Production Systems and Information Engineering
Volume 10 (2), pp. 27–36
doi: 10.32968/psaie.2022.2.3.

PERFORMANCE ANALYSIS OF LOW DIMENSIONAL WORD EMBEDDINGS TO SUPPORT GREEN COMPUTING

László Csépányi-Fürjes
University of Miskolc, Hungary
Department of Information Engineering
laszlo.csepanyi-furjes@uni-miskolc.hu

Abstract. It has become increasingly important to pay attention how much energy we use to operate various *Artificial Intelligence* (AI) and *Machine Learning* (ML) systems. In order to implement environmentally responsible solutions we need to reconsider our used storage resources and computational power. Training a natural language model is a time and energy demanding process. In recent years the language models are becoming extremely large and the trend is growing. The building process of these models are consuming an extremely large amount of computational power hence these demands huge amounts of energy. In our research we trained and evaluated low dimensional word2vec embedding models and analyzed their performance on building transition based dependency parsers to show that low dimensional models are still competitive and in many use cases may be sufficient.

Keywords: green computing, word2vec, transition based dependency parsing

1. Introduction

In recent years, there has been a growing need to provide more energy-efficient solutions to humanity in all different areas, computer science included. Green computing is aiming to examine the possibilities of reducing the environmental impact of computer technology [1]. In human-machine interaction the so called Natural Language Understanding (NLU) process plays a key role. To be able to implement AI based information systems we need to provide a solution that supports free communication with the human agent, using the least possible formalism. NLU methods among others are helping the AI agent to identify the intent of the human. Identifying the intent of a customer is crucial when implementing Robotic Process Automation (RPA) systems for heavily loaded customer services. Dependency parsing is a dependency grammar based method that discovers grammatical relations between words of a sentence. By recognizing these grammatical relations better NLU modules can be built. In this paper we analyze two groups of transition based dependency parsing algorithms namely stack based and list based [2]. According to the latest trends these algorithms are using high dimensional word embedding to train ML classifiers. This process demands extremely high computational power with a huge energy footprint. In our study we built low dimensional embeddings and examined the impact of the dimension on the dependency parsing Accuracy and Unlabelled Attachment Score (UAS).

2. Related work

Transition based dependency parsing algorithms are widely used and extensively explored in the field of Natural Language Processing (NLP) [3, 4, 5]. The majority of the research activities has been aiming to increase the accuracy of the algorithms and much less research was done with the aim of providing less energy consuming variants. Dependency parsing algorithms are highly dependent on word embedding and language models that are replacing the previously used techniques in recent years. The state-of-the-art language models are based on more sophisticated variants of word embeddings, that are using contextualization, like the most advance transformer models [6].

Zadeh et al. present a quantization technique that compresses the 32-bit parameters of BERT models to 3-bit. Their solution promises to keep the Accuracy the same level as before the compression [7]. Maronikolakis & Schütze suggest training the language model on a multiple domain setup to be able to save the time and energy spent on training different models in different domains [8]. Smalheiser et al. suggest using low dimensional near-comprehensive vector representation of words. They have used this approach to create word-word and text-text similarity metrics [9]. Schick & Schütze present a method that produces the same performance as GPT-3 using a significantly smaller parameter count model. Their solution emphasizes the task preparation phase where they are converting textual inputs into "cloze questions" [10].

3. The experiment

We first present the implemented transition based dependency parsing system and the algorithm variants. Then we describe the training and evaluation processes with the used parameters. Finally we show the results of the experiment.

3.1. The implemented system

At the beginning of the experiment we generated word2vec embedding models according to the dimension configurations:

$$[5, 10, 20, 30, 40, 50]$$

We used the org.deeplearning4j.models.word2vec.Word2Vec class from the Deeplearning4J library to generate the models. Fig. 1 shows the configuration of the Word2Vec class.

```
Word2Vec word2Vec = new Word2Vec.Builder()
    .minWordFrequency(minWordFrequency)
    .epochs(epochs)
    .layerSize(layerSize)
    .seed(42)
    .windowSize(windowSize)
    .iterate(iterator)
    .tokenizerFactory(tokenizerFactory)
    .build();
```

Figure 1. Word2Vec model configuration

In order to examine the performance of the trained word2vec models we implemented four transition based dependency parser algorithms in Java. These algorithms are the following: *Arc-Standard Stack Based (ASSB)*, *Arc-Eager Stack Based (AESB)*, *Non-Projective List Based (NPLB)* and *Projective List Based (PLB)*. We trained an LSTM neural network model from the Deeplearning4J Java library to predict the transitions of the mentioned dependency parser algorithms. The configuration of the LSTM neural network can be studied in Fig. 2.

The training process in our implementation consists of two main phases. The first one is the pre-training phase when the needed training and evaluation files are getting produced and the training phase when the neural network is being trained.

The pre-training phase is an iterative process in which the system is in different transition states. All information about the actual state is stored in the state object which serves as input for the oracle. Using the state object the oracle produces the gold transition and inserts the transition information into the training-transition file. Also all the features of the actual state are translated into embedded format and the embedded features are getting concatenated and inserted into the training-feature file. Once the training files are ready, the pre-training phase is done.

```
MultiLayerConfiguration lstm = new NeuralNetConfiguration
.Builder()
    .seed(123)
    .optimizationAlgo(OptimizationAlgorithm
        .STOCHASTIC_GRADIENT_DESCENT)
    .weightInit(WeightInit.XAVIER)
    .updater(new AdaGrad.Builder().learningRate(0.01D).build())
    .gradientNormalization(GradientNormalization
        .ClipElementWiseAbsoluteValue)
    .gradientNormalizationThreshold(0.5)
    .dropOut(0.5D)
    .list()
        .layer(0, new LSTM.Builder()
            .activation(Activation.TANH)
            .nIn(numInputs)
            .nOut(numHiddenNodes)
            .build())
        .layer(1, new RnnOutputLayer.Builder(LossFunctions
                .LossFunction.MCXENT)
            .activation(Activation.SOFTMAX)
            .nIn(numHiddenNodes)
            .nOut(numOutputs)
            .build())
    .backpropType(BackpropType.TruncatedBPTT)
    .tBPTTLength(30)
    .build();
```

Figure 2. LSTM model configuration

In the training phase the aforementioned training files are used to train the LSTM neural network model. The training-features are the input and the training-transition is the expected output of the neural network. Since the process is sequential the actual transition depends highly on the previous state's transition. This is why a recurrent neural network can be considered effective in this situation.

In the stack-based algorithm variants (*ArcStandard*, *ArcEager*) the state object contains a stack σ and an input buffer β that holds the appropriate tokens w_i. A token object includes a word of the sentence and its properties. The state object also contains the set of already calculated dependency edges, the list of transitions and the state-characteristic vector Fig. 3.

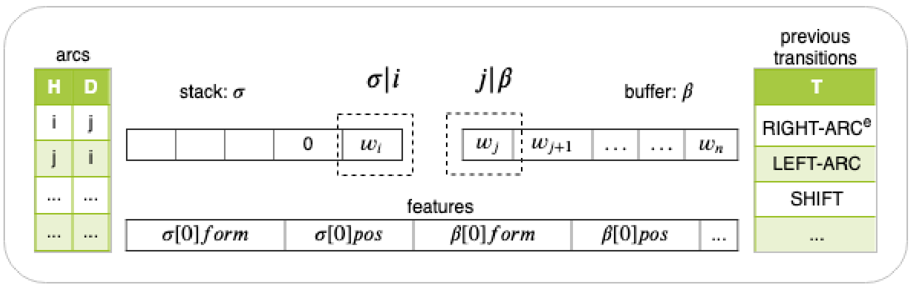

Figure 3. Stack based state object

The status object in the list-based variants (*NonProjectiveList*, *ProjectiveList*) is slightly different. In these variants, we store the token objects that have not yet been fully processed by the algorithm in two lists. The right headed list is the so called main list $\lambda1$, while the left headed list is a temporary list $\lambda2$. This state object also contains a buffer β, a set of already calculated dependency edges, a list of transitions, and also contains the state-characteristic vector Fig 4.

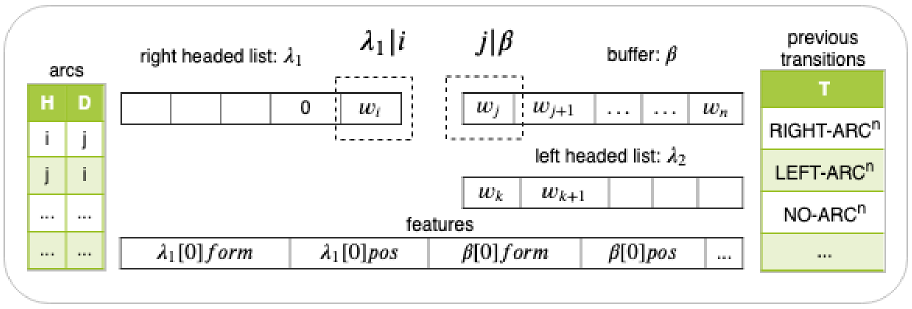

Figure 4. List based state object

In each iteration we create a new state object from the previous state and archive the previous ones. This process continues until the end state is reached, which means that the input buffer β is empty and there is no token waiting to be processed.

One of the key components of the training system is the oracle that produces the gold transition from the annotated text corpus Fig. 5.

3.2. Experiments and analysis

For the experiments we used an excerpt from the Hungarian Szeged Dependency Treebank (SZDT) corpus, which is the most significant Hungarian text

```
protected Transition getGoldTransition(final State state) {
    final Token stackTopToken =
        state.getTokenStack().getStackTopToken();
    final Token bufferHeadToken =
        state.getTokenBuffer().getBufferHeadToken();
    if (!stackTopToken.isRoot() &&
        stackTopToken.isDependantOf(bufferHeadToken)) {
        return ArcStandardTransition
            .LEFT(stackTopToken.getDeprel());
    }
    else if (bufferHeadToken.isDependantOf(stackTopToken) &&
        !state
            .getTokenBuffer()
            .bufferHeadHasUnprocessedDependants()) {
        return ArcStandardTransition
            .RIGHT(bufferHeadToken.getDeprel());
    }
    return ArcStandardTransition.SHIFT();
}
```

Figure 5. ArcStandard oracle

corpus [11]. This corpus is a large annotated dataset that contains examples of a number of linguistic phenomena, dependency relations included. We used 6817 training and 758 test sentences from the SZDT corpus.

Table 1. NN input vector size

Dimension	**Stack based variants**	**List based variants**
5	180	230
10	360	460
20	720	920
30	1080	1380
40	1440	1840
50	1800	2300

In our experiment we used the standard feature set that altogether consists of 36 predefined features in the stack based variants and 46 features in the list based variants. These features include words (FORM) and Part of Speech tags (POS) in combinations that are described in Nivre's paper [2]. The experiment focuses on the dimension of the embedding vectors, Fig. 6. We started with dimension 5 and trained our models in 6 steps until we reached dimension 50 (Table 1).

```
formEmbedding = Embedding{
    embeddingType = WORD2VEC,
    id = 'form_hu_szeged_7000_5_1',
    dimension = 5,
    minWordFrequency = 1,
    epochs = 1,
    windowSize = 10
}
posEmbedding = Embedding{
    embeddingType = WORD2VEC,
    id = 'pos_hu_szeged_7000_5_1',
    dimension = 5,
    minWordFrequency = 1,
    epochs = 1,
    windowSize = 10
}
```

Figure 6. Embedding configurations

3.3. Results

It is observable in Fig. 7 that the Accuracy starts increasing when we calculate higher dimension embedding vectors. At a certain level, around 30 the Accuracy value is stabilizing and stops raising any further significantly, Table 2. This phenomenon is observable for all studied algorithm variants.

Table 2. Accuracy

	DIM 5	**DIM 10**	**DIM 20**	**DIM 30**	**DIM 40**	**DIM 50**
ASSB	0.9257	0.9399	0.9470	0.9487	0.9505	0.9505
AESB	0.9227	0.9392	0.9455	0.9487	0.9487	0.9497
NPLB	0.9485	0.9590	0.9649	0.9670	0.9666	0.9666
PLB	0.9255	0.9399	0.9466	0.9490	0.9496	0.9509

A similar situation can be seen in the value of UAS as well, Table 3. Even though the UAS value is changing a bit more hectically, Fig. 8 suggests a similar conclusion as the Accuracy chart.

Reaching dimension 30 the LSTM classifier seems to be stabilizing and the success rate is not growing any further. Therefore it seems to be useless to

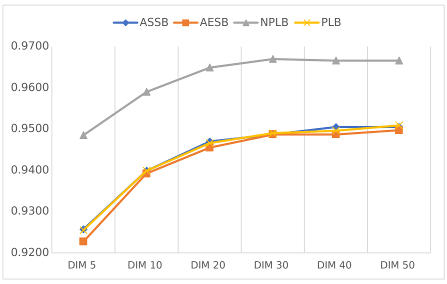

Figure 7. Accuracy

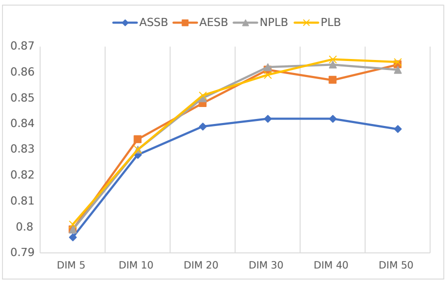

Figure 8. UAS

invest more computational power into the training process over this level of dimension.

Table 3. UAS

	DIM 5	DIM 10	DIM 20	DIM 30	DIM 40	DIM 50
ASSB	0.796	0.828	0.839	0.842	0.842	0.838
AESB	0.799	0.834	0.848	0.861	0.857	0.863
NPLB	0.799	0.830	0.850	0.862	0.863	0.861
PLB	0.801	0.830	0.851	0.859	0.865	0.864

Of course increasing the input dimension of the neural network increases the training time and energy as well. This can be observed in Fig 9.

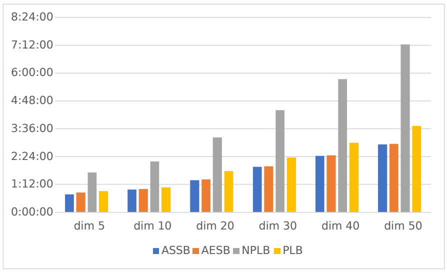

Figure 9. Training time

4. Conclusion and future work

We implemented a transition based dependency parser system that uses word2vec embeddings and an LSTM neural network classifier. The system includes four algorithm variants, two stack based and two list based ones. The aim of this study was to observe the effect of the embedding dimension change on the accuracy and UAS of the dependency parser. In order to save energy and to provide a "green" solution we wanted to identify an optimal embedding dimension size. This paper concludes that raising the embedding dimension over 30 produces very low increase in accuracy and UAS. At the same time the energy and time consumption of the training process raise dramatically. Keeping the dimension of the word embedding around 30 seems to be an optimal solution.

In the future it would be interesting to examine how the size of the training corpus affects the Accuracy and energy consumption. Is the embedding size or the NN input vector size causing the found effect of our experiment? What is the effect of the hidden layer size on the presented result? Also the quality of the input may affect the results which suggests to study the pre-processing phase of the NLP tasks further.

Acknowledgement. The described article was carried out as part of the 2020-1.1.2-PIACI-KFI-2020-00165 "ERPA - Development of Robotic Process Automation solution for heavily overloaded customer services" project implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the 2020–1.1.2-PIACI KFI funding scheme.

Furthermore, the GITDA (Governmental Information-Technology Development Agency, Hungary) is gratefully acknowledged for allocating computing resources used in this work.

References

[1] Amritpal, M., Ms, K., and Kaur, S.: Green computing: Emerging issues in it. *International Journal of Trend in Scientific Research and Development (IJTSRD)*, **3**, URL https://doi.org/10.31142/ijtsrd25311.

[2] Nivre, J.: Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, URL https://doi.org/10.1162/coli.07-056-R1-07-027.

[3] Chen, D. and Manning, C. D.: A fast and accurate dependency parser using neural networks. pp. 740–750, URL https://doi.org/10.3115/v1/D14-1082.

[4] Choi, J. D. and Palmer, M.: Getting the most out of transition-based dependency parsing. *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics (ACL '11): shortpapers*, **2**, (2011), 687–692, URL https://doi.org/10.5555/2002736.2002869.

[5] Gómez-Rodríguez, C., Shi, T., and Lee, L.: Global transition-based non-projective dependency parsing. *ACL 2018 - 56th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference (Long Papers)*, **1**, (2018), 2664–2675, URL https://doi.org/10.18653/v1/p18-1248.

[6] Devlin, J., Chang, M. W., Lee, K., and Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. *NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference*, URL https://doi.org/10.48550/arXiv.1810.04805.

[7] Zadeh, A. H., Edo, I., Awad, O. M., and Moshovos, A.: Gobo: Quantizing attention-based nlp models for low latency and energy efficient inference. *Proceedings of the Annual International Symposium on Microarchitecture, MICRO*, **2020-October**, (2020), 811–824, URL https://doi.org/10.1109/MICRO50266.2020.00071.

[8] Maronikolakis, A. and Schütze, H.: Multidomain pretrained language models for green nlp. *Adapt-NLP 2021 - 2nd Workshop on Domain Adaptation for NLP, Proceedings*.

[9] Smalheiser, N. R., Cohen, A. M., and Bonifield, G.: Unsupervised low-dimensional vector representations for words, phrases and text that are transparent, scalable, and produce similarity metrics that are not redundant with neural embeddings. *Journal of Biomedical Informatics*, **90**, URL https://doi.org/10.1016/j.jbi.2019.103096.

[10] Schick, T. and Schütze, H.: It's not just size that matters: Small language models are also few-shot learners. *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2339–2352, URL https://doi.org/10.18653/v1/2021.naacl-main.185.

[11] Vincze, V., Szauter, D., Almási, A., Móra, G., Alexin, Z., and Csirik, J.: Hungarian dependency treebank. *Development*, pp. 1855–1862, URL https://doi.org/10.1.1.890.1115.