UNIVERSITY OF CAMBRIDGE

Devan Kuleindiren
Robinson College
`dk503`

Computer Science Tripos - Part II Project

Language Modelling for Text Prediction

March 21, 2017

supervised by
Dr Marek Rei & Dr Ekaterina Shutova

Proforma

Name:	**Devan Kuleindiren**
College:	**Robinson College**
Project Title:	**Language Modelling for Text Prediction**
Examination:	**Computer Science Tripos – Part II, June 2017**
Word Count:	**?**
Project Originator:	Devan Kuleindiren & Dr Marek Rei
Supervisors:	Dr Marek Rei & Dr Ekaterina Shutova

Original Aims of the Project

The primary aim of the project was to implement and benchmark a variety of language models, comparing the quality of their predictions as well as the time and space that they consume. More specifically, I aimed to build an n-gram language model along with several smoothing techniques, and a variety of recurrent neural network-based language models. An additional aim was to investigate ways to improve the performance of existing language models on error-prone text.

Work Completed

All of the project aims set out in the proposal have been met, resulting in a series of language model implementations and a generic benchmarking framework for comparing their performance. I have also proposed and evaluated a novel extension to an existing language model which improves its performance on error-prone text. Additionally, as an extension, I implemented a mobile keyboard on iOS that uses my language model implementations as a library.

Special Difficulties

None.

Declaration

I, Devan Kuleindiren of Robinson College, being a candidate for Part II of the Computer Science Tripos, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose.

SIGNED

DATE

Contents

List of Figures

Chapter 1

Introduction

My project investigates the performance of various language models in the context of text prediction. I started by implementing a series of well-established models and comparing their performance, before assessing the tradeoffs that occur when you attempt to apply them in a practical context, such as in a mobile keyboard. Finally, I propose a novel extension to an existing model which aims to improve its performance on error-prone text.

1.1 Language Models

Language models (LMs) produce a probability distribution over a sequence of words, which can be used to estimate the relative likelihood of words or phrases occurring in various contexts. This predictive power is useful in a variety of applications. For example, in speech recognition, if the speech recogniser has estimated two candidate word sequences from an acoustic signal *'it's not easy to wreck a nice beach'* and *'it's not easy to recognise speech'*, then a language model can be used to determine that the second candidate is more probable than the first. Language models are also used in machine translation, handwriting recognition, part-of-speech tagging and information retrieval.

$$\begin{array}{rll}
\overbrace{\textit{Do you want to grab a}}^{w_1^k} & \overbrace{\underline{\textit{drink}}}^{w_{k+1}} & \overbrace{(0.327)}^{\mathbb{P}(w_{k+1}|w_1^k)} \\
 & \textit{coffee} & (0.211) \\
 & \textit{bite} & (0.190) \\
 & \textit{spot} & (0.084) \\
 & \vdots & \vdots
\end{array}$$

My project focuses on language modelling in the context of text prediction. That is, given a sequence of words $w_1 w_2 ... w_k = w_1^k$, I want to estimate $\mathbb{P}(w_{k+1}|w_1^k)$. For instance, if a user has typed *'do you want to grab a '*, then a language model could be used to suggest probable next words such as *'coffee'*, *'drink'* or *'bite'*, and these predictions could further be narrowed down as the user continues typing.

1.2 Motivation

Benchmarking

- Variety of language models, that vary in accuracy, memory consumption and speed.
- Combinations can be considered.
- Important tradeoffs when applying them to mobile.

There are two prominent approaches to language modelling, the first of which is the n-gram approach and the second of which is based on (recurrent) neural networks. language models and their various smoothing techniques, before proceeding onto neural network-based language models. Combinations of such models, which can be achieved by interpolating their probability distributions, will also be included in the benchmark.

New techniques for facilitating better predictions are constantly being developed for language models. However, it is important to realise that such models are being used increasingly in a mobile environment, where the computational and memory resources aren't as abundant.

Error-prone Text

One problem with existing language models is that their next-word predictions tend to be less accurate when they are presented with error-prone text. This isn't surprising, because they are only ever trained on sentences that do not contain any errors. Unfortunately, humans are not perfect, and they will make typographical mistakes, spelling mistakes and occasionally grammatical errors too. In this project I investigate ways to bridge the gap in performance between language model predictions on error-prone text and error-free text.

1.3 Related Work

- Google 1-Billion Word benchmark
- Chen and Goodman smoothing paper
- Work that's been done on language models with error-prone text

Chapter 2

Preparation

2.1 N-Gram Models

2.1.1 An Overview of N-Gram Models

Describe how they work, and the motivation for smoothing and backoff.

2.1.2 Smoothing Techniques

Add-One Smoothing

Katz Smoothing

Absolute Discounting

Kneser-Ney

Modified Kneser-Ney

2.2 Recurrent Neural Network Models

2.2.1 An Overview of Neural Networks

Give a brief introduction to neural networks. Explain backpropagation. Motivate why we need RNNs for language modelling.

2.2.2 Recurrent Neural Networks

Explain RNNs.

Vanilla Recurrent Neural Networks

Gated Recurrent Unit

Long Short-Term Memory

2.2.3 Word Embeddings

2.2.4 Backpropagation Through Time

2.3 Software Engineering

2.3.1 Starting Point

2.3.2 Requirements

2.3.3 TensorFlow

Perhaps this could go in the implementation section.

2.3.4 Tools and Technologies Used

Chapter 3

Implementation

3.1 Development Strategy

3.1.1 Version Control and Build Tools

3.1.2 Testing Strategy

3.2 System Overview

3.2.1 Interface to Language Models

3.3 N-Gram Models

3.3.1 Counting N-Grams Efficiently

3.3.2 Precomputing Smoothing Coefficients

3.4 Recurrent Neural Network Models

3.4.1 Long Short-Term Memory

3.4.2 Gated Recurrent Units

3.4.3 Word Embeddings

Show some cool embedding plots with PCA.

3.4.4 Parameter Tuning

Dropout, embedding, learning rate decay, momentum?, gradient clipping

3.5 Extending Models to Tackle Error-Prone Text

3.5.1 Preprocessing the CLC Dataset

3.5.2 Error Correction on Word Context

3.6 Mobile Keyboard

3.6.1 Updating Language Model Predictions On the Fly

Chapter 4

Evaluation

In this chapter, I will first describe the benchmarking framework that I built to evaluate the language models, before proceeding onto the results. The results section is threefold: firstly I will present the performance of the existing language models that I implemented, secondly I will focus on the tradeoffs faced when employing those models on a mobile device, and finally I will display my findings in language modelling on error-prone text.

4.1 Evaluation Methodology

In the context of text prediction, there are essentially two questions one might want to answer when evaluating a language model:

1. How accurately does the language model predict text?

2. How much resource, such as CPU or memory, does the language model consume?

In order to answer these questions, I implemented a generic benchmarking framework that can return a series of metrics that fall into one of the two aforementioned categories when given a language model. These metrics are outlined below.

4.1.1 Accuracy Metrics

Perplexity

Perplexity is the most widely-used metric for language models, and is therefore an essential one to include so that my results can be compared with those of other authors. Given a sequence of words $w_1^N = w_1 w_2 ... w_N$ as test data, the perplexity PP of a language model L is defined as:

$$\mathrm{PP}_L(w_1^N) = \sqrt[N]{\frac{1}{\mathbb{P}_L(w_1^N)}} = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{\mathbb{P}_L(w_i | w_1^{i-1})}} \tag{4.1}$$

where $\mathbb{P}_L(w_i | w_1^{i-1})$ is the probability computed by the language model L of the word w_i following the words w_1^{i-1}. The key point is that **lower values of perplexity indicate better prediction accuracy** for language models trained on a particular training set.

This somewhat arbitrary-looking formulation can be better understood from a touch of information theory. In information theory, the cross-entropy $H(p, q)$ between a true probability distribution p and an estimate of that distribution q is defined as:[1]

$$H(p, q) = -\sum_{x} p(x) \log_2 q(x)$$

It can be shown that $H(p, q) = H(p) + D_{KL}(p||q)$ where $D_{KL}(p||q) \geq 0$ is the Kullback-Leibler distance between p and q. Generally speaking, the better an estimate q is of p, the lower $H(p, q)$ will be, with a lower bound of $H(p)$, the entropy of p.

The perplexity PP of a model q, with respect to the true distribution p it is attempting to estimate, is defined as:

$$\mathrm{PP} = 2^{H(p,q)}$$

Language models assign probability distributions over sequences of words, and so it seems reasonable to use perplexity as a motivation for a measure of their performance. In the context of language modelling, however, we do not know what the underlying distribution of p is, so it is approximated with Monte Carlo estimation by taking samples of p (i.e. sequences of words from the test data) as follows:

$$\mathrm{PP}_L(w_1^N) = 2^{-\frac{1}{N}\sum_i \log_2 \mathbb{P}_L(w_i|w_1^{i-1})}$$

With a little algebra, this can be rearranged to give equation (4.1).

One issue with perplexity is that it is undefined if $\mathbb{P}_L(w_i|w_1^{i-1})$ is 0 at any point. To get around this in my implementation, I replaced probability values of 0 with the small constant `1e-9`. Results that use this approximation are marked.

Guessing Entropy

If a language model L_1 has assigned a higher probability to the correct next word w_c than another language model L_2, then it does not necessarily follow that w_c is ranked higher in the predictions of L_1 than in the predictions of L_2.[2]

Guessing entropy is a metric which focuses more on the final ordering of the word predictions made by a language model. It is defined as the binary logarithm of how far down a list of predictions (when sorted by probability) the actual next word is, averaged across the test data.

Average-Keys-Saved

It is typical for the top three next-word predictions to be displayed and updated as the user types in a mobile keyboard, as described in section ***TODO:*** XXX. Clearly, it is in

[1] Note that $H(p, q)$ is often also used to denote the joint entropy of p and q, which is a different concept.

[2] To see this, consider the following example: L_1 and L_2 both have a vocabulary of 5 words; L_1 assigns a probability of 0.3 to w_c, and assigns 0.35, 0.35, 0 and 0 to the remaining words; L_2 assigns a probability of 0.25 to w_c, and assigns 0.2, 0.2, 0.2 and 0.15 to the remaining words. In this example, w_c is given a higher probability by L_1 but ranked higher by L_2.

the interest of the mobile keyboard developer to minimise the amount of typing a user has to do before the correct prediction is displayed. Average-keys-saved is based on this incentive, and is defined as the number of keys that the user would be saved from typing as a result of the correct next word appearing in the top three predictions, averaged over the number of characters in the test data.

As an example, if the user is typing `science` and the word `science` appears in the top three predictions after they have typed `sc`, then that would count as 5 characters being saved, averaging at $\frac{5}{7}$ keys saved per character. Averaging over the number of characters in the test data ensures that the results are not biased by the data containing particularly long words, which are easier to save characters on.

4.1.2 Resource Consumption Metrics

Memory Usage

This is measured as the amount of physical memory in megabytes occupied by the process in which the language model under test is instantiated.

Average Inference Time

This is measured as the amount of time in milliseconds that the language model takes to assign a probability to all of the words in its vocabulary given a sequence of words, averaged over a large number of sequences.

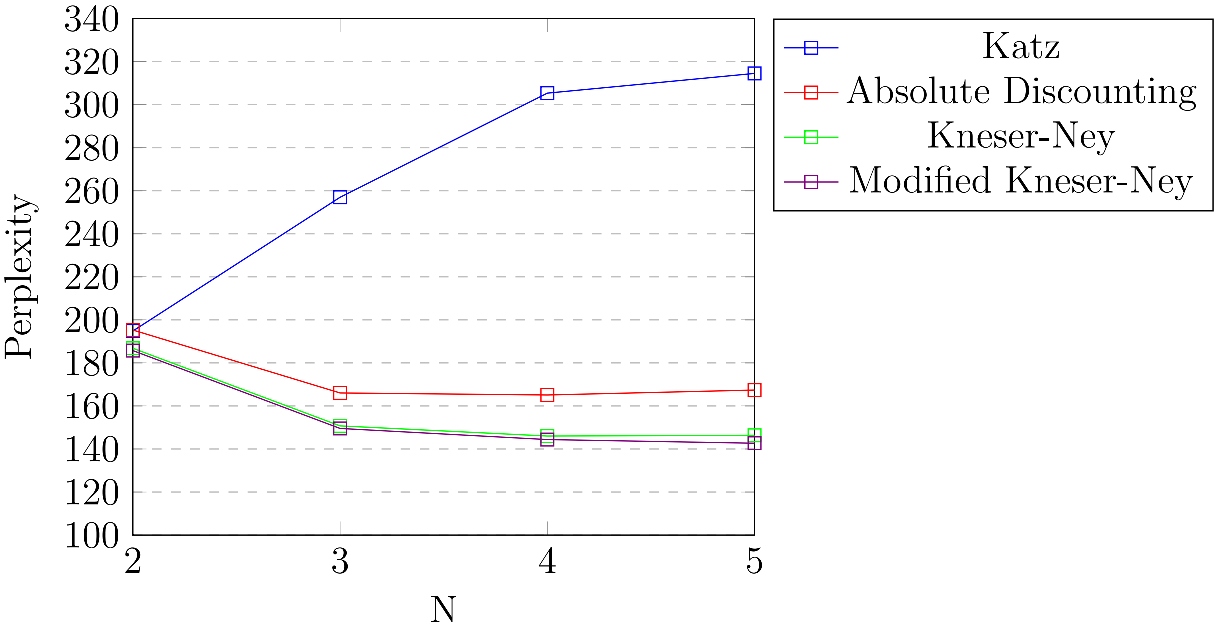
4.2 Results

4.2.1 Existing Models

Things I aim to evaluate here are:
N-Gram Models:

- Accuracy as a function of the amount of training data used (one line per LM), for various smoothing techniques *(on subset of 1BN word dataset)*.
- The effect of increasing N.

Perplexity of N-gram models trained on the PTB dataset.



Neural Models

- Accuracy as a function of the amount of training data used (one line per LM), for various RNN architectures *(on subset of 1BN word dataset)*.
- The effect of changing the number of hidden neurons.

All Models

- A comparison of the performance of all models *(on the PTB dataset)*.

Language Model	**Perplexity**	**Guessing Entropy**	**Average-Keys-Saved**	**Memory Usage (MB)**	**Average Inference Time (ms)**
3-gram	454267.15	1.97123	0.35014	**266.91**	62
3-gram + add-one	3764.96	1.97123	0.53063	266.94	**41**
3-gram + Katz	256.95	4.45221	0.68482	266.71	88
3-gram + absolute disc.	166.03	4.10462	0.72178	266.78	63
3-gram + KN	150.73	4.06848	0.72466	266.88	54
3-gram + modified KN	149.54	4.08662	0.72355	266.97	54
5-gram	195857061.27	0.23265	0.07167	737.36	130
5-gram + add-one	8610.45	0.23265	0.33886	737.30	63
5-gram + Katz	314.49	4.53360	0.67154	737.43	156
5-gram + absolute disc.	167.38	4.06856	0.72333	737.43	126
5-gram + KN	146.35	4.01120	**0.72598**	737.37	114
5-gram + modified KN	**142.68**	4.03443	0.72554	737.53	116
Vanilla RNN 256					
Vanilla RNN 512					
GRU 256					
GRU 512					
LSTM 256					
LSTM 512					
LSTM 512 interpolated with 5-gram + mod. KN					

Combinations of Models

- How combinations of models compare with respect to standalone ones *(on the PTB dataset).*

4.2.2 On a Mobile Device

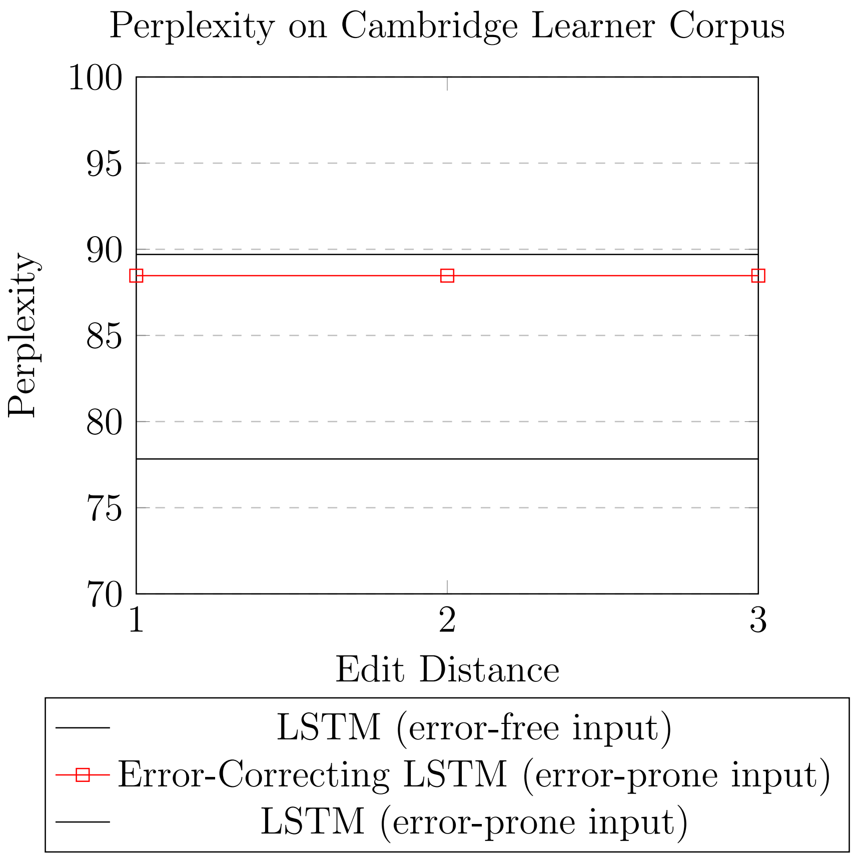
Here, I want to focus on the tradeoff between accuracy and resource consumption. Specifically, I could look at the following:

- The effect of increasing the minimum frequency for a word to be considered in the vocabulary. (I.e. the effect of changing the vocabulary size).
- The effect of changing the number of hidden layer neurons.
- The effect of using RNN vs GRU vs LSTM.
- (Perhaps also the effect of pruning on n-gram models).
- ALSO: Perhaps compare using float 32 vs float 16 in terms of memory vs accuracy.

4.2.3 On Error-Prone Text

Things I aim to evaluate here are:

- The hypothetical upper and lower bounds on accuracy (i.e. the LM results on correct input and on incorrect input respectively).
- The effect of using the vocabulary vs different sized dictionaries for determining if a word should be replaced or not.
- The effect of edit distance on performance.
- An intuitive explanation behind the gap remaining between the current performance and the upper bound. Perhaps some suggestions for future work.



Chapter 5

Conclusion

Appendix A

Project Proposal