Recurrent neural network language models in the context of under-resourced South African languages

Alessandro Scarcella

May 17, 2018

Abstract

Over the past five years neural network models have been successful across a range of computational linguistic tasks. However, these triumphs have been concentrated in languages with significant resources such as large datasets. Thus, many languages, which are commonly referred to as under-resourced languages, have received little attention and have yet to benefit from recent advances. This investigation aims to evaluate the implications of recent advances in neural network language modelling techniques for under-resourced South African languages. Rudimentary, single layered recurrent neural networks (RNN) were used to model four South African text corpora. The accuracy of these models were compared directly to legacy approaches. A suite of hybrid models was then tested. Across all four datasets, neural networks led to overall better performing language models either directly or as part of a hybrid model. A short examination of punctuation marks in text data revealed that performance metrics for language models are greatly overestimated when punctuation marks have not been excluded. The investigation concludes by appraising the sensitivity of RNN language models (RNNLMs) to the size of the datasets by artificially constraining the datasets and evaluating the accuracy of the models. It is recommended that future research endeavours within this domain are directed towards evaluating more sophisticated RNNLMs as well as measuring their impact on application focused tasks such as speech recognition and machine translation.

Contents

1 Introduction

1.1 Background to the investigation

Statistical language modelling is the task of fitting a probability distribution to sequences of linguistic units. When properly trained, language models can capture nuanced patterns and structural properties of linguistic sequences. This includes grammatical structure but also contextual detail [15, 16].

Accurate language models serve an important function in several natural language processing (NLP) applications. If trained on suitable text corpora, the knowledge synthesised in a language model can lead to significant performance gains when integrated with other subsystems for a specific task. The most prominent applications are machine translation and speech recognition. In these contexts, statistical language models represent the prior distribution of sequences of words. Several recent studies have demonstrated their effectiveness within these systems [6, 7, 14, 19]. In addition, language models can be used for other tasks like natural language tagging, question answering and summarisation.

1.2 Problem description

An important aspect of building accurate language models is the size of the text corpora [4]. Greater variety and quantity of sequences increase the generalisability of statistical language models and make it less likely that infrequent sequences have zero frequency counts. More concisely, increasing the amount of data should make it more likely that the model is being built using a representative sample. Accumulating large datasets has become feasible because of the amount of digital content. Thus, much research has been focused on the performance of models given abundant data [4, 16]. This research area has sometimes been referred to as *large scale language modelling* [16]. The size of the dataset, at least for well resourced languages, is no longer a notable constraint.

Although there has been growth in the amount of digital resources, much of digital content is concentrated in a few languages. *Under-resourced languages* is a phrase that encompasses multiple facets of a language but one of the qualifying characteristics is a lack of resources for NLP tasks [3]. South Africa has eleven official languages and, with the exception of English, all of these are considered under-resourced. By extension, building language technologies for South Africa is a challenge. A recent study on machine translation for South African languages is a good example of the challenge in building NLP systems [22]. A more general survey of automatic speech recognition (ASR) for under-resourced languages provided a comprehensive analysis of the challenges as well as the motivations for producing language systems in these contexts

[3]. The survey highlights studies on language modelling of two other African languages: Somali and Amharic [1, 24, 31]. A follow up study on Ahmaric looked at the relative importance of audio and text data for under-resourced ASR [23]. The common thread is that the quality and quantity of data affects performance and that various techniques, such as using different linguistic units, have been used in an attempt to mitigate the effect of limited resources.

1.3 Purpose of the research

Language models form part of the core for many NLP applications. Two relevant tasks in the South African context are the transcription of parliamentary dialogue using automatic speech recognition and the automatic translation of government documents into all the official languages. Both of these tasks require accurate language models that have to be built using sparse datasets.

Historically, n-gram language models have been the most accurate [12]. Recently, the advancements in training procedures, dataset size, dataset quality and computational power have lead to neural network based language models outperforming n-gram models [19]. More specifically, the best performing models have been recurrent neural network language models (RNNLMs). The landmark study was conducted by Thomas Mikolov in 2012 whereby a comprehensive analysis was conducted on several common datasets and RNNLMs were compared to n-gram benchmarks [19]. The study established that RNNLMs can improve the performance of statistical language models. Since then, RNNs have been established as the best performing language models in academia and industry [4, 16, 19].

The emergence of RNNLMs raises the question of whether RNNs can improve language modelling for under-resourced languages and, specifically, South African languages. There is currently no published work that analyses the effectiveness of RNNLMs on any South African text corpora. The aim is to appraise the performance of RNN language models against a suitable n-gram benchmark using four South African text datasets. The languages chosen have distinct linguistic properties that should provide further insight into the differences between n-grams and RNNLMs. The performance of individual and combination models will be assessed and interpreted.

1.4 Structure

The investigation will be conducted by first evaluating prior relevant work and stating the necessary theory within the literature review chapter. The next chapter, titled *Experiments*, will detail the tools, experimental setup and results obtained. Finally, there will be a *Conclusion* chapter that will summarise the findings of the investigation and identify areas for further inquiry.

2 Literature Review

2.1 Language Modelling

A language model is a statistical model for a sequence of linguistic units:

$P(\mathbf{L})$ where $\mathbf{L} = l_1, l_2, l_3, ..., l_n$ and l_i is unit i in a sequence of n such units.

This also enables one to assign probabilities to all possible subsequent units in a sequence [17] :

$P(l_i|l_1, l_2, ..., l_{i-2}, l_{i-1}) = \frac{P(L)}{P(l_1, l_2, ..., l_{i-2}, l_{i-1})}$

Although the task is simple to state, accurate language models enable sophisticated applications like automatic speech recognition and machine translation [17].

In addition, language seems to be an important part of human intelligence. Any true demonstration of artificial intelligence would likely have the ability to understand and process natural language. Indeed, the Turing test, albeit flawed, is based on natural language [32]. Language modelling, to some extent, requires the ability to understand natural language. Therefore, it is a reasonable task to use as a gauge for the intelligence of a system, at least in some dimension.

For the sake of readability, we will refer to sequences of words instead of linguistic units when discussing language modelling in general. However, the principles discussed apply to all linguistic units unless explicitly stated otherwise.

2.1.1 Evaluating language models

Evaluating the performance of a language model can be a difficult task. Firstly, since language modelling is usually a component of a larger system for a specific application, the metric of interest is generally the performance on that task [17]. This can be problematic as it can be challenging to ascertain the marginal effect of the language model. Even if all other components are kept the same while the language model is changed, there are interaction effects, with the acoustic or translation model for example, that may not be accounted for. Nonetheless, this form of evaluation is known as extrinsic evaluation [17]. Extrinsic valuation can be useful for tuning a system for a specific task without necessarily aiming to isolate the contribution of the language model.

Intrinsic evaluation aims to identify the strength of a language model independent of any application. Ideally, the intrinsic metric should be expressive of the language model's general ability across several tasks.

2.1.2 Perplexity

One simple and intuitive way to test out a language model is to assess the probability it assigns to a correct sequence that it has not been trained on. Perplexity is related to this idea but has some notable differences.

To best understand perplexity, it is important to refer to information theory and the concept of entropy. Entropy is an attempt to quantify the expected amount of information or uncertainty of a source. Although, uncertainty and information appear to be different properties, information theory links the two. For an event x_i, the quantity of information $I(x_i)$ defined by

$$I(x_i) = \log \frac{1}{P(x_i)} \tag{1}$$

is proportional to the uncertainty of that event i.e. the smaller the probability $P(x_i)$, the larger the amount of information it carries. The intuition is that rare events should provide more information than regularly occurring events.

Information entropy is an extension of this. For a given information source, which yields discrete outcomes x_i from the probability distribution of X, the information entropy, $H(X)$, of X, is the expected value of the quantity of information:

$$H(X) = E(I(X)) = \sum_S P(x_i)I(x_i) = \sum_S P(x_i) log \frac{1}{P(x_i)} = E[-logP(X)] \tag{2}$$

where:

S is the space of discrete outcomes [15].

The unit of information entropy is determined by the base of the logarithm. For example, a base two logarithm produces information entropy values in bits.

It is important to note that information entropy is maximised when there is a uniform probability distribution on the outcomes. Intuitively that makes sense as a uniform distribution indicates no specific knowledge regarding the outcome. Rather it represents a way to consider all outcomes equally likely as there is no insight as to what areas of the outcome space are more likely. Essentially, the occurrence of any specific outcome is relatively informative since the presumption is that all events are equally likely. In addition, since all outcomes are equally probable, there are no outcomes that are highly probable and, by definition, carry less information. In contrast, the information entropy of a source is zero if the probability of a specific outcome is equal to one. That is, if an outcome is certain it does not provide any information.

When the probability distribution that the data is being drawn from is unknown but there is a model of the distribution, the *cross-entropy* between

the two distributions can allow one to infer properties of the entropy of the unknown distribution. In theory, to compute the cross-entropy we would need to consider the entire set of all possible sequences. This is, of course, infeasible. However, according to the Shannon-McMillan-Breiman theorem, an acceptable estimate of the cross-entropy can be obtained, given a suitable length sequence, by:

$$H(p,m) = -\lim_{n\to\infty} \frac{1}{n} log\, m(w_1 w_2 ... w_n) \tag{3}$$

for an unknown probability distribution p and a model m [2]. Where $m(w_1 w_2 ... w_n)$ is the model's probability estimate of the word sequence of length n.

The cross-entropy serves many purposes but for language modelling, an important property is that, given two models, the model that has the lowest cross-entropy is the better model of the data generating distribution. A more thorough discussion of entropy and cross entropy can be found elsewhere in the literature [15, 17].

Finally, the perplexity(PP) of a source (S), $PP(S)$, is defined by:

$$\begin{aligned} PP(S) &= 2^{H(W)} \\ &= 2^{-\frac{1}{N} log P(w_1 w_2 w_3 ... w_n)} \\ &= P(w_1 w_2 w_3 ... w_n)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 w_3 ... w_n)}} \\ &= \sqrt[N]{\prod_{i=1}^{n} \frac{1}{P(w_i | w_1 ... w_{i-1})}} \end{aligned} \tag{4}$$

where $H(W)$ is the cross-entropy of a model on some data from the source. As mentioned earlier, it is explicit that perplexity is related to the probability of the data sequence. As is clearly seen in (4), maximising the probability of the data is equivalent to minimising the perplexity.

Perplexity can be interpreted in many ways and can be used for several purposes. Most importantly for the purpose of the experiments in the investigation, perplexity serves as a metric for how well the language model performs on unseen data. Lower perplexity in this context, implies less unexpected, low probability, outcomes in the sequence i.e. the sequence was predictable for the model. In addition, perplexity is indicative of certain attributes of a language. If an accurate model is used, the perplexity of a representative data source will provide information regarding the language's overall complexity [15]. As a result, comparing the performance of language models on two different languages might provide more information about the

differences in linguistic structure of the languages rather than the differences in the language model.

2.2 N-grams

2.2.1 Introduction

N-grams are a very simple approach to language modelling and were the most effective family of models up until very recently [19]. Fundamentally, n-grams are non-parametric language models based on the relative frequency of sequences.

Take the simple example of a two word sequence. The probability of that specific sequence is computed according to the number of times it occurred in some training corpus relative to all other two word sequences. Explicitly, that would be computed as:

$$P(w_i, w_{i+1}) = \frac{C(w_i, w_{i+1})}{\sum_{\mathbf{W}} C(w_i, w_{i+1})} \tag{5}$$

where:

C is the count of a sequence within the training corpus.
w_i and w_{i+1} are the first and second word of a sequence.
$\mathbf{W}$ is the the entire set of two word sequences.

If it were required that one predict the second word given the first word of a sequence, the probability distribution would be estimated by the count of each possible word being preceded by the first word. That is:

$$P(w_{i+1}|w_i) = \frac{C(w_i, w_{i+1})}{\sum_{w_{i+1}} C(w_i, w_{i+1})} \tag{6}$$

The only difference between (5) and (6) is the denominator. We sum over all two word sequences starting with the first word w_i. The denominator in (6) is equivalent to the count of w_i in the corpus, therefore:

$$P(w_{i+1}|w_i) = \frac{C(w_i, w_{i+1})}{C(w_i)} \tag{7}$$

The above examples demonstrate the use of a *bigram* since the model is based on sequences of two words. N-grams model sequences of N words. In theory, if it were required that the final word of a sequence be predicted, one would like to set N such that the entire sequence is considered in the prediction. Otherwise, if the probability of an entire sentence should be estimated, N would be set to the length of the sentence. Ideally, all available information should be included in an estimate. However, as N increases the number of

potential sequences grows exponentially. The number of possible sequences of length N using a vocabulary of size V is V^N. Therefore, for longer sequences, it is unlikely that there will be enough occurrences of the range of potential sequences within a corpus to form reliable estimates.

To overcome this, firstly, smaller n-grams are used as an approximation for the probability given the entire context of linguistic units.

$$P(l_i|l_1, l_2, ..., l_{i-1}) \approx P(l_i|l_{i-N+1}, ..., l_{i-1}) \tag{8}$$

Secondly, smaller n-gram models are chained together when required to estimate the probability of longer sequences. For example, a six word sentence could be assigned a probability using a chain of bigrams as:

$$P(l_1, l_2, .., l_6) \approx P(l_1)P(l_2|l_1)P(l_3|l_2)P(l_4|l_3)P(l_5|l_4)P(l_6|l_5)$$

2.2.2 Smoothing

As mentioned above, the number of potential sequences for a given n-gram grows exponentially with n. Therefore, in order to learn accurate distributions over higher order n-grams, the datasets need to be large. However, even using large corpora, the training data is unlikely to be representative enough to allow the language model to encode all the relevant information. More specifically, some n-grams may have been unseen in the data and hence get assigned a probability of zero. This can lead to brittle models that are too dependent on the training data. In order to make n-gram language models more general, there are smoothing techniques that redistribute the density estimates so that sequences that had zero counts in the training data are no longer estimated as zero probability sequences.

Importantly, smoothing only relates to sequences constructed using the vocabulary of the training dataset. That is, smoothing aims to deal with specific sequences that are unseen but that consist of words that have appeared in the training data in other sequences. Words that appear in the test set but were not in the training corpus are known as out of vocabulary (OOV) words and are a separate matter that will be discussed later in this section [17].

The most basic concept to deal with the problem is to add a single count to all n-grams. This is known as Laplace smoothing. Alternatively, one can add an arbitrary constant to all n-grams. Fractions can be added even though it is impossible to have a fraction of a count. This can be useful as adding a full count may lead to rare n-grams being estimated as more frequent than is actually the case [9]. Although conceptually simple, these techniques are unreliable [17].

An alternative approach is to gather information from lower order n-grams to inform the probability assigned to higher order n-grams with zero counts. This can provide valuable information regarding the actual density of zero frequency n-grams. There are two general ways to incorporate this information into the estimate.

The first is known as backoff. Backoff gathers information for zero count n-grams by gradually reducing the context until an n-gram with some non-zero count is found. For example if there is a trigram with zero counts in the training corpus, backoff will first look at the bigram count. If the bigram count is non-zero, the trigram will be adjusted, according to the bigram count only. If the bigram count is zero, only then will the unigram be considered.

The second method is interpolation. Interpolation adjusts zero frequency counts by considering *all* lower order n-grams i.e. not only the highest order n-gram with non-zero counts.

For example, if we were to use a trigram, densities could be assigned as follows:

$$P^*(w_3|w_1, w_2) = \lambda_1 P(w_3|w_2, w_1) + \lambda_2 P(w_2|w_1) + \lambda_3 P(w_1)$$

where:

P^* is the adjusted probability
and λ_i is a weighting

This is an instance of interpolation known as linear interpolation. The specific details of how each of these two general methods (backoff and interpolation) adjust zero counts vary and is an active research area. For example, linear interpolation computes the counts of a higher order n-gram according to a weighted sum of all lower order n-grams. However, it can be said that all smoothing techniques redistribute the counts or densities and then normalise in order to derive a valid probability distribution. This can be done by dividing all the counts by the original n-gram count total plus the additional counts that are added through the smoothing:

$$P(S_i) = \frac{C_i + k}{TC + NC}$$

where:

S_i is a specific n-gram.
C_i is the original count of n-gram S_i within the training corpus.
k is the constant that is added to all n-grams for smoothing purposes.
TC is the total count of n-grams in the training corpus.
NC is the total additional counts added for smoothing purposes.

Alternatively the original n-gram counts can be multiplied by a discounting factor so that the denominator remains the same [17]:

$$\begin{aligned} *C_i &= (C_i + k)\frac{TC}{TC + NC} \\ P(S_i) &= \frac{*C_i}{TC} \end{aligned}$$

where:

$*C_i$ is the adjusted count after multiplying by the discounting factor.

A detailed discussion of all the techniques available will not be done as it is not the focus of the research question. Instead, the focus will be on Kneser-Ney smoothing as this is the smoothing method that will be used to establish the n-gram baseline results.

Kneser-Ney smoothing is an interpolated smoothing technique. It is an extension of a technique known as absolute discounting [21]. Absolute discounting is a technique that normalises the distribution by taking a fixed amount off the count of all non-zero count n-grams.

Kneser-Ney discounting builds on this idea by modifying the way that unigram probabilities are estimated. The technique deviates from pure counts by instead trying to measure the diversity of the contexts in which a word appears. This is effective as some words appear frequently but their frequency is concentrated in few contexts i.e. they are preceded by the same word or words. Kneser-Ney counters this by assigning unigram probabilities in proportion to the amount of different times a word appears with a distinct preceding word.

Therefore the computation for Kneser-Ney unigrams is:

$$P_{KN}(w_i) = \frac{|w_{i-1} : c(w_{i-1}w_i) > 0|}{|w_{j-1}w_j : c(w_{j-1}w_j) > 0|} \tag{9}$$

The numerator is the count of distinct contexts (bigrams) that the word w_i appears in. The denominator is a count of all distinct bigrams in the corpus. This ensures that there is a valid probability distribution. Of course, this is just the unigram distribution, an example of a complete interpolated Kneser-Ney smoothing bigram would be defined as:

$$P_{KN}(w_i|w_{i-1}) = \frac{max(c(w_{i-1}w_i) - k, 0)}{c(w_{i-1})} + \lambda_{w_{i-1}} P_{KN}(w_i) \tag{10}$$

where k:

is some constant discount, usually between 0 and 1. There are two well known methods for choosing k. The original Kneser-Ney method defines k as follows:

$$k = \frac{n_1}{n_1 + 2n_2}$$

where n_1 is the number of single count n-grams and n_2 is the number of n-grams with a count of two.

It was later found that a slightly more sophisticated method works better. This is known as *modified* Kneser-Ney smoothing [5] and is defined as:

$$\begin{aligned} D &= \frac{n_1}{n_1 + 2n_2} \\ k_1 &= 1 - 2D\frac{n_2}{n_1} \\ k_2 &= 2 - 3D\frac{n_3}{n_2} \\ k_{3+} &= 3 - 4D\frac{n_4}{n_3} \end{aligned}$$

where:

D is an intermediary term
k_1 is the discount for single count n-grams
k_2 is the discount for two count n-grams
k_{3+} is the discount for n-grams with a count of three or more

This discounting scheme, sometimes referred to as Chen and Goodman's modified Kneser-Ney discounting, has been shown to perform the best and is widely used [5].

One may look at the absolute discount as a decrease in count of amount k, or a decrease in probability of amount $k/c(w_{i-1})$. This amount is discounted for each distinct word that follows w_{i-1} . There are $|w_i : c(w_{i-1}w) > 0|$ such words. Therefore, in order to make the discounting and redistribution balance, $\lambda_{w_{i-1}}$ is set as:

$$\lambda_{w_{i-1}} = \frac{k}{c(w_{i-1})} |w_i : c(w_{i-1}w_i) > 0|$$

The right hand side of the equation, represents the total amount of discounting for bigrams starting with w_{i-1}. Since $P_{KN(w_i)}$ sums to one, this ensures that $P_{KN(w_i|w_{i-1})}$ is maintained as a valid probability distribution.

The extension to the n-gram case is:

$$P_{KN}(w_i|w_{i-n+1}^{i-1}) = \frac{max(c(w_{i-n+1}^{i-1}, w_i) - k, 0)}{\sum_{w_i}(w_{i-n+1}^{i-1}, w_i)} + \lambda(w_{i-n+1})P_{KN}(w_i|w_{i-n+2}^{i-1})$$

2.2.3 Out of vocabulary words

Out of vocabulary (OOV) words refer to words that are unseen in the training text but appear in the evaluation text. That is, the model that is built during the training phase has no knowledge to make any inferences about unseen words that appear in the test set.

This is usually dealt with by creating an *unknown token*. Anytime a word appears for the first time in training, it is treated as an instance of the *unknown token*. The second instance of a word results in a word being given its own token. This way, all unseen words in the test data can be modelled based on the proxy of rare words in the training data.

2.3 Neural Networks and Neural Language Modelling

The most effective class of neural networks (NNs) for language modelling are recurrent neural networks (RNNs) [4, 16, 19]. These networks are an extension of feedforward neural networks (FNNs) [10].

Neural networks can be viewed as several stacks of linear models with non-linear transformations between the stacks. The output of one model, after being transformed, serves as an input for another. They are referred to as networks as they are composed of many functions that in aggregate represent a model [10].

Since interesting problems are rarely linear, neural networks contain modules called nonlinear activation functions. These serve to make the network more flexible in what it can model. Thus, instead of output from one linear model being fed directly into another, a nonlinear transformation of the output is used as input for subsequent modules.

There are several different kinds of activation functions. The study of these functions is an active area of research [10]. However, only the logistic sigmoid activation function will be described as it is the activation function that will be used in the models that were used for the experiments:

$$\sigma(x) = \frac{1}{1 + e^{(-x)}} \tag{11}$$

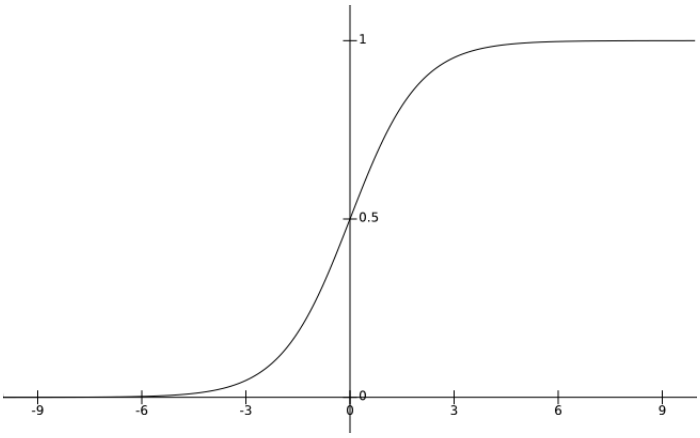

Figure 1: Sigmoid function

Activation functions are sometimes referred to as squashing functions because they usually have a very limited range. The logistic sigmoid function has a (0,1) range. Within a neural network the input to the activation function will be a linear combination of values from previous layers.

Neural networks are perhaps best represented visually:

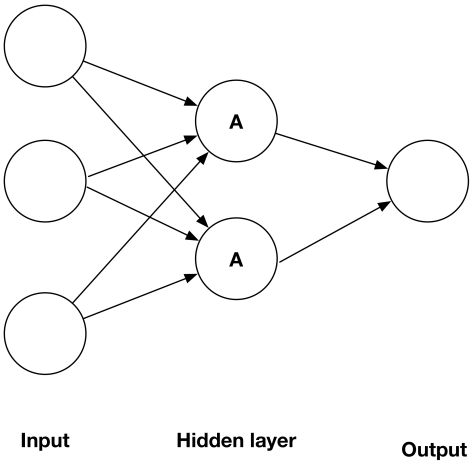

Figure 2: Simple neural network with one hidden layer.

This is an example of a FNN. The input is mapped to output without any temporal considerations.

Each node in the hidden layer is a nonlinear transformation, represented by the symbol **A**, of a linear combination of the nodes in the previous layer.

The illustration above depicts a model with a three dimensional input, one hidden layer with two neurons and an output with one state. Layers indicate the position of a group of nodes within the hierarchy between input data and output. The dimensionality of the output can be adjusted according to the task. Some tasks may require a distribution over some output state space while others will require a scalar quantity output. The output layer often has a different transformation. For example, if it were required that the output be a probability distribution, the output layer could be normalised with a softmax function:

$$s(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}} \tag{12}$$

where $s(y_i)$ is the estimated probability of output state i.

Instead of trying to comprehend an entire neural network layer, it might be easier to analyse each node within the layer separately:

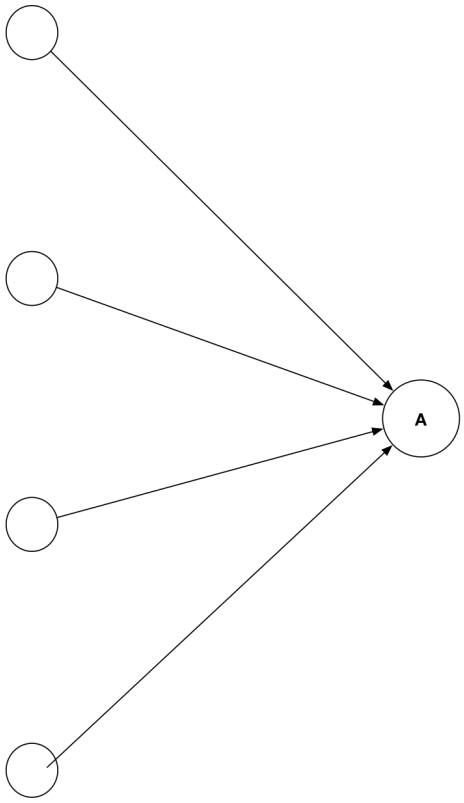

Figure 3: Input to a single node.

Thus, the relationship between two layers can be interpreted as several vector to scalar transformations that in aggregate produce a new layer [10].

Each line represents a separate weight. The weights are the parameters that are adjusted when the model is trained. Higher level decisions about the structure of the network may be chosen using cross validation but are not adjusted during the training procedure. Some of the architectural decisions of a neural network are the number of layers, the width or number of neurons in each layer and the choice of activation function.

2.3.1 Feedforward Neural Networks

A feedforward neural network statically maps from inputs to outputs through a multilevel structure of linear combinations and nonlinear activation functions. There is no concept of time or consideration of prior outputs built into the model. For a three layer neural network, the symbolic representation is:

$$
\begin{aligned}
H &= \sigma(X^T W_1) \\
Y &= \psi(H^T W_2) = \psi(\sigma(X^T W_1)^T W_2)
\end{aligned}
$$

where:

X is the input data vector.
Y is the model output.
H is the hidden layer vector.
σ is the nonlinear activation function.
ψ is the output layer tranformation.
W_i is the weights between layer i & $i+1$.

Neural networks also have a set of parameters known as bias terms. For each neuron of the network, a bias element, in addition to input from the previous layer, is included in the computation to provide the network with more flexibility.

That is:

$$
H_i = \sigma(H_{i-1}^T W_i + B_i) \tag{13}
$$

where:

H_i and H_{i-1} are neuron vectors at layers i and $i-1$, respectively.
σ is the nonlinear activation function.
W_i is the weights between layer i & $i+1$.
B_i is the vector of bias terms between layer $i-1$ & i.

The inclusion of the bias term can also be done by augmenting the input vector with a scalar value of 1 and including an additional weight, relating to the scalar 1, in the weight vector.

2.3.2 Recurrent Neural Networks

Feedforward networks can be used to model sequential data but they are not suited for these tasks [19]. They statically approximate functions using input and output vectors that have no temporal attributes. Recurrent neural networks are specifically designed for sequence modelling. In theory, they can map data from an arbitrary length of prior inputs to the current output [13]. Language modelling is a sequential data task and therefore it is unsurprising that RNNs have been effective in this setting.

Fundamental to recurrent neural networks is the idea of having a persistent model across the entire time series. That is, the weights of the network are the same irrespective of the position of the data point being processed [10]. This has several advantages. Firstly, the model is more flexible. It can process sequences of varying length. In addition, the information processed from sequences of varying lengths are aggregated and this should lead to more effective models.

RNNs process data very similarly to FNNs except that in addition to the raw input from the current time-step, hidden layer neurons from the prior time step are considered:

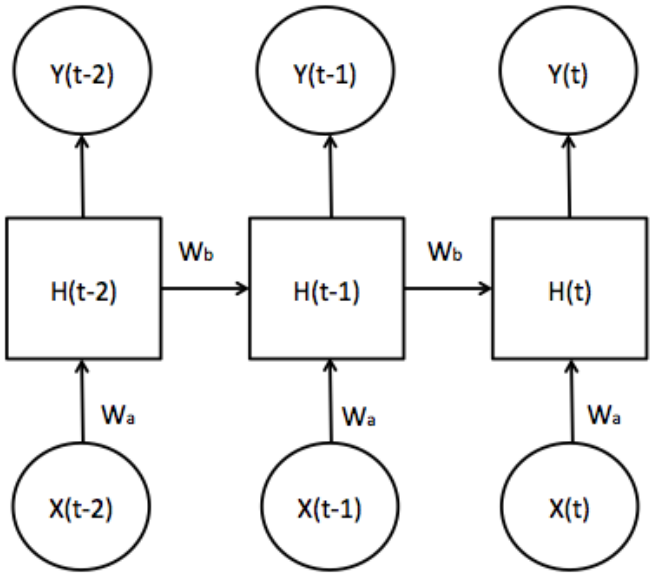

Figure 4: Recurrent neural network

Importantly, although the output at the current time-step only depends explicitly on the current input and the hidden neurons from the previous time-step, those hidden neurons are explicitly dependent on prior inputs. At each step, the hidden neurons can be viewed as a digest of all the previous inputs in the time series. Hence, theoretically, RNNs can model arbitrary length sequences.

The computation at the hidden layer in the illustration will consist of two parts:

$$H(t) = \sigma(X_t^T W_a + H_{t-1}^T W_b) \tag{14}$$

The first part is the same as a FNN whereby the raw input from the current timestep is processed. These inputs are weighted via W_a in the equation. The second computation depends on the previous timesteps hidden neurons. These are weighted via W_b in the equation. Another way to think about the computation is to view it as a singular large weight vector that gets multiplied by the concatenation of the raw input and the hidden neurons from the previous step.

Thus:

$$\begin{aligned} H(t) &= \sigma(Z_t^T W) \\ \text{where} &: \\ Z_t &= X_t \frown H_{t-1} \\ W &= W_a \frown W_b \\ \text{and} \quad \frown &\text{ represents the concatenation of two vectors} \end{aligned}$$

2.3.3 Optimisation

Thus far, the discussion has focused on how a neural network maps from inputs to outputs. However, it has yet to be detailed how the parameters of the model, the weights, are obtained.

Neural networks are primarily trained using gradient based optimisation methods. Optimality is defined by an objective function. Optimisation is the process of trying to find the maximum or mininum of some function by adjusting the arguments of the function. A more comprehensive discussion on objective functions will follow, however, it is sufficient for now to say that it is a function of the model weights and that the function value should, ideally, correlate with the performance of the model on a specific task.

Gradient descent is an optimisation technique that incrementally adjusts the model parameters based on the gradient vector - the vector of the partial derivatives of the objective function with respect to the weights. Depending on the nature of the optimisation, the parameters are adjusted either i) in the direction of or ii) in the opposite direction of the gradient vector.

However, as these models are hierarchical, computing the partial derivatives becomes more complicated. Backpropagation is an algorithm to compute the partial derivatives with respect to individual parameters, within the neural network structure, in an efficient manner and therefore allows models to be

trained faster.

A computational graph can be defined as a series of nodes and operations [10]. Nodes are variables, which may be multidimensional. Operations are functions that take input and produce output in the form of variables. Since neural networks are hierarchical, the effect that changing a weight has on the objective function depends on the operations that are further along on the computational graph. That is, there are a sequence of computations and in order to compute the effect of a change, one must consider all the effects that would occur further along in the sequence as a result of that change. Naturally, the final operation has a direct effect on the objective function. Therefore one can compute the partial derivative of the final operation fairly easily. Using that, it is possible to compute the partial derivatives further back in the computational graph. Informally, that is the backpropagation algorithm. It is an efficient method of recursively applying the chain rule to compute the partial derivatives of parameters within a computational graph.

For a symbolic representation, take a simple computational graph:

$$\begin{aligned} z &= f(y) \\ y &= g(w, x) \\ w &= h(u) \end{aligned}$$

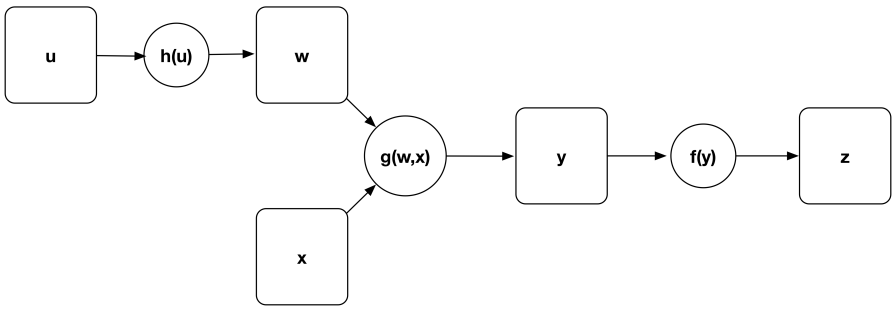

Figure 5: Computational graph

Working from z backwards, the partial derivatives are:

$$
\begin{aligned}
\frac{\partial z}{\partial z} &= 1 \\
\frac{\partial z}{dy} &= \frac{\partial z}{\partial y} \\
\frac{\partial z}{\partial x} &= \frac{\partial y}{\partial x}\frac{\partial z}{\partial y} \\
\frac{\partial z}{\partial w} &= \frac{\partial y}{\partial w}\frac{\partial z}{\partial y} \\
\frac{\partial z}{\partial u} &= \frac{\partial w}{\partial u}\frac{\partial z}{\partial w} = \frac{\partial w}{\partial u}\frac{\partial y}{\partial w}\frac{\partial z}{\partial y}
\end{aligned}
$$

This example is based on scalars however it can be extended easily to vectors and tensors of any dimensionality. In the case of a vector, one would need to compute the Jacobian matrix.

The final equation shows two mathematically equivalent expressions of $\frac{\partial z}{\partial u}$. Computationally, however, one may be more efficient. Backpropogation speeds up the computation of the partial derivatives by limiting the repetition of computing subexpressions [10]. In this example, computing all the partial derivatives seperately would result in $\frac{\partial y}{\partial w}\frac{\partial z}{\partial y}$ being computed twice. However, with more complex computational graphs, such as neural networks, the number of unnecessary computations would significantly slow down the running time and potentially make the training procedure intractable [10]. Backpropogation mitigates the wasted computation by storing the subexpression values of the layer immediately after the layer of the partial derivatives one is trying to compute. In the example, $\frac{\partial z}{\partial w}$ would be stored in memory and would be plugged into the calculation of $\frac{\partial z}{\partial u} = \frac{\partial w}{\partial u}\frac{\partial z}{\partial w}$. Thus, avoiding the need to recompute the expressions $\frac{\partial y}{\partial w}$ and $\frac{\partial z}{\partial y}$.

2.3.4 Backpropogation through time

Optimising the weights of a RNN can be done using gradient descent and the backpropogation algorithm. Using backpropagation on RNNs is known as backpropagation through time (BPTT), but rather than a fundamental change to the algorithm the only difference is the way that the RNN is conceptualised. For an RNN with one hidden layer, each time-step can be thought of as an additional layer in a FNN at a certain point in time [19]. That is, an RNN, which is inherently temporal, can be viewed as a deep, atemporal FNN. Since the weight matrix persists over time, each time-step or 'layer' will have the same parameters. One could update the parameters at the current step using all information from prior time-steps but this would be computationally expensive. In addition the performance gains tend to diminish with increased number of

backpropogation time steps. Therefore, an emerging question is how many time-steps in the past should be considered for updating the weights. It has been suggested that five steps should be sufficient and that information beyond five steps may still be captured due to the nature of RNNs [19].

2.3.5 RNN language modelling performance

The successful application of RNNs to language modelling was first demonstrated in 2012 [19]. RNN language models were shown to be the state of the art by producing the highest accuracy rates on well established datasets. In addition, the investigation showed that RNN language models could improve the performance of larger systems, where the language model is a component, such as speech recognition and machine translation. Importantly, the RNNs used were rudimentary models. They had a single hidden layer and made use of sigmoidal activation functions. The size of the hidden layer, relative to more current models, would be considered small. Despite this, the results demonstrated that fairly basic RNN models could outperform state of the art n-gram models that had been highly optimised.

Since Mikolov's seminal investigation, there has been much progress in neural network research and specifically deep learning [11, 28]. The phrase deep learning refers to neural network models that have several hidden layers. More narrowly, there has been significant progress in the area of RNN language modelling [4, 16]. The reasons for the performance gains can be vaguely summarised as larger networks, increased sophistication of modules within the network and variations in the morphological units being modelled. It is important to note that these developments have happened almost entirely in the domain of large scale language modelling for well resourced languages.

3 Experiments

3.1 Structure

The aim of the experimental section is to empirically ascertain the relevance of RNNLMs for under-resourced South African languages. First, it is important to establish a comparative performance metric. The performance of an n-gram model with smoothing is a suitable benchmark as it had represented the state of the art before the emergence of RNNs.

For each family of models, model selection will be conducted according to a ten-fold cross validation process. Once the model has been selected, a final evaluation will be done on a hold out set so that the performance of the selected model will be a representative metric.

The models will be evaluated on four South African text corpora. The quality of the data and the characteristics of the languages vary but they are of the same size. A more detailed discussion will follow in the data subsection.

This section will first establish a baseline n-gram performance metric before evaluating RNNLMs. Finally, combinations of models will be assessed.

3.2 Cross validation procedure

After shuffling the order of the sentences, ten percent of the dataset is set aside as an evaluation set. This is to be used for the final evaluation once the optimal model has been chosen. The remaining sentences, the development set, is used to conduct model selection.

For each of the four languages, a ten fold cross validation procedure was conducted in order to find the best model for each family of models. The development set is split into ten folds. Each fold forms part of the training set in eight of the ten iterations of the cross validation. A fold will also represent the test set for one iteration and validation set for another.

For every unique hyper-parameter setting, the perplexity was recorded across the ten folds and averaged out to get an estimate of model performance. The standard error of the mean of the perplexity across all ten folds will be used as a measure of the robustness of the perplexity estimate. The standard error is defined below:

$$SE = \frac{\hat{\sigma}}{\sqrt{10}}$$

where:

$$\hat{\sigma} = \sqrt{\frac{\sum_{i=1}^{10}(PPL_i - \bar{PPL})^2}{10}}$$

where:

PPL_i is the perplexity measurement for fold i
$\bar{PPL}$ is the mean of the perplexity measurements across the ten folds.

Finally, once the model has been chosen, the parameters of the model will be learned using the entire development set and the model performance will be evaluated on the evaluation set.

3.3 Dataset

The text data used for the experiments are four text corpora, each consisting of roughly 150 000 sentences. The datasets represent four different official South African languages. Namely, isiZulu, Xitsonga, English and Afrikaans.

The data used for this investigation were proprietary corpora prepared by the Multilingual Speech Technology Group from North West University. Although the entire compiled corpora is not available to the public, some of the original sources are accessible.

The isiZulu [8, 26, 29] and Xitsonga [8, 18] were compiled from multiple sources while the Afrikaans [27] and English [25] datasets were extracted from single sources.

Before conducting any analysis, it is best to briefly characterise the datasets:

Dataset characteristics					
	Sentences	**Unique words**	**Words**	**Words/Sentence**	**Characters**
Xitsonga	150 000	40 460	2 588 344	17.26	17 726 319
IsiZulu	147 233	283 714	2 877 072	19.54	21 730 628
Afrikaans	150 000	68 665	2 885 712	19.24	14 273 325
English	150 000	38 484	3 670 973	24.47	23 323 181

Table 1: Size of the four datasets in several linguistic units

The datasets appear very similar according to the table above with the exception of the isiZulu text. The number of unique words in the isiZulu corpus is noticeably larger than in the other datasets. This is likely due to the agglutinative nature of the language.

It is also noteworthy that although English is considered an adequately resourced language, the English dataset used in these experiments is of the same quality and quantity as the other languages. Therefore the results from the English dataset should not be viewed as a benchmark for an adequately resourced language. Instead it is used to further explore the effectiveness of RNN models on small text corpora.

3.4 N-gram model selection

N-grams serve as a benchmark against which to analyse all subsequent models. When used in conjunction with smoothing techniques, n-grams, prior to RNNLMs, were the best performing models. In addition, n-grams do not require much computational power. These two factors have made n-grams ideally suited for applications that require language modelling. Therefore, an analysis of n-gram performance on the South African datasets is a reasonable first step.

3.4.1 Setup

It is established that n-grams perform best with modified Kneser-Ney smoothing [12, 19]. Thus, different smoothing techniques will not be evaluated. The order of the n-gram is the only parameter that will be evaluated during the cross-validation procedure.

3.4.2 Software: SRILM

SRILM is a statistical language modelling toolkit that is built in C++. It allows one to create and evaluate n-gram language models with a variety of settings and hyper parameter choices. In addition, the toolkit provide functionality for the language models to interface with speech recognition tools and tools for other applications [30]. It is well established as a reliable software tool for n-gram language modelling. For the purposes of the experiments, the toolkit settings of interest are the order of the n-gram and the smoothing technique. SRILM provides sufficient functionality for these settings.

3.4.3 SRILM: settings

The only relevant setting for the experiments, as mentioned within the subsection 3.4.1, is the order of the n-grams. However, SRILM provides several other options to modify the training and testing of language models. Since the objective is to set a credible baseline for comparative purposes, it was decided that the full range of options would not be tested. Instead, based on prior literature, other than the order of the n-grams, the discounting method and

the minimum frequency (which is described in the next paragraph), the default settings were used.

SRILM provides an option whereby a specific n-gram, a sequence of n words, needs to have a minimum frequency if it is to be included in the model. If an n-gram has a lower count than the threshold, then it will be assigned a zero count instead. The default value is two for all orders of n-grams. However, this was changed to one for n-grams of order three or higher. The intuition is that for sparse data, there may only be one instance of certain higher order sequences and thus every instance should inform the model. In addition to smoothing techniques, this should make the models more reliable.

3.4.4 Results

The figure and the table below show the perplexity as a function of the n-gram order.

Across all the languages, bigrams perform substantially better than unigrams. Thereafter, the n-grams performance improves with the order, albeit at a diminishing rate. Although, six-grams produce the lowest cross validation error across all the languages, the improvement over the four-gram is marginal.

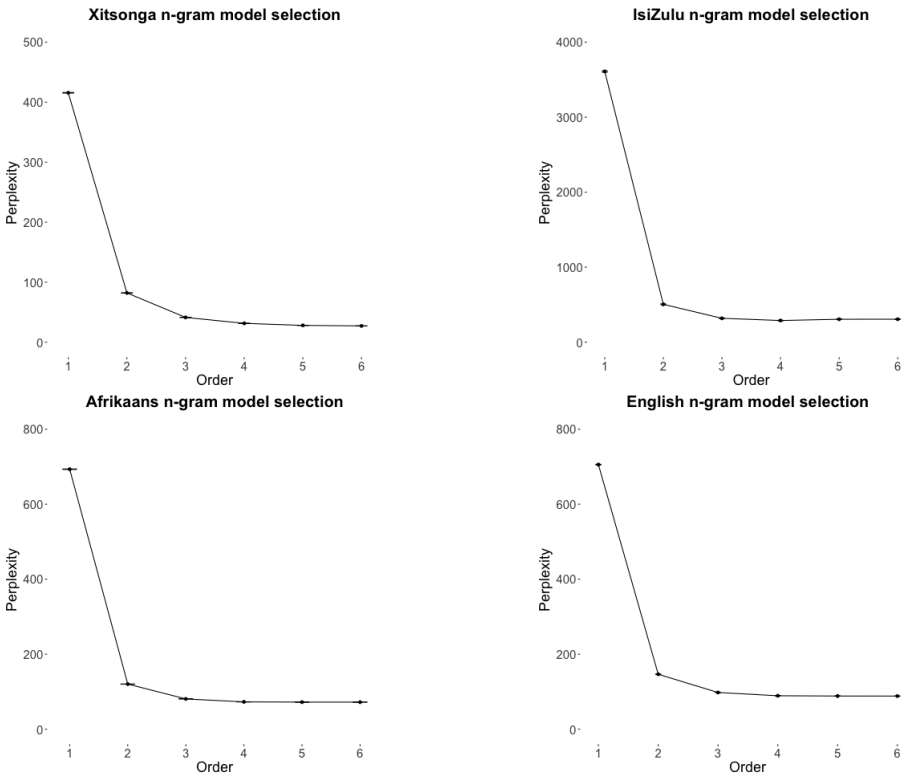

Figure 6: Cross validation of n-gram order for each language.

N-gram results: Perplexity						
	Unigram	**Bigram**	**Trigram**	**Four-gram**	**Five-gram**	**Six-gram**
Xitsonga	455.5 (0.5)	94.4 (0.4)	45.5 (0.3)	34.0 (0.2)	30.2 (0.2)	29.3 (0.3)
IsiZulu	5886.0 (9.4)	794.4(5.8)	500.1 (4.4)	460.9 (4.1)	457.3 (4.3)	457.1 (4.3)
Afrikaans	693.3 (0.7)	120.3 (0.2)	80.8 (0.2)	72.8 (0.2)	72.3 (0.2)	72.1 (0.1)
English	705.6 (0.8)	146.5 (0.3)	97.8 (0.2)	89.2 (0.2)	88.4 (0.2)	88.3 (0.2)

Table 2: n-gram perplexity and sample standard deviation in brackets

3.5 RNN Model Selection

3.5.1 Setup

The RNN model used has a single hidden layer. The reason for this is to set a baseline for RNNLMs using a rudimentary RNN. The primary hyper-parameter of interest is the size of the hidden layer. The size should correlate with greater performance up until some point. In addition, the number of time steps that the backpropagation algorithm will consider for every update, will be varied in order to see how relevant it is to the accuracy of the model. The learning rate will not be cross validated as the toolkit introduced below incorporates a reflective process whereby the learning rate is adjusted based on results from the validation set.

The size of the hidden layer varies between five neurons and three hundred neurons. The aim is to see how the size and complexity of the language model, the number of neurons in the hidden layer specifically, correlates with the mean perplexity. It will be important to analyse the rate at which the performance of the model improves with respect to changes in the size of the hidden layer. This will indicate how much extra performance is gained at the expense of the additional computational resources needed for the training and storing of the model.

3.5.2 Software: RNNLM

RNNLM is an abbreviation for the recurrent neural network language modelling toolkit. It was made available as an open source tool for researchers and practitioners to use in order to improve and speed up their language modelling workflow [20]. RNNLM allows users to build simple, single layered recurrent neural network language models. The models can be trained, tested and sampled from using the toolkit. In addition, RNNLM provides functionality that allows the language model to interface with software for specific applications such as speech recognition. In Tomas Mikolov's seminal dissertation, RNNLM was the tool used in the experiments [19]. Therefore, it seems suitable to use the RNNLM toolkit for the first attempt at modelling a specific language using RNNs. Any potential extensions of this investigation can proceed fairly straightforwardly by using more sophisticated RNNs.

3.5.3 RNNLM: settings

RNNLM allows one to specify several hyperparameters. The most rudimentary settings are the size of the hidden layer and the number of steps that the errors will be propagated.

One interesting option is the number of classes. Classes are used to lessen the computational requirements of computing the output distribution. Since the output distribution is over the entire vocabulary, this can be an expensive computation. Classes aim to deal with this by clustering words into classes that form subsets of the vocabulary. Then, instead of computing a distribution over the vocabulary, the model estimates the distribution on the words within a class that is most relevant. More details on class based models can be found in [12, 19]. The RNNLM default value of 100 classes was used for all the models.

The learning rate, i.e. the coefficient that is used to adjust the parameters, has a default value of 0.1. After each iteration, the performance on the validation data is used as a guide as to whether the learning rate needs to be adjusted [19]. It was decided that this default mechanism would be used as it seems reasonable and has produced credible results in prior work [19].

3.5.4 Results I

The change in performance across different numbers of backpropagation steps were negligible. There was less than a 1% difference in perplexity between the models that were identical other than the number of backpropogation steps. Therefore the results of the entire parameter search will not be presented, however, the results from the changes in hidden layer size were insightful and are presented in the figures below.

In order to more clearly view the effect of changes in the size of the hidden layer, the results are split into two sections where the first focuses on a hidden layer that has between five and thirty neurons, and the second section views the experiment in its entirety. This allows one to view the smaller models with greater granularity that would not be possible otherwise.

These results in the figures below show that relatively small changes in the model size have significant effects on the perplexity of the language model. As expected, there are diminishing returns to increases in the size of the model, however, there is clearly still an indication that bigger models should provide further accuracy gains.

Importantly, the analysis of smaller models provides some insight as to the degree of parsimony that is possible for a certain level of performance. That is, if the goal is not to maximise accuracy unconditionally, but instead is to

obtain a certain level of accuracy with the fewest resources, this analysis could be useful. Certain applications where the language model would have to be trained on a mobile phone or with other limited compute power could find the nature of this analysis useful.

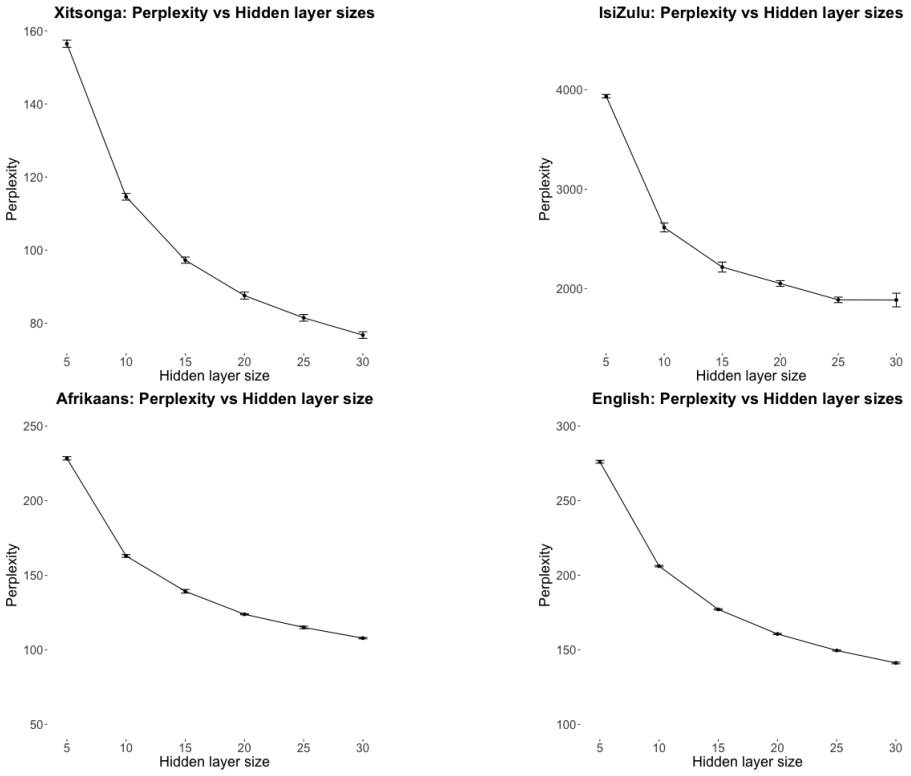

Figure 7: RNNLM: hidden layer size 5-30

RNN model selection: I						
Hidden layer size:	**5**	**10**	**15**	**20**	**25**	**30**
Xitsonga	156.5 (1.0)	114.6 (0.9)	97.3 (0.9)	87.6 (1.0)	81.5 (0.9)	76.8 (0.9)
IsiZulu	3935.6 (17.5)	2615.6 (44.8)	2217.1 (50.2)	2051.8 (28.3)	1887.4 (27.4)	1885.7 (68.56)
Afrikaans	228.4 (1.1)	162.9 (0.9)	139.2 (1.2)	123.8 (0.4)	115.0 (0.9)	107.8 (0.5)
English	276 (0.91)	206 (0.47)	177 (0.44)	161 (0.40)	150 (0.41)	141 (0.60)

Table 3: Perplexity estimates and sample standard deviation in brackets for RNN models with 5-30 hidden layer neurons. All models in the table were learned using backpropogation through time with five timesteps.

3.5.5 Results II

The graphs below illustrate the perplexity as a function of hidden layer size for the entirety of the experiments. That is, across all four languages, the size of the hidden layer was varied between 5 and 300 neurons. Whereas the table shows this same relationship but only for layer sizes of 50 to 300.

Although most of the models show increases in performance as they increase in size, the rate of improvement clearly decreases as the models get bigger. It is evident that the returns in accuracy have somewhat plateaued at three-hundred neurons and therefore there does not seem to be good reason to increase the size of the models any further. The best performing model within this size range will serve as a representative figure for RNNLMs for the investigation.

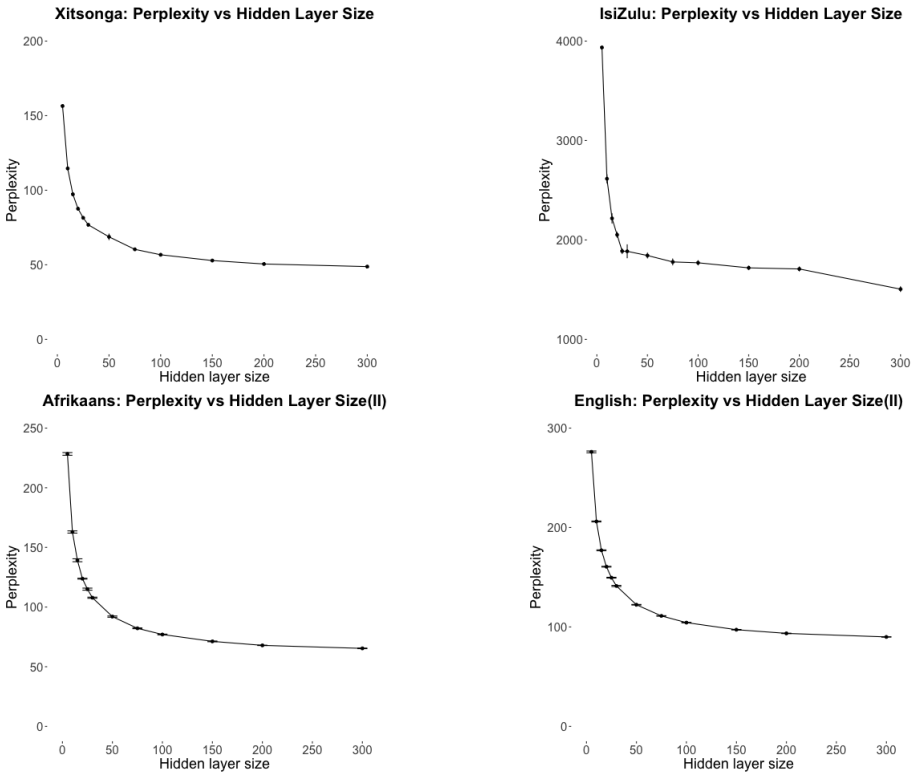

Figure 8: RNNLM: hidden layer size 50-300

RNN model selection: II						
Hidden layer size:	**50**	**75**	**100**	**150**	**200**	**300**
Xitsonga	68.8 (2.1)	60.4 (0.8)	56.7 (0.8)	52.8 (0.8)	50.5 (0.7)	48.8 (0.7)
IsiZulu	1843.4 (28.1)	1778.9 (32.7)	1770.4 (24.4)	1719.3 (21.2)	1708.1(23.4)	1505.3(26.8)
Afrikaans	92.0 (0.5)	82.2 (0.4)	77.0 (0.3)	71.2 (0.3)	67.9 (0.2)	65.4 (0.3)
English	122.3 (0.3)	111.1 (0.3)	104.4 (0.3)	97.2 (0.3)	93.5 (0.2)	89.9 (0.2)

Table 4: Perplexity estimates and sample standard deviation in brackets for RNN models with 50-300 hidden layer neurons. All models in the table were learned using backpropogation through time with five timesteps.

3.6 Preprocessing: punctuation

3.6.1 Introduction and setup

Preprocessing of the text data is usually necessary before any language models can be built. One important aspect of preprocessing is the removal of punctuation marks. An interesting question is what, the magnitude of the effect that punctuation has on the language model accuracy is.

In the previous sections, all results were obtained using datasets that had punctuation removed. For this investigation, the initial isiZulu and Xitsonga datasets had punctuation marks throughout. This section aims to quantify the extent to which punctuation influences language modelling. An experiment was conducted using the unprocessed isiZulu and Xitsonga datasets. RNNLMs of varying sizes were trained on the punctuated datasets and were compared to results on the unpunctuated datasets.

The processing is not a universal removal of all punctuation marks. For example, hyphens used for compound words and apostrophes used for contractions are not removed. These punctuation marks are components of words that are considered as unique tokens. All punctuation marks that did not form part of a token were removed from the text data.

The structure of the experiments is similar to the cross-validation experiments in section 3.5. All RNNLMs were trained using backpropagation through time going back five time steps. As in section 3.5, the smallest model had five neurons and the largest model had three-hundred.

3.6.2 Results

Both Xitsonga and isiZulu show an increase in perplexity once the punctuation is removed. The effect is much larger in the isiZulu dataset where the perplexity more than doubles across models of all sizes. One potential explanation for this is that much of what the language model had learnt about the unprocessed dataset was the placement of punctuation marks. This constituted a larger proportion of the underlying pattern, that the model had learnt, in the isiZulu dataset than the Xitsonga dataset.

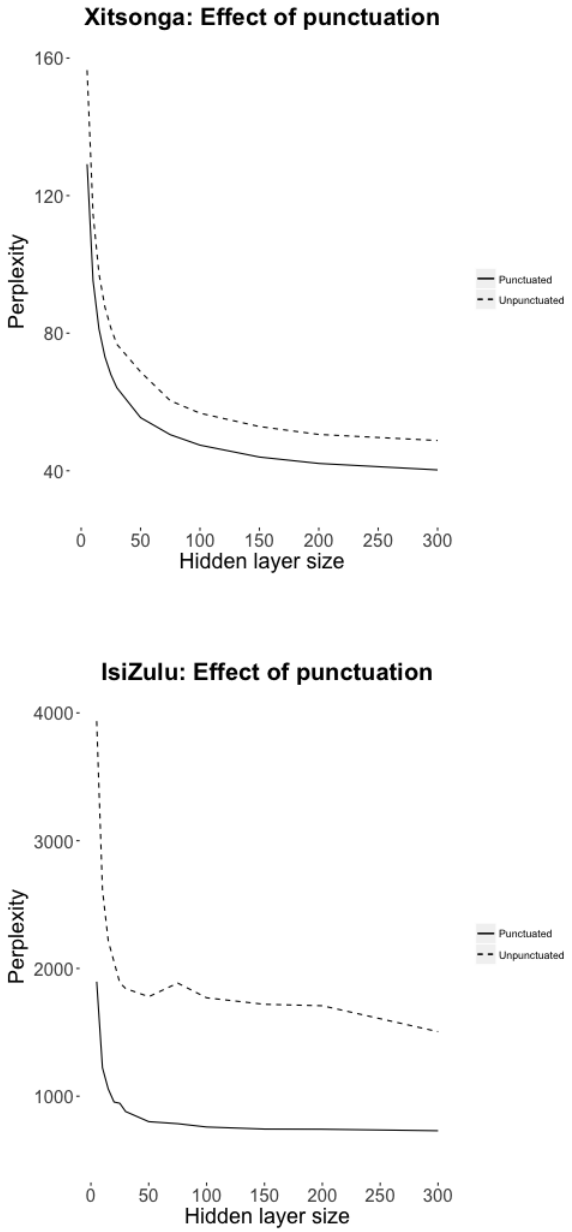

Figure 9: RNN language model performance on punctuated and unpunctuated text corpora.

RNN perplexity: punctuation						
Hidden layer size:	**50**	**75**	**100**	**150**	**200**	**300**
Xitsonga punctuation	55.4	50.5	47.4	43.9	42.1	40.2
Xitsonga no punctuation	68.8	60.4	56.7	52.8	50.5	48.8
Xitsonga mean % change	24%	20%	20%	20%	20%	21%
IsiZulu punctuation	800.9	785.1	759.1	742.9	741.7	729.8
IsiZulu no punctuation	1885.7	1778.9	1770.4	1719.3	1708.1	1505.3
IsiZulu mean % change	135%	126%	133%	132%	130%	106%

Table 5: A comparison of RNNLM performance on punctuated and unpunctuated datasets.

3.7 Linear interpolation

3.7.1 Introduction

This section will explore the performance of models that are a combination of RNNLMs and n-grams. More specifically, the models will be combined using linear interpolation.

Linear interpolation is a simple method that derives an estimate of the probability of a sequence through a weighted sum of the estimates of the individual models. This method is rudimentary but still provides results comparable with more sophisticated methods like log-linear interpolation [19].

In the context of a linear combination between an n-gram and a RNNLM, this can be explicitly expressed as:

$$P_{interpolation}(W) = \lambda_1 P_{RNNLM}(W) + \lambda_2 P_{N-gram}(W) \tag{15}$$

where:
W is a sequence of words
λ_1 and λ_2 are weights.

For the experiments in the investigation, the sum of the weights added up to one. That is,

$\lambda_2 = 1 - \lambda_1$ and
$0 \leq \lambda_1 \leq 1$

It is expected that some interpolated model should outperform any individual model. This was demonstrated in Mikolov's investigation [19].

3.7.2 Setup

For each of the four languages, a set of linear interpolated combination models were evaluated. For each language various weight combinations between models

were tested. For a given weight combination, the combination model was evaluated on a single fold of each development set. The primary motivation for not testing the interpolated models across all ten folds is for computational parsimony. However, a single fold should still provide some insight into what the ideal weighting would be for the best performing model. The weighting of the RNNLM was varied between 0.1 and 0.9 in increments in 0.1.

3.7.3 Results

The perplexity of the combination models are shown in the graphs below:

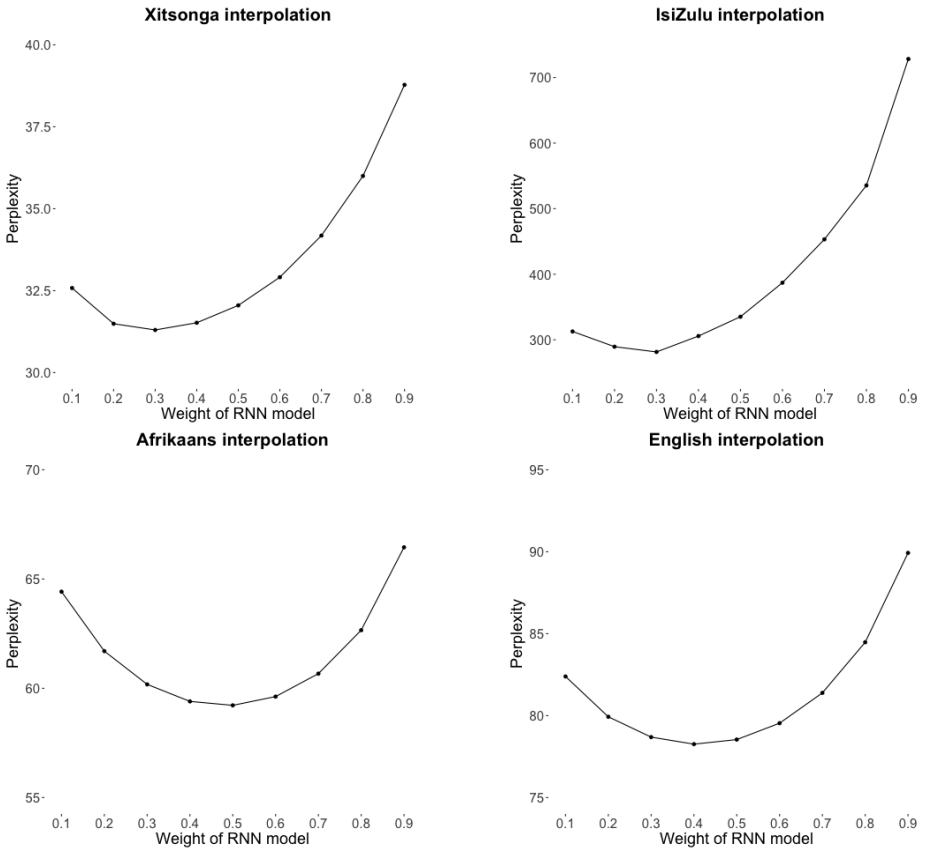

Figure 10: Perplexity scores as a function of the weight of the RNNLM in a linearly interpolated model.

The table below compares the results from the best performing interpolated model for each language with the results from the lowest perplexity RNN and n-gram model. These results are derived based on experiments conducted on the development set. The results for single n-gram and single RNN models differ from previous sections as the models were retrained. In the next section (3.9) a similar comparison will be done using the test set as the basis for perplexity.

Model type:	**Single n-gram**	**Single RNN**	**Linearly interpolated model**
Xitsonga	30.5	48.8	29.6
IsiZulu	457.1	1503.3	283.7
Afrikaans	69.0	65.4	54.4
English	86.3	89.9	76.3

Table 6: Perplexity values for: i) a 5-gram, ii) a RNN with a hidden layer of 300 neurons and iii) linear interpolated models. The weight of the interpolated model is based on a cross validation analysis. All RNN models in the table were learned using backpropagation through time with five time-steps.

3.7.4 Interpretation

In accordance with the literature and expectations, some interpolated mixture between an n-gram and an RNNLM produced a lower perplexity score than any individual model. For Afrikaans and English the most accurate interpolated model consisted of a roughly equal weighting between the n-gram and RNNLM. The best performing Afrikaans model had an equal weighting between the models while the English interpolated model assigned the RNNLM slightly less at 0.4. The most accurate Xitsonga and isiZulu interpolated models had a higher weighting on the n-gram model. The RNNLM had a weight, in the lowest perplexity interpolated model, of 0.3 for both isiZulu and Xitsonga.

The isiZulu experiment is a very good demonstration of the merits of interpolation. In isolation the n-gram model far outperforms the RNNLM. The extent of the disparity between the models might suggest that any interpolated model would perform worse than the n-gram in isolation because the RNNLM is far worse. However, even in this case, a linear interpolated model outperformed the n-gram. The superiority of the n-gram model was reflected in the best performing model assigning a weight of 0.7 to the n-gram model.

In summary, linear interpolation provided additional accuracy to the individual models by ostensibly combining complimentary aspects of the RNNLMs and n-grams. These results are consistent with the literature [19]. In the final evaluation section below, a single interpolated model for each language will be appraised and compared to the best performing individual models. The weights for the interpolated models will be informed by the results from this section.

3.8 The effect of dataset size

3.8.1 Introduction and setup

The principal question in this dissertation is to investigate how different language models perform in an under-resourced environment. One component of this is the size of the datasets. Thus far, the investigation has been done on datasets of approximately 150 000 sentences each.

This section aims to further investigate the extent to which the size of the dataset effects the perplexity of the model. This should provide insight regarding what constitutes a satisfactorily sized dataset. It will also aim to provide a sense of how quickly a language model improves with respect to the amount of data it is built with.

Shadowing the previous sections, the analysis will be done on all four languages. For each language, a RNNLM will be trained using a subset of the training data defined in section 3.2. Three different sized subsets - 25%, 50% and 75% of the original training data - will be used to build RNNLMs which will then be tested on the original sized test set data. The performance of these models will be compared to the models in section 3.5.

3.8.2 Results

The results show a fairly dramatic increase in the perplexity of the models as the dataset size is reduced. Using a dataset that is 75% of the original dataset has a significant effect across all four languages. The reduction in model performance continues, in a roughly linear relationship, as the dataset size decreases.

This could indicate that the quality of a language model is very sensitive, within this proximity, to small changes in the amount of data. Therefore gathering additional data, even if it is resource intensive to do so, should provide satisfactory performance gains.

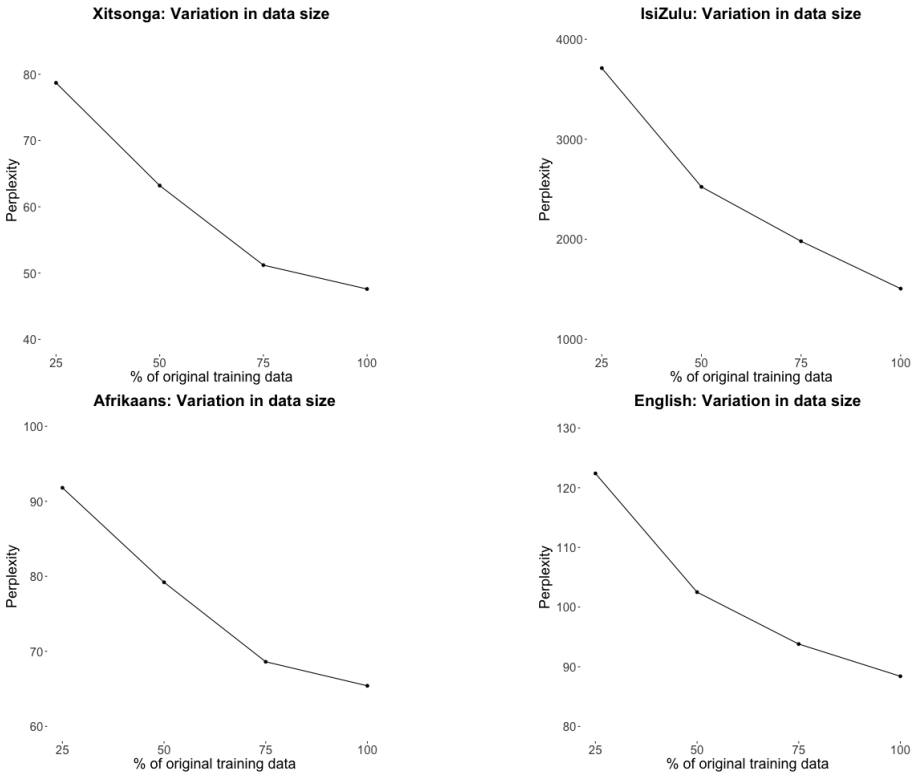

Figure 11: Performance of RNNLMs with reduced training data

3.9 Final evaluations

This section will provide a summary of the performance differences for the models that were chosen using cross-validation. The models will be tested on the evaluation set – the ten percent of data that was set aside before model selection. The rest of the the data was used to build the models. In addition, composite models that are linear interpolations between RNN and n-gram models, are tested and reported on. Linear interpolations between two different models usually provide performance gains over each of the individual models as seen in the literature and in section 3.7.

Model type:	**Single n-gram**	**Single RNN**	**Linearly interpolated model**
Xitsonga	30.5	46.9	28.8
IsiZulu	451.9	1506.3	283.6
Afrikaans	69.0	62.7	54.1
English	86.2	88.4	76.5

Table 7: Perplexity values for: i) an order 5 n-gram, ii) a RNN with a hidden layer of 300 neurons iii) linear interpolated models. The weight between the RNNLM and the n-gram is language specific and was chosen based on the best performing model in section 3.7. All RNN models in the table were learned using backpropagation through time with five time-steps.

3.9.1 Interpretation

The final evaluation reveals the n-gram models perform better across all the languages except Afrikaans. This is consistent with the literature relating to simple single layered RNNLMs where the performance of n-grams and RNNLMs performed similarly [19].

Particularly noteworthy is the degree to which the n-gram model outperforms the RNN model on the isiZulu dataset. In conjunction with the fact that the isiZulu dataset has, by a considerable margin, the most unique tokens–this may be indicative of an underlying cause of performance differences between RNN and n-gram models in this setting. One interpretation of these results is that RNN models require much more data than n-grams to fit complex functions. The isiZulu dataset is more diverse than the other three languages in the investigation and the language itself is agglutinative, which implies that each word contains more information than in non-agglutinative languages. As a result, even though the amount of sentences and words are approximately the same across all the datasets, the isiZulu dataset could be seen as having less instances of linguistic patterns from which to learn a representative language model. Given that the n-gram performed considerably better in this context, it may be indicative that there is a minimum threshold of data required for RNNLMs to become effective. It could also be suggestive of the effectiveness of the state of the art smoothing techniques used in the n-gram model.

Despite the performance of the RNNLMs in isolation, RNNs seem to be complimentary to n-gram models based on the performance of the interpolated models. Across all four languages, models that formed some linearly weighted combination of the RNN and n-gram models performed significantly better than any isolated model. Again, this is consistent with the literature [19]. This complementarity may suggest that RNN and n-gram models encode different properties of text. Perhaps n-gram models learn common sequences better while RNNs are better able to store longer term dependencies and structure.

Lastly, the perplexity of the final evaluations for both the n-gram, RNNLMs and interpolated models are within a reasonable range of the estimates computed in prior sections. This confirms that the estimates were reliable.

4 Conclusion

4.1 Summary

The purpose of the investigation was to conduct a rudimentary analysis of the effectiveness of RNNLMs within the context of South African under-resourced languages. To do this, a research toolkit that allows for the training and testing of simple, single-layered neural networks was used and was benchmarked against state of the art n-gram language models.

Consistent with literature, n-grams performed better than basic RNNLMs [19]. However, the combination of the two models provided significant improvements over the individual models. The fact that combination models, constructed using linear interpolation, provided noteable improvements over, previously state of the art, n-grams is an important finding.

Since there is no literature on prior attempts to build RNNLMs for South African languages, the results of the investigation imply that all previously built language models could potentially be improved using linear interpolation with RNNLMs. This could be significant for other NLP applications such as speech recognition and machine translation.

In addition to the primary research question, the investigation uncovered some information relating to the importance of preprocessing text datasets. An experiment relating to punctuation revealed that a significant amount of the performance on an unprocessed dataset was the model learning the punctuation patterns. This serves as evidence of the necessity of preprocessing.

4.2 Recommendations

Although the investigation revealed some information regarding RNNLMs, there is still much left to be analysed. Firstly, there are a host of modifications that can be made to RNNLMs to improve performance. These include changes to the activation function, depth of the network, optimisation techniques and modern network modules such as LSTMs. These changes should provide additional performance improvements [4, 16].

Secondly, this investigation focused solely on intrinsic valuation of language models. A clear follow on question is whether the performance gains, presented here, carry over into applications like speech recognition and machine translation.

References

[1] Nimaan Abdillahi, Nocera Pascal, and Bonastre Jean-François. "Automatic transcription of Somali language". In: *Proceedings of the Annual Conference of the International Speech Communication Association, Interspeech* (2006), pp. 289–292.

[2] Paul H Algoet and Thomas M Cover. "A sandwich proof of the Shannon-McMillan-Breiman theorem". In: *The Annals of Probability* 16.2 (1988), pp. 899–909.

[3] Laurent Besacier, Etienne Barnard, Alexey Karpov, and Tanja Schultz. "Automatic speech recognition for under-resourced languages: A survey". In: *Speech Communication* 56 (2014), pp. 85–100.

[4] Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, and Phillipp Koehn. "One Billion Word Benchmark for Measuring Progress in Statistical Language Modeling". In: *Proceedings of the Annual Conference of the International Speech Communication Association, Interspeech* (2013).

[5] Stanley F Chen and Joshua Goodman. "An Empirical Study of Smoothing Techniques for Language Modeling". In: *Computer Speech & Language* 13 (1999), pp. 359–394.

[6] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (2014), pp. 1724–1734.

[7] George E Dahl, Dong Yu, Li Deng, and Alex Acero. "Context-Dependent Pre-Trained Deep Neural Networks for Large-Vocabulary Speech Recognition". In: *IEEE Transactions on Audio, Speech, and Language Processing* 20 (2012), pp. 30–42.

[8] Roald Eiselen and Martin J. Puttkammer. "Developing Text Resources for Ten South African Languages". In: *LREC*. 2014, pp. 3698–3703.

[9] William A Gale and Kenneth W Church. "What's wrong with adding one". In: *Corpus-Based Research into Language. Rodolpi*. 1994.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[11] Ian Goodfellow, Yoshua Bengio, Aaron C. Courville, and Geoffrey E. Hinton. "Deep Learning". In: *Nature* (2015), pp. 436–444.

[12] Joshua T Goodman. "A bit of progress in language modeling". In: *Computer Speech & Language* 15 (2001), pp. 403–434.

[13] Alex Graves. *Supervised Sequence Labelling with Recurrent Neural Networks*. Jan. 2012.

[14] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. "Speech recognition with deep recurrent neural networks". In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing* (2013), pp. 6645–6649.

[15] Xuedong Huang, Alex Acero, and Hsiao-Wuen Hon. *Spoken Language Processing: A guide to theory, algorithm and system development*. Prentice Hall, 2001.

[16] Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. "Exploring the Limits of Language Modeling". In: *CoRR* abs/1602.02410 (2016).

[17] Daniel Jurafsky and James H. Martin. "Chapter 4 N Grams". In: *Speech and Language Processing*. 3rd. 2015. Chap. 4.

[18] Cindy McKellar. "An English-Xitsonga SMT system for the government domain". In: *PRASA*. 2014, pp. 229–234.

[19] Tomáš Mikolov. "Statistical language models based on neural networks". PhD thesis. Brno University Of Technology, 2012.

[20] Tomáš Mikolov, Stefan Kombrink, Anoop Deoras, Lukáš Burget, and Jan Černocký. "RNNLM — Recurrent Neural Network Language Modeling Toolkit". In: *Proceedings of ASRU 2011* (2011), pp. 1–4.

[21] Hermann Ney, Ute Essen, and Reinhard Kneser. "On Structuring Probabilistic Dependences in Stochastic Language Modelling". In: *Computer Speech & Language* 1 (1994), pp. 1–38.

[22] Daniel R van Niekerk. "Exploring unsupervised word segmentation for machine translation in the South African context". In: *Proceedings of the 2014 PRASA, RobMech and AfLaT International Joint Symposium* (2014), pp. 202–206.

[23] Thomas Pellegrini and Lori Lamel. “Are audio or textual training data more important for ASR in less-represented languages?” In: *SLTU*. 2008, pp. 2–6.

[24] Thomas Pellegrini and Lori Lamel. “Investigating automatic decomposition for ASR in less represented languages”. In: *Interspeech*. 2006.

[25] Parliament of the Republic of South Africa. 2017. URL: `https://www.parliament.gov.za/` (visited on 12/05/2017).

[26] Justus Roux, Philippa Louw, and Thomas Niesler. “The African Speech Technology project: An assessment”. In: *LREC* (2004), pp. 93–96.

[27] RSG. 2017. URL: `http://www.rsg.co.za/` (visited on 12/05/2017).

[28] Jürgen Schmidhuber. “Deep Learning in Neural Networks: An Overview”. In: *Neural Networks* 61 (2015), pp. 85–117.

[29] Dirk Snyman, Gerhard B. Van Huyssteen, and Walter Daelemans. “Cross-Lingual Genre Classification for Closely Related Languages”. In: 2012, pp. 132–137.

[30] Andreas Stolcke. “SRILM - an extensible language modeling toolkit”. In: *Interspeech*. 2002, pp. 901–904.

[31] Martha Yifiru Tachbelie, Solomon Teferra Abate, and Laurent Besacier. “Using different acoustic, lexical and language modeling units for ASR of an under-resourced language– Amharic”. In: *Speech Communication* 56 (Jan. 2014), pp. 181–194.

[32] Alan Turing. “Computing machinery and intelligence”. In: *Mind* 49 (1950), pp. 433–460.