Investigating Danish Texts using Zipf's law and Neural Networks

Niels Bohr Institute
Submitted March 27, 2017

***Abstract*—In this article we present the results of a two part project. The first part involves investigating the compatibility of different variations of Zipf's law with four different forms of Danish texts; the Bible, news articles, Wikipedia articles and subtitles. For each type of text we compare the idealized Zipf's law with three different variations of it using maximum likelihood ratios. We also generate data using Monte Carlo simulations, fit parameters for each of the three versions. These fits are compared to the data sets using a Kolmogorov–Smirnov goodness-of-fit test.**

We find that two of the three alternative hypotheses fit better to all four data sets compared to the null-hypothesis, and that of these two the simplest (a pure power law) fits the data the best. The second part involves setting up a Recurrent Neural Network that is capable of generating Danish bible verses, and a qualitative and quantitative analysis of its ability to imitate the Danish language. We find that after 60,000 iterations the network can imitate words in Danish bible verses with an accuracy of $(99.3 \pm 0.3)\%$.

You certainly have my attention with this well written abstract

PACS numbers: 02.50.Ng, 02.70.Uu, 07.05.Mh

***Keywords*—Zipf's law, Maximum Likelihood Estimation, Monte Carlo, Likelihood Ratio, Recurrent Neural Network, Long Term Short Memory**

PART 1: ZIPF'S LAW

I. INTRODUCTION

Danish is a North Germanic Language and is spoken by around six million people. The Danish online dictionary consists of around 125,000 words [1], and the frequency of which the different words occur in sentences, articles, subtitles etc. vary. The words in a text can be ranked after the frequency at which they occur, and the relationship between the number of occurrences of a word and its rank can be quantitatively described by *Zipfs's law*. Zipfs's law is an empirical law formulated in the mid-20th century, and it describes a regularity in the usage of language [4]. It states that, for a sufficiently large amount of words in a text, the usage of a word will decline proportionally to the inverse rank:

I don't know what is meant here by 'after'

$$n \propto \frac{1}{r^{\alpha}}, \tag{1}$$

for α close to 1. E.g., the second most used word will occur about half as many times as the most used and so on. Zipf's law also has an alternative formulation describing the relationship between the number of words appearing n times and n. If, in total, there are 100 different words that each appear only one time in the text, this formulation of Zipf's law suggests that 50 words will be counted two times, etc.:

$$f(n) \propto \frac{1}{n^{\beta}}, \tag{2}$$

where $\beta = 1 + 1/\alpha$ [4]. In other words, the frequency of word occurrences is taken as the independent variable. Ziph's law in general is said to describe the distribution of many other ensembles, such as employees in a firm, population of cities, believers of religions [2].

In this part of the project we use large samples of Danish texts to compare three different models of Ziph's law against the null hypothesis, $\alpha = 1$, by doing (maximum) likelihood ratio (LR) tests. We also use maximum likelihood (ML) estimation on the three models, fitting the parameters, and comparing data generated by Monte Carlo simulation with the original data using a Kolmogorov-Smirnov test. The text analysis in this part of the project largely follows the same procedures as [4], but the methods are applied to Danish texts instead.

II. THEORY

Since Zipf's law describes a discrete rank versus frequency relationship, the normalization of the probability mass function (PMF) includes the discrete sum of the ranks n one to infinity:

$$\sum_{n=1}^{\infty} f(n) = 1. \tag{3}$$

The normalisation of eq. (2) yields the null hypothesis H_0 with $\alpha = 1 \rightarrow \beta = 2$:

$$f_0(n) = \frac{6}{\pi^2 n^2}. \tag{4}$$

The first alternative hypothesis, H_1 ($\alpha \neq 1$), that we wish to test against H_0 is given by:

$$f_1(n) = \frac{1}{\zeta(\beta, 1) n^{\beta}}, \tag{5}$$

for $\beta > 1$, and where $\zeta(\beta, 1) = \sum_{k=1}^{\infty} (1+k)^{-\beta}$ denotes the Riemann-Zeta function [4].

A power law in the PMF, $f(n)$, does not lead to a power law in the corresponding complementary cumulative distribution, $S(n)$, given by $S(n) = \sum_{n'=n}^{\infty} f(n')$; the probability of getting a frequency larger than or equal to n. However, if the power law is instead in the cumulative distribution, $S_2(n) = \left(\frac{1}{n}\right)^{\beta-1}$ for $\beta > 1$, we use that:

$$f(n) = S(n) - S(n+1), \tag{6}$$

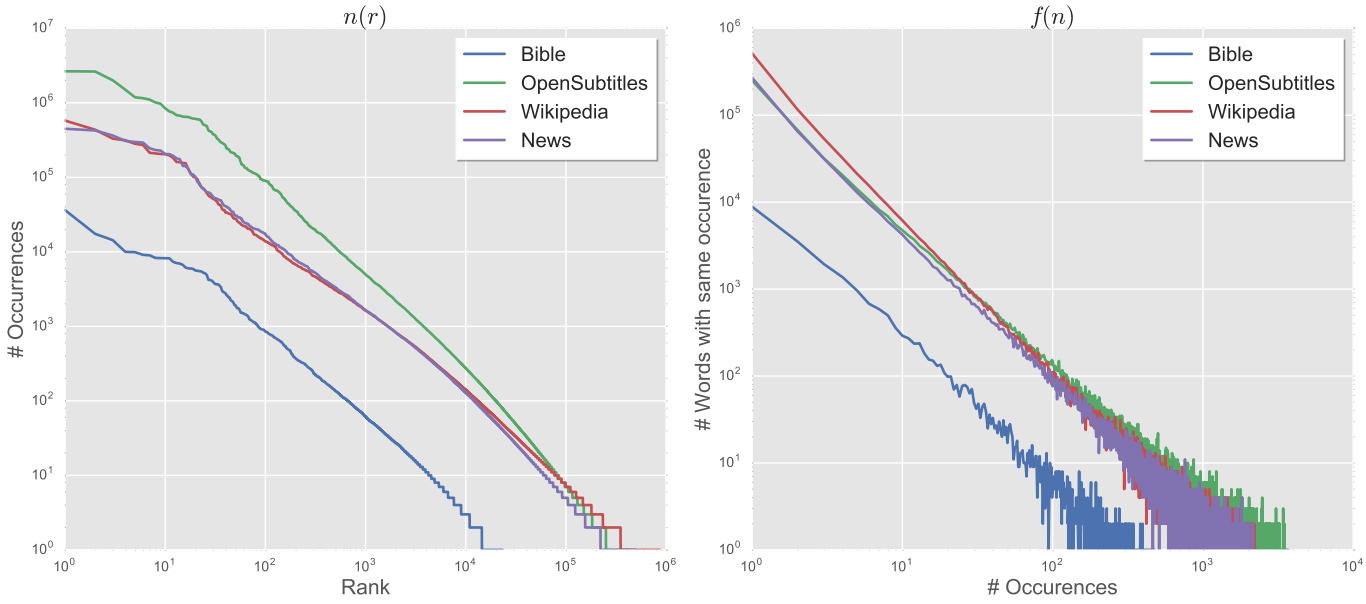

Fig. 1. Plots showing the 4 texts sorted after (**Left**) the number of times each word appears in the texts as a function of rank (most to fewest) $n(r)$ and (**Right**) the frequency of words occurring the same number of times $f(n)$

and obtain the second expression for $f(n)$:

$$f_2(n) = \left(\frac{a}{n}\right)^{\beta-1} - \left(\frac{a}{n+1}\right)^{\beta-1}. \tag{7}$$

This is the second hypothesis, H_2, that we wish to investigate. For the third distribution and hypothesis, we will test the frequency distribution derived by B. Mandelbrot [3]. In this formulation, the power law is contained in an underlying theoretical frequency-rank relation $n(r)$. f_3 takes the (slightly rewritten) form :

Any reason as to why the f_x have the functional forms they do, e.g, beta-function?

$$f_3(n) = \frac{B(n+1-\beta,\beta)}{B(2-\beta,\beta-1)} \tag{8}$$

for $1 < \beta < 2$. Here $B(x, y)$ is the beta function (defined in term of the gamma function $\Gamma(x)$) and not the beta distribution.

The difference between the three distributions f_1, f_2 and f_3 is clearest when n is small. f_1 appears as a straight line (with a slope of $-\beta$), whereas f_2 and f_3 respectively are convex and concave (seen from above) in a log-log plot. The null hypothesis H_0 is a perfectly straight line (slope of -2) in a log-log plot, which is also how it is most often recognized.

III. Methods

The texts used in this part of the project are the Danish Bible (both new and old testament), various Danish newspaper articles, various Danish Wikipedia articles and various Danish subtitles. The Danish Bible was downloaded in its entirety from the Gutenberg Project website [5][6]. The news articles and Wikipedia articles were gathered from the Leipzieg Corpora Collection webpage [7] and each contain 1,000,000 words. Finally, the subtitles were gathered from a github repository by Hermit Dave[8] and contains 501,080 words.

The texts were sorted two times; one time by the frequency by which each word appear in the text and another time by using this frequency as the independent variable (effectively counting how many words only appear one time in the text, how many two times, etc.). The first sorting is shown in the left plot of Fig. 1 and the second in the right plot.

What about misspellngs ?

Corral and Ferrer [4] show that in the first case the fit results of the exponent (α or β) are biased. This, together with the actual shape of the plots in Fig. 1, motivated us to focus only on using the frequency as the independent variable (i.e. only use the right part of Fig. 1), which also means that the parameter of interest is β.

How are they biased?

Since only one of the hypotheses is nested (H_1) compared to H_0, we needed to compute the necessary test statistics to perform a likelihood ratio test for the other two hypotheses. We simulated 10000 pseudo experiments each with 10000 events sampled from our null-hypothesis, eq. (2), using a Discrete Inverse Transformation Monte Carlo Method. Since this is a discrete method, there is no analytical function to invert. Instead, a uniformly distributed number $u \in [0, 1]$ is generated. Then one finds the value of k such that:

$$F(x_{k-1}) < u < F(x_k), \quad F(x_k) = \sum_{i=1}^{k} f(x_i). \tag{9}$$

The value of k is then distributed according to $f(x)$ [9].
For each pseudo experiment we calculated the likelihood ratio between H_0 and H_i for $i \in \{1, 2, 3\}$ being the alternative hypotheses given by equations (5), (7) and (8). The likelihood

ratio is given as:

$$LR = -2 \ln \frac{L_0}{L_i} \tag{10}$$

The resulting distributions of likelihood ratios are shown in Fig. 2.

Did you actually fit the distribution to a chi-squared?

As can be see from Fig. 2 only the likelihood ratio values between H_1 and H_0 are χ^2-distributed, as expected according to Wilk's theorem for nested hypotheses[10]. The two other distributions are shaped closer to Gaussian distributions. Integrating from $-\infty$ and up, we find the upper 3σ confidence limit (CL). For the three hypothesis we found: $\mathrm{CL}_1 = 10.98, \mathrm{CL}_2 = -5.28, \mathrm{CL}_3 = -363.18$.

Having set up the test statistics based on the MC simulation, we performed a hypothesis test between H_0 and the different hypothesis. For each of the four texts, we perform a maximum likelihood (ML) fit using iMinuit[16] as the minimizer. This is done on the real data, and not using the MC simulated data. From the ML fit, the likelihood ratio (LR) is used as the test statistic to either reject H_0 at 3σ or not. If the LR is below the 3σ CL the alternative hypothesis is no-better than the null hypothesis, and we exclude that specific function from further analysis.

For the remaining hypotheses we perform a Maximum Likelihood Estimate (MLE) of the fit parameter β to find its value and uncertainty. Since all our models are 1D, performing the MLE and finding the uncertainty of the fit parameter is done by scanning through the parameter space and using the lowest LLH-value as the initial value for iMinuit to properly minimize the (negative) LLH.

To test whether or not the remaining hypotheses actually fit the data, we use Kolmogorov-Smirnov as the goodness of fit test statistics. Since testing the fitted model to the data it was fitted to would positively bias the p-value [4], we generate 1000 MC samples for each hypothesis based on the fit parameter β and compare these MC-generated samples to the actual data. To generate discrete samples based on the different hypotheses, we follow Moreno's [4] method of first using the Inverse Transformation Method on f_2 and then the Hit-and-Miss Method to generate numbers from f_1 and f_3.

IV. Results

Performing the likelihood ratio test between the three hypotheses on each of the four data sets give us the following values for each alternative hypothesis:

	Bible	News	Wikipedia	Subtitles
LR_1	6862.33	33451.35	12135.18	89191.34
LR_2	7588.24	18398.98	-12353.52	73115.06
LR_3	5238.35	32552.20	7650.96	88869.78

TABLE I. Table containing the calculated likelihood ratio values for each data set and each alternative hypothesis.

When we compare the values from table I with the 3σ confidence limits found earlier, we see that there is a case where we cannot reject H_0 compared to H_2 at 3σ confidence. We therefore exclude f_2 from the further analysis. For the hypotheses that passed the hypothesis tests we fit them to each of the 4 data sets. The graphs for f_1 and f_3 fitted to the Bible data sets is shown in figure 3. Similar graphs for the other data sets can be found in appendix A. From the fits we extract the values of β as shown in table II.

Why is this singular data test negative, while all the others are positive? Excluding f_2 based on this single point is not GOP a great idea because it looks like the result was the effect of a bug. If it is real, then there should be some discussion here as to why it is negative.

From these fitted values of β we generate 1000 data points for each alternative hypothesis via Monte Carlo and do a Kolmogorov−Smirnov test to test the goodness of fit of the alternative hypotheses to the data. The resulting p-values can be seen in table III. From table III we see that $f_1(n)$ seems to generally fit the data sets the best. $f_3(n)$ gives quite bad fits for all data sets.

We therefore conclude that in almost all cases the 3 alternative hypotheses were better than the null hypothesis, and that $f_1(n)$ fits the data the best.

	Bible	News	Wikipedia	Subtitles
β_1	1.598 ± 0.006	1.770 ± 0.002	1.884 ± 0.001	1.664 ± 0.001
β_3	1.467 ± 0.003	1.572 ± 0.001	1.619 ± 0.001	1.522 ± 0.001

TABLE II. Table containing the fitted values of β with corresponding 1σ uncertainties for each data set and each alternative hypothesis. $f_2(n)$ used on the Wikipedia data set was rejected during the hypothesis test.

	Bible	News	Wikipedia	Subtitles
P_1	1.15×10^{-2}	1.62×10^{-2}	1.24×10^{-6}	9.07×10^{-4}
P_3	4.26×10^{-12}	7.34×10^{-16}	8.39×10^{-11}	1.91×10^{-37}

TABLE III. Table containing the resulting p values from the Kolmogorov−Smirnov test for each data set and each alternative hypothesis.

Nice work and clearly explained.

Part 2: Recurrent Neural Network

I. Introduction

In addition to the more classical statistical method in Part 1, we also employ a different and more modern machine learning approach to the problem of analyzing Danish language; training a neural network on the largest Danish open text we were able to find: The Danish Bible.

Citation(s)?

We trained a specific form of neural network (NN) called a Recurrent Neural Network (RNN) on the data set, which consists of 78,165 lines of bible verses (the Old Testament[5] and the New Testament[6] combined), in total a 4 MB file. We optimize the RNN training, and in the end we were able to predict new bible verses and evaluate the performance of the RNN's output.

II. Neural Network Principle

Given a fixed input (e.g. an image), classify it as one of a fixed number of categories. This is a typical task for a neural network. This is what is shown in the leftmost part of Fig. 4,

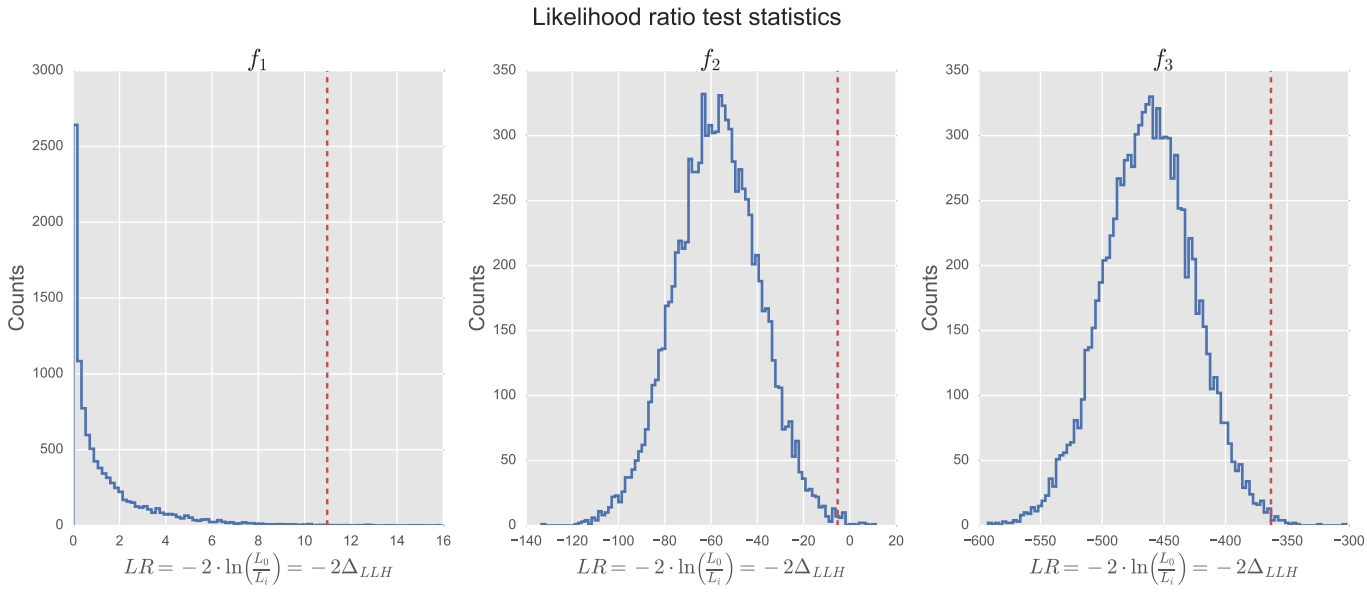

Fig. 2. Histograms of likelihood ratio values from 10000 pseudo experiments each with 10000 data points sampled from H_0. The ratios are between H_0 and (**left**) $f_1(n) = 1/(\zeta(\beta,1)n^{\beta})$, (**middle**) $f_2(n) = (1/n)^{\beta-1} - (1/(n+1))^{\beta-1}$ and (**right**) $f_3(n) = B(n+1-\beta,\beta)/B(2-\beta,\beta)$. The red line is the upper 3σ confidence limit.

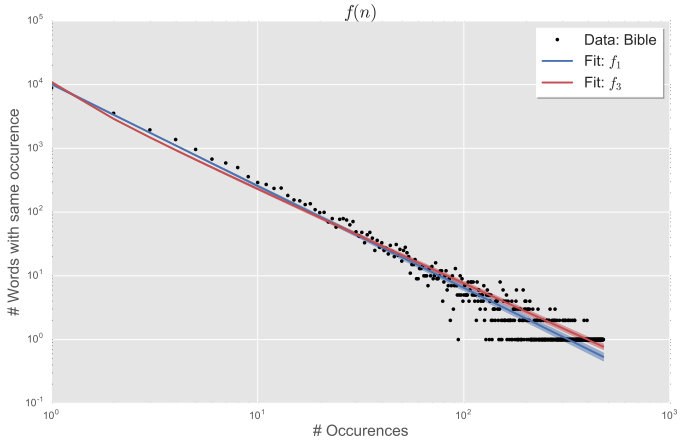

Fig. 3. Plot showing $f_1(n)$ and $f_3(n)$ fitted to the Bible data set. The fits include the 5σ confidence interval.

where red is the input, green is the internal state of the neural network and blue is the output. Compared to the one-to-one example, consider the case where you want a computer to write a caption for given image. Or, in our case, where we want to train a NN to compose bible verses based on a whole book. In these examples, there ~~are~~ no fixed input or output. This is where RNNs outpowers vanilla NNs by having many so-called internal states (green middle boxes in Fig. 4) [11]. The internal state is affected by earlier inputs – more or less how humans also think, making choices based on memory. In text prediction this internal memory is exactly what we are training in the RNN; to remember the last X characters and decide what the next character should be based on earlier patterns. In theory, it would be possible for a RNN to have as many internal states as we have characters in the training text, however, in practice there is a limit to this – and after reaching this limit, the RNN becomes progressively worse[12]. A specific form of RNN are the so called Long Short Term Memory Networks (LSTM). They are tailored specifically to store information from only the latest number of characters, but to keep this information for a long time. For a further explanation, see [12]. In our case, LSTM can be seen as a better-working version of RNN – which itself is an improvement over vanilla NN.

An interesting citation.

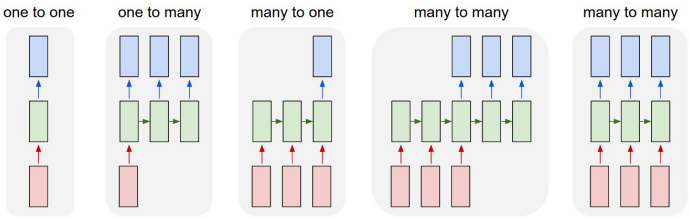

Fig. 4. Difference between vanilla neural network (**left**) and RNNs (**right**). Vanilla neural networks take only a fixed input (red) and generate a fixed output (blue) based on its internal state (green). RNNs allow for a more flexible use with many in- and outputs each with a internal state that has memory based on prevous inputs.

In practice, we make use of Justin Johnson's excellent module "Torch-RNN" which implements a character-based LSTM [13]. Torch-RNN is a high performance implementation of LSTM in the light weight scientific computing framework "torch7"[14]. The real advantage Torch-RNN has, compared

These quotation marks are the wrong way.

to other LSTM modules, is a moderately user friendly implementation of running the computations on the GPU compared to the CPU. Using CUDA to allow computations on the GPU strongly decreases the training time. Compared to a Kera's[15] implementation based on TensorFlow with only CPU-support, Torch-RNN was around 7 times faster (4 hours compared to about 30 hours, training time). We found that using 4 layers each having 512 internal states (and a dropout rate of 0.5) gave the best results. Nice work!

III. Results of neural network analysis

Viewing the output as a function of the number of iterations gives an impression on how the output of the algorithm gradually improves. Keep in mind that Torch-RNN is character based and not word-based, so it does not simply look at which word combinations often appear next to each other[1]. The algorithm therefore has to learn Danish by just looking at patterns at the character-level. After X iterations it outputs[2]:

```
Iteration 100:
    æøH  go9 ule?
i nn  ase rnn  r e M de  st   k    g va
ie rl  o  doæs sl are  h m   rsnsn s ng

Iteration 200:
(leg de fander han keltune Han Lal sam
  o  den kker de tad tår Hærer, fæl ot
  Eande sem Jås Perteng. or

Iteration 300:
8Ubølte for for var skal den du er mig mevlet
Mudtede på mige Lavtan, der Je sig mig og.
 10.  ner og deret da føste der den hjimten d

Iteration 500:
9.  Se, hver Lingen eder skal en Søn; Jårdet
har han står vidte i og skal de levede til
sig og handne den til hele og det ned blive
de som Adas,

Iteration 1,000:
24.  Da de siger ham.
 12.  Siger er sagde som den og førte i det
      med min Tjener Sted, Judas Borge,
      bryder mit Vand, stort de brænder.

Iteration 60,000:
2.  Så tog Saul det Hus af Assyrerkongen
    til dem, hans Sønner skal opbrændes
    for Moabs Konge, og Moses så sig der
    for at advare dem;
16. og han sagde til Josua: "Er det, som
    jeg giver Israeliterne hen som en til
    min Trælkvinde, som har have dine Fædre
    og din Ord laver sin Fædrenehus's
    Brødre, thi Retten er din Tjener.
```

Initially the algorithm mostly generates random letters. Around iteration 200 it starts learning some of the short words. At iteration 300 numbers begin appearing and at iteration 500 it learns about more advanced punctuation. However, it still does not really make any syntactical sense. From iteration 1,000 and forward sentences which makes sense start slowly emerging, and at iteration 60,000 it becomes difficult to distinguish between generated text and real bible verses.

To quantify the effectiveness of the RNN, we generate 2,000 characters from iteration 100 to 1,500 in steps of 100 and calculate the fraction of generated valid words. We use all the words in the Bible that appear more than 1 time (to account for some spelling errors in the online public version of the bible) as a dictionary. This dictionary yields 14,618 distinct Danish words to compare the generated sentences with. The results are seen in Fig. 5. We see that after only around 500 iterations the correctness of the generated sentences is larger than 90% with a slow asymptotic behaviour. At 60,000 iterations the correctness is $(99.3 \pm 0.3)\%$. Are the errors misspellings or complete gibberish?

These numbers show the great strength of LSTM RNNs; that they are able to correctly generate new bible verses based on a single text file as input. After only about 1000 iterations the words starts to make sense, although more iterations are needed to correctly imitate Danish syntax. We also found that using the GPU significantly reduced training time of the RNN by up to a factor of 7. Okay. How long did it take to train?

Good check.

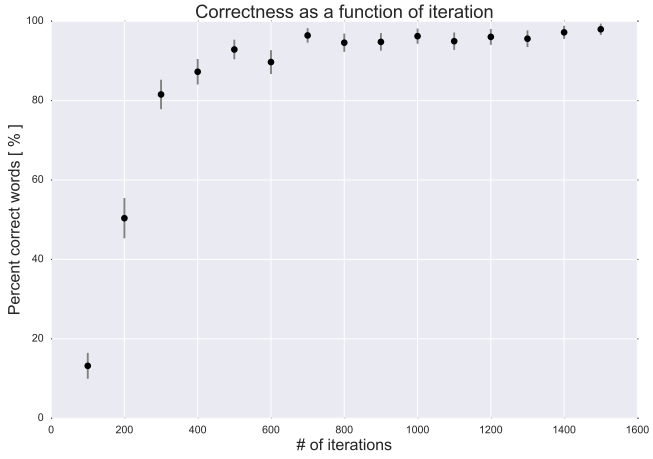

Fig. 5. Fraction of correctly generated words of our neural network as a function of iteration number. For illustration purposes, the shown uncertainties are 2σ. The neural network is a RNN using LSTM trained on the Danish Bible.

Part 3: Conclusion

The project consisted of two parts. The first focused on a quantitative analysis of the word frequency of the Danish language, testing different versions of Ziph's law. The most compatible formulation of Ziph's law was found also to be the most simple model, $f_1(n) \propto 1/n^{\beta}$, which consistently

[1] This further has the advantage of being able to predict not just text, but also general code (like C-code or LaTeX-code) [11].

[2] Sampling it with a temperature setting of 0.5

yielded good results for the different danish texts. Further work on this part of the project could include examining entirely different languages with Ziph's law.

The second part contained a quantitative and qualitative analysis of text generated by a recurrent neural network, Torch-RNN, training on the Danish Bible. After 60,000 iterations of this algorithm, $(99.3 \pm 0.3)\%$ of the words generated was actual words present in the Bible. Using the GPU for the calculations decreased the iteration time by a factor of 7. Further work on this part of the project could include training on different (more modern) Danish texts.

REFERENCES

[1] Nyhedsbrev fra Den Danske Ordbog, nr. 8, april 1997.

[2] Zipf's law holds for phrases, not words, Jake Ryland Williams, December 2014 [Accessed on March 27, 2017] http://www.nature.com/articles/srep12209

[3] B. Mandelbrot - On the theory of word frequencies and on related Markovian models of discourse, pages 214-218. American Mathematical Society, Providence, RI, 1961. [Accessed on google books March 27, 2017]

[4] Moreno-Sánchez I, Font-Clos F, Corral Á (2016) Large-Scale Analysis of Zipf's Law in English Texts. PLoS ONE 11(1): e0147073. doi:10.1371/journal.pone.0147073

[5] Bibelen, Det nye Testamente. 2000. EBook-nr. 2143. Project Gutenberg; [Accessed on March 27, 2017]. https://www.gutenberg.org/ebooks/2143

[6] Bibelen, Det gamle Testamente af 1931. 2000. EBook-nr. 2144. Project Gutenberg; [Accessed on March 27, 2017]. https://www.gutenberg.org/ebooks/2144

[7] Leipzig Corpora Collection Download Page. c1998-2017. Leipzig University: Institute of Computer Science; [Accessed on March 27, 2017]. http://wortschatz.uni-leipzig.de/en/download/

[8] Github: Repository for Frequency Word List Generator and processed files. c2016. Hermit Dave; [Accessed on March 27, 2017]. https://github.com/hermitdave/FrequencyWords

[9] Karl Sigman (2010). Inverse Transformation Method

[10] S. S. Wilks (1938). "The Large-Sample Distribution of the Likelihood Ratio for Testing Composite Hypotheses". The Annals of Mathematical Statistics. 9: 60–62. doi:10.1214/aoms/1177732360.

[11] Andrej Karpathy (2015). "The Unreasonable Effectiveness of Recurrent Neural Networks" [Accessed on March 27, 2017]. https://karpathy.github.io/2015/05/21/rnn-effectiveness/

[12] Christopher Olah (2015). "Understanding LSTM Networks" [Accessed on March 27, 2017]. https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[13] Justin Johnson (2016). "Efficient, reusable RNNs and LSTMs for torch", [Accessed on March 27, 2017]. https://github.com/jcjohnson/torch-rnn/

[14] Torch: A SCIENTIFIC COMPUTING FRAMEWORK FOR LUAJIT. Ronan, Clément, Koray and Soumith. (Accessed on March 27, 2017). http://torch.ch/

[15] Keras: Deep Learning library for Theano and TensorFlow. (Accessed on March 27, 2017). https://keras.io/

[16] Iminuit. c2012. Piti Ongmongkolkul [Accessed on March 27, 2017]. http://iminuit.readthedocs.io/en/latest/index.html

It is absolutely correct to include the date when citing websites. Nice job.

APPENDIX A
PLOTS

- Fascinating topic that was excellently well executed and explained.
- very few grammatical errors, although any errors are somewhat ironic given the topic of your project
- including the word count frequency and predictive text NN could almost be considered two separate projects, so the amount of work was impressive.
- my minor criticisms are a lack of discussion about the neg. data entry in table 1, and a limited explanation of the LTSM NN given that you had at least 0.5 pages of text left blank. But, these are truly minor.

9.9/10

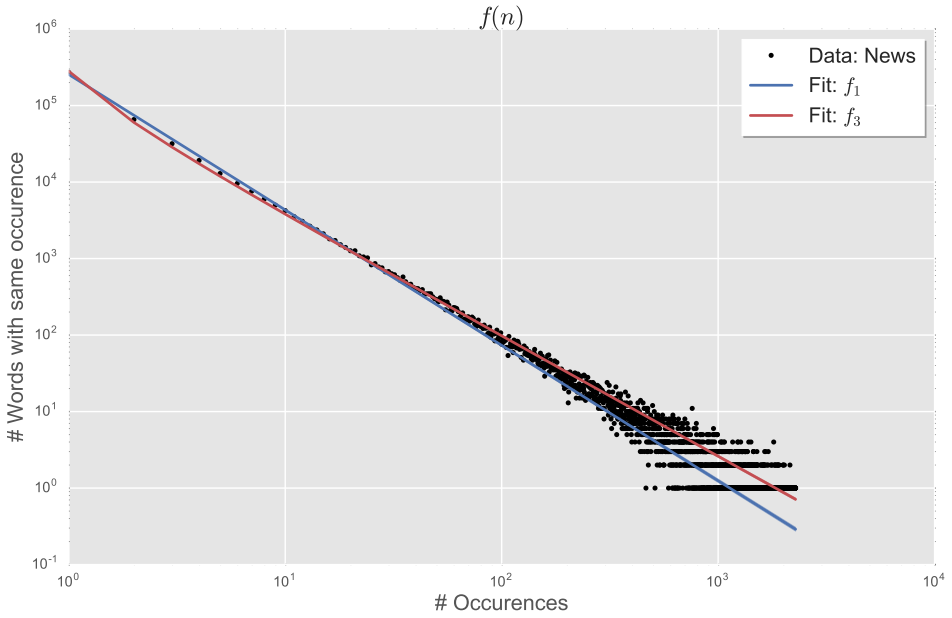

Fig. 6. Plot showing $f_1(n)$, $f_2(n)$ and $f_3(n)$ fitted to the News data sets. The fits include 5σ error bars, but these are too small to be discerned.

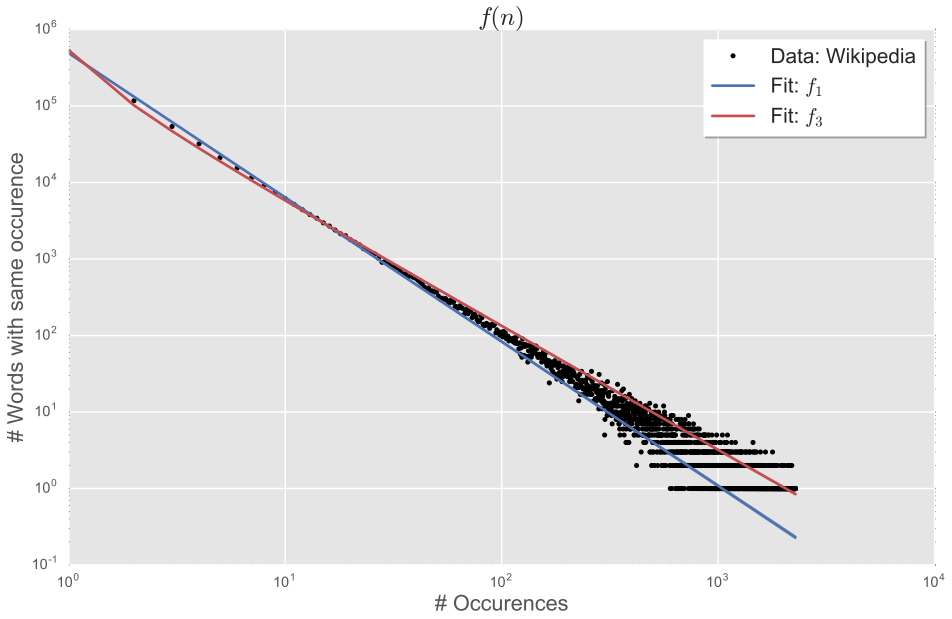

Fig. 7. Plot showing $f_1(n)$ and $f_3(n)$ fitted to the Wikipedia data sets. The fits include 5σ error bars, but these are too small to be discerned.

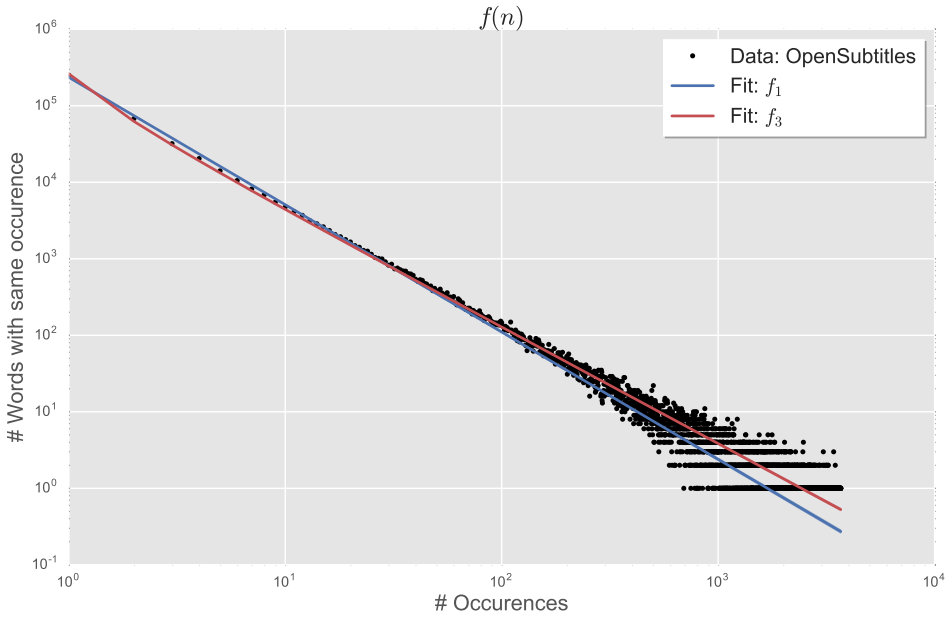

Fig. 8. Plot showing $f_1(n)$ and $f_3(n)$ fitted to the OpenSubtitles data sets. The fits include 5σ error bars, but these are too small to be discerned.