UNIVERSITY OF CAPE TOWN

DEPARTMENT OF COMPUTER SCIENCE

CS/IT Honours Final Paper 2020

Title: Low Resource Language Modelling for South African Languages

Author: Jared Shapiro

Project Abbreviation: LOW-LM

Supervisor(s): Jan Buys

Category	*Min*	*Max*	Chosen
Requirement Analysis and Design	*0*	*20*	0
Theoretical Analysis	*0*	*25*	0
Experiment Design and Execution	*0*	*20*	20
System Development and Implementation	*0*	*20*	10
Results, Findings and Conclusions	*10*	*20*	20
Aim Formulation and Background Work	*10*	*15*	10
Quality of Paper Writing and Presentation	*10*		10
Quality of Deliverables	*10*		10
Overall General Project Evaluation (*this section allowed only with motivation letter from supervisor*)	*0*	*10*	0
Total marks		**80**	

Low Resource Language Modelling for South African Languages

Jared Shapiro
University of Cape Town
Cape Town, South Africa
jar.shapiro@gmail.com

ABSTRACT

In the recent two decades, there has been exponential growth in the amounts of text of various types that have become available online. This significant increase in data has allowed language models, which are core components of many natural language processing applications, to greatly increase in quality. Thus, the quality of these applications has increased dramatically as well. Despite these great advances in the field of language modelling, much of the focus has been on language modelling for languages that have large amounts of training data available on them, for language models to use. This has resulted in many low-resource languages receiving far less focus. We trained and evaluated two types of language models, namely n-gram language models and feedforward neural network language models, on the low-resource South African languages of isiZulu and Sepedi. We evaluated each of these types of model's performance and compared their performance against each other. Both types of models perform similarly to each other when trained and evaluated on both languages, however, feedforward neural network language models perform slightly better than n-gram language models when trained and evaluated on datasets of the language isiZulu, while n-gram language models perform slightly better than feedforward neural network language models when trained on datasets of the language Sepedi. These results are explored and avenues for future research are briefly discussed.

CCS CONCEPTS

• **Computing methodologies** → Neural networks; **Machine translation**; • **Information systems** → *Language models.*

KEYWORDS

Language Modelling, Neural Networks, Low-Resource Languages

1 INTRODUCTION

Language modelling is a key component in natural language processing, which is a core component for many types of applications, such as information retrieval, voice recognition, machine translation, spelling correction, and question answering [3, 8, 12, 15]. Language Models assign probabilities to sequences of words [7], thereby, they are defined as a probability distribution of a set of strings over a given context [4, 15].

Recent advances in language modelling have yielded significant improvements in performance, however, these improvements have largely been seen for models that are typically trained on large, high-quality datasets. The performance of language models is largely dependent on the size of the training data available for these models [9]. Thus, when language models are trained on data that is insufficient in terms of size, these models perform poorly or produce undesirable results [9]. For language models that are based on widely used languages such as English, this has largely not been an issue, as accumulating large amounts of data has recently become much easier due to the amount of digital content available online [16]. However, many languages, such as South African languages, lack large amounts of data [16]. These languages which do not have large amounts of data are commonly referred to as "low-resource languages".

The languages typically studied for language modeling are typologically1 very different from most African languages, such as isiZulu or Sepedi. African languages are considered as morphologically rich languages [13], which means that the language's grammatical relations are indicated by changes in the words, rather than the words' relative position in the sentence. African languages are also agglutinative, where the words in the language are composed of the combination of smaller morphological units. The nature of these languages can lead to significantly large vocabulary sizes, where there are many unique words that appear relatively seldom, despite unique "sub-words" that appear relatively often.

This study aims to determine whether these recent advances will also improve performance in low-resource South African Languages. This is done by evaluating the performance of a traditional n-gram language model and a feedforward neural network language model (FFNNLM) on a selection of these low-resource South African languages. The most accurate iterations of these models is evaluated and compared against each other. This is done by obtaining datasets for each language and suitably cleaning them of unwanted artifacts. The open-source implementation of each type of model is then adapted for training and evaluation on these selected languages. The hyperparameters of each model are then optimised, which requires many training runs. The resulting models of these optimisation processes are then evaluated based on the chosen evaluation metrics on a test set for each language from a specific language class, namely Nguni and Sotho-Tswana. These languages are isiZulu and Sepedi. These metrics are then analysed to determine whether there is a significant difference between the two models in terms of performance. We hypothesise that for both languages, FFNNLMs will slightly outperform n-gram language models.

The rest of this paper is organized as follows: Section 2 provides background of the language models used in this study, the performance metrics used to measure the performance of these models, as well as reviewing similar work. Section 3 describes the datasets used in this study and what processes were done on those datasets. Section 4 explains the model development process. Section 5 presents the experiment's results, while Section 6 analyzes and discusses these results, including the opportunity for future work that could extend this study. Finally, Section 7 concludes this study.

2 BACKGROUND AND RELATED WORK

One of the most challenging problems that the field of natural language processing faces, is that of data sparsity, where there is not a sufficient amount of training data to model a language accurately [1], as well as what is commonly referred to as the "curse of dimensionality", which refers to how a word sequence on which the model will be tested, is often different from word sequences seen in the training data [2]. This necessitates using other methods of estimating the probability of a word, with various techniques being used by the most popular language models.

The joint probability of all words $P(w_1, w_2, ...w_n)$ can be decomposed as follows, using the chain rule, into the probability of each word given its preceding context [9]:

$$\begin{aligned} P(w_1^n) &= P(w_1)P(w_2|w_1)P(w_3|w_1^2)...P(w_n|w_1^{n-1}) \\ &= \prod_{k=1}^{n} P(w_k|w_1^{k-1}) \end{aligned} \tag{1}$$

Equation (1) defines a statistical language model, which is represented by the conditional probability of the next word, given the word's history.

2.1 N-grams

N-gram models are one of the simplest widely used language models [6, 14], which perform well across a variety of language modellings tasks. N-gram models make use of the approximation that the probability of a word depends entirely on the previous $(n - 1)$ words. This approximation relates to the idea of Markov models, which states that we can predict the probability of some future unit without looking too far into the historical context of that future unit. This approximation relates to n-gram models in terms of future words and their history [5, 9]. The general approximation for n-grams relating to the calculation of the probability of a future word in a sequence is as follows:

$$P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-N+1}^{n-1}) \tag{2}$$

An example of using this general approximation can be shown with the use of the most common n-gram model, the trigram language model [6]. A trigram model computes the probability of a word by considering the context of the two words that precede it: $P(w_n|w_{n-2}^{n-1})$. This probability can be approximated by using the maximum likelihood estimate (MLE), which can be calculated by counting the frequency at which the word sequence $(w_{n-2}w_{n-1}w_n)$ appears in some training corpus, and dividing that value by the frequency at which the word sequence $(w_{n-2}w_{n-1})$ appears [5, 9]. This equation is as follows:

$$P(w_n|w_{n-2}^{n-1}) = \frac{C(w_{n-2}w_{n-1}w_n)}{C(w_{n-2}w_{n-1})} \tag{3}$$

Where the function C refers to the frequency of a word sequence.

This approximation can be done for any order n-gram, with the general case of MLE n-gram parameter estimation being as follows:

$$P(w_n|w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1}w_n)}{C(w_{n-N+1}^{n-1})} \tag{4}$$

Potential insufficient amounts of data and the inherent sparseness in statistical modelling may cause the use of the MLE to lead to undesirable results for many applications, as these issues may lead to the assignment of zero probability to unseen sequences in the training data [5]. This would lead to inaccurate or undefined probability estimates, as an event that was not observed in some training data could potentially be seen in some test data [5]. In the case where there is a significantly large amount of training data, data sparsity still becomes an issue if one expands the model [5]. To address this issue, smoothing techniques, which largely produces probabilities very close to the MLE, can be used by modifying the MLE to produce more accurate results [5]. This modification involves changing the probability distributions to be more uniform, by increasing significantly low probabilities and decreasing significantly high probabilities [5]. An example of this would be the increase in the probability of a sequence that was initially assigned a probability of zero.

2.1.1 *Smoothing.*

No matter how much data is available for use, smoothing techniques can almost always improve the performance of language models without much effort [3].

Within the context of n-gram models, Modified Kneser-Ney [5] smoothed models achieve the best performance when compared to all other smoothing techniques, as well as performing significantly better than models that are smoothed with regular Kneser-Ney smoothing [5]. As its name suggests, Modified Kneser-Ney is a modification of regular Kneser-Ney smoothing, which has the following form:

$$P_{KN}(w_i|w_{i-n+1}^{i-1}) = \begin{cases} \frac{max[C(w_{i-n+1}^{i})-D,0]}{\sum_{w_i} C(w_{i-n+1}^{i})} & \text{if } C(w_{i-n+1}^{i}) > 0 \\ \gamma(w_{i-n+1}^{i-1})P_{KN}(w_i|w_{i-n+2}^{i-1}), & \text{if } C(w_{i-n+1}^{i}) = 0 \end{cases} \tag{5}$$

where $\gamma(w_{i-n+1}^{i-1})$ is chosen to make the distribution sum to 1, and D is a fixed discount, where $D \leq 1$.

Modified Kneser-Ney [5] differs from regular Kneser-Ney smoothing as instead of using a single discount D for all nonzero counts ,as in Kneser-Ney smoothing, three or more parameters D_1, D_2, D_{3+} are applied to n-grams that have been seen once, twice, and three or more times, respectively, in the training data.

2.2 Feed-Forward Neural Network Language Model

The first type of neural networks introduced to the field of language modelling was Feedforward Neural Networks (FFNNs), which adopted the paradigm of supervised learning, which refers to output targets being provided for each input pattern, which is followed by explicitly correcting the errors of the network [19]. In contrast to traditional count-based n-gram models, FFNNs estimate probabilities of words and sequences via the use of a neural network, rather than using smoothing methods to estimate probabilities. Although FFNNs differ from n-grams in terms of probability estimation, both models are based on the Markov assumption [18].

FFNNs are built up from three different layers; the input layer, hidden layer, and output layer, which consist of input, hidden, and output units respectively. While there can be multiple hidden layers in FFNNs, there is always only one input and output layer [10]. Layers in FFNNs are fully-connected, which means that each unit in a layer processes all the separate outputs of the units in the previous layer, as input. Layers that have at least one layer succeeding them also have units that send their output to all separate units in the following layer. Each neural unit in FFNNs output values that are referred to as activation values, which are calculated by taking a weighted sum of its inputs, adding a bias term, and applying a non-linear function to this resulting value. This activation value can be represented using vector notation as follows [10]:

$$y = a = f(z) = f(w \bullet x + b) \tag{6}$$

where w is the weight vector, x is the input vector, b is the bias scalar, f is the activation function, a is the activation value, z is the intermediate output of the node, and y is the final output of the network, which in this case, consists of a single unit.

The three most popular non-linear functions used by FFNNs to compute activation values are the sigmoid, tanh, and rectified linear unit (ReLU) [10]. The sigmoid function can be defined as follows:

$$a = \sigma(z) = \frac{1}{1 + \exp^{-z}} \tag{7}$$

Tanh function:

$$a = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{8}$$

ReLU function:

$$a = max(f(x), 0) \tag{9}$$

FFNNs are able to perform some hidden layer computation more efficiently by using a single matrix W for the weights of some hidden layer. This enables the hidden layer computation to be done with simple matrix operations [10]. Each hidden unit in some hidden layer has a weight vector w, and a bias scalar b. Thus, these parameters can be represented for some hidden layer by combining them for each hidden unit i with weight w_i and bias b_i, into a single weight matrix W and a single bias vector b. W now represents all the parameters from some hidden layer, with each element W_{ij} of W representing the weight applied to the ith input unit x_i to the jth hidden unit h_j. After these steps, hidden layer computation can now be done much more efficiently. This can be done by multiplying the input vector, x, by the weight matrix W, and adding the bias vector b to it to produce an intermediary result. This intermediary result can now be used as input to the activation function. The resulting value h, can be given or "fed forward" to successive hidden layers or the output layer as their input. This output from the hidden layer can be defined as follows [10]:

$$h = \sigma(Wx + b) \tag{10}$$

The output layer also has a weight matrix, U, that is used in with its inputs (outputs from the previous hidden layer) and bias scalar. It must be noted that although the output of hidden layers can be real-valued numbers, the result of the output layer is used to make classification decisions [10]. Thus, this final result of the output layer must be focused on the case of classification. The softmax function converts a vector of real-valued numbers into a vector that encodes a probability distribution and can be used as the activation function for the output layer. It is defined as follows:

$$softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{d} e^{z_j}} 1 \leq i \leq d \tag{11}$$

where z is the vector of the intermediate output of the output layer that has dimensionality d.

A two-layer FFNN that uses ReLU as the activation function for all intermediate outputs, except the final layer intermediate output, which uses the softmax function, can be represented as follows:

$$\begin{aligned} z^{[1]} &= W^{[1]}a^{[0]} + b^{[1]} \\ a^{[1]} &= ReLU(z^{[1]}) \\ z^{[2]} &= W^{[2]}a^{[1]} + b^{[2]} \\ a^{[2]} &= softmax(z^{[2]}) \\ \hat{y} &= a^{[2]} \end{aligned} \tag{12}$$

where the superscripts in square brackets refer to the layer number, and $\hat{y}$ is the final output of the network, which, in this case, is a vector that encodes a probability distribution.

Some of the advantage that FFNNs have over traditional n-gram language models are their capacity to handle much longer word histories, they do not require smoothing techniques, and they can generalize to unseen data better through the use of embeddings, which are used to represent the history or prior context of a word [10].

2.3 Performance Metrics

Intrinsic evaluation methods measure the quality of a language model independent of applications. These methods include cross-entropy and perplexity, which are derivatives of the probability that the model assigns to test data. Intrinsic evaluation metrics give us a measure of how well the model fits the data we evaluate against [9]. In practice, the use of raw probabilities as a metric of performance is not used. Instead, cross-entropy and perplexity are used.

Cross-entropy is useful if the probability distribution p that generated some data, is unknown to us. The cross-entropy $H(W)$ of a model $M = P(w_i|w_{iN+1}...w_{i1})$ on a sequence of words W is defined as [5, 9] :

$$H(W) = \frac{1}{N} \log P(w_1 w_2 ... w_N) \tag{13}$$

where N is the total number of words in the data evaluated against. This value can be interpreted as the average number of bits needed to encode each word in the test data, using the compression algorithm associated with the specific language model [5].

The perplexity $PP(W)$ of a model can be defined as the exp of the model's cross-entropy value [5]:

$$\begin{aligned} PP(W) &= 2^{H(W)} \\ &= P(w_1 w_2 ... w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 ... w_N)}} \\ &= \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1 ... w_{i-1})}} \end{aligned} \tag{14}$$

The lower the cross-entropy and perplexity of a model is, the better the model is said to perform [5].

Although cross-entropy and perplexity measure the performance of language models independent of applications, these performance metrics cannot be compared across language models with different vocabulary sizes, as lower vocabulary sizes decrease the cross-entropy and perplexity of a language model. Thus, the key performance metric used in this study is bits-per-character (BPC), which can be used to compare language models across different vocabulary sizes. BPC can be defined with the use of the model's cross-entropy value as follows:

$$BPC(W) = H(W) * \frac{T}{C} \tag{15}$$

Where T refers to the number of tokens in a dataset, and C refers to the number of characters in a dataset.

The reason we can use cross-entropy to calculate BPC, is because cross-entropy is a measure of $information/event$. If the cross-entropy is calculated with log to the base 2, then these units of information are bits. This means that a cross-entropy calculation using log to the base 2 is a measure of $bits/event$. We can regard the events as being tokens related to datasets. Thus, to calculate BPC, or $bits/characters$, we simply need to multiply the cross-entropy by the total number of tokens over the total number of characters of a test set.

2.4 Related Work

The research field of language modelling has seen significant improvements in recent history, however, the majority of the research has been concentrated on languages with significantly large datasets [15]. Thus, there has not been much research on how these recent strides in the performance of language models relate to low-resource languages, such as South African languages.

A 2014 study by Gandhe, Metze, and Lane [8] investigated and compared the performance of a modified Kneser-Ney [11] n-gram language model, Feedforward Neural Network Language Model, and a Recurrent Neural Network Language Model (RNNLM), which were trained on a significantly limited amount of training data. The results of this study demonstrated that, under low-resource conditions (approx. 100k training tokens), FFNNLMs perform better than RNNLMs. The low-resource languages modelled on were Tagalog, Pashto, Cantonese, Turkish, and Vietnamese. The study showed that as the size of the training data increased, the relative improvement from the modified Kneser-Ney model to Neural Network models increased as well.

A 2018 study by Scarcella [15] also investigated and compared modified Kneser-Ney N-gram language models to basic RNNs on a severely limited amount of language model training data. The languages that this study modelled on were Xitsonga, IsiZulu, Afrikaans, and English. It was shown that the n-gram models used in this study performed better across all languages except Afrikaans, with the n-gram model having a particularly significant improvement over the RNN model on the IsiZulu dataset. This study also showed that a linear interpolation of the two models performed significantly better than each individual model.

3 DATASET DESCRIPTION AND PREPROCESSING

For the language models used in this study to be able to train on South African language data and evaluate these models on datasets of these specific languages, it was necessary to acquire datasets for each language that the language models will train and be evaluated on, namely isiZulu and Sepedi. As such, this study required South African language corpora of sufficient size and quality.

3.0.1 Autshumato Project Corpora. The Autshumato project[1] was initiated in 2007, and is funded by the South African Department of Arts and Culture. The project provides resources such as parallel

[1]Datasets are available at https://sourceforge.net/projects/autshumato/files/Corpora/

corpora for three pairs of South African languages, namely English-Afrikaans, English-Sepedi, and English-isiZulu. These are sentence-level, aligned corpora, constructed via a combination of automatic and manual alignment techniques, and sourced from the South African government domain.

3.0.2 Dataset Preprocessing. There were many unwanted artifacts in the datasets obtained, mostly owing to most of them being obtained from web-scraped articles, which needed to be stripped out for this research. These unwanted artifacts included HTML tags, URLs, Twitter tags, as well as English web-related sentences such as software warnings. It was therefore necessary for these datasets to be pre-processed, or cleaned, before use. These datasets were then partitioned into their training, validation, and test sets. The training set was produced by obtaining the first 80

4 MODEL DEVELOPMENT AND IMPLEMENTATION

This study used Google Colab[2], a service that provides a notebook environment that runs entirely in the cloud, while also providing the use of a GPU. The current study uses this notebook environment to download and run all of the necessary programs to construct the language model, as well as providing the ability to write and execute python code to achieve this purpose.

4.1 Tokenization

The datasets used in this study have significantly large vocabularies, owing mainly to the extensive use of agglutination in South African languages. Thus, to reduce the vocabulary size of language models trained on the segmented training sets available, Byte-Pair encodings [17] (BPE) was used as a tokenization strategy, where common groups of characters may be grouped into single tokens, resulting in a far smaller vocabulary size. This was necessary, as an overly large vocabulary often leads to the poor performance of language models. Before any of the language models in this study were built or evaluated for a specific language, a BPE vocabulary, which could vary in size, was created from the training set of that language, and this vocabulary was then used to encode the identification numbers of these tokens to a new file for the original training, validation, and testing datasets. The language models were then trained on these encoded files, as well as evaluated on the encoded validation and test datasets.

4.2 N-gram Language Model

For our n-gram language model, KenLM[3], an n-gram with modified Kneser-Ney smoothing estimation and inference implementation was used. The KenLM implementation supports high-performance estimation by map-reduce parallelisation. To use KenLM, the library needed to be downloaded, built, and setup.

Once KenLM was successfully implemented, the open-source Huggingface Tokenizers[4] library was used to train a byte-level BPE tokenizer model, with a specified vocabulary size, on a selected training dataset of a specific language, namely isiZulu or Sepedi, from the data source. Using this BPE vocabulary, the original training, validation, and test datasets were encoded. Their encoded identification numbers(IDs) were then written, space-separated, to separate files respectively. KenLM was then used to train an n-gram model of chosen *n*, on the new training dataset of space-separated IDs, which produced a language model. This model was created as an arpa formatted file, a text format that is supported by language modelling toolkits. This file was then converted to a binary format, klm, to improve performance relating to evaluation. This model was then evaluated on the newly encoded test dataset. The evaluation metrics perplexity and bits-per-character (BPC) were then calculated and recorded for later use. This process was repeated through multiple cycles, with each cycle having a different set of parameter values, to determine the best parameter values which produced the most accurate model.

Both the vocabulary size and *n*, of the nth order n-gram, were able to be changed through every cycle, and these variables were our parameters that were optimised over many iterations. We evaluated the BPE vocabulary size over the range {1000,10000}, with an interval size of 1 thousand within this range. We evaluated the order of n-gram in the range {0,5}, with an interval size of 1 within this range.

Each language model was optimised by changing the parameters over multiple cycles, with the parameters being changed in accordance with which values led to the language models achieving better performance metrics. The best performing language models for each language were then directly compared to each other in terms of performance.

4.3 Feed-Forward Neural Network Language Model

For our Feedforward neural network language model (FFNNLM), this study used the Pytorch implementation[5] to train and evaluate our FFNNLM on our previously encoded datasets. Once the implementation was set up, the model was trained on the encoded training dataset of a specific language, namely isiZulu or Sepedi, and evaluated on the encoded verify and test datasets of the same language. Each language model relating to a specific language was optimised to produce the most accurate results possible.

A range of hyperparameters was considered to best optimize the FFNNLMs. We evaluated the BPE vocabulary size in the range {2000,10000}, with an interval size of 2 thousand within this range. We evaluated the size of the word embeddings in the range {500,2500} with an interval of size 250 within this range, hidden units per layer in the range {500,2500} with an interval size of 250 within this range, the number of hidden layers sizes {1,2,3}, and learning rates in the range {5,25} with an interval size of 5 within this range. We also evaluated batch sizes {10,20,30}, sequence lengths {35,70,140,280}, dropout percentages applied to the layers in the range {30,80} with an interval size of 5 within this range, tying the word size of the word embeddings to the softmax weights, and the order of the n-grams in the range {0,5} with an interval of 1 within this range. It was decided that 50 training cycles for each model were necessary,

[2]Google Colab is available for use at https://colab.research.google.com/
[3]KenLM is available at https://github.com/kpu/kenlm
[4]Huggingface Tokenizers is available at https://github.com/huggingface/tokenizers

[5]Pytorch including FFNNLM support is available at https://github.com/janmbuys/word-language-model

Table 1: Dataset sizes for each language, including each partitioned dataset.

Language	Family	Dataset Sizes (Thousands)					
		Sentences			Tokens		
		Training	Validation/Test	Total	Training	Validation/Test	Total
isiZulu	Nguni	22	3	28	264	33	330
Sepedi	Sotho-Tswana	27	3.5	34	547	68.5	684

as more training cycles did not improve performance by a significant amount and each cycle added increased training time of the model.

After each FFNNLM was evaluated, the resulting perplexity was analysed and the BPC metric was calculated. The BPC was then used to determine new values to set the parameters to, based on whether there was an increase or decrease in the performance of the model. This process underwent many iterations with different sets of hyperparameter values to best determine the hyperparameter values which gave the best possible performance metrics of the resulting language model. Once these results were collected, and there was confidence that further tuning of the model parameters would not lead to significant improvements in the model's performance metrics, we concluded the optimisation process for FFNNLMs for the specific language.

Once the optimisation process for FFNNLMs trained and evaluated on datasets relating to both languages was done, the final results of the best performing FFNNLMs and their sets of parameters were tabulated, analysed, and compared in terms of their performance.

The best results achieved for the n-gram language model were then compared to the best results achieved for the FFNNLM model for each language modelled on, as well as their overall performance.

5 RESULTS

5.1 N-gram Language Model

Preliminary results from testing many different parameter values demonstrated that the 6-gram language model consistently outperformed other n-grams for both languages, across all BPE vocabulary sizes tested. For this reason, the testing was later focused to only test 6-gram (5-order n-gram) language models at different vocabulary sizes.

Table 2 illustrates the best performing n-gram language models' BPC performance metric, as well as each model's hyperparameters, namely the BPE vocabulary size and the n-gram order of the model. As can be seen in Table 2, the language model trained and evaluated on the isiZulu language scored a performance metric of 1.904 BPC with a BPE vocabulary size of 6 thousand tokens, 1.885 BPC with a BPE vocabulary size of 4 thousand tokens, 1.854 BPC with a BPE vocabulary size of 2 thousand tokens, and 1.832 BPC with a BPE vocabulary size of 1 thousand tokens. The language model trained and evaluated on the Sepedi language scored a performance metric of 1.732 BPC with a BPE vocabulary size of 6 thousand tokens, 1.720 BPC with a BPE vocabulary size of 4 thousand tokens, 1.710 BPC with a BPE vocabulary size of 2 thousand tokens, and 1.705 BPC with a BPE vocabulary size of 1 thousand tokens.

Table 2: Summary of the best performing KenLM n-gram models and their bits-per-character (BPC) performance metric on the isiZulu and Sepedi datasets. For each model trained on a specific language, that model's hyperparameters, namely BPE vocabulary size and n-gram order, is shown. The best performing model's BPC score is shown in bold for each language modelled on.

	Hyper-Parameters		
Language	Vocabulary Size	n-gram order	BPC
isiZulu	6000	5	1.904
isiZulu	4000	5	1.885
isiZulu	2000	5	1.854
isiZulu	1000	5	**1.832**
Sepedi	6000	5	1.732
Sepedi	4000	5	1.720
Sepedi	2000	5	1.710
Sepedi	1000	5	**1.705**

Analysis of these results shown in Table 2 reveals that the best performing isiZulu n-gram language model, which scored 1.832 BPC, was the isiZulu model that used a BPE vocabulary size of 1 thousand tokens, and was of n-gram order 5. The best performing Sepedi n-gram language model, which scored 1.705 BPC, was the Sepedi model that used a BPE vocabulary size of 1 thousand tokens, and was of n-gram order 5. It must be noted that the language models consistently performed best when using a lower BPE vocabulary size. This was true for language models trained on both the isiZulu and Sepedi BPE datasets. It can also be seen that language models trained and evaluated on the Sepedi language consistently performed better than the language models trained and evaluated on the isiZulu language.

5.2 Feed-Forward Neural Network Model

Preliminary results from testing many different parameter values across their different ranges demonstrated that certain parameters, when set to specific values, always yielded the best performing models across the different BPE vocabulary sizes tested. Thus, we have chosen to only present the best results across the range of the best performing models that used different BPE vocabulary sizes.

These hyperparameters and their chosen values, which remained the same across the best performing models tested, were the size of word embeddings of value 2000, the hidden units per layer of value 2000, the number of hidden layers of value 1, the learning rate of value 10, the dropout percentage of value 0.7, the batch size of value 10, the sequence length of value 280, and tying the word embeddings to the softmax weights.

Table 3 illustrates the best performing FFNNLMs' BPC performance metric, as well as each model's hyperparameters, namely the BPE vocabulary size and the n-gram order of the model. As can be seen in Table 3, the language model trained and evaluated on the isiZulu language scored a performance metric of 1.822 BPC with a BPE vocabulary size of 10 thousand tokens, 1.815 BPC with a BPE vocabulary size of 8 thousand tokens, 1.824 BPC with a BPE vocabulary size of 6 thousand tokens, 1.851 BPC with a BPE vocabulary size of 4 thousand tokens, and 1.911 with a BPE vocabulary size of 1 thousand tokens. The language model trained and evaluated on the Sepedi language scored a performance metric of 1.716 BPC with a BPE vocabulary size of 10 thousand tokens, 1.716 BPC with a BPE vocabulary size of 8 thousand tokens, 1.724 BPC with a BPE vocabulary size of 6 thousand tokens, 1.744 BPC with a BPE vocabulary size of 4 thousand tokens, and 1.817 with a BPE vocabulary size of 2 thousand tokens. All of the language models presented in Table 3 were of n-gram order 2, except for the isiZulu language model trained on a BPE vocabulary of 2 thousand tokens, which was of n-gram order 3.

Analysis of these results shown in Table 3 reveals that the best performing isiZulu FFNNLM, which scored a BPC of 1.815, was the isiZulu model that used a BPE vocabulary size of 8 thousand tokens and was of n-gram order 2. The two best performing Sepedi FFNNLMs, which both scored a BPC of 1.716, were the Sepedi models that were of n-gram order 2 and used a vocabulary size of 10 thousand and 8 thousand BPE tokens. It can also be seen that language models trained and evaluated on the Sepedi language consistently performed better than the language models trained and evaluated on the isiZulu language.

Table 3: Summary of the best performing feedforward neural network models at different BPE vocabulary sizes and their bits-per-character (BPC) performance metric on the isiZulu and Sepedi datasets. For each model trained on a specific language, the BPE vocabulary size and n-gram order, is shown. The best performing model's BPC score is shown in bold for each language modelled on.

	Hyper-Parameters		
Language	Vocabulary Size	n-gram order	BPC
isiZulu	10000	2	1.822
	8000	2	**1.815**
	6000	2	1.824
	4000	2	1.851
	2000	3	1.911
Sepedi	10000	2	**1.716**
	8000	2	**1.716**
	6000	2	1.724
	4000	2	1.744
	2000	2	1.817

5.3 Comparison Between n-gram Model and Feed-Forward Neural Network Model

Table 4 illustrates a direct comparison between n-gram models using KenLM and FFNNLMs. The table shows the best performing model/s for each model type across both languages tested. The table shows a summary of each model's hyperparameters and illustrates each model's BPC score. As can be seen from this table, for the language of isiZulu, the best performing FFNNLM, which used a BPE vocabulary size of 8 thousand tokens and was of n-gram order 2, outperformed the best performing n-gram model slightly, as the BPC score for the best performing isiZulu FFNNLM was 1.815, while the best performing isiZulu n-gram model achieved a BPC score of 1.832. For the language of Sepedi, the best performing n-gram model, which used a BPE vocabulary size of 1 thousand tokens and was of n-gram order 5, outperformed the best performing FFNNLM slightly, as the BPC score for the best performing Sepedi n-gram model was 1.705, while the best performing Sepedi FFNNLM achieved a BPC score of 1.716.

Table 4: The best performing models across both model classes, and their bits-per-character (BPC) performance metric on the isiZulu and Sepedi datasets. For each model trained on a specific language, that model's hyperparameters, namely BPE vocabulary size and n-gram order, as well at the model type, is shown. The best performing model's BPC score is shown in bold for each language modelled on.

		Hyper-Parameters		
Language	Model	Vocab Size	n-gram order	BPC
isiZulu	n-gram	1000	5	1.832
	FFNNLM	8000	2	**1.815**
Sepedi	n-gram	1000	5	**1.705**
	FFNNLM	10000	2	1.716
	FFNNLM	8000	2	1.716

It must be noted that across the two different models, n-gram language models achieved better results as the size of the BPE vocabulary was lowered, whilst this was not the case for FFNNLMs.

6 DISCUSSION

Since FFNNLMs, like n-gram models, also make use of n-grams at a basic level, we found it unexpected that the best-performing FFNNLMs used trigrams, as the best performing n-gram models in this study were exclusively 6-gram models. The reasons for this difference in optimisation are unclear, however, many of the FFNNLMs tested that used 6-grams in their model came close to achieving the same performance as the best performing FFNNLMs at certain BPE vocabulary sizes. This leads us to believe that FFNNLMs that made use of 6-grams may have been able to achieve the best performance, as although thorough testing and optimisation was done, more stringent and precise optimisation of the FFNNLMs, possibly across expanded ranges of hyperparameters, could have led to more accurate models. Future work could explore whether this could be achieved.

It must be noted that although Sepedi language models, across both types of language models tested, achieved significantly better performance metrics than isiZulu language models, this was likely due to roughly half the number of tokens in the training data available to our models for the language of isiZulu when compared to the number of tokens in the training data available to our models for the language of Sepedi.

It was unexpected that the best n-gram models performed very close to the level of the FFNNLMs when trained and evaluated on the isiZulu datasets and that the best n-gram models outperformed the best FFNNLMs when trained and evaluated on the Sepedi datasets. We expected that the FFNNLMs would outperform n-gram models for both languages by a slight, but significant margin. We suspect that further optimisation may have allowed FFNNLMs to outperform n-grams for the language of Sepedi and that given more training data, the performance gap would swing in FFNNLMs favor.

7 CONCLUSIONS

The implementation and analysis of the performance of language models when modelled on low-resource languages, such as the South African languages isiZulu and Sepedi, is important as language modelling is a key component in natural language processing, which is used for many types of applications, and language modelling on low-resource languages has not been widely studied.

In this study, we implemented, optimised, and tested many language models of two different types, namely traditional n-gram models and feedforward neural network language models, on the low-resource languages of isiZulu and Sepedi. Both types of models were evaluated and then compared against one another, in terms of their performance, which was measured in bits-per-character. Our results show that for both the languages of isiZulu and Sepedi, in low-resource conditions, traditional n-gram models and FFNNLMs performed similarly in terms of performance, however, the most accurate FFNNLM performed slightly better than the most accurate n-gram model when trained and evaluated on the language of isiZulu. The opposite is true when these models were trained on the language of Sepedi, where the best performing traditional n-gram model slightly outperformed the best performing FFNNLM.

In terms of possible future work, n-gram models using different types of smoothing methods could be implemented, tested, and analysed to determine if there is an uplift in performance over the results achieved in this study. Our results obtained in this study have the potential to assist researchers to evaluate which types of language models would work best in low-resource settings, especially in the context of South African low-resource languages, specifically isiZulu and Sepedi.

REFERENCES

[1] Ben Allison, David Guthrie, and Louise Guthrie. 2006. Another look at the data sparsity problem. In *International Conference on Text, Speech and Dialogue*. Springer, 327–334.

[2] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *Journal of machine learning research* 3, Feb (2003), 1137–1155.

[3] Catherine Chavula and Hussein Suleman. 2016. Assessing the Impact of Vocabulary Similarity on Multilingual Information Retrieval for Bantu Languages. In *Proceedings of the 8th Annual Meeting of the Forum on Information Retrieval Evaluation (FIRE '16)*. Association for Computing Machinery, New York, NY, USA, 16–23. https://doi.org/10.1145/3015157.3015160

[4] Stanley F Chen and Joshua Goodman. 1999. An empirical study of smoothing techniques for language modeling. *Computer Speech & Language* 13, 4 (1999), 359–394.

[5] Stanley F Chen and Joshua Goodman. 1999. An empirical study of smoothing techniques for language modeling. *Computer Speech & Language* 13, 4 (1999), 359–394.

[6] Alexander Clark, Chris Fox, and Shalom Lappin. 2013. *The handbook of computational linguistics and natural language processing*. John Wiley & Sons.

[7] Jurafsky Daniel and James H Martin. 2019. N-gram Language Models. In *Speech and Language Processing* (3 ed.). Chapter 3.

[8] Ankur Gandhe, Florian Metze, and Ian Lane. 2014. Neural Network Language Models for low resource languages. In *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*. Carnegie Mellon University, International Speech and Communication Association, 2615–2619.

[9] Dan Jurafsky. 2000. *Speech & language processing*. Pearson Education India.

[10] Daniel Jurafsky and James H. Martin. 2019 (accessed May 12, 2020). *Speech and Language Processing*. http://https://web.stanford.edu/~jurafsky/slp3/.

[11] Reinhard Kneser and Hermann Ney. 1995. Improved backing-off for M-gram language modeling. In *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, Vol. 1. IEEE, 181–184. https://doi.org/10.1109/icassp.1995.479394

[12] B. Ndaba, H. Suleman, C. M. Keet, and L. Khumalo. 2016. The effects of a corpus on isiZulu spellcheckers based on N-grams. In *2016 IST-Africa Week Conference*. 1–10.

[13] Laurette Pretorius and Sonja Bosch. 2009. Exploiting Cross-Linguistic Similarities in Zulu and Xhosa Computational Morphology. In *Proceedings of the First Workshop on Language Technologies for African Languages*. Association for Computational Linguistics, Athens, Greece, 96–103. https://www.aclweb.org/anthology/W09-0714

[14] Ronald Rosenfeld. 2000. Two decades of statistical language modeling: Where do we go from here? *Proc. IEEE* 88, 8 (2000), 1270–1278.

[15] Alessandro Scarcella. 2018. *Recurrent neural network language models in the context of under-resourced South African languages*. Ph.D. Dissertation. University of Cape Town.

[16] Alessandro Scarcella. 2018. *Recurrent neural network language models in the context of under-resourced South African languages*. Ph.D. Dissertation. University of Cape Town.

[17] Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural Machine Translation of Rare Words with Subword Units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Berlin, Germany, 1715–1725.

[18] Martin Sundermeyer, Hermann Ney, and Ralf Schlüter. 2015. From feedforward to recurrent LSTM neural networks for language modeling. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 23, 3 (2015), 517–529.

[19] Joe Tebelskis. 1995. *Speech recognition using neural networks*. Ph.D. Dissertation. Carnegie Mellon University.