

Tensor Product of Representations in Service of Low-Resource Languages

G. Arshinov, S. Kosyak, D. Samsonova, F. Tyers, M. Voronov

June 2020

Abstract

Polysynthetic low-resource languages are poorly treated with standard language modeling approaches. In this paper a hypothesis that word-segment embeddings based on tensor product of representations show better performance for low-resource languages compared to conventional word- and char-based models is checked. In order to prove that a pipeline that allows to process low-resource polysynthetic languages was developed. Using Neural Sequence Labeling Toolkit (Yang and Zhang 2018) to train a segmenter on a Chukchi corpus, a raw Chukchi corpus was segmented and the *iiksiin* (Schwartz, Haley, et al. 2019-2020) model was employed to create the embeddings. After that we tested them on language modelling task and evaluated the results, which showed a notable increase in performance compared to regular approaches.

Tags: TPR, Chukchi, language modeling, polysynthetic, low-resource, NLP

1 Introduction

Most traditional NLP frameworks target languages that do not have much inflection (e.g. (Mikolov et al. 2013)). Such frameworks treat *pug* and *pugs* as separate words. It works reasonably well for such analytical languages as Chinese or English. However, there are polysynthetic languages that feature extensive morphology and cannot be efficiently processed this way.

In this work we will model the Chukchi language. Let us consider the following two examples (examples (1) and (2)). They clearly demonstrate the fact that it is inefficient to process such languages with traditional frameworks such as Word2Vec.

- (1) wełə-tko-ra-jpə-ŋ
goods-ITER-dwelling-ABL-AD
in the shop
- (2) q-wełə-tko-ra-nta-ŋ-e=ʔəm
2.S/A.SUBJ-goods-iter-dwelling-GO.DO-IRR-2/3SG.S=EMPH
go to the shop!

Moreover, many polysynthetic languages are minority languages. It is spoken in north-east Russia by approximately 5100 people and is marked as "threatened" (*Ethnologue. Languages of the World. Chukchi* 2020). Therefore, there is very little language data available for Chukchi.

So, our goal is to develop a pipeline that can process low-resource languages with extensive inflectional morphology.

2 Related work

The idea that some smaller segments can be used as representations was suggested in (Smolensky 1990). In this article Smolensky suggests "a formalization of the idea that a set of value variable pairs can be represented by accumulating activity in a collection of units each of which computes the product of a feature of a variable and a feature of its value" (Smolensky 1990, p. 159). He suggests using tensor product to accumulate representations of smaller structures into bigger ones.

This way, a word *pugs* will be treated not only as a whole but also as a combination of two morphemes.

Before we may continue, we should provide the definition of a tensor product.

Definition 1 *Let V_1 and V_2 be two vector spaces. A space W furnished with a map $(x_1, x_2) \mapsto x_1 \cdot x_2$ of $V_1 \times V_2$ into W , is called the tensor product of V_1 and V_2 if the two following conditions are satisfied:*

- i. $x_1 \cdot x_2$ is linear in each of the variables x_1 and x_2 .*
- ii. If (e_i1) is a basis of V_1 and (e_i2) is a basis of V_2 , the family of products $e_i1 \cdot e_i2$ is a basis of W .*

(Serre 1977, p. 8)

The idea to use tensor product of representations to process a natural language, was implemented in 2019 in a tool called *iiksiin* (Schwartz, Tyers, et al. 2020; Schwartz, Haley, et al. 2019-2020). It "constructs a sequence of morpheme tensors from a word using Tensor Product Representation" (Schwartz, Haley, et al. 2019-2020). We will further cover the way this tool functions.

3 Data

For our language model experiments we decided to use Chukchi language (Tyers n.d.). We chose it because it is low-resource and hence, may prove the concept with the least amount of available data. Our corpus consisted in fiction and folklore texts in Cyrillic transliteration. We also had some data serving as a gold standard for segmentation, though it was written in Latin alphabet. Table 1 shows some statistics for the data.

To use the corpus, we had to do several things: At first, we fixed the apostrophes after *h* and *k* to have *ḥ* and *ḳ*. We then removed invalid characters and fixed the ones in wrong language, meaning *C* (U+0043) and *C* (U+0421). Finally, we fixed the

| | sentences | words |
|---------------|-----------|--------|
| Corpus | 33331 | 151667 |
| Gold standard | 1006 | 4417 |

Table 1: Preprocessed corpus statistics

apostrophes after the vowels putting ? before the vowel, so that *a'* would be ?*a*. Fixing the segmentation standard also involved these steps, though at the beginning we had to transliterate it to Cyrillic alphabet. Unfortunately, we had to review some of the sentences manually to make sure the segmentation worked correctly. Table 2 shows the statistics for the post-processed corpus.

| version | changes | sentences | words |
|---------|--------------------|-----------|--------|
| v2 | н' and κ' | 33331 | 151667 |
| v3 | invalid characters | 33323 | 151585 |
| v4 | ? before the vowel | 33323 | 151585 |

Table 2: Post-processed corpus statistics

The example of changes in the data can be seen in the Table 3.

| version | changes |
|---------|---------------------------|
| v1 | а'ачек>∅ эты н>ин>ив>к'ин |
| v2 | а'ачек>∅ эты н>ин>ив>џин |
| v3 | а'ачек>∅ эты н>ин>ив>џин |
| v4 | ?аачек>∅ эты н>ин>ив>џин |

Table 3: Changes in data

One of the questions we asked ourselves was if our data fixes may be incorrect and would significantly effect the quality of the segmentation model, so we ran it using different versions; the results will be later described.

Evidently, there are not many resources to use both for segmentation training and validation, so we decided to manually validate a piece of the output of the segmentation model in order to have more data to rely on. After the corpus segmentation data had to be put into the tensor-making model; the output of the segmentation model had to be converted from BMES format to the segmented sentences with delimiters.

4 Segmentation

The TPR model requires moderately large dataset of texts segmented into morphemes for training. Initially, we had only 1000 segmented sentences in Chukchi and that was not sufficient enough for getting any meaningful training results. To extend our training set, we obtained an unsegmented Chukchi language corpus and segmented it automatically.

To achieve any satisfactory segmentation quality, we tested several different approaches varying from rule-based to neural net based solutions. At first, we tried using an LSTM sequence-to-sequence model. We used the OpenNMT library (Klein et al. 2017), that is suitable for solving various sequence-to-sequence tasks, mainly machine translation. We took a word-level tokenized sentence as an input sequence and an arrangement of morphemes and their respective glosses as an output sequence. We used 770 examples for the training and 130 ones for evaluation. The resulting accuracy of 0.33 was, obviously, not enough to rely on this model.

Later, we tried using a rule-based approach. We discovered an in-progress project (Andriyanets and Tyers 2018) that was based on finite state transducing. We tested this tool and got the accuracy of 76.2 %, that was still not satisfactory. After we decided, that the rule-based approach is not the best possible way to achieve what we pursue, we reformulated the task: the main goal of the segmenter was to show where are the borders between morphemes, not identify them or gloss. Considering this fact, the task was restated as character-level sequence tagging. This allowed us to use the Neural Sequence Labeling toolkit (Yang and Zhang 2018), that leveraged convolutional neural network with conditional random field based output layer. To train this model we used 1315 unique words and 146 ones for test. The model was fed words without any context, these words were treated as “sentences”. Each character was assigned one of the four labels: B-MORPH, M-MORPH, E-MORPH, which stand for beginning, middle and end of morpheme. One more label is S-MORPH, that stands for a single character morpheme. The output of the model is a sequence of the aforementioned tags. We trained during 1000 epochs, the 879th of which gave the most accurate results. This model showed 91% F-1 rate for morpheme segmentation.

The final evaluation metrics are shown the Table 4:

| Accuracy | Precision | Recall | F1-measure |
|----------|-----------|--------|------------|
| 0.9577 | 0.9193 | 0.9131 | 0.9162 |

Table 4: Segmentation evaluation stats

5 Tensor Product Representation

Before using *iiksiin* we had to create a readme file, fix a bug in the code and rewrite a *Makefile*.

Now we provide the detailed explanation of how *iiksiin* works. The first step is to generate alphabet Σ for the Chukchi corpus Σ^* and the dictionary of morpheme tensors. These tensors are produced by the following formula:

$$(1) \quad repr(m) = \sum_{i=1}^n \left(oneHot(s_i, \Sigma) \otimes oneHot(r_i, m) \right)$$

Where:

- *oneHot* – one hot encoding function
- \otimes – tensor product (in this case equals to the outer product)
- *s* – symbol in the morpheme
- *r* – role (index of a symbol within the morpheme)
- *n* – number of symbols in the morpheme

Here we provide an example:

$$\begin{aligned}
 repr(caab \in \{a, b, c, d\}^*) &= \begin{pmatrix} 0 & 0 & 1 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix} + \\
 &+ \begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & 1 & 0 & 0 \end{pmatrix} + \\
 &+ \begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 & 1 & 0 \end{pmatrix} + \\
 &+ \begin{pmatrix} 0 & 1 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 0 & 0 & 0 & 1 \end{pmatrix} = \\
 (2) \quad &= \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}
 \end{aligned}$$

The next step is to generate tensors for each word $w \in m^*$ in the following way: we sum the outer product of two one-hot vectors for each symbol s_i in a word w . The length of first one-hot vector is equal to the length of alphabet Σ . The symbol index in the alphabet stands for its position in the vector. The length of the second vector equals to the length of the morpheme m . The symbol index in the morpheme stands for its position in the vector.

$$(3) \quad repr(w) = \sum_{i=1}^n \left(repr(m_i) \otimes oneHot(m_i, w) \right)$$

Where n is a number of morphemes in a word.

Example:

$$\begin{aligned}
 (4) \quad repr(\{caab, bd\}) &= \\
 &= \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 0 & 1 \end{pmatrix} = \\
 &= \left(\begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \right)
 \end{aligned}$$

The resulting third-rank tensors are very sparse. So, they should be converted into first-rank tensors (vectors) with a neural network algorithm. So, the result is a vector space which we call a tensor product of representations.

6 Evaluation

To evaluate the quality of the tensor representations of natural language we have decided to train awd-lstm-lm (Merity et al. 2020) language model.

This language model was chosen due to the fact that it performs the results for polysynthetic languages close to the state-of-the-art and its code is freely distributed and allowed to use.

The LSTM-model was trained on characters, words and segments (with tensor representation as pre-trained embeddings) and the perplexity of each language model was measured, the results are in the Table 5.

| Input format | Preplexity |
|-------------------------------------|------------|
| Character | 2677.94 |
| Word | 3930.33 |
| Segment (with pertained embeddings) | 623.53 |

Table 5: LSTM-model performance

According to the results, the tensor representation makes a significant improvement on the language model rate of perplexity.

Results are to be analyzed and described.

7 Conclusion

In this paper we test the hypothesis that word-segment embeddings based on tensor product of representations show better performance for low-resource languages compared to conventional word- and char-based models. To prove that we developed a pipeline that allows to process low-resource polysynthetic languages. Firstly, we used Neural Sequence Labeling Toolkit (Yang and Zhang 2018) to train a segmenter on a Chukchi corpus. Later, we segmented a raw Chukchi corpus using it. Secondly, we used *iiksiin* (Schwartz, Haley, et al. 2019-2020) to create the embeddings. After that we tested them in action and evaluated the results, which showed a notable increase in language modeling performance.

Thanks

We would like to show our appreciation to the HSE expeditions which visited Chukotka and collected the corpus of texts in Chukchi, which gave us an opportunity to test the hypothesis using the corpus they made. We would also like to mention that this research was supported in part through computational resources of HPC facilities at NRU HSE.

A List of abbreviations

- ITER – iterative aspect
- ABL – ablative case
- AD – archaic dative
- 2.S/A.SUBJ-...-IRR-2/3SG.S – nonimperfective subjunctive mood, subject is singular, in second person
- EMPH – emphatic clitic

References

- Andriyanets, Vasilisa and Francis Tyers (Aug. 2018). “A prototype finite-state morphological analyser for Chukchi”. In: *Proceedings of the Workshop on Computational Modeling of Polysynthetic Languages*. Santa Fe, New Mexico, USA: Association for Computational Linguistics, pp. 31–40. URL: <https://www.aclweb.org/anthology/W18-4804>.
- Ethnologue. Languages of the World. Chukchi* (2020). URL: <https://www.ethnologue.com/language/ckt>.
- Klein, Guillaume et al. (July 2017). “OpenNMT: Open-Source Toolkit for Neural Machine Translation”. In: *Proceedings of ACL 2017, System Demonstrations*. Vancouver, Canada: Association for Computational Linguistics, pp. 67–72. URL: <https://www.aclweb.org/anthology/P17-4012>.
- Merity, Stephen et al. (2020). *awd-lstm*. URL: <https://github.com/neural-polysynthetic-language-modelling/awd-lstm-lm>.
- Mikolov, Tomas et al. (2013). “Distributed Representations of Words and Phrases and Their Compositionality”. In: *Proceedings of the 26th International Conference on Neural Information Processing Systems. NIPS’13*. Lake Tahoe, Nevada: Curran Associates Inc., pp. 3111–3119. URL: <http://dl.acm.org/citation.cfm?id=2999792.2999959>.
- Schwartz, Lane, Coleman Haley, et al. (2019–2020). *iiksiin*. URL: <https://github.com/neural-polysynthetic-language-modelling/iiksiin>.
- Schwartz, Lane, Francis Tyers, et al. (2020). *Neural Polysynthetic Language Modelling*. arXiv: 2005.05477 [cs.CL].
- Serre, Jean-Pierre (1977). *Linear Representations of Finite Groups*. Trans. by Leonard L. Scott. Springer-Verlag New York, Inc.
- Smolensky, Paul (1990). “Tensor Product Variable Binding and the Representation of Symbolic Structures in Connectionist Systems”. In: *Artificial Intelligence*, pp. 159–216.
- Tyers, Francis (n.d.). *chukchicorpus*. URL: https://gitlab.com/itml_lab/UD_Chukchi-Anguema.
- Yang, Jie and Yue Zhang (2018). “NCRF++: An Open-source Neural Sequence Labeling Toolkit”. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*. URL: <http://aclweb.org/anthology/P18-4013>.