Low Resource Language Modelling Literature Review

Stuart Mesham
stuart.mesham@gmail.com
University of Cape Town
Cape Town, South Africa

ABSTRACT

Recent state-of-the-art results in language modelling have been achieved using large models trained on billions of words. South African languages typically have a limited amount of data available. We aim to determine whether the same techniques that have led to improvements in English also improve performance in other languages by comparing the performance of different model architectures and training strategies on South African languages. In this review we describe different language models ranging from traditional models to the most recent. We include an overview of language modelling concepts as well as approaches to modelling low-resource languages.

CCS CONCEPTS

• **Computing methodologies** → **Natural language processing**;

KEYWORDS

natural language processing, language modelling, low-resource language modelling, n-gram, long short-term memory, transformer

1 INTRODUCTION

Language modelling has valuable applications such as machine translation, voice recognition, information retrieval, spelling correction and question answering [12, 20, 37, 49, 68]. However, modern language models typically require large amounts of data to train. Low-resource languages therefore may see reduced benefit from recent advances in language modelling. Many of South Africa's widely spoken languages have limited data available for language modelling; making research into low-resource language modelling relevant to the country. We focus on three types of language models: n-grams, Long Short-Term Memory (LSTM) models and transformer models.

N-grams are a traditional class of language models which achieve good performance in a variety of contexts despite their relative simplicity. N-grams serve as a useful baseline against which to compare more complex models.

LSTM language models have pushed state-of-the-art performance in language modelling over the past decade. They are complex neural networks with high computational requirements for training.

Transformer models have recently emerged as an alternative to LSTMs which allow faster training times due to increased parallelisation of computation. Models such as GPT-2 have also produced state-of-the-art results [57].

We aim to evaluate these three types of models on South African languages to determine whether recent advancements improve performance in a low-resource setting.

2 LANGUAGE MODELLING THEORY

A language model assigns a probability to a sequence of words. Given the start of a sentence, the model tries to predict the next word, assigning a probability to each of the words that could potentially follow [17].

2.1 Language Model Definition

Formally, given a sequence of n words, $W_1^n = w_1, ..., w_n$ a language model m assigns a probability $P(W_1^n)$ to the sequence. Typically, this is achieved using the chain rule of probability as follows:

$$\begin{aligned} P(W_1^n) &= P(w_1)P(w_2|w_1)P(w_3|w_1, w_2)...P(w_n|w_1, w_2, ..., w_{n-1}) \\ &= \prod_{k=1}^{n} P(w_k|W_1^{k-1}) \end{aligned} \tag{1}$$

Here the model is assigning a probability to each word given the previous words in the sentence. The probability of the sequence is given by the product of the probabilities of each word.

2.2 Model Evaluation

The quality of a language model can be evaluated either extrinsically or intrinsically. Extrinsic evaluation measures a model's usefulness in some downstream task such as speech recognition or machine translation whereas intrinsic evaluation uses statistical measures to assess a the model's quality. For the purposes of our research, we will assess language models in isolation. Thus only intrinsic evaluation measures will be considered. The intrinsic measure most commonly used to evaluate language models is perplexity. Below, we give brief explanations of the concepts of entropy, cross-entropy and their relation to perplexity.

2.2.1 Entropy. In the field of information theory, entropy is a basic measure of information. The entropy, H, of a random variable X with sample space S and a probability distribution function p is given by the equation:

$$H(X) = -\sum_{x \in S} p(x) \log_2 p(x) \tag{2}$$

The entropy describes the average amount of information produced per event observed. Since the log in the above equation is in base 2, the amount of information is measured in bits. The equation therefore describes the average number of bits of information produced per observation of X [17, 61].

2.2.2 Cross-Entropy. To determine the entropy of a language, we would need to know it's true probability distribution which in practice is not known. We only have a language model which approximates the distribution. In this scenario, the cross-entropy of the model is used. The cross-entropy of one probability distribution on another is a measure of the level of divergence of the two

distributions [23, 36]. Given a sample of text, the cross-entropy of a model is estimated as follows:

$$H(W_1^n) = -\frac{1}{n}\log P(W_1^n) \tag{3}$$

[17]

The more accurately the model approximates the true distribution of the language, the lower the cross-entropy. It is for this reason that perplexity, the most commonly used model evaluation metric, is derived from cross-entropy [17].

2.2.3 Perplexity. When comparing two language models, to determine which model is of a higher quality we compare the perplexity of each model on a sequence of words. Perplexity is defined as follows:

$$\text{Perplexity}(W_1^n) = 2^{H(W_1^n)} \tag{4}$$

The better model will have a lower perplexity [17].

Perplexity can also be interpreted as the weighted average branching factor, describing the average number of possible words which can follow any given word, weighted by the probability the model assigns each next word [17].

3 N-GRAM LANGUAGE MODELS

Historically, one of the most prominent types of language model has been the n-gram. To assign a probability $P(w_n|w_1, w_2, ..., w_{n-1})$ that the word w_n is the next word in the sequence $w_1, w_2, ..., w_{n-1}$, n-grams simplify the task using the Markov assumption [40] that this probability can be approximated by a maximum likelihood estimation using only the $N-1$ previous words as follows:

$$P(w_n|w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1}w_n)}{C(w_{n-N+1}^{n-1})} \tag{5}$$

where C is the count of the occurrences of a sequence of words in the training corpus [17, 61]. For example, a trigram ($N = 3$) language model assigns the probability $P(w_n|w_{n-1}, w_{n-2})$ by counting the frequency with which these three words appear consecutively in a training corpus. Consider the word sequence "the cat sat on the". To assign a probability that the next word is "mat" the model counts the frequency of the trigram "on the mat" in the training corpus relative to the frequency of all trigrams starting with "on the". In this example, "the cat sat on the" is termed the **context** and "mat" is termed the **target**.

3.1 Zero-Probability N-Grams

Some n-grams may never occur in the training set due to practical limitations on its size. This means that the model will assign some n-grams a probability of zero. If such an n-gram appears in the test set, the entire test set will be given a probability of zero. This is an issue since the perplexity is then undefined due to its inverse relation to the probability. Below we discuss techniques called backoff, interpolation and smoothing which address this issue. In the context of low resource languages, it is expected that small training sets will result in many zero-probability n-grams. It is therefore important that these techniques are explored.

3.2 Backoff and Interpolation

Consider the trigram model evaluating the probability of the word "mat" in the earlier example. Suppose that the trigram "sat on the mat" does not occur in the training data. We will need to make use of lower order n-grams to estimate this probability.

3.2.1 Backoff. Continuing with this example, its probability can be estimated using a bigram model since this may not have a zero count. Should the bigram also have a zero-count, one could then look to the monogram to estimate this probability. This strategy is called backoff. The probabilities of higher order n-grams are estimated by "backing off" to lower order n-grams. Additionally, when backing off, probability mass must be redistributed from higher order n-grams to lower order n-grams. This algorithm is known as Katz backoff [33]. Another algorithm known as Good-Turing backoff combines Katz backoff with a smoothing algorithm called Good-Turing smoothing [13].

3.2.2 Interpolation. One way to utilise multiple-order n-gram models simultaneously is to perform a fixed linear interpolation of their results. In the example above, the probability of the target "mat" could be calculated using:

$$\begin{aligned} P(\text{"mat"}|\text{"the cat sat on the"}) =& \lambda_3 P(\text{"mat"}|\text{"on the"}) \\ &+ \lambda_2 P(\text{"mat"}|\text{"the"}) \\ &+ \lambda_1 P(\text{"mat"}) \end{aligned} \tag{6}$$

where $\sum_i \lambda_i = 1$ so that equation 6 gives a well formed probability distribution. The λ parameters can be conditioned on the context, for a more sophisticated interpolation [17].

3.3 Smoothing

Smoothing is used to shift probability mass from n-grams with high probabilities to those with lower probabilities. This prevents zero-probability n-grams and in practice typically improves the performance of the model.

3.3.1 Laplace Smoothing. Adding one to every frequency count such that equation 5 becomes:

$$P(w_n|w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1}w_n) + 1}{C(w_{n-N+1}^{n-1}) + V} \tag{7}$$

prevents zero-probability n-grams since the numerator cannot be zero. This is known as laplace or add-one smoothing [30]. In this equation, V is the vocabulary size.

3.3.2 Add-k Smoothing. Similarly, adding k to each frequency count can be used to adjust the level of smoothing.

$$P(w_n|w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1}w_n) + k}{C(w_{n-N+1}^{n-1}) + kV} \tag{8}$$

Increasing k increases the level of smoothing, shifting more probability mass away from higher probability n-grams and towards lower probability n-grams [31].

3.3.3 Kneser-Ney Smoothing. A more sophisticated smoothing algorithm called Kneser-Ney smoothing [34] was modified to the current most commonly used and typically best performing smoothing algorithm: modified Kneser-Ney smoothing [13].

4 SUB-WORD TOKENIZATION

So far we have considered language modelling at the word level; with words being the smallest unit each language model has taken as input and provided as output. However, there are many ways in which text can be divided, such as into whole phrases, words, small groups of characters or individual characters. In more general terms, we call these atomic units tokens. In many cases it is beneficial to break words into sub-word tokens and model the sequence of these tokens instead of whole words. This process is particularly well suited to languages in which words are commonly formed by agglutination or compounding [60]. This makes sub-word tokenization particularly relevant to our research, since South African languages typically make heavy use of agglutination.

4.1 Character-Level Modelling

One of the most straightforward approaches to breaking words down into smaller tokens is to break them down into individual letters and have the language model predict text character by character. This is difficult for many reasons. For example, the language model will now need to learn to recognise and generate words. Another reason being that the number of tokens is high, increasing the amount of computation required to train and evaluate models and requiring the models to learn very long-range dependencies [4].

4.2 WordPiece

First developed for the task of segmenting text in Asian languages, one approach is termed the WordPiece model [59]. Initially, one token is used to represent each character. A language model is then trained using this tokenization and it's performance is measured. Two tokens are then combined; the combination that gives the greatest improvement in measured performance is selected. This step is done by retraining and evaluating the language model for each potential combination. This process is repeated until either a desired vocabulary size is reached or until the improvement in model performance after each combination event is small. The algorithm is $O(n^2)$ with respect to the vocabulary size at each iteration, with n^2 language models being trained. Various optimisations can be applied such as only evaluating token combinations which appear in the training data.

It is worth noting that the language models used in the original implementation of this algorithm were n-grams. This algorithm may be impractically slow if complex models such as large neural language models are used as they are computationally expensive to train. However, this does not preclude the use of WordPiece tokenization as a pre-processing step before training complex language models. A simple, computationally cheap language model could be used to train the WordPiece model whilst a complex model is trained with the resulting tokenization and used for the main language modelling task.

WordPiece subword tokenization has been used in Google's neural machine translation system [68].

4.3 Byte Pair Encoding

A data compression algorithm called Byte Pair Encoding (BPE) finds the pairs of adjacent bytes which occur most frequently and replaces these pairs with a single byte that is not used elsewhere in the data [21]. This process is repeated to compress the data. This algorithm has been successfully adapted for use in language modelling. Starting with one token being used to represent each character, the most frequently occurring pairs of tokens are merged to form a new token. This process is repeated for either a predefined number of iterations, or until a desired vocabulary size is reached [60].

BPE has been widely adopted in language modelling and has been used with many influential models such as the transformer [67], ConvS2S [22], GPT [56] and GPT-2 [57].

4.4 Probabilistic Sub-Word Sampling

For any given word, there are many possible sub-word encodings. The WordPiece and Byte Pair Encoding strategies determine a single (ideally, the most useful) sub-word encoding for each word in the vocabulary. However, in some cases it may be beneficial to have multiple encodings for each word. One strategy for this is to probabilistically sample sub-word encodings [35]. Each time a word is encoded, it's sub-word encoding is randomly sampled from a distribution where encodings deemed to be more useful are given higher probability. In the case of neural language models, word encodings can be re-sampled on each training epoch, adding noise to the training data and achieving a regularizing effect. This novel form of regularization may prove useful in modeling South African languages with limited training data.

5 NEURAL LANGUAGE MODELS

Artificial neural networks can be used in language modelling. There are many different neural model architectures with different properties.

5.1 One-Hot Encoding

To input words into a neural network, they must first be converted to vector representations. The most straightforward approach to this is one-hot encoding. All words in the vocabulary V are given an index from 0 to $|V| - 1$. To encode a word with index i, a vector x of length $|V|$ is constructed containing all zeros except x_i, which has the value 1.

5.2 Embeddings

When making a next-word prediction, the context is provided as input to the neural network in the form of continuous vector embeddings of the preceding words. More formally, an embedding matrix is applied to transform each one-hot-encoded word to a continuous vector representation.

One of the key advantages of neural language models over n-grams is that modeling on embeddings instead of exact words allows neural models generalise better [8]. Word-vector embeddings are created such that words with similar meanings have similar embeddings. Sentences which have never been seen before can be assigned a high probability by the model if they are constructed from words with similar embeddings to those of words in a sentence that existed in the training corpus.

5.3 Feed-Forward Neural Networks

One of the simplest architectures for neural language models is the feed-forward neural network [8]. This architecture makes the same Markov assumption as the n-gram model in that only a limited window of the previous n words are used by the model. One of the original proposed neural language models predicts $P(w_t|w_{t-1}, ..., w_{t-n+1})$ using:

$$x = (W_e u_{t-1}, W_e u_{t-2}, ..., W_e u_{t-n+1}) \tag{9}$$

$$\hat{y} = \text{softmax}(b + W_o^{(1)} x + W_o^{(2)} \tanh(d + W_h x)) \tag{10}$$

$$\begin{aligned} b, \hat{y}, u_{t-n+1}, ..., u_{t-1} &\in \mathbb{R}^V \\ x &\in \mathbb{R}^{mn} \\ d &\in \mathbb{R}^h \\ W_e &\in \mathbb{R}^{m \times V} \\ W_h &\in \mathbb{R}^{h \times mn} \\ W_o^{(1)} &\in \mathbb{R}^{V \times mn} \\ W_o^{(2)} &\in \mathbb{R}^{V \times h} \end{aligned}$$

[8] Here u_i is the one-hot encoding of word w_i, V is the vocabulary size, W_e, W_h, $W_o^{(1)}$, $W_o^{(2)}$, b and d are trainable parameters and $\hat{y}$ is the predicted next word probability distribution over the vocabulary. The hidden layer size h is an adjustable hyperparameter. Of note is how W_e is re-used to create an m dimensional vector representation of each word. W_e is a word embedding matrix where column j is a vector representation of the word with index j in the vocabulary.

5.4 Pre-Trained Word Embeddings

Word embeddings do not necessarily need to be trained as part of the main language model; they can alternatively be learned separately in an unsupervised manner independently of a language modelling task. Typically, a simple model with an objective function related to a neural language model is trained. This model itself may not perform a useful task, but part of the trained model is extracted and used as pretrained word embeddings for a downstream language modelling task. Examples of such pre-trained embeddings include embeddings trained with continuous bag-of-words models [44], skip-gram models [46], GloVe [53] and FastText [10].

6 RECURRENT NEURAL LANGUAGE MODELS

Recurrent Neural Networks (RNNs) are neural network models designed to process variable length input sequences $x_1, ..., x_T$ [19]. The sequence is processed iteratively; the hidden state h_t and output $\hat{y}_t$ at timestep t are calculated using:

$$h_t = f(U x_t + W h_{t-1} + b) \tag{11}$$

$$\hat{y}_t = g(V h_t + c) \tag{12}$$

[24] Here, f and g are activation functions and weight matrices U, V and W as well as bias vectors b and c are trainable parameters. In the context of language models, $\hat{y}$ is typically a predicted next word probability distribution over the vocabulary.

6.1 Simple RNN Language Model

A simple RNN language model can be constructed by letting $x_1, ..., x_T$ be the context word vectors, f be a sigmoid activation function, g be a softmax activation function and length of the output vector $\hat{y}$ be the vocabulary size [45]. The model can then be given a sequence of words and predict a probability distribution for the next word at each timestep.

6.1.1 Vanishing or Exploding Gradient Problem. During training, the error signal must be backpropagated across the hidden states at each timestep. This requires repeated multiplication of the error signal by the same weight values which are shared across timesteps. This causes an exponential growth or decrease of the error signal resulting in highly unstable error signals at the earlier timesteps. This is known as the exploding or vanishing gradient problem and poses a challenge when attempting to train RNNs on longer sequences [9, 52]. The error signal instability at earlier timesteps reduces an RNN's ability to learn long-term dependencies.

One method of dealing with this problem is gradient clipping, where gradients with norms exceeding a given threshold are re-scaled down to that threshold [52]. Another approach, often used in conjunction with gradient clipping is using model architectures designed to reduce the problem such as gated RNNs.

6.2 Gated Recurrent Neural Networks

Gated RNN architectures are designed to mitigate the vanishing gradient problem and improve the ability of RNNs to learn long-term dependencies by including various "gates" within the model. These gates break the chain of multiplication that otherwise occurs over successive timesteps through recurrent connections, by having an additive rather than multiplicative relationship between the contexts of successive timesteps. The successive contexts are related by an addition operator instead of the multiplicative recurrent connection weights of the original RNN architecture. This allows the gradient to propagate across timesteps without exponential decay, thereby reducing the vanishing or exploding of the gradient [52].

Long Short-Term Memory (LSTM) models are widely used gated RNNs [25]. LSTMs use a system of memory cells and gate units to achieve the gating effect described above. Another gated recurrent architecture is the Gated Recurrent Unit (GRU) [14]. This has fewer gates and learned parameters than the LSTM.

6.3 LSTM Language Models

LSTM-based language models show improved performance over the original RNN language models [63]. The current state of the art, the Mogrifier LSTM, extends LSTM models by adding a learned "preprocessing step" for the input and the hidden state at each timestep [42]. The hidden state is used to gate and modify the input. Subsequently the modified input is used to gate and modify the hidden state. This process is repeated for r rounds. The final resulting input and hidden state are used as input for the LSTM at the current timestep. This model achieves state-of-the-art performance on many benchmarks. The authors hypothesise that the lack of broader context in word representations used with LSTM models

imposes a bottleneck on generalisation ability and that the Mogrifier LSTM overcomes this bottleneck by conditioning the word representations on the hidden state of the LSTM.

6.4 Encoder-Decoder RNNs

The encoder-decoder architecture enables a model to take a variable length sequence of tokens as input, compute a fixed-length vector representation (an encoding) of the input and use this to generate a variable length output sequence. A GRU model with an encoder-decoder architecture was proposed for machine translation [14]. Building on this, a deep encoder-decoder model with stacked LSTM layers was proposed as a language model [64]. The increase in depth was found to improve performance.

6.4.1 Long-Term Dependencies. Despite its improved performance, the deep LSTM model has issues with learning long-term dependencies. For example, in the translation task, when the source sentence is input in reverse, the performance improves [64]. This is hypothesised to be due to the closer proximity of the first words of the source sentence to the first words of the target sentence; thus the number of recurrent steps through which the information must propagate is reduced. This idea was later built on by the introduction of the attention mechanism which aims to reduce the number of connections through which information must propagate in order to model long-term dependencies.

6.4.2 Attention . Under the encoder-decoder framework, RNNs are limited by the amount of information that can be stored in a fixed length context vector. Information about words further back in the sentence is often lost. A novel architecture called RNNSearch extends the encoder-decoder framework by giving the decoder direct access to the encoder's hidden states at every timestep of the encoding process [7]. At each timestep, the decoder takes as input a weighted sum of the encoder hidden states. The decoder learns to predict the distribution of these weights and is therefore said to have learned which input timesteps to "pay attention" to at each step of the decoding process.

7 CONVOLUTIONAL NEURAL LANGUAGE MODELS

Widely used in the field of computer vision, is a neural architecture called a convolutional neural network (CNN) [38]. A convolutional layer consists of numerous filters; single feed-forward neurons with only spatially local connections to the input layer which "slide" over the input layer, computing an output for each position. These filters can be conceptualised as "feature detectors" which output the presence or absence of some feature at each position in the input layer.

Since each filter typically has a small number of weights which are reused at each position, a convolutional layer generally has much fewer weights than a fully connected layer of the same size. Thus in the context of language modelling, a CNN can be used as an alternative to a feed-forward language model with fixed context width [18, 22]. Convolutional layers can be used to capture larger context windows than would be possible with fully connected layers. One advantage of convolutional models over recurrent models is that the computation of convolutional layers can be parallelised, yielding faster training times.

8 TRANSFORMER LANGUAGE MODELS

Recurrent neural networks require the hidden states for each timestep to be sequentially computed. This is a performance bottleneck for modern hardware capable of highly parallel computing. A model architecture called a transformer network eliminates this bottleneck by relying entirely on attention mechanisms instead of recurrent connections for propagating information across timesteps [67].

8.1 Model Architecture

The original transformer model is used for translation and has an encoder-decoder structure [67]. For the task of language modelling, only the decoder section of the architecture is used [39]. In this section we describe the decoder-only architecture used by GPT-2 [57]. The model uses learned embeddings of length d_{model} to represent each token.

8.1.1 GPT-2 Architecture. The decoder-only architecture of GPT-2 consists of N stacked identical layers. Each layer both takes as input and produces as output a vector of length d_{model} for each timestep. On a high level, the GPT-2 architecture is as follows:

$$H^{(0)} = UW_e + SW_p \tag{13}$$

$$H^{(i)} = \text{transformer\_block}(H^{(i-1)}) \forall i \in [1, N] \tag{14}$$

$$P(u) = \text{softmax}(H^{(N)} W_e^T) \tag{15}$$

[56] where

$$U, P(u) \in \mathbb{R}^{d_t \times V}$$

$$S \in \mathbb{R}^{d_t \times L}$$

$$W_e \in \mathbb{R}^{V \times d_{\text{model}}}$$

$$W_p \in \mathbb{R}^{L \times d_{\text{model}}}$$

$$H^{(0)} ... H^{(N)} \in \mathbb{R}^{d_t \times d_{\text{model}}}$$

Here d_t is the length of the current input, L is the maximum input length, V is the vocabulary size, U is the input text representation where row i is the one-hot encoding for the word at position i in the text, S is a position matrix where row i is the one-hot encoding for the position of word i (S can be conceptualised as the identity matrix truncated to d_t rows), W_e is the word embedding matrix and W_p is the position embedding matrix, $H^{(i)}$ is the output of hidden layer i and $P(u)$ is the predicted next word probability distribution over the vocabulary. W_e and W_p are trainable weight matrices. Each row t of $H^{(i)}$ contains the output of hidden layer i at timestep t.

The transformer_block of each layer has the following structure:

$$A^{(0)} = \text{per\_timestep\_layer\_norm}(H) \tag{16}$$

$$A^{(1)} = \text{MultiHead}(A^{(0)}, A^{(0)}, A^{(0)}) + H \tag{17}$$

$$A^{(2)} = \text{per\_timestep\_layer\_norm}(A^{(1)}) \tag{18}$$

$$\text{transformer\_block}(H) = \text{feed\_forward}(A^{(2)}) + A^{(2)} \tag{19}$$

[11] where the per_timestep_layer_norm function applies LayerNorm [6] independently to each row (corresponding to each timestep) of it's input matrix:

$$\begin{aligned}\text{per\_timestep\_layer\_norm}(X) &= \text{Concat}(\text{norm}_1, ..., \text{norm}_{d_t}) \\ \text{where norm}_i &= \text{layer\_norm}(X_{i*})\end{aligned} \tag{20}$$

and LayerNorm is applied to a row using:

$$\text{layer\_norm}(x) = a \odot \left(\frac{x - \text{mean}(x)}{\sqrt{\text{var}(x) + \epsilon}} \right) + b \tag{21}$$

[11] Here $a, b \in \mathbb{R}^{d_m}$ are trainable weights which are not shared with subsequent applications of layer_norm.

Finally, the feed_forward network is applied with:

$$\text{feed\_forward}(X) = \text{GELU}(XW_{\text{ff}}^{(1)} + b^{(1)})W_{\text{ff}}^{(2)} + b^{(2)} \tag{22}$$

$$X \in \mathbb{R}^{d_t \times d_{\text{model}}} \tag{23}$$

$$W_{\text{ff}}^{(1)} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{ff}}} \tag{24}$$

$$W_{\text{ff}}^{(2)} \in \mathbb{R}^{d_{\text{ff}} \times d_{\text{model}}} \tag{25}$$

$$b^{(1)} \in \mathbb{R}^{d_{\text{ff}}} \tag{26}$$

$$b^{(2)} \in \mathbb{R}^{d_{\text{model}}} \tag{27}$$

Here $W_{\text{ff}}^{(1)}$, $W_{\text{ff}}^{(2)}$, $b^{(1)}$ and $b^{(2)}$ are trainable weights. These are not shared with subsequent applications of feed_forward.

8.1.2 Attention Mechanism. The model uses an attention mechanism similar to that mentioned in section 6.4.2. However, instead of a single attention being computed, a multi-headed approach is taken. There are h attention "heads" each of which independently learns and computes it's own scaled dot-product attention function:

$$\begin{aligned}\text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O \\ \text{where head}_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)\end{aligned} \tag{28}$$

$$W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$$
$$W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$$
$$W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$$
$$W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$$

The input keys, queries and values are all of dimension d_{model} ($Q, K, V \in \mathbb{R}^{d_t \times d_{\text{model}}}$). Each head h_i has its own set of trained weight matrices W_i^Q, W_i^K and W_i^V which transform the keys queries and values to dimensions d_k, d_k and d_v respectively before passing them to the attention function in equation 30. Finally, the outputs of the heads are concatenated and transformed with W^O to yield an output of dimension d_{model}, the same dimension as the input to the multi-head attention layer.

The scaled dot product attention computation is performed within each head as follows:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{29}$$

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{30}$$

Again, Q, K and V are matrices containing queries, keys and values for each timestep, but unlike equation 28, $Q, K \in \mathbb{R}^{d_t \times d_k}$ and $V \in \mathbb{R}^{d_t \times d_v}$. The function computes a weighted sum of V over each timestep (the rows of V). The weight given to each timestep is determined by the dot product similarity of the corresponding key and query vectors, scaled by $\frac{1}{\sqrt{d_k}}$. The transformation QK^T simultaneously computes the dot products of the key and query vectors for each timestep.

8.1.3 Masking. During training, each self-attention head of the decoder has access to the entirety of the target sentence, including the current word being predicted and future words. This information leak enables the model able to "cheat" and simply learn to base it's prediction of the current word on the current word appearing in the training set, instead of learning to make predictions from the context words. To prevent this, any of the values of the input to the attention softmax which allow the model to "cheat" are set to $-\infty$ during training, thereby preserving the auto-regressive property of the model.

8.2 Positional Encoding

Unlike RNNs, the transformer architecture does not intrinsically encode information about the positions of tokens in the input. Instead, an explicit "positional encoding" is added to the embedding of each word before its input to the network.

8.2.1 Absolute Positional Encodings . In absolute positional encoding schemes, for each token in the input sequence, a positional encoding of length d_{model} is generated as follows:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}}) \tag{31}$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}}) \tag{32}$$

[67] Here i is the dimension and pos is the position. The values of dimensions with even indices i are given by equation 31 and the values of dimensions with odd indices are given by equation 32. A useful property of this encoding is that given any fixed offset k, PE_{pos+k} can be computed as a linear transformation of PE_{pos}.

8.2.2 Relative Positional Encodings. A modification of absolute positional encoding is introduced as a necessity for the Transformer-XL model [16]. The recurrent nature of the model makes absolute position encoding ineffective. Modification of the encoding as well as the attention mechanism allows the model to learn relative offsets when deciding which input to attend to.

8.2.3 Learned Positional Encodings. Alternatively, positional encodings can be learned as shown in. Under this scheme, we add the positional encodings to the word embeddings as described in section 8.2.1, but instead of using sine and cosine functions to generate the positional encodings, the encodings are learned as trainable parameters as shown in equation 13 [22]. Numerous recent variations of transformer networks use learned positional encodings including GPT-2 as described above [56, 57].

8.3 Recent Transformer Models

8.3.1 GPT. Training a language model for some specific task can be done in two steps. First a transformer model with a generative (next word prediction) head is trained on unlabeled text. This first step is task agnostic. Second, a classification head is added alongside generative head. The model is then jointly trained on a target task using the classification head and next word prediction using the generative head [56]. The core transformer architecture is a variant of a decoder-only [39] architecture based on the original transformer [67]. The model uses learned positional embeddings, $N = 12$ decoder layers, a hidden layer size of $d_{\text{ff}} = 3072$ in the feed-forward network, $d_{\text{model}} = 768$ and a context size of 512 tokens.

The generative pre-training is shown to improve performance on the target tasks. The authors show that generative language models implicitly learn some target tasks. For example, it is demonstrated that a purely generative model can be used for zero-shot sentiment analysis by simply appending the word "really" to the input text and comparing the output probabilities of "positive" and "negative" [56].

8.3.2 GPT-2. A subsequent strategy takes the insight that generative models must implicitly learn numerous tasks to its extreme. A large generative model is trained on a large dataset [57]. The dataset, called WebText, contains text extracted from 45 million articles which had been linked to by users of Reddit, a social media platform. Only links which had at least 3 karma (a measure of user interest) at the time of dataset creation are included. The model architecture, based on GPT, contains modifications to the decoder layers and is described in section 8.1.1. Models of varying sizes are trained; the largest being a 1.5 billion parameter model with $N = 48$ layers, a hidden layer size of $d_{\text{ff}} = 3072$ in the feed-forward network, $d_{\text{model}} = 1600$ and a context size of 1024 tokens.

Instead of altering and fine-tuning the model on target tasks, the model is given natural text prompts to perform specific tasks. For example, by appending "tl;dr:" to the input string, the model can be made to perform text summarisation. In the dataset, an article or paragraph is often followed by "tl;dr:" and a summary of an article or paragraph. In order to predict the next words after "tl;dr:" the generative model must implicitly learn text summarisation. Similarly, the model can be made to perform question answering by appending "A:" to the input. The authors additionally demonstrate various other tasks that the model can be made to perform using natural text prompts.

8.3.3 Transformer-XL. Practically, the original transformer architecture can only be trained with a limited context size due to computational resource requirements, making it unable to learn dependencies with a greater temporal offset than this window. A recent architecture called Transformer-XL [16] extends the maximum offset of learned dependencies by introducing recurrent connections into the model.

While conventional RNNs process sequences one input at a time, the Transformer-XL processes segments of multiple inputs at a time, with recurrent connections between segments. When a new segment is processed, the hidden states of the previous segment are kept in memory. In the original transformer architecture, hidden states are able to attend to the layer below them [67], but in the Transformer-XL architecture they are also able to attend the layer below them in the previous segment. In order for the attention mechanism to be able to address these, the relative positional encoding scheme described in 8.2.2 is used. Together, the recurrent connections and relative position encodings allow the transformer-XL to learn longer term dependencies. Additionally, the segment size, otherwise termed attention length, can be increased during evaluation (without needing to retrain the model) allowing a further increase in the effective context window of the model.

9 LOW RESOURCE LANGUAGE MODELLING

When modelling low-resource languages, techniques such as adjusting model architecture and training as well as using data augmentation can be employed to maximise performance.

9.1 Model Optimisation and Regularisation

In a low-resource setting, a model can fit an overly complex function to the available training data, attempting to accurately describe every data point. This means the model may fit the statistical noise in the data instead of more meaningful patterns and fail to generalise to test data [65]. In order to prevent this, we attempt to manage the complexity of the function that the model learns.

A model sensitive to fluctuations in the data is said to have high variance. On the other hand, a model that is unable to learn sufficiently from the patterns in the data is said to have high bias [29]. Decreasing variance typically increases bias; thus a careful balance between the bias and variance of the model must be found: the model must have a low enough bias to adequately fit the data whilst not having a high enough variance to overfit.

9.1.1 Model Size. The more learnable parameters a model has, the greater its variance. Thus, one way to control the variance of a model is by tuning the number of weights in the model. This is typically done by changing the number and sizes of the hidden layers. A recent study compared the performance of different transformer network depths for the translation of South African languages [66]. Another describes a strategy for automatically tuning the size of transformer networks in the context of low-resource translation [48].

9.1.2 Optimisation. There are also many existing gradient-descent-based optimisation algorithms and they typically have numerous parameters that can be tuned. This can include parameters such as learning rate, momentum and stopping conditions [43]. Deliberate tuning of these parameters can improve model performance.

9.1.3 Regularisation. Deep neural language models are complex and generally have high variance making them prone to overfitting in low-resource settings. This effect can be mitigated using various regularisation strategies.

L1 and L2 regularisation add a loss function term to penalize large weight values, preventing the model from being overly reliant on any particular weight [50]. Another technique called dropout temporarily hides a random subset of neurons during each training step [62]. This adds noise and similarly prevents the model from being overly reliant on any particular neuron. These techniques are broadly applicable to neural networks and can be used in feed-forward neural networks, RNNs and transformers [8, 43, 67].

Weight tying shares weights across layers, reducing the number of parameters and hence reducing overfitting [27, 55]. A notable example of this is the sharing of the word embedding matrix at both input and output layers of GPT and GPT-2 [56, 57].

When training RNN models, the batch length of backpropagation sequences and breakpoints between them can also be randomised during training to provide additional noise to reduce overfitting [43].

9.1.4 Sub-word Regularisation. Sub-word segmentation is ambiguous. Probabilistically sampling sub-word segmentations as discussed in section 4.4 can be used as a form of regularisation. This technique additionally constitutes a form of data augmentation and has been shown to improve performance on translation tasks with low-resource languages [35].

9.1.5 Normalisation. During training, the input distributions to each layer of a neural network change as weights in the previous layers are updated. This is called covariate shift. Instability results from weights of subsequent layers being trained on distributions which continuously shift during training. By normalising the values passing through internal layers of neural networks, the covariate shift is reduced [28]. In other words: the input distributions to intermediate layers are made more consistent. This increases stability during training and in practice results in faster training convergence. Additionally, modifications to the normalisation of self-attention layers in transformer models have been shown to improve model performance in a low-resource setting [51].

9.2 Ensemble Language Models

State-of-the-art performance is sometimes achieved using an ensemble of different types of models. For example, the outputs of different models may be averaged or interpolated with learned parameters to yield a combined model with better performance than any of the smaller models individually [3, 32]. Notably, on the South African languages isiZulu, Xitsonga, English and Afrikaans, an ensemble model interpolating the outputs of RNN and n-gram models achieves improved performance over n-gram only models, the previously best-performing model [58].

9.3 Cross-Lingual and Modelling

In some instances, information from high-resource languages can be transferred to low-resource languages. Cross-lingual word representations can be pre-trained using monolingual text from a high-resource language in combination with a lexicon mapping between the high and low-resource languages [3]. These pre-trained word representations can improve the performance of language models trained on limited monolingual text in the low-resource language. A similar approach is to train a single model on multiple monolingual corpora [15, 47, 54]. Alternatively a single model can be trained on many language pairs with parallel corpora to obtain universal representations that can improve information transfer to low-resource languages [5, 15, 26].

9.4 Related Work on African Languages

Comparisons of transformer and convolutional models for machine translation of Southern African languages have recently been performed. The results favoured transformer models over convolutional models. The use of byte-pair encoding also yielded performance improvements [1, 2, 41]. Their sample outputs showed that the translation models were able to generate somewhat coherent sentences suggesting that although training data is limited, it may be possible to successfully perform some language modelling tasks on South African languages.

10 CONCLUSIONS

In the past decade there has been a shift to complex neural architectures such as LSTMs, convolutional models and transformers. Previously, n-grams were the dominant language model. N-grams' simplicity and well studied optimisations such as backoff, interpolation and smoothing algorithms make them a good baseline model for novel language modeling tasks. LSTMs and Transformers now achieve state-of-the-art results.

Accompanying the shift to neural models, there have been improvements in the input and output representations used for language modelling. Many different sub-word tokenization schemes have been developed as well as strategies for learning meaningful word representations and generating pre-trained representations.

Recent models require significant computational resources and are trained on increasingly large datasets. Many regularisation and optimisation techniques are employed to improve model performance. Generative pre-training has also been used to obtain current state-of-the-art results. Furthermore, in some low-resource settings, cross-lingual information transfer has been shown to improve language model performance.

In light of these developments, we feel that an evaluation of n-grams, LSTMs and transformer models on low-resource languages will provide valuable insights as to how much low-resource languages benefit from recent advances in language modelling.

REFERENCES

[1] Jade Abbott and Laura Martinus. 2019. Benchmarking Neural Machine Translation for Southern African Languages. In *Proceedings of the 2019 Workshop on Widening Natural Language Processing*. Association for Computational Linguistics, Florence, Italy, 98–101.

[2] Jade Z Abbott and Laura Martinus. 2018. Towards neural machine translation for African languages. *Neural Information Processing Systems 2018 Workshop on Machine Learning for the Developing World* (2018).

[3] Oliver Adams, Adam Makarucha, Graham Neubig, Steven Bird, and Trevor Cohn. 2017. Cross-Lingual Word Embeddings for Low-Resource Language Modeling. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*. Association for Computational Linguistics, Valencia, Spain, 937–947. https://www.aclweb.org/anthology/E17-1088

[4] Rami Al-Rfou, Dokook Choe, Noah Constant, Mandy Guo, and Llion Jones. 2019. Character-level language modeling with deeper self-attention. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 3159–3166.

[5] Mikel Artetxe and Holger Schwenk. 2019. Massively Multilingual Sentence Embeddings for Zero-Shot Cross-Lingual Transfer and Beyond. *Transactions of the Association for Computational Linguistics* 7, 0 (2019), 597–610. https://transacl.org/index.php/tacl/article/view/1742

[6] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450* (2016).

[7] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015*.

[8] 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.

[9] Y. Bengio, P. Simard, and P. Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks* 5, 2 (1994), 157–166.

[10] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics* 5 (2017), 135–146. https://doi.org/10.1162/tacl_a_00051

[11] Jan Buys. 2020. The GPT-2 Transformer. (2020). work in progress.

[12] 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

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

[14] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. 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)*. 1724–1734.

[15] Alexis Conneau and Guillaume Lample. 2019. Cross-lingual Language Model Pretraining. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 7059–7069. http://papers.nips.cc/paper/8928-cross-lingual-language-model-pretraining.pdf

[16] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive Language Models beyond a Fixed-Length Context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Florence, Italy, 2978–2988. https://doi.org/10.18653/v1/P19-1285

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

[18] Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier. 2017. Language Modeling with Gated Convolutional Networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70 (ICML'17)*. JMLR.org, 933–941.

[19] Jeffrey L Elman. 1990. Finding structure in time. *Cognitive science* 14, 2 (1990), 179–211.

[20] Alexander Franz and Brian Milch. 2002. Searching the Web by Voice. In *Proceedings of the 19th International Conference on Computational Linguistics - Volume 2 (COLING '02)*. Association for Computational Linguistics, USA, 1–5. https://doi.org/10.3115/1071884.1071887

[21] Philip Gage. 1994. A new algorithm for data compression. *C Users Journal* 12, 2 (1994), 23–38.

[22] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. 2017. Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 1243–1252.

[23] I. J. Good. 1956. Some Terminology and Notation in Information Theory. *Proceedings of the IEE - Part C: Monographs* 103, 3 (1956), 200–204.

[24] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. Sequence Modelling: Recurrent and Recursive Nets. In *Deep Learning*. MIT Press, Chapter 10, 374. http://www.deeplearningbook.org/contents/rnn.html

[25] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[26] Haoyang Huang, Yaobo Liang, Nan Duan, Ming Gong, Linjun Shou, Daxin Jiang, and Ming Zhou. 2019. Unicoder: A Universal Language Encoder by Pre-training with Multiple Cross-lingual Tasks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Association for Computational Linguistics, Hong Kong, China, 2485–2494. https://doi.org/10.18653/v1/D19-1252

[27] Hakan Inan, Khashayar Khosravi, and Richard Socher. 2016. Tying word vectors and word classifiers: A loss framework for language modeling. *arXiv preprint arXiv:1611.01462* (2016).

[28] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Francis Bach and David Blei (Eds.), Vol. 37. PMLR, Lille, France, 448–456. http://proceedings.mlr.press/v37/ioffe15.html

[29] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2013. The Bias-Variance Trade-Off. In *An introduction to statistical learning*. Vol. 112. Springer, Chapter 2.2.2, 33–42.

[30] Harold Jeffreys. 1948. Section 3.23. In *Theory of Probability* (2 ed.). Clarendon Press, Oxford.

[31] W. E. Johnson. 1932. Probability: The Deductive and Inductive Problems (appendix). *Mind* 41, 164 (1932), 409–423.

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

[33] Slava Katz. 1987. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE transactions on acoustics, speech, and signal processing* 35, 3 (1987), 400–401.

[34] 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

[35] Taku Kudo. 2018. Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Melbourne, Australia, 66–75. https://doi.org/10.18653/v1/P18-1007

[36] S. Kullback and R. A. Leibler. 1951. On Information and Sufficiency. *The Annals of Mathematical Statistics* 22, 1 (1951), 79–86.

[37] Ankit Kumar, Ozan Irsoy, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus, and Richard Socher. 2016. Ask Me Anything: Dynamic Memory Networks for Natural Language Processing. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48 (ICML'16)*. JMLR.org, 1378–1387.

[38] Yann LeCun, Bernhard E Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne E Hubbard, and Lawrence D Jackel. 1990. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*. 396–404.

[39] Peter J. Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. 2018. Generating Wikipedia by Summarizing Long Sequences. In *International Conference on Learning Representations*.

[40] A. A. Markov. 1913. Essai d'une recherche statistique sur le texte du roman "Eugene Onegin" illustrant la liaison des epreuve en chain ('Example of a statistical investigation of the text of "Eugene Onegin" illustrating the dependence between samples in chain'). *Izvistia Imperatorskoi Akademii Nauk (Bulletin de l'Académie Impériale des Sciences de St.-Pétersbourg)* 7 (1913), 153–162. English translation by Morris Halle, 1956.

[41] Laura Martinus and Jade Z Abbott. 2019. A Focus on Neural Machine Translation for African Languages. *arXiv preprint arXiv:1906.05685* (2019).

[42] Gábor Melis, Tomáš Kočiský, and Phil Blunsom. 2020. Mogrifier LSTM. In *International Conference on Learning Representations*.

[43] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2018. Regularizing and Optimizing LSTM Language Models. In *International Conference on Learning Representations*.

[44] Tomas Mikolov, Kai Chen, Greg S. Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. (2013).

[45] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*. https://www.isca-speech.org/archive/archive_papers/interspeech_2010/i10_1045.pdf

[46] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 3111–3119.

[47] Phoebe Mulcaire, Jungo Kasai, and Noah A. Smith. 2019. Polyglot Contextual Representations Improve Crosslingual Transfer. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, Minneapolis, Minnesota, 3912–3918. https://doi.org/10.18653/v1/N19-1392

[48] Kenton Murray, Jeffery Kinnison, Toan Q. Nguyen, Walter Scheirer, and David Chiang. 2019. Auto-Sizing the Transformer Network: Improving Speed, Efficiency, and Performance for Low-Resource Machine Translation. In *Proceedings of the 3rd Workshop on Neural Generation and Translation*. Association for Computational Linguistics, Hong Kong, 231–240. https://doi.org/10.18653/v1/D19-5625

[49] 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.

[50] Andrew Y. Ng. 2004. Feature Selection, L1 vs. L2 Regularization, and Rotational Invariance. In *Proceedings of the Twenty-First International Conference on Machine Learning (ICML '04)*. Association for Computing Machinery, New York, NY, USA, 78. https://doi.org/10.1145/1015330.1015435

[51] Toan Q Nguyen and Julian Salazar. 2019. Transformers without tears: Improving the normalization of self-attention. *International Workshop on Spoken Language Translation* (2019).

[52] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *International conference on machine learning*. 1310–1318.

[53] Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.

[54] Telmo Pires, Eva Schlinger, and Dan Garrette. 2019. How Multilingual is Multilingual BERT?. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Florence, Italy, 4996–5001. https://doi.org/10.18653/v1/P19-1493

[55] Ofir Press and Lior Wolf. 2017. Using the Output Embedding to Improve Language Models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*. 157–163.

[56] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training. (2018). https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf

[57] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language Models are Unsupervised Multitask Learners. (2019). https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf

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

[59] Mike Schuster and Kaisuke Nakajima. 2012. Japanese and korean voice search. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, Kyoto, Japan, 5149–5152.

[60] 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.

[61] C. E. Shannon. 1948. A Mathematical Theory of Communication. *Bell System Technical Journal* 27, 3 (jul 1948), 379–423. https://doi.org/10.1002/j.1538-7305.1948.tb01338.x

[62] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research* 15, 1 (2014), 1929–1958.

[63] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. 2012. LSTM neural networks for language modeling. In *Thirteenth annual conference of the international speech communication association*.

[64] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to Sequence Learning with Neural Networks. In *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 3104–3112.

[65] Igor V Tetko, David J Livingstone, and Alexander I Luik. 1995. Neural network studies. 1. Comparison of overfitting and overtraining. *Journal of chemical information and computer sciences* 35, 5 (1995), 826–833.

[66] Elan van Biljon, Arnu Pretorius, and Julia Kreutzer. 2020. On optimal transformer depth for low-resource language translation. In *AfricaNLP Workshop 2020*.

[67] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). Curran Associates, Inc., 5998–6008. http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

[68] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *CoRR* abs/1609.08144 (2016). http://arxiv.org/abs/1609.08144