Machine Translation using the English to IsiZulu parallel corpus

Chloë Smith
Wits University
1877342

Jesse Bristow
Wits University
1875955

Dino Anastasopoulos
Wits University
1900661

Abstract

Many difficulties are encountered when trying to build machine translation models for low resourced languages. In this paper, we investigate how different tokenization techniques and RNN architectures can impact the results of a translation model from English to IsiZulu.

1 Introduction

Machine translation is an incredibly useful area of research, but up until recently, there has been a large group of languages (and by extension, people) that have been left out. Neural Machine Translation (NMT) models involving deep learning can be extremely data hungry, but in order to produce quality models, we need quality data. Fortunately, there is a growing community around creating clean, curated corpuses for low-resourced languages, and in this paper, we investigate the performance of some RNN architectures on an isiZulu parallel corpus (Mabuya et al., 2021). The goal is to produce quality translations from English to isiZulu, which we hope is achievable provided a somewhat small, but clean corpus.

2 Background

Deep learning has been successfully utilised in many tasks such as computer vision applications, and more recently in natural language processing (NLP). Feedforward Neural Networks(NNs) have been used to some success using a bag-of-words representation of texts as input, but these lose the sequential information of the text. Convolutional NNs improve on this, keeping contextual information during training, and can also make use of attention. RNNs (Recurrent Neural Networks) tackle this problem by taking into account the inherently sequential nature of text, more closely mimicking how we understand language. RNN encoder-decoder models can be used (Cho et al., 2014), in which the encoder maps variable-length input to fixed length embeddings, and the decoder maps these back to a variable-length target output, which can be used in an NN model, or incorporated into a more traditional statistical NLP model (Cho et al., 2014). RNNs for NLP are most often combined with an attention network, and they consist of either LSTM (Long Short-Term Memory) or GRU (Gated Recurrent Unit) units, but in general, RNNs may use any non-linear activations.

RNNs take the general form of a neural network consisting of a hidden state **h** which is updated at each time step **t** producing output **y**, which may feed into further layers. At each time step, the hidden state is a non-linear function (e.g. sigmoid, or LSTM) of the hidden state at the previous time-step, and the input.

It has been found that increasing the depth of the encoder portion of a network can both increase BLEU scores and decrease loss, while increasing depth in the decoder portion improves loss (Barone et al., 2017). We also see that label smoothing, multi-head attention and layer normalisation can significantly improve performance and stability of RNN-based NMT models (Chen et al., 2018).

In the context of low-resourced languages, the trend is to use data augmentation (an obvious path when the presented issue is a lack of data) and hand-crafted techniques to reduce reliance on deeper architectures (Wang et al., 2021). Another obstacle with language in general that is exacerbated for low-resourced languages is the problem of sub-words. These can result in a large proportion of rare words that should be tokenized into their roots. Common tokenization methods often handle this problem poorly, where for example, even though a model has seen a word like ”alcohol”, it may not see the connection to another word ”alcoholic”, or ”chocoholic”, even though a human would easily see the connection even if the word was new to

them. There has been a rise in the use of Byte Pair Encoding (BPE), a lossless compression algorithm, to encode natural language based on the most common n-grams in a corpus (Sennrich et al., 2016), negating the need for back-off dictionaries when encountering rare words.

Some of the more common encoder techniques include vector embedding, in our case specifically using a Keras (a Python deep learning API) embedding layer, which takes in an index or list of indices (i.e., vocabulary indices) and returns a dense vector of fixed size (Chollet et al., 2015).

These techniques are often used in NN-style natural language processing models, but they have shown great utility in Statistical Machine Translation (SMT) as well, in which word segmentation or tokenization is arguably even more significant. In this area, there is great emphasis on exploiting linguistic knowledge to structure models (Nießen and Ney, 2000), but the strategies used are often conservative. BPE presents a more aggressive tokenization strategy that may bridge the gap where the problem of rare words or lack of specific knowledge about the language can be overcome.

In the following sections, we present a methodology and the results of using some combination of encoders and/or tokenizers, and RNN architectures to produce an NMT model for English to isiZulu. We look at some common tokenizers as well as Byte Pair Encoding (BPE), and the possible influence of language structure on their effects on performance.

3 Methodology

3.1 Data

Sourced from Umsuka English - isiZulu Parallel Corpus

Evaluation Dataset:
998 English sentences with 2 corresponding isiZulu translations each. Each sentence translated by 2 professional English-to-isiZulu translators, since isiZulu is morphologically complex and translations can be quite different.

Testing Dataset:
4739 English sentences and their corresponding isiZulu translations.

The English sentences are sampled from the News Crawl dataset and therefore are skewed to young, white and male perspectives.

3.1.1 Pre-processing

We needed to perform some pre-processing on the data before it could be used for training. Initially when we tried to train on the data we noticed that the computer would quickly run out of available RAM. We realised that this was due to the vocabularies of both languages being overly large. In order to over come this issue we needed to remove some data in such a way to reduce the vocabulary sizes. What we did was just remove any data points that contained more than 20 English words or more than 20 Zulu words. We then split the remaining data into training, validation and testing data.

3.1.2 Tokenization

In order for the models to take in the sentences as input we first needed to tokenize all of the sentences and then convert all of the tokens to integers. We used used two different types of tokenizers in order to compare which tokenizer would produce better results. Once all of the sentences were tokenized, each token was given a unique id which was used to represent each token as an integer. The model could then read in a sentence as input after it had been converted to an array of integers.

Standard Tokenizer:
The first tokenizer we used was the standard keras tokenizer. Basically this tokenizer will just split a sentence into it's separate words and then assign each word(token) an id which is based on how frequently that word appears.

Byte-Pair Encoding Tokenizer:
The second tokenizer we used was a Byte-Pair Encoding tokenizer. This tokenizer first learns which tokens it can assign, given the maximum possible number tokens as a parameter. Essentially it learns these tokens by determining which combination of pairs within it's current vocabulary it can find most frequently in the data and then combining those pairs into new tokens that we can add to the vocabulary. This process is repeated until the vocabulary of tokens is equal to the maximum possible vocabulary. We set our maximum possible vocabulary to be 10000 and then trained one BPE tokenizer on the English sentences and then another on the Zulu sentences.

3.1.3 Padding

Finally once all of the data had been tokenized we had to pad all of the tokenized sentences in order

to make them all the same shape. We needed to do this because the model only accepted a fixed sized input and produced a fixed sized output. This was easily achieved by appending 0's to the tokenized sentences until they were the same shape as the longest tokenized sentence.

3.2 Models

3.2.1 Model 1 - Simple RNN with Standard Tokenization

For our baseline model 1 we just setup a simple RNN that would run the data through some recurrent layers and then some dense layers until finally predicting the output translation. We used this setup to allow for the predictions of words to be based on the words that appeared previously in the sentence. This model took in data that was pre-processed with the standard tokenization.

3.2.2 Model 2 - Simple RNN with Byte-Pair Encoding

Model 2 is the same as model 1 except that we used the BPE tokenization to pre-process the data.

3.2.3 Model 3 - RNN with Word Embedding and Standard Tokenization

For model 3 we added an embedding layer to the baseline model. We added this to allow for the model to have some sense of the meaning of words as well.

3.2.4 Model 4 - RNN with Word Embedding and Byte-Pair Encoding

Model 4 is the same as model 3 except that we used the BPE tokenization to pre-process the data.

3.2.5 Model 5 - Bidirectional RNN with Word Embedding and Standard Tokenization

For model 5 we used a bidirectional RNN instead of just a forward pass RNN with the word embedding. We added this so that the predictions of words would be based on the words that appeared before and after this word in the sentence.

3.2.6 Model 6 - Bidirectional RNN with Word Embedding and Byte-Pair Encoding

Model 6 is the same as model 5 except that we used the BPE tokenization to pre-process the data.

3.2.7 Model 7 - Bidirectional RNN with Word Embedding, Encoder-Decoder and Standard Tokenization

For model 7 we added an encoder-decoder to the bidirectional RNN with the word embedding. We added this in order to try and extract the important information from the inputs and capture the meaning of the sentences more efficiently. Thus allowing the model to understand the input better in order to provide better predictions

3.2.8 Model 8 - Bidirectional RNN with Word Embedding, Encoder-Decoder and Byte-Pair Encoding

Model 8 is the same as model 7 except that we used the BPE tokenization to pre-process the data.

4 Results and Analysis

Figure (1) displays the graphs that show training loss vs epoch, validation loss vs epoch, training accuracy vs epoch, and validation accuracy vs epoch respectively. For each of these graphs we plot the results for all 8 models. Table (1) shows the final training loss, training accuracy, validation loss and validation accuracy for all 8 models. Table (2) displays the Bleu score calculated on the training data, and the Bleu score calculated on the testing data for all 8 models. Figure (2) displays the training Bleu scores from table (2) as a bar graph with the red bars indicating the models that used the standard tokenization and the blue bars indicating the models that used the BPE tokenization. Figure (3) displays the testing Bleu scores from table (2) in the same format as figure (2).

Table 1: Accuracy and Loss

Model	Train Loss	Train Acc	Val Loss	Val Acc
1	1.2911	0.7335	5.8239	0.5804
2	1.4706	0.7246	3.6206	0.6860
3	0.2429	0.9607	6.4931	0.5620
4	0.1791	0.9516	4.5984	0.6627
5	0.1324	0.9848	6.6252	0.5607
6	0.2719	0.9415	4.6694	0.6675
7	1.9809	0.6227	5.8928	0.5712
8	1.8533	0.7005	3.7763	0.6687

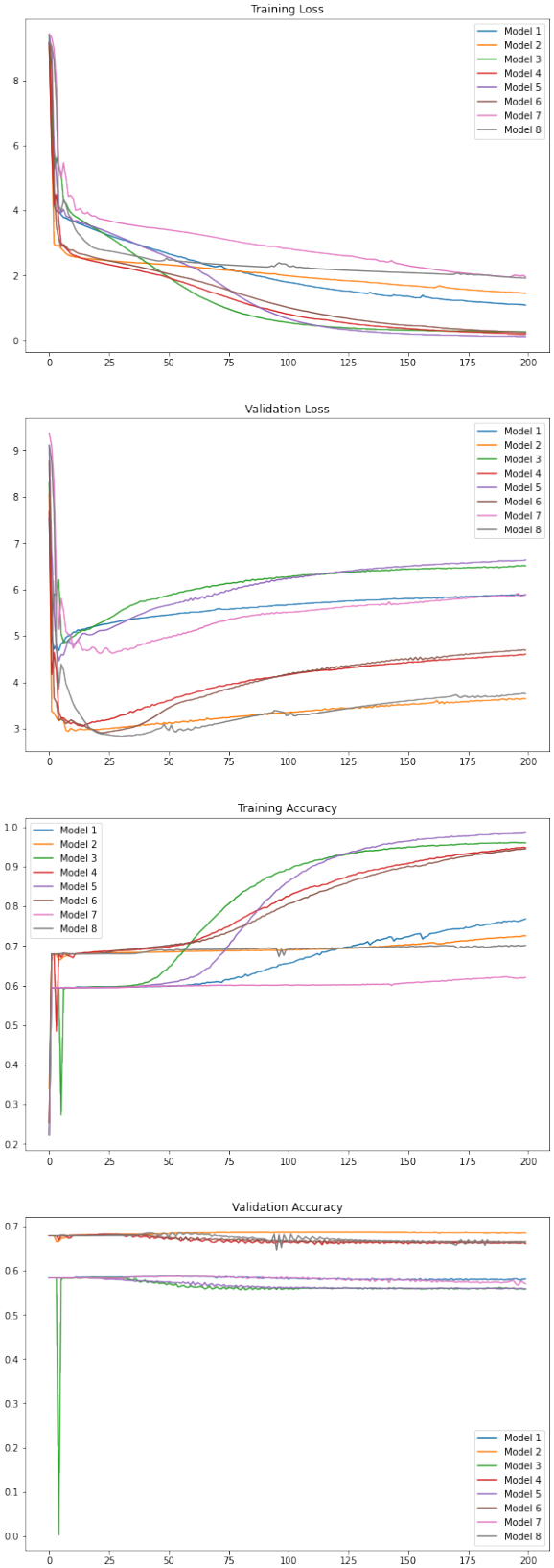

Figure 1: Accuracy and Loss

4.1 BLEU Scores

One of the most common metrics for a translation model is the BLEU (BiLingual Evaluation Understudy) score: individual translated sequences are compared to a set of good quality reference translations, and these similarity scores are averaged across the corpus, giving a metric between 0 and 1 (in practice usually multiplied by 100). While there is debate on the usefulness of BLEU for comparison, as it is not always a reliable measure of improvement (Doddington, 2002), it is a very common and easily interpretable metric.

Table 2: BLEU Scores

Model	**Training Score**	**Testing Score**
1	14.85	19.05
2	2.32	1.62
3	72.54	4.9
4	71.8	2.02
5	78.23	8.35
6	67.4	2.68
7	3.79	35.97
8	1.99	14.79

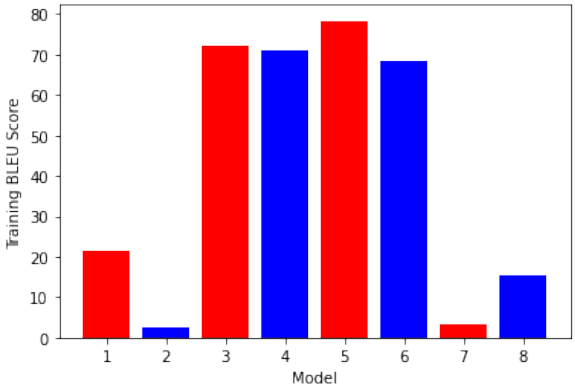

Figure 2: Training Data BLEU Scores

Figure 3: Testing Data BLEU Scores

4.2 Discussion of Results:

As we can see from figure (1), all of the models start off learning very slowly. After some time the models 3,4,5 and 6 start to learn quicker and produce better training accuracies/losses. In the end these 4 models produce good training accuracies/losses/bleu scores, although we can see that none of the validation accuracies/losses improve across any of the models and also all of the final validation bleu scores are poor. The reason for this is probably due to the small size of the data set used because the model doesn't learn about enough words or sentences during training in order to successfully make predictions on unseen words or sentence structures during testing.

As we can see the standard tokenizer produces better accuracies/losses and bleu scores in general compared to the BPE tokenizer, although the tokenizer doesn't have as much of an impact as the model architecture. The reason the results may appear worse for the BPE tokenizer is because the sentences are tokenized to longer lengths which means that the model has to make more token predictions per input sentence. Thus it is more difficult for the model to learn the correct predictions.

We can see that the word embedding definitely improves the results of the baseline model, and we can also see that the bidirectional RNN slightly improves the results compared to just using a regular RNN. The addition of the encoder and decoder worsens the results of the model. This may be due to the fact that the encoder and decoder need more data or time to learn better.

4.3 Example Translations

These translations were computed using the best performing model, that with the highest BLEU score, model 5 (Bidirectional RNN with WordEmbedding and Standard Tokenization). We can clearly see in the examples below that the model is highly overfitting to the training data.

4.3.1 Example from Training Set

English source sentence:
his solo albums in the 60s were some of the most brilliant and breathtaking beautiful sounds i've ever heard
isiZulu ground truth:
ama-albhamu akhe e-solo eminyaka yawo-60 angamanye anezingoma ezimnandi kakhulu kwengake ngazizwa
isiZulu prediction:
ama albhamu akhe e solo eminyaka yawo 60 angamanye anezingoma ezimnandi kakhulu kwengake ngazizwa
Google translate from predicted isiZulu to English:
his solo albums of the 60s are some of the most beautiful songs I have ever heard

4.3.2 Example from Testing Set

English source sentence
one user said: "gordon is a typical negative vote leave curmudgeon.
isiZulu ground truth:
omunye umsebenzisi uthe: "ugordon uyi-curmudgeon ejwayelekile yevote leave
isiZulu prediction:
i jose uthe uthe white elibheka ukunambitheka ukhona
Google translate from predicted isiZulu to English:
jose said he said white looking taste is present

5 Impact

The results of this experiment appear to point to the need for some sort of regularisation, as we found that with the basic RNN architecture that we made small modifications to, training and validation accuracy would increase rapidly and then hit a plateau, at which point training accuracy continued to increase while validation accuracy slowly decayed. We are hesitant to point to very deep architectures, however, as the problem of very small volumes of clean data is still present.

6 Conclusion

Using our standard RNN architecture for comparison, a bi-directional RNN with basic embedding and standard tokenization learned the most from training looking at accuracy, while a bi-directional RNN with BPE using an encoder-decoder network had the best testing accuracy. Neither, however, had very good BLEU scores or anecdotally "good" translations. This suggests we may be using completely misleading measures of accuracy during training, and a topic for further study would be the effect of different loss functions on the performance of a model indicated by some set of metrics other than basic accuracy.

References

Antonio Valerio Miceli Barone, Jindrich Helcl, Rico Sennrich, Barry Haddow, and Alexandra Birch. 2017. Deep architectures for neural machine translation. *CoRR*, abs/1707.07631.

Mia Xu Chen, Orhan Firat, Ankur Bapna, Melvin Johnson, Wolfgang Macherey, George F. Foster, Llion Jones, Niki Parmar, Mike Schuster, Zhifeng Chen, Yonghui Wu, and Macduff Hughes. 2018. The best of both worlds: Combining recent advances in neural machine translation. *CoRR*, abs/1804.09849.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation.

Francois Chollet et al. 2015. Keras.

George Doddington. 2002. Automatic evaluation of machine translation quality using n-gram co-occurrence statistics. In *Proceedings of the Second International Conference on Human Language Technology Research*, HLT '02, page 138–145, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Rooweither Mabuya, Jade Abbott, and Vukosi Marivate. 2021. Umsuka english - isizulu parallel corpus. Thank you to Facebook Research for funding the creation of this dataset.

Sonja Nießen and Hermann Ney. 2000. Improving smt quality with morpho-syntactic analysis. pages 1081–1085.

Surangika Ranathunga, En-Shiun Annie Lee, Marjana Prifti Skenduli, Ravi Shekhar, Mehreen Alam, and Rishemjit Kaur. 2021. Neural machine translation for low-resource languages: A survey.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units.

Rui Wang, Xu Tan, Renqian Luo, Tao Qin, and Tie-Yan Liu. 2021. A survey on low-resource neural machine translation.