</td>
</tr>
<tr>
<td>A.3</td>
<td>Sample 3</td>
<td>54</td>
</tr>
<tr>
<td>A.4</td>
<td>Sample 4</td>
<td>54</td>
</tr>
<tr>
<td>A.5</td>
<td>Sample 5A</td>
<td>55</td>
</tr>
<tr>
<td>A.6</td>
<td>Sample 5B</td>
<td>55</td>
</tr>
<tr>
<td>A.7</td>
<td>Sample 6</td>
<td>56</td>
</tr>
<tr>
<td>A.8</td>
<td>Sample 7</td>
<td>56</td>
</tr>
<tr>
<td>A.9</td>
<td>Sample 8</td>
<td>57</td>
</tr>
</tbody>
</table>
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Popularity trends, “異世界転生”</td>
<td>2</td>
</tr>
<tr>
<td>1.2</td>
<td>Google trends, “異世界転生”</td>
<td>3</td>
</tr>
<tr>
<td>3.1</td>
<td>Artificial neural network</td>
<td>11</td>
</tr>
<tr>
<td>3.2</td>
<td>Convolutional neural network</td>
<td>12</td>
</tr>
<tr>
<td>3.3</td>
<td>Word vector projection</td>
<td>17</td>
</tr>
<tr>
<td>6.1</td>
<td>Memory module overview</td>
<td>28</td>
</tr>
<tr>
<td>6.2</td>
<td>Muted softmax</td>
<td>28</td>
</tr>
<tr>
<td>6.3</td>
<td>Wave example</td>
<td>31</td>
</tr>
<tr>
<td>6.4</td>
<td>Example model using a memory module</td>
<td>33</td>
</tr>
<tr>
<td>6.5</td>
<td>LSTM implementation using memory module</td>
<td>34</td>
</tr>
<tr>
<td>8.1</td>
<td>Length/rating</td>
<td>43</td>
</tr>
</tbody>
</table>
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Short story distribution over time</td>
<td>24</td>
</tr>
<tr>
<td>5.2</td>
<td>Short story distribution over genre</td>
<td>24</td>
</tr>
<tr>
<td>5.3</td>
<td>Discarded tokens based on minimum occurrence in source corpora</td>
<td>25</td>
</tr>
<tr>
<td>7.1</td>
<td>Accuracy of different models</td>
<td>36</td>
</tr>
<tr>
<td>7.2</td>
<td>Confusion matrix, end state</td>
<td>39</td>
</tr>
<tr>
<td>7.3</td>
<td>Confusion matrix, average state</td>
<td>39</td>
</tr>
<tr>
<td>7.4</td>
<td>Confusion matrix, sum state</td>
<td>40</td>
</tr>
<tr>
<td>7.5</td>
<td>F-measures, sum state</td>
<td>40</td>
</tr>
<tr>
<td>7.6</td>
<td>Accuracy per granularity</td>
<td>41</td>
</tr>
<tr>
<td>8.1</td>
<td>Bias from using last_updated value 2013 vs 2015</td>
<td>45</td>
</tr>
</tbody>
</table>
## List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A)NN</td>
<td>(Artificial) Neural Network</td>
</tr>
<tr>
<td>API</td>
<td>Application Program Interface</td>
</tr>
<tr>
<td>CNN</td>
<td>Convolutional Neural Network</td>
</tr>
<tr>
<td>DNN</td>
<td>Deep Neural Network</td>
</tr>
<tr>
<td>EOF</td>
<td>End Of File</td>
</tr>
<tr>
<td>GPU</td>
<td>Graphical Processing Unit</td>
</tr>
<tr>
<td>GRU</td>
<td>Gated Recurrent Unit</td>
</tr>
<tr>
<td>INF</td>
<td>INFINITY</td>
</tr>
<tr>
<td>LSTM</td>
<td>Long Short Term Memory</td>
</tr>
<tr>
<td>MLP</td>
<td>Multi Layer Perceptron</td>
</tr>
<tr>
<td>MM</td>
<td>Memory Module</td>
</tr>
<tr>
<td>NaN</td>
<td>Not a Number</td>
</tr>
<tr>
<td>POS</td>
<td>Part Of Speech</td>
</tr>
<tr>
<td>ReLU</td>
<td>Rectified Linear Unit</td>
</tr>
<tr>
<td>RNN</td>
<td>Recursive Neural Network</td>
</tr>
<tr>
<td>UTF</td>
<td>Unicode Transformation Format</td>
</tr>
<tr>
<td>GB</td>
<td>Good Bad</td>
</tr>
<tr>
<td>GNB</td>
<td>Good Neutral Bad</td>
</tr>
<tr>
<td>MMConv2D</td>
<td>Memory Module with CONVolutional Pooling</td>
</tr>
<tr>
<td></td>
<td>in 2 Dimensions</td>
</tr>
<tr>
<td>MMConv2DX2</td>
<td>Memory Module with two sets of CONVolutional Pooling in 2 Dimensions</td>
</tr>
</tbody>
</table>
## List of Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$</td>
<td>collection of short stories</td>
<td>$\mathbb{R}^3$</td>
</tr>
<tr>
<td>$\mathbf{S}$</td>
<td>short story matrix</td>
<td>$\mathbb{R}^2$</td>
</tr>
<tr>
<td>$\mathbf{s}$</td>
<td>short story token vector</td>
<td>$\mathbb{R}$</td>
</tr>
<tr>
<td>$r$</td>
<td>rating</td>
<td>score</td>
</tr>
<tr>
<td>$\omega$</td>
<td>size of short story collection ($S$)</td>
<td>count</td>
</tr>
</tbody>
</table>
1 Introduction

1.1 Popularity as a quantifiable concept

The popularity of a short story is dependent on numerous factors, the actual content being just one of them. It depends on time, as literary trends change both in genre, topic, as well as in writing style. It also depends on place. A hugely popular theme in a specific social group or geological place is not necessarily popular in some other social group or place. Certain themes are invented or revitalized through hugely popular works, resulting in a sharp increase in both works of a similar kind and of interest thereof among readers, consequently contributing to the popularity of the works as a collective whole.

Comparing, for instance, the number of works uploaded with the keyword “異世界転生” (“being reborn in a different world”) between January 2009 and February 2015 (Figure 1.1), we see a clear example of this in action.

Up until December 2009, there were a total of 0 updates. In the months following this, there were 1, 3, 0, 1, 0, 1, and 0 updates respectively. Compare this to the 70, 89, and 99 updates in December 2014 up to February 2015. Google Trends gives a more universal picture of the same trend (Figure 1.2), showing a nearly identical picture, although from the perspective of a reader, as opposed to an author.[1]

Given this, and many other unmentioned and unknown factors, the task of predicting the popularity of a short story seems insurmountable. Indeed, this task has only seen one other attempt at a quantitative analysis prior to this thesis.[2]

In the Machine Learning field, there are no readily available tools for processing very long sequences of data (such as a short story in its entirety), and the alternatives that do exist each have their drawbacks. Consequently, a lot of research efforts go into analyzing e.g. Twitter content, which has a fixed-length limit per “tweet” (message) of 140 characters.[3]

Recurrent neural networks (RNN:s) — a form of neural net which processes sequences of data and maintains a form of state that is updated each step — are able to efficiently learn to construct a representation of the content being processed, but there is no proven method for learning to

---

1 As mentioned in Figure 1.1, due to how updates are counted, there is a slight underrepresentation for older results, as each short story’s updated value refers to the latest time that short story was updated.

2 “Success with Style: Using Writing Style to Predict the Success of Novels” by Ashok, V. et al.

3 A Google Scholar search for ‘neural network twitter’ gives over 800 thousand results, whereas ‘neural network “short stories”’ gives less than four thousand.
Figure 1.1: Popularity trends between January 2009 and February 2015 for the keyword “異世界転生” (“being reborn in a different world”). Note that the data is based on updates, which means there is a slight underrepresentation the further back in time you go, as each short story is only represented once, at its latest updated date. The accumulated distinct updates (bottom) consequently only accounts for one short story once.
1. Introduction

3

Figure 1.2: Worldwide Google trends results for “異世界転生”.

represent longer sequences of data, even though LSTM:s (Long Short Term Memory units) significantly reduce the forgetfulness tendencies seen in vanilla RNN:s.

This thesis discusses ways to predict popularity for short stories written in Japanese within a given time frame. It exclusively looks at the content of each short story, without using or investigating POS tags (part of speech tags, such as “noun” or “verb”) or other natural language tools. It doesn’t take into account when the story was written, or any other available “metadata”. In the thesis, existing models such as the LSTM are discussed, and a new extension to neural networks called a memory module is introduced, and a comparison is made of the individual performance of each model at the given task.

1.2 Time series data

As the short stories analyzed are in Japanese, the concept of “words” becomes slightly ambiguous, as the language does not include the concept of word separation (space). The term “word” is nonetheless used in its generic definition, and simply refers to a part of a sentence split up into pieces somehow. The thesis sometimes abstracts away from the specific task of analyzing short stories, to the more general task of analyzing time series data, to emphasize that the findings presented are in no way restricted to natural language processing tasks. A time series is defined as follows:

A time series is a sequence of data points made:

1. over a continuous time interval
2. out of successive measurements across that interval

4The phrase “きれいなお花” (a beautiful flower) can be split into “綺麗なお花”, “綺麗なお花”, “綺麗なお花”, and “綺麗な お花”. Some people consider the な (“na”) in so called な-adjectives to be attached to the word (綺麗な), while other people consider な to be a separate particle. 花 (flower) is often referred to with the honorary prefix お (“o”), so many people would consider お花 to be one word, but dictionaries often exclude the honorary prefix, so other people would consider お to be separate from the dictionary word 花. In short, there is no precise definition for how words are split up.
3. using equal spacing between every two consecutive measure-
4. ments

4. with each time unit within the time interval having at most

one data point

The chronologically ordered words making up a short story can be seen
as a series of measurements (words) made (1) over a continuous time interval
(chronological iteration over the words) (2) out of successive measurements
across that interval (the measurement of the word itself) (3) using equal
spacing between every two consecutive measurements (each word is roughly
the same “span”) (4) with each time unit within the time interval having
at most one data point (there are never two words in the same “spot”).
Consequently this generalization is deemed to be justified.

1.3 Research problem

We have a short story $S_\ell$. The short story is made up of words, which
we refer to as *tokens*. The short story additionally has a popularity score
associated with it. We call this the rating of $S_\ell$ and name it $r_\ell$. $\ell$ is the label
of the short story, which we will use to refer to $S_\ell$ and $r_\ell$.

Although we say that the tokens are words, in reality each token is a
representation of a given word, in the form of a fixed length vector. As such,$S_\ell$ can be considered a matrix, where each column is one token vector.

Finally, we have a rating function $r(\ell) = r_\ell$. It plays an important role
later, when the rating is redefined in section 5.3 (p. 25).

Given $S_\ell$ comprised of $n_\ell$ tokens $\vec{s}_{\ell,1} \cdots \vec{s}_{\ell,n_\ell}$ and its corresponding
rating function $r(\ell)$, can a system be developed which predicts $r(\ell)$ given $S_\ell$?
I.e. does some function exist that satisfies Eqn. (1.1) for all $\ell$?

$$f(S_\ell) \approx r(\ell) \quad (1.1)$$

We generalize this further and introduce one last definition, namely
$S$, a *collection* of $\omega$ short stories $S_0, \cdots, S_\omega$. Thus, we have a collection $S$,
containing short stories such as $S_\ell$, which are made up of tokens $\vec{s}_{\ell,i}$.

1.4 Purpose

The purpose of the research was to determine whether $f(S_\ell)$ in Eqn. (1.1)
does indeed exist, and to attempt to determine its form or an approxi-
mation thereof.

A secondary, but arguably more important purpose was to investi-
gate alternative, preferably more robust approaches for analyzing very long
sequences of data without suffering from forgetfulness.

A third purpose was to determine if the prediction accuracy improves
by transforming the given short story piece by piece (token by token) into
some form of $n$-dimensional proxy object, and to then predict from that object instead of predicting directly from the short story itself. In other words, would we benefit from performing a mapping $\mathbb{R}^{x \times y} \to \mathbb{R}^n$ (where $x$ is the token size, $y$ the number of tokens, and $n$ some arbitrary value)?

1.5 Goal

Having access to $f(S_\ell)$ in Eqn. (1.1) opens up a realm of possibilities, in particular in the literary realm, but potentially also in any other case involving time-series data mapped to some overarching “digest quantifier.” By analyzing a user’s written content, the system could pinpoint “weak” and “strong” points, and even suggest alternatives that would increase the overall score. A publisher could use the tool to “flag” exceptionally good submissions (or to filter out exceptionally bad ones). While the latter may not be all that desirable, the former could prove to be a powerful tool in particular for aspiring authors. One goal is to create such a product.

Finding a method for analyzing extremely long time series data without the problem of forgetting context would open up a realm of possibilities in the machine learning field. As noted, instead of short stories, it could just as well be music or recorded data from some physical phenomenon. A second goal is to invent and present such a method to the community.

Last but not least, pinpointing what makes a short story popular, i.e. pleasing to the reader, as a fundamental problem, is an intriguing and as yet unexplored realm that promotes the question: is there some structure or logic, even if temporally or culturally dependent, which can be used to distinguish highly popular works from impopular ones? If there is, and it was defined via some model, what impact would that definition have on the literary realm as it stands today?

1.6 Method

The method is described separately in Chapter 4 (p. 19).

1.7 Delimitations

The project was quite a bit bigger than the average master thesis, and as such it was necessary to focus efforts on only the core essentials, rather than model fine tuning and similar. As such, the new model proposed is still crude and, with proper fine tuning, could most likely give much better results than those presented here. In addition, there are several limitations with and open questions related to the model which are mentioned separately. These are discussed but not resolved in Chapter 8 (p. 42).
1.8 Ethics

It is necessary to raise the question of the ethical implications of an implementation such as the one proposed in this thesis. In section 1.5 above, the potential publisher using the tool to filter out bad submissions is an example of a use case which is of a questionable nature. Specifically, it runs the risk of automatically denying potentially good content, due to flaws in the system. Then again, if the algorithms are public, an author may use the same tool to evaluate their own work and perhaps even identify and resolve shortcomings in their works.

In the case of an aspiring author attempting to write a best selling novel, the system might mistakenly flag content as popularizing or impopularizing, even if the opposite happens to be the case. It is a highly subjective matter, and it would be quite unfortunate if the next Shakespeare was demotivated and lost their will to write, or ended up writing something completely generic due to destructive feedback.

1.9 Thesis structure

The thesis is divided into ten chapters.

Introduction

Chapter 1 (p. 1) introduces the topic, giving background information on the research topic, its goals, and on ethical implications.

Related Work

Chapter 2 (p. 8) discusses research similar to the one presented in this thesis.

Machine Learning

Chapter 3 (p. 10) introduces the concept of machine learning, describing key theoretical concepts, in particular LSTM:s and how they differ from the proposed memory model.

Method

Chapter 4 (p. 19) describes the method used in the evaluation of the system.

Short Stories

Chapter 5 (p. 23) discusses the short stories used as input to the system, including how they were preprocessed and arranged.
1. Introduction

Memory module

Chapter 6 (p. 27) goes into detail on the specific implementation used in this thesis.

Results

Chapter 7 (p. 35) contains the results of the evaluation, in the form of reports with data from the evaluation.

Analysis

Chapter 8 (p. 42) contains the analysis, where the implications of the evaluation results are discussed.

Open questions

Chapter 9 (p. 46) brings up open questions that arose in the development of the model.

Conclusions

Conclusions and suggestions for future research are presented in Chapter 10 (p. 48).
Recursive neural networks: In “Reasoning With Neural Tensor Networks for Knowledge Base Completion”, Richard Socher et al introduce an “expressive neural tensor network suitable for reasoning over relationships between two entities.”[6] This is a sub-problem of the more generic “understanding content” problem addressed in this thesis, with the biggest difference being that no explicit entity reasoning is performed here.

In his doctor’s thesis “Recursive Deep Learning For Natural Language Processing and Computer Vision”, Richard Socher presents a number of recursive deep models which “can predict an underlying hierarchical structure, learn similarity space for linguistic units of any length and classify phrase labels and relations between inputs.”[7] Again, while no explicit direction is made to enforce the models to learn similarity space or the like, the thesis explores a similar aspect of natural language processing, namely comprehension.

The 2016 paper “Deep Sentence Embedding Using the Long Short Term Memory Network: Analysis and Application to Information Retrieval” introduces a model that performs sentence embedding using LSTM:s, where a sentence is processed one word at a time, and the end result is a semantic “representation” of the given sentence.[8] While the authors emphasize that “the proposed model generates sentence embedding vectors that are specially useful for web document retrieval tasks,” it is nonetheless related to this thesis, in particular in the attempt to “translate” a sequence of inputs (words) into a more compact format (embedding vector). As with most other works, the big difference is that the paper focuses on single sentences, rather than entire documents.

Popularity or success prediction of literary works: Not a lot of research seems to have been done in the area of judging quality or popularity of literary works. Some examples exist, however. The 1953 paper “The content characteristics of bestselling novels” takes a qualitative approach to the problem, but refers to sales, rather than popularity. By necessity (it being “manual”), the focus is on word usage and structure.[9]

“Automatic Metrics for Genre-specific Text Quality” (2012) proposes new metrics for text quality that “utilize the unique properties of different genres.” The paper takes a qualitative approach, with a focus on non-fiction.[10]

Last but not least, “Success with Style: Using Writing Style to Predict the Success of Novels” (2013) by Ashok, V. et al, takes a quantitative approach
to determining the success of novels, as quantified by sales, downloads, received prizes etc., by investigating the writing style. They claim to be the first to take a quantitative approach.\textsuperscript{[2]}

The research done is quite similar, except they use Project Gutenberg and use download count as a measure of success. The paper investigates correlations between the success of a given story and

- distribution of part-of-speech (POS) tags,
- distribution of grammar rules,
- distribution of constituents,

and other aspects. Meanwhile, this thesis takes a generalized approach without a specific focus, giving the model itself the task of finding correlations as best it can. Ashok’s team focuses on finding correlations between specific types of words, finding that “less successful books rely on verbs that are explicitly descriptive of actions and emotions (e.g., ‘wanted’, ‘took’, ‘promised’, ‘cried’, ‘cheered’, etc.), while more successful books favor verbs that describe thought-processing (e.g., ‘recognized’, ‘remembered’), and verbs that serve the purpose of quotes and reports (e.g., ‘say’). Also, more successful books use discourse connectives and prepositions more frequently, while less successful books rely more on topical words that could be almost cliché e.g., ‘love’, typical locations, and involve more extreme (e.g., ‘breathless’) and negative words (e.g., ‘risk’).”\textsuperscript{[2, p. 1757]}

Generally speaking, the kind of task attempted in this thesis already has real world applications, albeit in slightly different shape. For example, Epagogix, a UK based company, performs natural language processing analysis to “create innovative tools and solutions for the hard decisions that senior company directors need to make.” The company claims to develop “customised artificial intelligence solutions,” in order to “help clients to better predict outcomes.”\textsuperscript{[11]}
3 Machine learning

In this chapter we go into detail on the various machine learning technologies used or referred to in this thesis. In particular, we will discuss LSTM:s, and compare these to the memory module described in Chapter 6 (p. 27). In subsequent chapters it is presumed that the reader is familiar with the topics presented here, but explicit references are sometimes made as deemed appropriate.

3.1 Deep and shallow neural networks

A neural network is a universal approximator model with, traditionally, an input layer, a hidden layer, and an output layer, fully connected using so called weights (floating point values). See Figure 3.1. Non-linear activation functions such as tanh or sigmoid are inserted in between layers, which enables the universal approximation property (see section 3.6).\footnote{Indeed, without them, the model couldn’t even solve the XOR problem\cite{13}. This is because two matrices $A$ and $B$ connected without a non-linear function in between them has an equivalent matrix $C \equiv A \times B$. It has been proven that a single layer neural network can only solve linear problems, and since $A \times B \equiv C$ is a single layer, this limitation holds here as well.}

Neural networks with more than one layer (such as the one above) are sometimes called multi-layer perceptrons (MLP:s).

In the last five or so years, great progress has been made in the field of deep neural networks, where the above 3-layer network is considered shallow. Deep neural networks — or DNN’s — comprise multiple hidden layers. Previously, it was considered impossible to train such deep-layered networks, but with the invention of dropout (see section 3.8.1), and the ReLU (rectified linear unit) activation function (section 3.6.3), this has become a possibility.\cite{14}, \cite{15}

All neural networks are trained according to the same principle. Given a set of inputs $X$ and a set of outputs $Y$, $X$ is fed into the network in batches, and a loss function\footnote{There are unsupervised neural networks (without an answer part), which are usually used to discover a better way to represent the actual input. These still follow the same principle, except $X = Y$.} is used to determine the combined error of the outputs in the batch compared to the corresponding values in $Y$. The loss function simply gives a numeric value indicating how “bad” the model is performing (how much it is “losing”), and the aim is to minimize it. A gradient is calculated for the weights in each layer, in reverse order, which

\textsuperscript{3}Sometimes called cost function.
minimizes the loss function, and the weights are updated — trained — to produce a smaller error the next time they see the same batch of inputs.

Neural networks were tested. These were not able to generalize when trained on the short stories, possibly due to underfitting.

### 3.2 Over- and underfitting

Overfitting is when a model, such as a neural network, has so many parameters that it can make special cases for certain parts of the data. This looks good, as the model accuracy increases for these specialized cases, but doesn’t increase the performance of the model. This is because these special cases only apply to the specific data used when training the model, and it will not perform any better on unseen data. In fact, these discrepancies may cause the model to make poor judgements, effectively lowering the resulting accuracy.

Underfitting is when a model has too few parameters to generalize on the input. No matter how much time you spend training the model, it will not be able to come up with a good representation that generalizes the inputs onto the outputs. This is obviously bad, because the model will never converge.

As noted at the end of section 3.1 above, regular neural networks were not able to generalize on the short stories, possibly due to underfitting. However, neural networks extended with something called a memory module were able to generalize to some extent. See Chapter 6 (p. 27) and Chapter 7 (p. 35).
3. Machine learning

![Convolutional neural network](image)

**Figure 3.2:** Convolutional neural network in $\mathbb{R}^2$. The initial convolutional layer generates four *feature maps* from the input. These are subsampled (using e.g. max-pooling) into smaller feature maps, which are passed through additional convolutional layers and subsampled any number of times. [16]

3.3 Training and validation

In principle, presuming $X$ and $Y$ are sufficiently generic and consistent, the network should be able to generalize on the same kind of data not in $X$. In other words, by adjusting its weights for $X$ and $Y$, the network learns a pattern which applies everywhere, not just for $X$. This latter part is confirmed through something called validation, where data has been set aside and kept hidden from the model until it is time to validate it. The model’s performance on the validation data is ultimately the benchmark used to determine its quality.

3.4 Convolutional neural networks

Convolutional neural networks (see Figure 3.2), or CNN’s, are “biologically-inspired variants of MLPs.” [17] They are traditionally used to emulate vision, and are most commonly present in vision related models, such as in image recognition tasks. They are introduced here because one of the alternative memory module architectures makes arbitrary use of convolutional layers (see Chapter 6 (p. 27)). As mentioned in section 1.4 (p. 4), finding a mapping between $\mathbb{R}^{x \times y} \rightarrow \mathbb{R}^n$ was one of the purposes of this research, and this is experimented with using CNN’s.

Convolutional layers take data in e.g. two dimensions as input ($n = 2$), and look at the input, split into smaller pieces, producing a set of new output images called *features*, based on their weights.

Convolutional layers are usually paired with a pooling layer of some kind, often max-pooling (which simply picks the largest value among the candidate values).

It may seem strange to bring up a technology normally used for vision related tasks in a thesis addressing a natural language processing problem, but convolutional pools have been used with success in this field in recent works, such as in “Character-level Convolutional Networks for Text Classification”
3. Machine learning

(2016) by Xiang Zhang et al., although they admittedly only use convolutional layers in one dimension.[18]

3.5 Recurrent neural networks


An RNN is essentially a neural network layer that is “repeated” over each input in a sequence of inputs. Since it is repeated, it only has one set of weights, which applies to every input. RNN:s are particularly suited for natural language processing or other time series processing.

One major issue with RNN:s is that they tend to forget about states as they iterate forward. For example, given the following sentence

He realized he was running a bit late so he picked up the pace.

A regular RNN will often forget that the subject (he) is male when it gets to word 10, despite this knowledge being presented at words 1 and 3.

It was for this particular reason that the LSTM was invented. The LSTM consists of several neural network layers, which determine what to keep and what to forget for each iteration. Consequently, an LSTM would not suffer from the issue above, and would be able to remember the pronoun when it gets to word 10.[19]

The Gated Recurrent Unit (GRU) was “motivated by the LSTM unit but is much simpler to compute and implement.”[21, p. 3]

LSTM:s were tested extensively. While they are powerful tools for teaching a model to recognize semantic and grammatical relations[22], the model did not turn out to perform quite as well when the task required iterating over longer content, such as a short story in its entirety. This was the case both when training the LSTM to predict the rating value directly, and when using an LSTM pre-trained on character prediction inside of a bigger network with the aim of predicting the rating. This is discussed further in Chapter[9] (p. 46).

3.6 Activation functions

A number of activation functions exist which play an important role in neural networks. The activation function is what adds the non-linearity to the network, and without it, the matrix layers of the network could be replaced by an equivalent matrix by performing the multiplication part beforehand (thus rendering the layering part redundant).
3. Machine learning

3.6.1 Sigmoid ($\sigma$)

A common activation function is the sigmoid function in Eqn (3.1). Its derivative is defined in Eqn (3.2).

\[
S(t) = \frac{1}{1 + e^{-t}} \quad (3.1)
\]

\[
S'(t) = S(t)(1 - S(t)) \quad (3.2)
\]

3.6.2 Hyperbolic tangent function (tanh)

Another common activation function is tanh, the hyperbolic tangent function, Eqn (3.3). Derivative in Eqn (3.4).

\[
f(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \quad (3.3)
\]

\[
f'(z) = 1 - (f(z))^2 \quad (3.4)
\]

3.6.3 Rectifier

Recently, the rectified linear unit (ReLU) has become a valuable activation function for use in particular in deep neural networks, standing as the most used activation function in deep learning tasks in 2015. See Eqn (3.5) and the derivative Eqn (3.6).

\[
f(x) = \max(0, x) \quad (3.5)
\]

\[
f'(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{if } x \leq 0 \end{cases} \quad (3.6)
\]

3.6.4 Softmax

Softmax is a general purpose function which "squashes" a $K$-dimensional vector $\vec{z}$ of arbitrary real values to a $K$-dimensional vector $\vec{\sigma}(\vec{z})$ of real values in the range $(0, 1)$ that add up to 1. It is depicted in Eqn (3.7), with the derivative left out.

\[
\sigma(\vec{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}; \quad 1 \leq j \leq K \quad (3.7)
\]

3.7 F-measures

In information retrieval, F-measures — or the F1-score — are used to determine the proficiency of the model, in terms of how accurate it is,
looking at both precision (selecting incorrect items) and recall (including correct items). The F1-score is an average value of the precision and recall as seen in Eqn. (3.8). $P$ is the precision and $R$ is the recall.

$$F1 = \frac{2P \times R}{P + R}$$ (3.8)

In cases where there are multiple sets of F-measures, one can use the micro average method to calculate precision and recall (Eqns (3.9) and (3.10)), where $tp_i$ and $tn_i$ are the number of true positive and true negative values, and $fp_i$ and $fn_i$ are the number of false positive and false negative values, for set $i$).

$$P_{micro} = \frac{\sum_i tp_i}{\sum_i (tp_i + fp_i)}$$ (3.9)

$$R_{micro} = \frac{\sum_i tp_i}{\sum_i (tp_i + fn_i)}$$ (3.10)

The micro average F1 score then is the F1 score of the given precision and recall, according to Eqn. (3.8).

One can also use the macro average method (Eqns (3.11) and (3.12)).

$$P_{macro} = \frac{1}{|P|} \sum_i P_i$$ (3.11)

$$R_{macro} = \frac{1}{|P|} \sum_i R_i$$ (3.12)

$P_i$ is precision $i$ and $R_i$ is recall $i$. Note that $|P| = |R|$.

For sets of roughly equal sizes, the two methods should give roughly the same result. For data where the sets are skewed, i.e. there are more instances of a certain $i$, the micro average method gives a bias toward the dominating set(s), whereas the macro average method gives a bias toward the minority set(s). [26], [27]

3.8 Miscellaneous techniques

3.8.1 Dropout

Dropout is when nodes in a neural network layer are disabled, at random, during training. By forcing the network to learn using only a random subset of its available nodes, it is forced into making more generalized decisions, which in turn reduces overfitting (see section 3.2). [28]

Note that dropout is only used during training. Nodes are not disabled during validation or prediction. To compensate for the weakness induced by the lack of nodes, the node signals are modified while dropout is enabled to give stronger outputs relative to the number of nodes being disabled.

For instance, with a 50% dropout, the signals of every node is doubled.
3.8.2 One-hot vectors

One-hot vectors are vector representations of an index-based array of content. Suppose we have a dictionary of words based on the two sentences “He likes cats.” and “She likes dogs.” We go through this dictionary and assign a number to each new word in order, so we get something like this:

1. he
2. likes
3. cats
4. she
5. dogs

We now want to represent these as one-hot vectors. We have a total of 5 words, so each vector is 5 elements in length.

1. $(1, 0, 0, 0, 0) \rightarrow \text{he}$
2. $(0, 1, 0, 0, 0) \rightarrow \text{likes}$
3. $(0, 0, 1, 0, 0) \rightarrow \text{cats}$
4. $(0, 0, 0, 1, 0) \rightarrow \text{she}$
5. $(0, 0, 0, 0, 1) \rightarrow \text{dogs}$

As you can see, each column only has a 1 in one place. Because of this, when feeding a one-hot vector into a neural network, which has connections between all nodes, we get a one-to-one mapping for each word to its own set of weights, creating an automatic trainable lookup system inside the neural network! For instance, it is trivial to make a matrix that tells us whether the pronoun is he $(1, 0)$, she $(0, 1)$, or neither $(0, 0)$ by creating a $(5 \times 2)$ matrix as such:

$$
P = \begin{pmatrix}
1 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 1 \\
0 & 0
\end{pmatrix}
$$

This gives us

1. $(1, 0, 0, 0, 0) \times P = (1, 0)$
2. $(0, 1, 0, 0, 0) \times P = (0, 0)$
3. $(0, 0, 1, 0, 0) \times P = (0, 0)$
4. $(0, 0, 0, 1, 0) \times P = (0, 1)$
3. Machine learning

Figure 3.3: Top left half of an example of word vectors projected onto $\mathbb{R}^2$. Words like “goddess” and “god” are found in close proximity, as are e.g. numbers.\[29\]

5. $\begin{pmatrix}0,0,0,1\end{pmatrix} \times P = (0,0)$

I.e. the “he” vector $\times P$ gives $(1,0)$, “likes” $\times P = (0,0)$, and “she” $\times P = (0,1)$.

The drawback with this approach is of course that we need a tremendous amount of weights to represent data with large one-hot vectors.

One thing to note is that one-hot vectors are probability distributions which say that there is a 100% probability that the given element (the 1) is true, and a 0% probability that all other elements are true. This ties in well with the softmax activation function mentioned in section 3.6.4 which also generates a probability distribution. It is often the case that one-hot vectors are used as the output, and softmax as the final activation function, for categorization problems (i.e. where each output node represents one category).

3.8.3 Word vectors

Building on the idea of one-hot vectors, which give an automatic trainable lookup system inside a neural network, with the intention of addressing their biggest drawback in that they demand so much space, were the word vectors, followed by the paragraph vectors. Word vectors are dense embeddings representing a distribution of words (a dictionary, or corpus) in an $n$-dimensional space. Word vectors are built from existing texts by looking at the relation between words and attempting to predict surrounding words given a sequence of text.\[30\]–\[32\]

The resulting vectors have some impressive qualities, such as the now famous example of “king” - “man” + “woman” = “queen”. Projecting the word vectors onto a two-dimensional space shows several other similar clusterings appearing naturally through the algorithm. See Figure 3.3.
The length of the word vectors is chosen by the user. This alleviates the issue with several hundred thousand length one-hot vectors, but the one-to-one lookup property is of course lost to some extent.
4 Method

The research was empirical. The given realm has seen very little research (only one other paper takes a quantitative approach to a similar problem), and exploring the feasibility of addressing the problem was a major focus. However, it was also exploratory; a number of follow up questions and problems arose from the results.

The research was quantitative. A large number of short stories were collected and analyzed, using a large number of different models. The analysis was done using neural network models, where the models were sometimes pretrained to predict the next character of a given sequence before they were used to predict the rating value, and sometimes trained directly to predict the rating, with no pretraining.

No hard-coded rules or sub-goals, other than the character prediction used in the pretraining part, were used. Furthermore, as was stated previously, pretraining was not used in all cases.\footnote{The final model presented in Chapter\textsuperscript{7} (p. 35) did not use pretraining.}

The short stories were restricted to specific time periods, based on the “last updated” value, to circumvent bias related to trends. They were divided into three sets: training (75%), test (5%), and validation (20%). Training was used to teach the models, test was used to validate the models in-training, and to track the best achieving model parameters, which were stored away for validation, and validation was used to determine the final accuracy of the resulting models.

Thresholds for the rating value were defined based on the available train set short stories, so that each threshold included an equal number of stories. This introduced an imbalance in the validation set later, and consequently, the validation set was trimmed down to match the train set.

Word vectors and one-hot vectors were both evaluated. In the former case, tokenization (sentences split into words) was evaluated, as was per-character delimited word vectors. For practical reasons, the latter was only evaluated using per-character delimitation.

During training, a model’s current performance was measured as its perfect, near, gnb (good/neutral/bad), and gb (good/bad) values. These were evaluated individually.
4. Method

4.1 Preparatory work

Preparation for the neural network training involved tokenization and word vector generation. This part assumes that data is available in the form of separate, UTF-8 encoded text files. It also assumes that metadata for the popularity or rating of the short stories is available as well.

4.1.1 Tokenization

Two types of tokenization were used: per-character tokenization and corpus-based tokenization.

In the former case, each UTF-8 character was considered to be one token. In the case of word vectors, each character was fed into the word vector library as if it was a word. To achieve this, all data was tokenized by simply inserting a space character in between every single character in each text.

In the latter case, all data used in word vector generation was tokenized using the unidic-neologd corpus included in the Kuromoji software library.[33]

4.1.2 Word vectorization

The word vector library used was Gensim’s doc2vec, using the TaggedLineDocument method. Each short story was preprocessed, converting newlines into tilde (\texttt{/tilde}) symbols. The entire collection of short stories was then fed into doc2vec, one story per line.[34]

The above approach was used for both character based tokenization and corpus based tokenization.

4.1.3 Rating-to-short story mapping

Short stories were associated with a given genre and rating through their filenames. I.e. a file would be named “gg\_rrrrrr\_id.txt”, where \( gg \) is the genre in numeric form (two digits with leading zero), \( rrrrrr \) is the rating in numeric form (six digits with leading zeroes), and \( id \) is the ID of the short story, used to associate it with other metadata properties as appropriate.

4.1.4 Threshold determination

To balance the different outputs in the system, ratings were divided into \( m = 8 \) thresholds, so that each threshold \((0, \cdots, m - 1)\) would have an equal amount of short stories, or alternatively an equal amount of samples.\(^2\)

The algorithm for the former is defined as follows:

\(^2\)These two approaches are different since each short story is of a specific length. Longer short stories will “take up” more space in the latter case, whereas each story will take the same amount of space in the former case.
4. Method

1: function calcThresholds(dir, m)
2:   ratings ← {}
3:   texts ← 0
4:   for f ∈ listdir(dir) do
5:     genre, rating, id ← f.split('_') >> id unused
6:     ratings[rating] ← ratings[rating] + 1 >> ∅ (null) ≡ 0
7:     texts ← texts + 1
8:     idealcount ← texts/m
9:     ccount ← idealcount
10:   thresholds ← []
11:   for r ∈ ratings.keys.sorted do
12:     ccount ← ccount - ratings[r]
13:     if ccount ≤ 0 then
14:        ccount ← idealcount
15:     thresholds.append(r)
16: return thresholds

As noted on line 6, if ratings[rating] is updated for the first time, it becomes 0 + 1 = 1. This is the same as saying “check if ratings contains rating; if it does not, set it to 1, otherwise increment it by 1”.

This function ensures that there are an equal amount of short stories for each of the m nodes. It only applies to the training data, which means the validation data required adjustments (randomly trimming out superfluous input files) to not be imbalanced.

4.2 Sampling data

Data was sampled using a variety of strategies via a tool called the sequencer. This can be thought of as a simple sample generator, but is explained more in detail in section 5.4 on page 26.

4.3 Memory module implementation

The memory module, described later in Chapter 6 on page 27, was implemented according to the following schematic:

1. Create an empty dictionary memdict, and empty arrays Xs, Xmem, and y-rating.

2. While the sequencer has more data:

   (a) Sequence the next chunk from the sequencer, where a chunk consists of an array of (filename, sample, filepos) triples, ending with a null triple

   (b) Preprocess: for each filename not in memdict, add a zeroed memory of the memmod size to memdict for the given filename
(c) Iterate over triples in chunk:
   i. If null triple, or if we have seen the filename before:
      A. Train on \{input = Xs, mem-in = Xmem, rating-out = y-rating\}.
      B. Reset Xs, Xmem, y-rating
      C. Stop iterating if null triple
   ii. Set mem ← memdict[filename]
   iii. Set X ← input based on sample
   iv. Set R ← rating based on rating
   v. Add mem, X, R to Xmem, Xs, y-rating arrays
   vi. Set memdict[filename] to the new mem-out for the given parameters (X, R, mem)

   The above process is repeated for appropriate lengths of time, depending on the various learning strategies used.
5 Short stories

The short stories used in this thesis all came from the same source, a popular online short story site in Japanese which includes metadata such as popularity, rating (the latter defined using bookmark count as well as users voluntarily “grading” short stories), first publication date (used to determine the age of a short story in section 5.3), as well as last updated date (used to determine time ranges in section 5.1).

Beyond the above short stories, the entire collection of classics available at 青空文庫 (Aozora Bunko) was used in certain preprocessing parts.\[35\]

5.1 Selection

Short stories were picked from a variety of genres, including romance, science fiction, fantasy, history, and so on. The stories were purposefully picked mainly from within two specific time ranges\[1\] — March through June 2013, and March through October 2015 — to minimize the interference of external factors related to e.g. trends rising and falling over time (see, however, section 8.3 on page 45). TABLE 5.1 shows the actual number of short stories used over the given time period. TABLE 5.2 shows the distribution over the given genres.

Experiments were made on specific genres, in particular romance (comprising 37.3% of all short stories) and fantasy\[2\] (comprising 18.6%). Experiments were also made indiscriminate of genre.

5.2 Vectorization

Short stories were sometimes tokenized (which, in the case of Japanese, means inserting a separator character (usually space) in appropriate places throughout the text) and turned into a word vector dictionary using paragraph vectors\[32\]. Tokenization was done using the “unidic neologd” corpus included in Kuromoji\[33\].

In some cases, the vectorization not only included the input content, but also all the content from Aozora Bunko (青空文庫), the Japanese equivalent

---

\[1\]The time range applies to the last time the short story was updated.
\[2\]Fantasy was preferred over literature even though the latter had more short stories, simply because the former is a more defined genre, perceived to be easier to learn by the system. Chapter 2 (p. 35) discusses this further, but genre-specific training ultimately turned out to be inferior to indiscriminate training.
5. Short stories

<table>
<thead>
<tr>
<th>Month</th>
<th>2013</th>
<th>2014</th>
<th>2015</th>
<th>*</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan</td>
<td>1</td>
<td>46</td>
<td>25</td>
<td>72</td>
<td>0.6</td>
</tr>
<tr>
<td>Feb</td>
<td>2</td>
<td>36</td>
<td>17</td>
<td>55</td>
<td>0.4</td>
</tr>
<tr>
<td>Mar</td>
<td><strong>533</strong></td>
<td>51</td>
<td><strong>2342</strong></td>
<td>2926</td>
<td>23.9</td>
</tr>
<tr>
<td>Apr</td>
<td><strong>586</strong></td>
<td>29</td>
<td><strong>1724</strong></td>
<td>2339</td>
<td>19.1</td>
</tr>
<tr>
<td>May</td>
<td><strong>625</strong></td>
<td>25</td>
<td><strong>2014</strong></td>
<td>2664</td>
<td>21.8</td>
</tr>
<tr>
<td>Jun</td>
<td><strong>551</strong></td>
<td>31</td>
<td><strong>1931</strong></td>
<td>2513</td>
<td>20.6</td>
</tr>
<tr>
<td>Jul</td>
<td>134</td>
<td>19</td>
<td><strong>484</strong></td>
<td>637</td>
<td>5.2</td>
</tr>
<tr>
<td>Aug</td>
<td>79</td>
<td>29</td>
<td><strong>269</strong></td>
<td>377</td>
<td>3.1</td>
</tr>
<tr>
<td>Sep</td>
<td>53</td>
<td>13</td>
<td><strong>222</strong></td>
<td>288</td>
<td>2.4</td>
</tr>
<tr>
<td>Oct</td>
<td>43</td>
<td>22</td>
<td><strong>126</strong></td>
<td>191</td>
<td>1.6</td>
</tr>
<tr>
<td>Nov</td>
<td>35</td>
<td>30</td>
<td>24</td>
<td>89</td>
<td>0.7</td>
</tr>
<tr>
<td>Dec</td>
<td>35</td>
<td>19</td>
<td>22</td>
<td>76</td>
<td>0.6</td>
</tr>
<tr>
<td>Sum</td>
<td>2677</td>
<td>350</td>
<td>9200</td>
<td>12227</td>
<td>—</td>
</tr>
<tr>
<td>%</td>
<td>21.9</td>
<td>2.9</td>
<td>75.2</td>
<td>—</td>
<td>100.00</td>
</tr>
</tbody>
</table>

Table 5.1: Distribution of short stories over time in the selected three year period 2013 through 2015.

<table>
<thead>
<tr>
<th>Genre</th>
<th>2013</th>
<th>2014</th>
<th>2015</th>
<th>Sum</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adventure</td>
<td>33</td>
<td>7</td>
<td>47</td>
<td>87</td>
<td>0.7</td>
</tr>
<tr>
<td>Comedy</td>
<td>188</td>
<td>30</td>
<td>236</td>
<td>454</td>
<td>3.7</td>
</tr>
<tr>
<td>Essay</td>
<td>28</td>
<td>5</td>
<td>163</td>
<td>196</td>
<td>1.6</td>
</tr>
<tr>
<td>Fairy-tale</td>
<td>22</td>
<td>2</td>
<td>42</td>
<td>66</td>
<td>0.5</td>
</tr>
<tr>
<td>Fantasy</td>
<td>1013</td>
<td>104</td>
<td>1154</td>
<td>2271</td>
<td>18.6</td>
</tr>
<tr>
<td>History</td>
<td>21</td>
<td>6</td>
<td>44</td>
<td>71</td>
<td>0.6</td>
</tr>
<tr>
<td>Horror</td>
<td>62</td>
<td>10</td>
<td>80</td>
<td>152</td>
<td>1.2</td>
</tr>
<tr>
<td>Literature</td>
<td>197</td>
<td>19</td>
<td>2925</td>
<td>3141</td>
<td>25.7</td>
</tr>
<tr>
<td>Military</td>
<td>36</td>
<td>7</td>
<td>44</td>
<td>87</td>
<td>0.7</td>
</tr>
<tr>
<td>Mystery</td>
<td>26</td>
<td>6</td>
<td>25</td>
<td>57</td>
<td>0.5</td>
</tr>
<tr>
<td>Poetry</td>
<td>19</td>
<td>0</td>
<td>79</td>
<td>98</td>
<td>0.8</td>
</tr>
<tr>
<td>Romance</td>
<td><strong>650</strong></td>
<td><strong>91</strong></td>
<td><strong>3823</strong></td>
<td><strong>4564</strong></td>
<td><strong>37.3</strong></td>
</tr>
<tr>
<td>SF</td>
<td>118</td>
<td>31</td>
<td>134</td>
<td>283</td>
<td>2.3</td>
</tr>
<tr>
<td>School</td>
<td>147</td>
<td>15</td>
<td>141</td>
<td>303</td>
<td>2.5</td>
</tr>
<tr>
<td>Other</td>
<td>117</td>
<td>17</td>
<td>263</td>
<td>397</td>
<td>3.2</td>
</tr>
<tr>
<td>*</td>
<td>2677</td>
<td>350</td>
<td>9200</td>
<td>12227</td>
<td>—</td>
</tr>
<tr>
<td>%</td>
<td>21.9</td>
<td>2.9</td>
<td>75.2</td>
<td>—</td>
<td>100.00</td>
</tr>
</tbody>
</table>

Table 5.2: Distribution of short stories over genres in the selected three year period 2013 through 2015.
Table 5.3: Discarded tokens based on minimum occurrence in source corpora

<table>
<thead>
<tr>
<th>MIN OCC</th>
<th>DISCARDED</th>
<th>TOTAL</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>415702</td>
<td>570753</td>
<td>72.8%</td>
</tr>
<tr>
<td>30</td>
<td>438943</td>
<td>570753</td>
<td>76.9%</td>
</tr>
<tr>
<td>200</td>
<td>514635</td>
<td>570753</td>
<td>90.2%</td>
</tr>
<tr>
<td>500</td>
<td>536897</td>
<td>570753</td>
<td>94.1%</td>
</tr>
<tr>
<td>1000</td>
<td>549169</td>
<td>570753</td>
<td>96.2%</td>
</tr>
</tbody>
</table>

of Project Gutenberg, where classical literary works are made available to the general public free of charge. [35], [36]

Vectorization was done with different sets of parameters, which define how frequently a word must occur to be included in the resulting dictionary. Words excluded are turned into “unknown” tokens. See Table 5.3. A minimum occurrence of 30 gave the best results.

5.3 Rating

A number of different functions for rating were tried, such as the bookmark count of the short story, as well as the site specific rating, which included not only bookmark count but also ratings given by readers. User ratings are divided into two parts: grammar and word usage, and content. Each type has a rating between 0 and 5.

The latter function turned out to give the best results and was chosen for the duration of the final evaluation. Because the age of a short story directly affected its rating (the longer a story existed, the higher its rating tended to be), the rating was transformed according to Eqn. (5.1), where age(ℓ) gives the age of the short story, in days.

\[
r'(ℓ) = 1000 \frac{r(ℓ)}{\text{age}(ℓ)} \quad (5.1)
\]

The new rating, dependent on the age of the story, would now give “rating per day” (in thousandths, which gave ratings in the interval \([0 \ldots 509857]\), i.e. the most rapidly increasing short story (in terms of rating) received an average of 510 points per day.

This rating was passed through a threshold mapping which ensured that the number of short stories for each type of rating was the same. With 8 nodes, the mapping would look like in Eqn. (5.2).

\[
\tau(r) \to (0..7) \quad (5.2)
\]
5. Short stories

In practice, this is implemented as in Eqn. (5.3), where a boolean operation returns a 1 for truth and a 0 for falsity. $z_0, \cdots, z_6$ are the respective numeric thresholds for each node.

$$\tau(r) = \sum_{i=0}^{6} (r > z_i)$$  \hspace{1cm} (5.3)

5.4 Sequencing

A variety of methods for presenting the data to the model in the training phase were tried. We refer to the selection of samples as sequencing, and each method is referred to as a sequencing strategy.

We have a collection of training data $S$ consisting of $\omega$ short stories $S_0, \cdots, S_\omega$, which are in turn composed of a sequence of tokens, e.g. $s_{0,0}, \cdots, s_{0,n_0}$. We have a set of samples $Q$ which contains all so far seen samples. For each selected sample $q$, we update $Q$ so that $Q \leftarrow Q \cup \{q\}$. We require that $q \notin Q$ for all $q$ prior to this update operation. The following sequencing strategies were tested:

- **Random**: the next sequence in the training data is $q = s_{i, \text{rand}(0..n_i)}$, where $i = \text{rand}(0..\omega)$.

- **Universal-Iterate**: define an indexer $I(i)$ for each short story. Initially, $I(i) = 0$ for all $i$. Until we have reached the desired number of samples, iterate $i$ over each short story in order and set $q = s_{i,I(i)}$ and let $I(i) \leftarrow I(i) + 1$. For all cases where $I(i) > |S_i|$, skip over that entry.

- **Universal-Iterate-Spin**: works like Universal-Iterate, except when $I(i) > |S_i|$, let $I(i) \leftarrow 0$, i.e. start over from the first token in all short stories that have been exhausted.

- **Universal-Iterate-Waves**: given an initial wave length $w_i$, a wave growth $w_g$, and a wave modifier $w_m$, start by setting the cap $w = w_i$, and read from each input one token at a time just like Universal-Iterate above. Once the $w$th token has been read, reset $I(i)$ to 0 for all $i$ and set $w \leftarrow w \cdot w_m + w_g$.

- **Universal-Expand**: define a counter $C = 0$. For each sequencer request of samples, increase $C$ by 1, and read samples from the first $C$ files until the desired sample count is reached. If EOF (end of file) is reached, increase $C$ by 1. Uses the $I(i)$ indexer defined above as normal.

In section 6.3 (p. 30) we discuss the Universal-Iterate-Waves approach more in detail.
6 Memory module

In this chapter, the memory module is introduced. The memory module was invented during the research to address the general perceived underfitting problem with non-recursive neural network models and the problem with recursive neural networks’ bad performance when training on longer sequences of text. In particular, it addresses the second and third purposes mentioned in section 1.4 (p. 4) — to investigate alternative, more robust approaches for analyzing very long sequences of data without suffering from forgetfulness, and to determine if we would benefit from performing a mapping $\mathbb{R}^{x\times y} \rightarrow \mathbb{R}^{n}$.

Several other techniques related to training a memory module are introduced as well, such as muted softmax (section 6.2) and waves (section 6.3). Finally, we go into detail on the memory module implementation in section 6.4.

6.1 Memory module

The memory module (see Figure 6.1) is a regular neural network model with a persistent memory attached to it. There is one memory instance for each problem (short story).

Each iteration takes one token from the input short story. The token representation and the current memory module state are both given as two separate inputs into a graph model (MEM-IN and Input 0 in Figure 6.1). The model processes the inputs and eventually merges the old memory module together with the resulting state for the current token into a new memory state (MEM-OUT). From this state, a response is derived (RATING). This is done via back propagation as normal.

The new memory state is then used as part of the input for the next token in the sequence (Input 1). In other words, the memory module is updated through the neural network and retained for each iteration.

A number of additions or tweaks to existing algorithms were necessary to facilitate the memory module training phase, namely muted softmax (section 6.2) and waves (section 6.3). The memory module implementation is also discussed more in detail in section 6.4.

6.2 Muted softmax

Regular softmax transforms a vector $\tilde{z}$ of $K$ values $z_1, z_2, \ldots, z_K$ into a probability distribution $\tilde{z}_*$ where $\sum_{z_*, \in \tilde{z}_*} z_* = 1.0$ according to Eqn. (3.7).
6. Memory module

Figure 6.1: Memory module overview: each token is processed left to right, with an output memory state as a byproduct of processing a rating for the given problem up until the given token.

Figure 6.2: Muted softmax for $2 \leq n \leq 10$. Red line shows incorrect ($i \neq j$), blue line shows correct ($i = j$) node.
Softmax is used to categorize things where the answer is usually the highest probability entry in $\vec{z}_*$. It is obvious that $\text{argmax}(\vec{z}_*) = \text{argmax}(\vec{z})$, so why bother using softmax? The reason lies in the training part. The input is given as a one-hot vector — a vector $\vec{w}$ with a single value at 1 (the right answer) and all other values at 0. By applying softmax, the loss function calculated e.g. as the difference between the output vector $\vec{z}_*$ and the given answer $\vec{w}$ becomes more stable: if the previous layer (which generated $\vec{z}$) has an intense reaction to a given input and pushes the right answer node to a value of, say, 300, without softmax this would be considered 299 points away from the right answer, because the answer is 1.0 and $|300.0 - 1.0| = 299$, whereas with softmax, this intense peak at the correct value would push the other values toward zero and the given answer toward 1, matching the one-hot vector more closely.

One problem with using one-hot vectors and softmax is that we are expecting a certain consistency in the input/output mapping. With how the short story popularity prediction was done here, this consistency was lacking in particular in the initial iterations, which caused problems with the network. Consider the following inputs:

- Story 0: “Once upon a time, there was a ···”; popularity value 3. $\vec{w}_0 = \{0, 0, 0, 1, 0, 0, 0, 0\}$
- Story 1: “Once upon a time, a young man ···”; popularity value 0. $\vec{w}_1 = \{1, 0, 0, 0, 0, 0, 0, 0\}$
- Story 2: “Once upon a time, there lived a ···”; popularity value 7. $\vec{w}_2 = \{0, 0, 0, 0, 0, 0, 0, 1\}$
- ...

Although each story looks about the same in the beginning, the popularity value (the answer) differs widely. Attempting to teach a network that these stories have their respective popularity values ultimately resulted in malfunctions. The model was basically told that “Once upon a time” had the answer of all three versions of $\vec{w}$ above.

As the network progressed through a short story, the probability that it would look identical (i.e. have an identical memory state) to another story with a different popularity value diminishes rapidly. It is thus in this early stage that muted softmax became necessary. Muted softmax is defined as in Eqn (6.1).

$$sm_m(t, i, j) = \begin{cases} \frac{t}{|s|} + \frac{1}{n}, & \text{if } i = j \\ \frac{1}{n} & \text{otherwise} \end{cases} \quad (6.1)$$

1In the form of not-a-number or infinity values in the network model weights.
6. Memory module

We will briefly prove that Eqn. (6.1) produces a probability distribution with a sum of 1 at all times.

**Lemma 1.** Muted softmax produces a probability distribution with a sum of 1.

**Proof.** With \( n \) being the number of output nodes (i.e. the length of the \( \vec{w} \)-vector), \( i \) being the correct answer, \( t \) being the index of the token being processed, \( \vec{s} \) being the short story being processed, and \( |\vec{s}| \) being the number of tokens in \( \vec{s} \),

\[
\sum_{j=1}^{n} sm_m(t, i, j) = sm_m(t, i, i) + (n - 1)sm_m(t, i, j; j \neq i) \quad (6.2)
\]

\[
= \frac{t}{|\vec{s}|} + \frac{1 - \frac{t}{|\vec{s}|}}{n} + (n - 1)\frac{1 - \frac{t}{|\vec{s}|}}{n} \quad (6.3)
\]

\[
= \frac{t}{|\vec{s}|} + n \frac{1 - \frac{t}{|\vec{s}|}}{n} = \frac{t}{|\vec{s}|} + 1 - \frac{t}{|\vec{s}|} = 1 \quad (6.4)
\]

While softmax is applied to the resulting vector of the neural network, i.e. \( \vec{z} \), the muted softmax function is applied to the one-hot vector \( \vec{w} \), i.e. the provided answer in the training phase. Softmax is still used as the activation function in the network as normal. By using muted softmax instead of one-hot vectors as the desired output, the inconsistency issues present above are alleviated.

Figure [6.2] shows what the muted softmax looks like for all \( t \) for the two cases \( i = j \) and \( i \neq j \), for different values of \( n \). While regular one-hot vectors are fixed, the muted softmax is a function of the time proportional to the length of the story. At the last token (1.0 in the figure), the muted softmax converges with the one-hot vector. Depending on \( n \), the initial value differs, but in all cases, with \( t \to 0 \), it follows that \( sm_m(t, i, j) \to \frac{1}{n} \) for all \( j \).

Consequently, the popularity rating for all short stories is “each rating equivalently probable”, and this moves towards the one-hot vector equivalent over time. The break-down observed in the weights of the model stopped occurring with this change.

### 6.3 The waves approach

The sequencer strategy described in section [5.4 (p. 26)] that ultimately worked best was UNIVERSAL-ITERATE-WAVES. We describe it in more detail in this section.

Consider a completely randomized set of weights whose task is to update the memory state based on a state derived from the next token in line. The network produces an updated memory state which is stowed away, after which the network is updated. Updating these weights using back
propagation, the previously stored memory state is now slightly “outdated”, as it would have had different values with the updated weights. The error introduced in the previous iteration accumulates, spilling over into future iterations. Not only is the output wrong, but the input is wrong as well.

This discrepancy in the memory state accumulates for each iteration of the given short story, not only in the resulting updated state, but also in that the input memory state, from which the new state is derived, is already outdated to begin with.

Although unwieldy, the approach chosen to address this problem is here referred to as the waves approach (see Figure 6.3). It is very simple and is comprised of two parts: firstly, we train the network on successively longer pieces of the short stories, starting with a small number of tokens and then gradually increasing the count. I.e. with \( \omega = |\mathbb{S}| \) being the number of short stories, instead of training on all of \( \mathbb{S}_\ell \) for some \( \ell \), we train on \( \vec{s}_{0,0} \cdot \vec{s}_{\omega,10} \), then reset the memory state and start again at \( \vec{s}_{0,0} \) this time moving up to, say, \( \vec{s}_{\omega,20} \). We keep on resetting and increasing the length like this, with the intention of minimizing the discrepancy observed in the memory states as the model iterates through stories.

The second part of the waves approach lies in how the data is arranged. With \( \mathbb{S} \) being a matrix in \( \mathbb{R}^3 \) of all short stories and their respective tokens, the waves approach iterates through all stories at once. I.e. the model trains on token 1 for all \( \mathbb{S}_i \in \mathbb{S} \), then token 2, and so on, until it has reached the current cap. As such, there are \( \omega \) memory states in existence simultaneously.

The aim with this approach is to minimize the error in particular in earlier iterations, to increase the stability as the network iterates over the input.
6.4 Memory module implementation

We define a generic activation function as $\| ( \cdot )$ and a dense neural network layer as $\# ( i )$. Combinations are interpreted right to left, i.e. the combination $\# \| i$ means $i$ passed through the activation function $\|$, then passed into a dense neural layer as defined in $\#$, whereas $\# \| # i$ means $i$ passed through a neural network layer, whose results are then passed into an activation function.

The memory module at time step $t$ is generated from two equally sized sets of inputs, merged e.g. via concatenation, via a dense neural layer as in Eqn. (6.5), where $M^o_t$ is the updated memory, $M^i_t$ is the input memory, and $M^s_t$ is a state map based solely on the input token at $t$.

$$M^o_t = \# \left[ \begin{array}{c} M^i_{t,0} \\ M^i_{t,1} \\ \vdots \\ M^i_{t,m} \\ M^s_{t,0} \\ M^s_{t,1} \\ \vdots \\ M^s_{t,m} \end{array} \right]$$ (6.5)

Eqn. (6.5) insinuates that there is a $(2 \times m)$ matrix, but in reality this is a simple vector of length $2m$ with $M^i_t$ followed by $M^s_t$.

The model is recursive (see example in Figure 6.1). With $M^o_t = M^o_{t-1}$, we get

$$M^o_t = \# \left[ \begin{array}{c} M^o_{t-1} \\ M^s_t \end{array} \right] = \# \left[ \begin{array}{c} \# \left[ M^o_{t-2} \right] \\ \# \left[ M^s_{t-1} \right] \end{array} \right]$$ (6.6)

With $M^o_0 = \vec{O}$, we end up with

$$M^o_t = \# \left[ \begin{array}{c} \# \left[ \# \left[ \ldots \# \left[ \vec{0} \right] \right] \right] \\ M^s_{t-1} \\ M^s_t \end{array} \right]$$ (6.7)

Unfolding it this way, we get a very large recursive neural network which, using the same weights each time step, generates a representation in $M$ with the intent of optimizing the accuracy of the rating prediction.

The state component is completely arbitrary (see Figure 6.4), as long as it follows two rules: (1) it takes the input token at time step $t$ or in some range, in the case of e.g. an LSTM, and (2) it outputs a set of nodes that match the size of the memory module shape. As an example, we may have

$$M^s_t = \# \# \# ( i_t )$$ (6.8)

Dropout was left out of these representations, but was used in most of the dense layers.

Attempting to derive the transformation process of a single node in the memory module is convoluted, due to (1) the recursive nature of $M^o$ and (2) the interconnectivity between nodes — every node in the output $M^o_t$ is affected by every node in the input $M^i_t$ and every node in the state $M^s_t$. In
other words, a node in the memory module output at timestep $t$ is affected by every node in every memory state all the way back to timestep 1.

6.5 LSTM and memory module

At first glance, the memory module looks like a trimmed down version of a recurrent neural network (RNN), such as the LSTM or GRU. There are some fundamental differences, however. Figure 6.5 shows an example of an LSTM designed using a memory module and regular neural network layers. It is not possible to emulate a memory module using an LSTM or other recurrent neural network in this manner. Differences include:

- Memory modules are more generic than LSTM units, but architecturally simpler.
- Memory modules require more computing power and time compared to LSTM units.
- As seen in Figure 6.5, you can “emulate” an LSTM using a memory module but the reverse is not possible.
Figure 6.5: LSTM implemented using a neural network with a memory module component. $x$ indicates layer multiplication (component-wise), $+$ indicates addition, $\sigma$ is sigmoid activation, and $tanh$ is tanh activation.
7 Results

The results are described using a top-down hierarchical approach, starting with accuracy definition (section 7.1) and model selection (section 7.2), moving down to individual parameter tuning (section 7.3 and onward).

7.1 Measuring accuracy

Accuracy was measured in four granularities with decreasing level of difficulty. These granularities are here sometimes referred to as $n$-choice granularity, where $n$ is the number of available choices. With $A$ being the answer, $C$ being the correct value, and with outputs as $m = 8$ distinct nodes in the range $(0..7)$,

- **Perfect** is where $A = C$ (8-choice granularity),
- **Near** is where $|A - C| \leq 1$, i.e. the given answer is at most one step away from the correct value (no $n$),
- **Gnb** (good/neutral/bad) is where the nodes are divided into three groups $\vec{g} = (0..2, 3..4, 5..7)$ and where $g^{-1}(A) = g^{-1}(C)$, i.e. where the group index of $A$ must be equal to the group index of $C$. In other words, if $C \in (0..2)$ then $A$ must also be in $(0..2)$ (3-choice),
- **Gb** (good/bad) is where the nodes are divided into two groups $\vec{g} = (0..3, 4..7)$. I.e. $A$ must be in the same half as $C$ to be considered correct (2-choice).

A natural comparison for these is randomness. With $m = 8$, the base case becomes

- **Perfect** $= \frac{1}{8} = 12.5\%$,
- **Near** $\approx \frac{1}{8} \left(2 \frac{1}{4} + 6 \frac{3}{8}\right) \approx 34\%$,
- **Gnb** $\approx \frac{1}{8} \left(6 \frac{3}{8} + 2 \frac{3}{8}\right) \approx 34\%$,
- **Gb** $= \frac{1}{2} = 50\%$.

As the results were not very good for the finer granularities (higher $n$), the results in subsequent sections focus on the results for the 2-choice granularity Gb. In section 7.7 (p. 41), the complete results are listed.
7. Results

### Table 7.1: Accuracy of different models, as well as the base case, i.e. a random predictor.

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FIN</td>
<td>Avg</td>
<td>Sum</td>
</tr>
<tr>
<td><strong>Base</strong></td>
<td>50%</td>
<td>50%</td>
<td>50%</td>
</tr>
<tr>
<td><strong>LSTM</strong></td>
<td>43.48%</td>
<td>43.48%</td>
<td>34.78%</td>
</tr>
<tr>
<td><strong>MemMod</strong></td>
<td>64.30%</td>
<td>68.70%</td>
<td>69.07%</td>
</tr>
<tr>
<td><strong>MMConv2D-0</strong></td>
<td>66.12%</td>
<td>66.70%</td>
<td>67.67%</td>
</tr>
<tr>
<td><strong>MMConv2DX2-0</strong></td>
<td><strong>68.68%</strong></td>
<td><strong>69.12%</strong></td>
<td><strong>69.52%</strong></td>
</tr>
</tbody>
</table>

7.2 Model selection

Initial efforts went into training an LSTM network to understand text, and to use the pretrained LSTM component to predict the popularity of short stories. Even with LSTMs trained over long periods of time (months) on a K20 GPU, using the resulting LSTM to predict popularity did not work well at all.

Training an LSTM to predict popularity directly, i.e. skipping the pretrain step of predicting the next character in the sequence of characters, did not yield good results either.

Using a regular neural network (without an LSTM component) would quickly break the model, hitting NaN (not a number) or INF (infinity) values in the model weights. This happened even with networks with a large number of nodes per layer (4k+), and happened both with word vectors (300 length) and one-hot vectors (around 4k in length). See Chapter 7.3.

Extending a regular neural network with a memory module and using muted softmax (see section 6.2 (p. 27)) and waves (6.3) prevented the model from breaking in most cases, although it would still break in certain cases. See Chapter 7.4.

There were a number of models using the memory module included in the evaluation. Their accuracy, as well as that of an LSTM implementation, are listed in Table 7.1.

- **LSTM** is a regular LSTM model with 1k nodes, and a 4-layer processing component, training on 100 sequence long batches at a time.
- **MemMod** is a memory module as depicted in the example Figure 6.4 (p. 33).
- **MMConv2D-0** is a memory module with a single convolutional pool component (2D). The 0 stands for “no post-processing layers”\(^1\).

---

\(^1\)All memory module extended networks with convolutional pooling suffered from a severe drop in accuracy whenever post processing layers were present. See Chapter 9 (p. 46).
7. Results

• MMConv2DX2-0 is identical to the aforementioned model, except it has two convolutional pool components, directly connected to each other, instead of one.

7.3 Word vectors vs one-hot vectors

Training an LSTM on word vectors was in general giving worse performance for character prediction than an LSTM trained on one-hot vectors. There was no noticeable difference between using word vectors and using one-hot vectors when trying to teach an LSTM to predict a popularity score directly without pretraining.

The memory module extended neural network performed better with word vectors than one-hot vectors.

7.4 Softmax nodes vs real output

A number of different approaches were tested:

• Prob: probability softmax rating with $\tau$ thresholds; highest probability wins all,

• 7-Thresh: each rating upgrade adds a one from left to right, i.e. a rating of 3 gives $\{1, 1, 1, 0, 0, 0, 0\}$,

• 3-Bin: the rating is $n_0 + 2n_1 + 4n_2$, i.e. $111 = 7$, $001 = 1$, $010 = 2$, etc.,

• Float$^+$: a floating value between 0 and 1,

• Float$^\pm$: a floating value between $-1$ and 1.

Using a single floating point value as the output (Float$^+$), where $[0.0, \frac{1}{8} = 0.125]$ is a 0 rating, $[\frac{1}{8}, \frac{2}{8} = 0.25]$ is a 1 rating, and so on, tended to break the model after about a day of training, hitting NaN values as the output. The same goes for the Float$^\pm$ variant. Instead using 8 separate output nodes, and using softmax to generate a probability distribution of the most probable rating (Prob) worked the best, and was ultimately used in the final results.

This approach has its drawbacks, of course. There is no inherent correlation between a rating of 0 and a rating of 1, which would be desirable. A solution to this was attempted in 7-Thresh but this did not produce good results. 3-Bin was also outperformed by the regular Prob.

There were some issues resulting from these approaches. The models tended to focus on two nodes, mostly ignoring the other ones. See Tables 7.2 through 7.4.
7.5 Genre-specific vs general-purpose

Surprisingly, training and testing on all genres produced significantly better results than training on a given genre.

The most probable reason for this is that there simply wasn’t enough input for any given genre to sufficiently train a memory module enabled network. A more optimistic reason may be that the resulting model is general enough that genre specific differences don’t have a great enough impact to benefit the genre-specializing model.

7.6 End, average, and sum predictors

When predicting, for each input token, a new memory state was generated, and from said memory state a rating was predicted. Three methods were tried to determine the resulting rating of the short story as a whole:

- End state: only the end state, and consequently only the final rating value, is used. This makes sense if you expect the memory state to consistently track the popularity of the short story, and to retain this information while iterating. The approach ignores the fact that the network produces a rating for each iteration during training, however.

- Average state: the rating which received the most counts “wins” — e.g. if the predictions were 0, 3, 1, 3, 2, the resulting answer would be 3, because it has the highest occurrence count (2).

- Sum state: track the raw outputs (as floating point values) for each output node, and sum these together each iteration. The output node with the highest value indicates the winner. This differs from the average state above, in that outputs which are consistently high but not necessarily the highest output may end up the winner. It is arguably the most sensitive and accurate method of the three, and as can be noted in Table 7.4, this turned out to be the case in practice.

Table 7.2 shows the confusion matrix for the end state predictor, Table 7.3 for the average state predictor, and Table 7.4 for the sum state predictor. The sum state predictor has the highest accuracy, at 69.52%, with average coming in second at 69.12% and end last with 68.68%.

Table 7.5 shows the F-measures and micro/macro F1 averaged scores for the sum state predictor. The micro and macro averages are fairly similar, and are close in proximity to the resulting accuracy of 69.52% as well (Table 7.4).

It should be noted that all three tables 7.2 through 7.4 are derived from the same model, MMConv2DX2-0, using different predictors.
### Table 7.2: Confusion matrix, end state. Top shows all nodes, bottom shows good/bad distribution. Bottom indicates the end state had a 68.68% accuracy.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>Tot</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>35</td>
<td>0</td>
<td>451</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>89</td>
<td>581</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>0</td>
<td>116</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>56</td>
<td>184</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>0</td>
<td>135</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>47</td>
<td>190</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>0</td>
<td>120</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>46</td>
<td>180</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>0</td>
<td>176</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>113</td>
<td>311</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>0</td>
<td>124</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>150</td>
<td>289</td>
</tr>
<tr>
<td>6</td>
<td>17</td>
<td>0</td>
<td>76</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>224</td>
<td>320</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>0</td>
<td>34</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>175</td>
<td>215</td>
</tr>
<tr>
<td>Tot</td>
<td>114</td>
<td>0</td>
<td>1232</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>24</td>
<td>900</td>
<td>2270</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0-3</th>
<th>4-6</th>
<th>Tot</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-3</td>
<td>885</td>
<td>250</td>
<td>1135</td>
</tr>
<tr>
<td>4-7</td>
<td>461</td>
<td>674</td>
<td>1135</td>
</tr>
<tr>
<td>Tot</td>
<td>1346</td>
<td>924</td>
<td>2270</td>
</tr>
</tbody>
</table>

### Table 7.3: Confusion matrix, average state. Bottom shows a 69.12% accuracy.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>Tot</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>25</td>
<td>0</td>
<td>478</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>76</td>
<td>581</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>0</td>
<td>132</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>47</td>
<td>184</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>0</td>
<td>141</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>44</td>
<td>190</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0</td>
<td>131</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>46</td>
<td>180</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>0</td>
<td>187</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>114</td>
<td>311</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>0</td>
<td>137</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>141</td>
<td>289</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>0</td>
<td>90</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>220</td>
<td>320</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>0</td>
<td>41</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>170</td>
<td>215</td>
</tr>
<tr>
<td>Tot</td>
<td>67</td>
<td>0</td>
<td>1337</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>858</td>
<td>2270</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0-3</th>
<th>4-6</th>
<th>Tot</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-3</td>
<td>919</td>
<td>216</td>
<td>1135</td>
</tr>
<tr>
<td>4-7</td>
<td>485</td>
<td>650</td>
<td>1135</td>
</tr>
<tr>
<td>Tot</td>
<td>1404</td>
<td>866</td>
<td>2270</td>
</tr>
</tbody>
</table>
7. Results

Table 7.4: Confusion matrix, sum state. Bottom shows a 69.52% accuracy.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>Tot</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>31</td>
<td>0</td>
<td>470</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>79</td>
<td>581</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>0</td>
<td>126</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>49</td>
<td>184</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>0</td>
<td>140</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>44</td>
<td>190</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0</td>
<td>128</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>49</td>
<td>180</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>0</td>
<td>187</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>115</td>
<td>311</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>0</td>
<td>132</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>145</td>
<td>289</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>0</td>
<td>84</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>225</td>
<td>320</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>0</td>
<td>40</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>172</td>
<td>215</td>
<td></td>
</tr>
<tr>
<td>Tot</td>
<td>72</td>
<td>0</td>
<td>1307</td>
<td>0</td>
<td>0</td>
<td>13</td>
<td>878</td>
<td>2270</td>
<td></td>
</tr>
</tbody>
</table>

Table 7.5: F-measures and micro/macro F1 averaged scores for the sum state predictor.

<table>
<thead>
<tr>
<th>Set</th>
<th>Precision</th>
<th>Recall</th>
<th>F1-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-3</td>
<td>66.06%</td>
<td>80.26%</td>
<td>72.47%</td>
</tr>
<tr>
<td>4-7</td>
<td>74.86%</td>
<td>58.77%</td>
<td>65.74%</td>
</tr>
<tr>
<td><strong>Micro</strong></td>
<td><strong>69.52%</strong></td>
<td><strong>Macro</strong></td>
<td><strong>69.16%</strong></td>
</tr>
</tbody>
</table>
Table 7.6: Accuracy of different models for the different granularities.

<table>
<thead>
<tr>
<th>Model</th>
<th>Perfect</th>
<th>Near</th>
<th>Gnb</th>
<th>Gb</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base</td>
<td>12.5%</td>
<td>34%</td>
<td>34%</td>
<td>50%</td>
</tr>
<tr>
<td>LSTM</td>
<td>8.70%</td>
<td>34.78%</td>
<td>34.78%</td>
<td>34.78%</td>
</tr>
<tr>
<td>MemMod</td>
<td>18.81%</td>
<td>42.60%</td>
<td>58.37%</td>
<td>69.07%</td>
</tr>
<tr>
<td>MMConv2D-0</td>
<td>15.24%</td>
<td>36.83%</td>
<td>58.37%</td>
<td>69.07%</td>
</tr>
<tr>
<td>MMConv2DX2-0</td>
<td>15.24%</td>
<td>36.83%</td>
<td>58.55%</td>
<td>69.52%</td>
</tr>
</tbody>
</table>

7.7 Complete results

Table 7.6 shows the results for all granularities defined in Chapter 7.1 above, for the Sum predictor. The MemMod model outperforms the other models slightly at the higher granularities, but is surpassed by MMConv2DX2-0 for the lower half. The highest increase from the base (random) case is +24.55% at MMConv2DX2-0:Gnb (58.55% – 34%), an increase by 72.2%.

The reasons for the poor performance by the LSTM are not entirely clear, and are mentioned in Chapter 9 (p. 46).
8 Analysis

8.1 Model performance

No model was able to reliably predict the popularity of a short story at higher granularities. Even at the lowest, 2-choice granularity (Gb, good/bad), the highest performance seen did not reach the 70% mark.

8.1.1 Meta properties

This 30% discrepancy may possibly be attributed to many factors, such as trends or other “meta properties” unrelated to the content of the short story itself. For instance the popularity of the author of a given short story may play an important role in how popular it turns out to be.

That being said, since this 70% accuracy applies to the 2-choice granularity, the model has a great amount of leeway in terms of its approximations. If the author’s popularity would increase their output by 2 nodes, the model should still be able to compensate for most of this except for edge cases with “base” answers in the 2-3 node range. For instance, given a short story $S_x$, with a base rating of $\tau(r_x) = 1$, if an author’s popularity raises the rating by 2 points, so that $\tau(r'_x) = \tau(r_x) + 2 = 3$, this would still fall within the same half (0..3) in the Gb granularity. Since the stories are equally distributed across $\tau$ (as defined in section 5.3 (p. 25)), this means $\frac{2}{5} = 25\%$ of the stories suffer from the edge case where this would actually make a difference. If 10% of the authors would have the given effect, that would mean 2.5% of the lost accuracy is accounted for here.

During the research a correlation between the length of a short story and its popularity score was detected (Figure 8.1). The correlation indicates that for the given set of short stories, with $L(r)$ being the length of some short story whose popularity score is $r$, it holds that $L(r) \geq 503 + 7.26r$. This is a property which the system could be able to learn, but it would require some innovation in the training process, which currently is too localized.

Speculating further into various other properties might eventually justify the 30% loss. However, the model’s poor ability to make use of all output nodes remains an issue (see for instance Table 7.4, p. 40). This is discussed in Chapter 8.2.3 below.
8.1.2 LSTM performance

The LSTM model performed exceptionally poorly at the given task. A part of this bad performance is most likely due to the LSTM not usually being used to analyze content spanning several tens or hundreds of thousands of tokens. It may also be caused by a lack of parameter fine tuning, but this would then potentially be the case for the other models as well. This remains an open question and is mentioned in section 9, p. 46.

The poor performance observed was the case for an LSTM trained to map the input sequence into a rating directly, as well as for a model which included an LSTM pretrained using character prediction.

8.2 Memory module applicability

The memory module suffers from several issues, such as the inability to do regular batch training, and fluctuations in the training phase.

8.2.1 Batch training

Batch training over multiple tokens within the same short story is impossible, because the input at each step depends on (is calculated in) the previous step. This is somewhat alleviated by batch training over multiple short stories instead, keeping track of every memory state, but the training time is nonetheless drastically affected. On an NVidia K20 GPU, preparing and training on roughly 80 thousand samples took about seven minutes, processing about 200 samples per second. This may be resolved if better software support was made available. The library used did not inherently support such a thing as connecting one output to the input of the next iteration.
8.2.2 Training fluctuations

The model experienced heavy fluctuation in the accuracy for each pass during the training phase. After each training pass, test data was used to measure the accuracy of the model in its current state. This would oscillate quite drastically and would not give a very accurate picture of the resulting model performance on the validation data, in part due to the substantial amount of time required to predict on the test data resulting in using a rather low amount of inputs for this phase. A part of the problem here was deemed to lie in the learning rate used in the model, and consequently attempts to lower this were made, but notable differences in the fluctuations were not observed. It is probable that a more dynamic approach to learning rate would alleviate this problem entirely, but the nature of the memory module implementation, in how error accumulates over iterations, means even a tiny learning rate potentially triggers a cascade of errors very quickly. With high error, the learning rate is offset by the perceivedly high model inaccuracy.

8.2.3 Dormant nodes

In almost all cases, the model tended to group results into two distinct nodes, all but ignoring the others. This is most likely due to the use of softmax as the final activation function, because this decouples the individual nodes from each other in an unnatural way. Rating 0 ends up having nothing at all in common with rating 1 — or rather, rating 0 ends up having as little in common with rating 1 as it has with rating 2, 3, 4, 5, 6, and 7. Each rating becomes a distinct, independent category. As such, the approximations mentioned in 8.1.1 are, in practice, irrelevant.

Attempts to remedy this problem were made, but ultimately failed (see section 7.4, p. 37).

8.2.4 Performance

Ultimately, the memory module, despite its issues, outperformed existing alternatives. While the accuracy itself may not be all that impressive, the results indicate that the memory module approach is a potentially viable, new way to perform training in particular on long sequences of data.

New baselines for the respective granularities described in section 7.1 (p. 35) were defined. It is hoped that these will assist in future research on the topic in question, to better determine how well a model performs.

- **Perfect** = $\frac{1}{8} = 12.5\% \Rightarrow 18.81\% (+6.31\%)$, 
- **Near** $\approx \frac{1}{8} \left( 2\frac{1}{4} + 6\frac{3}{8} \right) \approx 34\% \Rightarrow 42.60\% (+8.60\%)$, 
- **GNB** $\approx \frac{1}{8} \left( 6\frac{3}{8} + 2\frac{2}{8} \right) \approx 34\% \Rightarrow 58.55\% (+24.55\%)$, 
- **Gb** = $\frac{1}{2} = 50\% \Rightarrow 69.52\% (+19.52\%)$. 
8. Analysis

<table>
<thead>
<tr>
<th>Rating</th>
<th>2013</th>
<th>2014</th>
<th>2015</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-10</td>
<td>403</td>
<td>46</td>
<td>2757</td>
</tr>
<tr>
<td>10-100</td>
<td>1511</td>
<td>160</td>
<td>2425</td>
</tr>
<tr>
<td>100-1k</td>
<td>606</td>
<td>110</td>
<td>2499</td>
</tr>
<tr>
<td>1-10k</td>
<td>154</td>
<td>28</td>
<td>1144</td>
</tr>
<tr>
<td>10-100k</td>
<td>3</td>
<td>6</td>
<td>362</td>
</tr>
<tr>
<td>100k-1M</td>
<td>0</td>
<td>0</td>
<td>13</td>
</tr>
</tbody>
</table>

Table 8.1: Bias from using last_updated value 2013 vs 2015.

8.3 Biases and trends

8.3.1 The last_updated value

The intention was to gather data so that biases and trends had a minimal impact on the collection. However, since data was collected from two periods with a gap in between them (2013 and 2015), some unintended bias was inadvertently introduced into the data set. The last_updated value was used as the basis for the time period, as the available API did not allow for searches based on “first uploaded”. Due to this, short stories with a last_updated in 2013 were consequently stories that had not been updated for several years, whereas those in 2015 had been updated within a couple of months. The differences between these two sets in terms of popularity were not significant, however (see Table 8.1). That being said, it is undetermined what other impacts this might have on the outcome of the experiment.
9 Open questions

This chapter lists open questions and problems that were raised during the research.

9.1 LSTM

A great deal of effort went into training LSTM:s to do character prediction on short stories. This was evaluated both on the actual loss in the training phase, as well as in manual analysis of the content produced when asking the LSTM to generate a short story itself. Examples of this can be seen in Appendix A (p. 53), although these are all in Japanese.

It was hypothesized that an LSTM performing very well at character prediction would have a good internal representation of the semantic properties of the short story. This would in turn be helpful when including the LSTM in a bigger model which predicted the rating for a short story, but this did not turn out to be the case.

In fact, there was no noticeable difference between a model which included a static LSTM (with no weight changes allowed) as part of it in learning to predict a rating, a model with a pretrained but dynamic LSTM (where the weights could be tuned to fit the rating task, even though they were already tuned to do character prediction), and a model with a randomized (untrained) LSTM performing the rating task directly.

The reasons for this are assumed to be due to the LSTM not entirely suitable for (without further fine tuning) the task of predicting an overall value based on a very long sequence of inputs, but this remains an open question.

9.2 Memory module fluctuations in training

There were heavy fluctuations in the training phase, which was presumed to be due to the learning rate parameter being set too high. It may also be due to the accumulating nature of the memory module training process where errors accumulate across iterations. A more robust method for training memory modules is necessary.
9.3 Convolutional neural networks with processing layers

The memory module models with convolutional neural network components would perform quite poorly at the task of learning to predict rating if one or several processing (neural network) layers were added after the CNN before the rating. The reason for this is unclear, but may be due to the added flexibility in the additional layers causing the convolutional pool to train too slow to be feasible.

9.4 Representation of output

The method chosen for the ratings was a threshold based classifier with 8 classes (0 being least popular and 7 being most popular). Attempts were made to use a non-discreet, floating point value as the output, which is obviously a better representation of the correlation between the different outputs, but when these were trained, the model broke down fairly quickly. Additional attempts were made using various alternatives to this, such as letting each node represent “one point” regardless of location, but this resulted in issues with representation as e.g. a ●●●●〇〇〇〇 4-rating would be 0% similar to a 〇〇〇〇●●●● 4-rating loss-wise, despite being 100% similar according to the rule. An activation function which “ordered” the outputs left-to-right according to strength would solve this problem, but such an activation function was never developed. Would this break the differentiability of the model? This area needs more thorough investigation. See section 7.4 (p. 37) for details.
10 Conclusions

A model was developed which could predict $r(\ell)$ with a 2-choice granularity in 69.5% of the cases using a new innovation called the memory module. While this outdid experiments using pre-existing models, there is simply too little existing research to determine if the model is ultimately successful or not, and, more importantly, a deeper analysis of the impact of metadata surrounding the short stories is required to formulate an estimate on the margin of inevitable error. If this margin (for 2-choice granularity) is close to 30%, the model is most definitely successful, and no other model will ever outperform it (for 2-choice granularity) without taking more than the tokenized short story content into consideration.

As such, the research problem formulated in section 1.3 (p. 4) remains unresolved. Instead, this thesis may serve as a potential basis for future research on this topic, as a new baseline (section 8.2.4 (p. 44)) for the respective granularities described in section 7.1 (p. 35).

The memory module introduced in Chapter 6 (p. 27) has great potential, but several of the unresolved issues related to training (see section 8.2 (p. 43)) makes it hard to assess whether it is a viable new tool or not.

While the convolutional pool variant outperformed the regular variant for lower-choice granularities, the differences are not significant enough to determine whether or not a mapping from $\mathbb{R}^{x \times y} \rightarrow \mathbb{R}^n$ is beneficial to the prediction accuracy (see section 1.4 on page 4).

Very few attempts have been made so far to tackle very long sequential data. Through the use of muted softmax, which addresses training phase issues, and the other techniques introduced here, it is hoped that this so far unexplored realm will receive more attention in future research.
Bibliography


A Sample stories

The overarching theme is inevitably "nonsense", but glimpses of bil-
liance can be gleamed here and there. The backslash character represents
the "unknown token" identifier. {{(letter)}} are names of people.

The parameters lists the number of nodes used in the LSTM being
trained, as well as (if available) the loss at the time. It also lists diversity,
which defines how much the model is allowed to improvise, i.e. stray from
the most probable next sequence when it is composing the story. Min-length
is a value indicating how many sequences the LSTM was trained on. A 100
min-length value means the LSTM was trained on 100 sequence long inputs
(in the case of a short story, this would be 100 tokens).

Most of the training here was done on a per-character basis (UTF-8
characters) as training on word vectors did not produce comprehensible
results. Training on UTF-8 sequences at byte level was attempted as well,
but the results are not presented here.

Notable is the consistent inability to distinguish between the various
"I" words, as demonstrated in e.g. A.2 on lines 1-2, and A.4 on line 11, where
the model indifferently switches between 僕 and 私 in the same sentence.
It is possible that the system interprets these words as referring to specific
individuals (as if they were names), hence causing the confusion.

The model quickly learns to be consistent with citations (「」 in
Japanese), but sometimes gets the starting and ending types wrong, as in
A.1 line 7. It is not perfect at this task, however, as can be seen in A.5 line
5.

It also learns to do chapter divisions and the like, as can be seen in
A.9

A.1 Sample 1

落ち着いた朗らかに粒われるチャンスは受け入れてね、うだった。ぷの前で
は、父親を従う弟の産みもなかった私は、情熱的な動揺を召喚するまで出た。

[A] 皇子のキスコールだし、こんなに包みさせて頂きたい。
お茶は出来るものだから分かってて、彼が楓を待つと
我の位磨を知った無賞
な彼が様々な言葉を断るため、末プランフじゃないよな、い筋繋
言うマイ
ムも穏やかな所にある。
初めてはいるがこの人を問い詰める鶴食は冷剥を把握しているのだが。
この難儀に応えなくと罪悪感でダグ中をもう参加しているあたしの一部の
言葉からは金髪さんの平然と弁当を上げてしまったものだ。
お嬢さんにならせがするが、酔った感じらしい。
「マントで水のような処理させておけた？帰る……お願いしたくないで……!?
だ、俺は知られるかもしれないんじゃ、
あるいは答えたんだと思う。
俺は、高日のスパルダやSクラスの散組のようで[B]の目を犬の憐により返事
をし、戻り、それは無料に身分を細めた。
俺は、何るんだ、すっかりしておりますよ。
何か気になる武力の質問をトレ錯のクラスの呑みを皆—————ポ
シュ可愛い女性である。

Parameters: 1k, 2.985, 1.0 diversity.

A.2 Sample 2
それを見て、俺はそれを見ている。
そう、私が、彼の仕事をしているのです。
そういえば、私はこの国の王子である。

Parameters: 512, 2.979, 0.5 diversity.

A.3 Sample 3
私は、この世界にいるのは、[C]との付き合いのテーブルに、目を閉じていた。
彼女が一番、それに気付いているのだろう。
これは、それが嫌だ。
そして[E]はお母様にお呼びになれたということではない。
それによって、この国の魔法は使い、悪役令嬢としては、全ての魔力があ
るのである。
私は、その時の家を出て行った。
そこには、少しだけ、今までのことではないだろうか。

Parameters: 1k, 2.823, 0.5 diversity.

A.4 Sample 4
髪を俺は育っている。
やっぱり近オさんが、眩しそうにヒロインから右扉を投げる。
店員の方が好きなのは、ダイブンッ!! 毎日薔사는していて行って下の荷
物を叶治するはずもない。
「わたい決まりになりました」
俺は号機ですよ。取り敢えず巻き求めて喋ると思ったんだ。誰とも動揺し
なかった。
その娘だが、今昔のライバル乙女、などとおもし殊物だって同じ正す!!
「だから、きっと部屋にじっと見坂だってあったのだろ」
結衣子は勝手に誰かが大好きですって……そんな虫を掘るのは、歓口にあ
るのです。私の補正は望や谷だったのです。
ちなみに執事しか自分のせいだと思われるのはもうどうだろう、中落する……
[D]と約束を受けていても。子ども従歪たれ誠!?
視界が違うわ。聞くけど、今はもっとした棚の中の二人に伴って迷惑をか
けてきたくし宣宗場に称するように。
僕は疲れているのよとは、私の耳もを思えば、市門に愛されている。血
の気を巻き込まれても想像もあるけれど、育て!
民のために選ん だ斗の
服は全て命をめたい。
この反応を褒めながら、近蹴、あの宇外の話だった事を思っていて、わたし
は悲鳴を上げ、変態を断られるとクリスティーナは、雪心と[...]
A. Sample stories

「後でカウンターに贈って買ってからです」
「ミド革！」
「はいはい、腐っております」
「お父様？」
「ああ。は、といっても侍女達の姿を見れば慈しんでいたわ。それにして
も訳の人間じゃない。でも時々は聞いてあげるわ」
お娘様、リリーが知らない態度を貫くのです。
「世種家の男子爵位にいるし、どうなっても本当に楽しみをさせたかった
からくすみめないように簡単に従う。言っておくりましたわ。す」
なんてラキであろうつもりのトリカスマンの体が端王家に置いてやる。だ
と、宜しくなく使うことになるような知っています。
「王族は伐を持てるけれど、その頃からお安凡な事ができるでしょう？ お気
持ちを出していたのですよSはわかりませんもの。軽く、都力無筆で大きな行
動をすると[F]の事を従の影を焼いているのだなら、私の[G]様に習わ
れていますわ」
「[H]様……どうか女気そうするなり、ご訳分ですか？」
「ええ、なかったんですけど、貴女は狙い供たちの一千年の[I]です。結構
町を見たい女が大国します」
若干ドキドキと怒りを傾げて余裕な刃が申し上げ

Parameters: 1k, 1.0 diversity.

A.7 Sample 6

「どうしてやっか？ 話しかけて、その時間に負けて免やとか悔しい」
「既渇説明を言えば、言う。お前が言うまでは――ある？」
「私、叶わない迷惑なども」
「夢に向かないで……」

Parameters: 1k, 331 min-length, 1.0 diversity.

A.8 Sample 7

ふと、今の私はそれを見て、少しずつずっと向かった。
彼はその手を取る。
[J]は少し合わせていた。
彼女の手を握り締め、口を開く。
「何をしているんだ」
「…………」
「………………」
ドアを開け、ほどきた後ろに、ピッリと音を立てて何かを落とす。
「お前に言われて、よく見ていたので、二人で、一緒に行くわ」
先輩は Ivyを手に入れた。
「明日も昼ご飯を食べたい」
「……」
A. Sample stories

先輩は無理矢理口に開いた。
「[K]の家に帰ると、行きましょう」
少しずつ分かる。
先ほどの部屋に行くことにした。
そして、その中で彼女が笑っているのを見る。
ふと、少しずつ分かる。
「…………」
私は目を見開いた。
[L]は苦笑してしまった。

Parameters: 1k, 342 min-length, 0.5 diversity.

A.9 Sample 8

それでも、新しい火を受けている私は、私の妹を一富させるだけで一瞬であるし、顔の花に手を洗い致し豊まった。
と療や上品の怪我をされると、どうやらそうは思わなかった。

※

もう驚いている私のさくに、等！顔も赤い。
まさか、あたしは[M]さんが……属性すら郎、酸味というけば好きになるから彼女がガッ……だとか、そうなることがあったとか、……慮を回してたんだから。

Parameters: 1k, 436 min-length, 1.0 diversity.