Neural Network Language Models for Low Resource Languages

Author1, Author2, Author3
Affiliation1, Affiliation2, Affiliation3
Address1, Address2, Address3
author1@xxx.yy, author2@zzz.edu, author3@hhh.com
{author1, author5, author9}@abc.org

Abstract

Each article must include an abstract of 150 to 200 words in Times New Roman 9 with interlinear spacing of 10 pt. The heading Abstract should be centred, font Times New Roman 10 bold. This short abstract will also be used for producing the Booklet of Abstracts (PDF) containing the abstracts of all papers presented at the Conference.

Keywords: keyword1, keyword2, keyword3

1. Introduction

2. Related Work

1. Language Models (Ngram vs NN)
2. Word Embeddings
3. Tokenization Sentencepiece

3. Experimental Setup

3.1. Corpus

We used the Mi'kmaq corpus built by (Maheshwari et al., 2018) and the script of the Passamaquoddy Texts[1]. The Mi'kmaq corpus was already tokenized, and the only additional cleaning steps taken were normalizing the different types of curly quote characters with ASCII single or double quotes and normalizing long hyphens characters with ASCII dashes. The Passamaquoddy corpus included English text and book annotations that were removed and the corpus was lowercased and tokenized. We divided the corpora and kept 80 percent for train, 10 percent for dev and 10 percent for test. All preliminary experiments were performed on the dev data to select hyperparameters, and the reported experiments use test data for model comparision.

3.2. Tokenization/Segmentation

SentencePiece[2] was used to tokenise the corpus into subwords to address the polysynthetic nature of the Mikmaq language. SentencePiece is an unsupervised text tokenizer and detokenizer. It implements subword units specifically byte-pair encoding and unigram language model from a corpus of raw sentences.

Byte Pair Encoding: It is a simple data compression technique that iteratively replaces the most frequent pair of sequential bytes with a single byte. For word segmentation, we initialise the symbol vocabulary with the character vocabulary. We then count the ocurrences of all symbol pairs and replace eachy occurrence of the most frequent pair (X, Y) with a merged symbol (XY). Each merge operation produces a new character ngram. For efficiency, we do not consider the pairs that cross word boundaries.

Unigram Language Model:(Kudo, 2018) proposed this segmentation based on the unigram language model. It makes one assumption that all subword occurrences are independent and the subword sequence is produced by the product of subword occurrence probabilities.

In SentencePiece, the whitespace is also treated as a token unlike traditional tokenization schemes. Consider the following example of an english phrase: *Hello World*.
The whitespace is first escaped with a meta symbol "_": *Hello_World*.
Then, segmented in different schemes as: *[He][llo][_World]*

The word level tokenisation excludes the whitespace leading to a loss of the original sequence of words, however, the sentencepiece tokenisations as described above append the whitespace in the token succeeding it, therefore preserving the original text.

Corpus	UNI	BPE	Words
Mi'kmaq			
Train	1,83,415	1,73,273	65,495
Dev	23,880	22,257	8263
Test	22,882	24,577	8696
Passamaquoddy-Maliseet			
Train	1,83,415	1,73,273	9,869
Dev	23,880	22,257	914
Test	22,882	24,577	952

Table 1: Number of tokens identified using different tokenization schemes

Sentence Piece trains the segmentation models on a fixed predetermined vocabulary, table 1 shows the number of tokens(subwords) identified in the corpus with BPE and Unigram algorithms.

3.2.1. Neural Network Language Models

We considered 2 RNN model architectures: LSTM networks (Hochreiter and Schmidhuber, 1997) and GRU net-

[1] https://www.gutenberg.org/ebooks/51200
[2] https://github.com/google/sentencepiece

works (Cho et al., 2014). We used Pytorch and their word level RNN language model as base.[3] The explored hyperparameters include the number of layers, the learning rate, the amount of dropout, and the embedding size. We further explored the use of weight tying which has been shown to improve the performance of language models by (Press and Wolf, 2017).

3.3. Evaluation Metrics

There are a few common metrics that are used to evaluate the performance of a language model,however, the use of subword units as tokens brings into question the validity of these conventional metrics like perplexity to compare these language models. We use two different evaluation metrics in order to achieve a better understanding of how our language models compare across different segmentations. In this section, we describe these evaluation metrics we used to evaluate our models:

Bits-Per-Characters: Perplexity is calculated as follows(Jurafsky and Martin, 2018b):

$$LTP = \sum_{i=1}^{N} \log P(w_i|w_1.....w_{i-1})$$

$$PPL = (2^{LTP})^{\frac{-1}{N}}$$

Language models are often intrinsically evaluated using perplexity, which considers the probabilities given to the correct sequences of words in a test corpus (Jurafsky and Martin, 2018b). However, perplexity values across closed vocabulary and open vocabulary language models are practically not comparable, since, it presents perplexity per prediction token where the prediction "token" differs across different segmentations as words, characters or subwords. To thus make these values comparable we normalise the perplexity as:

$$NPPL = \log PPL * (ntokens)$$

where ntokens is the number of tokens in the evaluation document.

Keystrokes Savings Rate: Keystrokes Savings Rate is closely associated to the downstream application of next word prediction in smartphones where many soft keyboards provide three suggestions for the next word. This metric measures the percentage reduction in keys pressed compared to letter-by-letter text entry.

$$KSR = \frac{keys_{normal} - keys_{prediction}}{keys_{normal}} * 100$$

We theorise that with an ideal language model a user could type the desired sentence by selecting words from the suggestions leading to the use of only as many keystrokes as the number of words in the sentence.Therefore, a language model with higher keystrokes saved is considered better. Since, this metric computes the absolute number of characters saved (keystrokes), it is independent of the nature of the tokens (word or subwords)and presents a fair comparison across the different segmentations and vocabulary sizes. We explore the metric for n=1,3,5,10 where n is the number of recommendations prompted to the user.

[3] https://github.com/pytorch/examples/tree/master/word_language_model

4. Experimental Results

4.1. Impact of Subword Tokenization Over Conventional Word Level Models

In Tables 2 and 3, we compare the performance of word level language models with subword level language models. For a morphologically rich language like Mi'kmaq, a subword language model will be highly effective.The keystrokes Savings Metric shows a dramatic improvement over the word level model with Byte Pair Encoding segmentation with 2,898 keystrokes saved. Interestingly, the unigram language model presents worse results than the word level model in both GRU and LSTM architecture. These results provide suitable evidence that subword information is useful for Mi'kmaq language modelling. Since, the GRU with BPE segmentation outperforms all models, we will use that for further experiments.

Model	Tokenization	Normalised Perplexity	
		Dev	Test
GRU	Word	1.02	1.05
	BPE	1.33	1.36
	UNI	1.34	1.37
LSTM	Word	0.96	0.99
	BPE	1.35	1.40
	UNI	1.35	1.38

Table 2: Normalised Perplexity for RNN Language models with different tokenization schemes

4.2. Differing Vocabulary Sizes for Tokenization

Subword Level models can be fitted with custom vocabulary sizes to improve their performance. We explored the effects of varying vocabulary sizes on the normalised perplexity and keystroke saved metrics. In section 4.1, we concluded that the BPE algorithm with a GRU tied architecture worked best for Mi'kmaq language models. Tables 4 and 5 show the performance of a GRU-Tied-BPE model for vocabulary sizes between 1000-8000 words.For both metrics evaluated we see a general trend of improvements for an optimal vocabulary size and then the scores become worse for large vocabulary sizes. This downward trend is indicative of the model learning longer words for large vocabulary sizes leading to poor results similar to a word level language model. We get the best overall results for a vocabulary size of 2000 words.

4.3. Word Embeddings

Pretrained word embeddings and using them to initialize a model's embedding layer has been shown to improve language model performance (Bojanowski et al., 2016). These

Model	Tokenization	n=1		n=3		n=5		n=10	
		Dev	Test	Dev	Test	Dev	Test	Dev	Test
GRU	Word	1.24	1.09	2.18	2.05	2.89	2.79	2.89	4.01
	BPE	1.23	**1.17**	2.79	**2.75**	3.36	**3.62**	5.30	**5.25**
	Unigram	0.22	0.30	1.13	1.05	1.97	1.81	3.19	3.04
LSTM	Word	1.08	0.90	1.76	1.65	2.35	2.31	3.50	4.01
	BPE	1.24	1.16	2.71	2.69	3.63	3.58	5.20	5.12
	Unigram	0.27	0.29	1.31	1.17	2.07	1.94	3.25	3.10

Table 3: Keystroke Savings Rate (higher is better) for RNN Language models with different tokenization schemes

Model	Tokenization	Normalised Perplexity	
		Dev	Test
GRU	1k	1.36	1.40
	2k	1.33	1.37
	4k	1.33	1.36
	8k	1.36	1.38

Table 4: Normalised Perplexity for RNN Language models with different tokenization schemes

embeddings are typically trained using a continuous bag-of-words(CBOW) or Skip-gram (Mikolov et al., 2013) approach.Fasttext is a state-of-the art implementation of these algorithms and it has the additional advantage of considering subword information to build embeddings (Bojanowski et al., 2016). FastText works by splitting words into character n-grams and learning vector representations for those n-grams. It then makes the vector representation of a word the sum of the representations of the n-grams that compose it. These character n-grams can then be used to share sub- word representation between words, and we can build word representations for new words using the character n-grams. This method is advantageous for Mi'kmaq as it is a polysynthetic language with multiple morphemes in every word (Johnson, 1996) furthermore, the corpus has 22k types and 83k tokens which means the model encounters a high frequency of out-of-vocabulary words while testing. Tables 6 and 7 compare the performance of a GRU with BPE tokenization with different embedding layers. The skipgram Fst embeddings are trained on a whitespace tokenized corpus while the Tokenized Fst are trained on the BPE-2k tokenized corpus.The word fst is a word level model that (Boudreau et al, 2019) showed to give sota results for Mi'kmaq corpus.

5. Conclusions

6. Bibliographical References

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2016). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (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)*, pages 1724–1734, Doha, Qatar, October. Association for Computational Linguistics.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Jurafsky, D. and Martin, J. H. (2018b). Chapter 3: Language Modelling. In *Speech and Language Processing*. Unpublished.

Kudo, T. (2018). Subword regularization: Improving neural network translation models with multiple subword candidates. In *ACL*.

Maheshwari, A., Bouscarrat, L., and Cook, P. (2018). Towards Language Technology for Mi'kmaq. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May 7-12, 2018. European Language Resources Association (ELRA).

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Neural and Information Processing System (NIPS)*.

Press, O. and Wolf, L. (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*, pages 157–163, Valencia, Spain, April. Association for Computational Linguistics.

Model	Vocabulary Size	n=1		n=3		n=5		n=10	
		Dev	Test	Dev	Test	Dev	Test	Dev	Test
GRU	1K	1.06	1.10	2.45	2.44	3.35	3.29	4.73	4.67
	2K	1.23	1.17	2.79	2.75	3.36	3.62	5.30	5.25
	4K	1.11	1.05	2.67	2.43	3.76	3.30	5.35	4.93
	8K	1.01	0.93	2.32	2.13	3.27	2.82	5.21	4.76

Table 5: Bits-per-character for RNN Language models with different Vocabulary Sizes

Model	Initialisation	n = 1		n = 3		n = 5		n = 10	
		Dev	Test	Dev	Test	Dev	Test	Dev	Test
GRU BPE	Word	1.24	1.09	2.18	2.05	2.89	2.79	2.89	4.01
	Normal	1.23	1.17	2.61	2.64	3.63	3.62	5.30	5.25
	Word Fst	1.06	1.10	2.45	2.44	3.35	3.29	4.73	4.67
	Skipgram Fst	1.19	1.20	2.52	2.53	3.65	3.60	5.07	4.99
	Tokenized Fst	1.32	1.32	2.65	2.66	3.81	3.78	5.34	5.26

Table 6: Keystrokes Savings Rate for RNN Language models with different initialisation methods

Model	Tokenization	Normalised Perplexity	
		Dev	Test
GRU	Word	1.021	1.053
	Normal	1.335	1.369
	Word Fst	0.910	0.941
	Skipgram Fst	1.343	1.368
	Tokenised Fst	1.332	1.359