Course Project: Dynamic Programming on Tensors for Solving the Problem of Dependency Parsing in NLP

Sergey Divakov*, Anastasia Koloskova†, Alfredo De la Fuente ‡ and Vladislav Pimanov§
Data Science Department, Skolkovo Institute of Science and Technology
Moscow, Russia
Email: *Sergei.Divakov@skoltech.ru, †Anastasiia.Koloskova@skoltech.ru,
‡Alfredo.DeLaFuente@skoltech.ru, §Vladislav.Pimanov@skoltech.ru

Abstract—**As part of the Optimization Methods - Numerical Linear Algebra courses final project, we explore the benefits of using tensor decomposition in order to reduce the asymptotically complexity of the well-known dynamic programming algorithm Inside-Outside (marginal probabilities computation) to infer PCFG grammars and improve the performance at parsing tasks. We benchmark our parsing quality with appropriate metric as well as the implementation speed using the treebank datasets available. We observe significant improvements by incorporating the proposed methodology.**

Keywords*—*Tensor decomposition, dynamic programming, NLP.

I. Introduction

Syntactic dependency parsing is one of the major problems in statistical natural language processing. There exists many algorithms to solve this problem, but we will focus on Probabilistic Contex Free Grammars (PCFG) with latent states, which is one of the most successful and mathematically interesting approaches.

Latent variable models have shown promising results in different areas in computational linguistics, bioinformatics and machine learning. Many different approaches have been recently suggested to solve the syntactic parsing task (spectral learning [4], recursive neural networks [5]), however regardless of the learning algorithm, the inference always relies on the inside-outside algorithm, which is a dynamic programming algorithm, allowing to calculate the marginal probabilities for spans of the parse tree.

The Inside-Outside algorithm has cubic complexity in the cardinality of non-terminal symbols. There is, however, a way to reformulate this problem in terms of dynamic programming on a tensor, with each step corresponding to some contraction. A relatively recent paper [1] uses this approach and applies the canonical tensor decomposition to reduce the complexity of the inference to linear in the number of non-terminals, with only a slight decrease in quality. Our goal is to try to improve on this result by testing different methods for tensor decomposition (Tucker and Tensor Train) and evaluate the quality of approximation both in terms of speed and precision.

In our project we will provide some basic notions regarding PCFGs and tensor decomposition methods; in order to implement the algorithm, perform different experiments and evaluate results.

II. Setup

For an integer $n \geq 1$, we let $[n] = \{1, 2, \ldots, n\}$.

A. Probabilistic Context-Free Grammars

We consider a probabilistic context-free grammar (PCFG) G in Chomsky normal form, which can be defined by a quintuple

$$G = (\mathcal{N}, \mathcal{L}, \mathcal{R}, \mathcal{P}, \pi), \tag{1}$$

where

1) $\mathcal{N}$ is the finite set of nonterminal symbols, $|\mathcal{N}| = m$.
2) $\mathcal{L}$ is the finite set of words (lexical tokens), $|\mathcal{L}| = n$.
3) $\mathcal{R}$ is a set of rules: $a \to bc$ or $a \to x$, where $a, b, c \in \mathcal{N}$, $x \in \mathcal{L}$.
4) $\mathcal{P}$ is a transition probability distribution: for each $(a \to x) \in \mathcal{R}$ and $a \to bc \in \mathcal{R}$ we have a parameter $p(a \to bc|a)$ and $p(a \to x|a)$.
5) π is an probability distribution for non-terminal symbols. For each $a \in \mathcal{N}$ we have π_a which is a probability of a being the root symbol of a derivation.

The parameters above satisfy the following normalization conditions:

$$\sum_{(a \to bc) \in \mathcal{R}} p(a \to bc|a) + \sum_{(a \to x) \in \mathcal{R}} p(a \to x|a) = 1,$$

for each $a \in \mathcal{N}$, and $\sum_{a \in \mathcal{N}} \pi_a = 1$.

B. Tensor

Definition 1 (Tensor). *A Tensor $T \in \mathbb{R}^{m \times m \times m}$ is a set of m^3 parameters $T_{i,j,k}$ for $i, j, k \in [m]$.*

Definition 2 (Tensor Contraction). *Given a tensor $T \in \mathbb{R}^{m \times m \times m}$ and vectors $v^1, v^2 \in \mathbb{R}^m$ we define a tensor contraction $T_{(1,2)} \in \mathbb{R}^{m \times m \times m}$ which is the function $T_{(1,2)} : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}^{1 \times m}$ defined as $T(v_1, v_2)_k = \sum_{i,j} T_{ijk} v_i^1 v_j^2$. Similarly, contractions $T_{(1,3)}$ and $T_{(2,3)}$ are defined.*

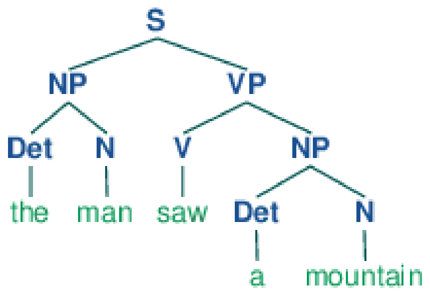

Fig. 1. Parse tree representation of *the man saw a mountain*

We use tensor representation of transition probabilities in grammar G and we denote by $T \in \mathbb{R}^{m\times m\times m}$ a tensor such that

$$T_{a,b,c} = p(a \to bc|a).$$

Similarly, we denote by $Q \in \mathbb{R}^{m\times n}$ a matrix such that

$$Q_{a,x} = p(a \to x|a).$$

Also, we introduce probabilities of each non-terminal to be a root, we denote it using a vector $\pi \in \mathbb{R}^{m\times 1}$ such that π_a is a probability of a to be a root.

C. *Parsing*

Parsing is the process of taking a string and a grammar and find a derivation from start category S for the input string, in other words, assigning proper trees to input strings based on the rules established by the grammar.

We must notice than often a sentence can give us an ambiguous parse, then how do we choose the best tree among all those returned? To solve this issue, we can assign probability values (weights) to the set of rules (PCFG) and rank parses by combining their weights. These probabilities can be learned from an annotated corpus (treebank) by using the inside-outside algorithm.

Definition 3 (Parse Tree)**.** *Let $z = x_1 \dots x_N$ be an input sentence. We denote by τ a parse tree of given sentence. The root of τ is S, leaves are terminal symbols $x_1 \dots x_N$. If tree node A has two children B and C then the rule $A \to BC$ is present in our grammar G, i.e. $A \to BC \in \mathcal{R}$ (see fig. 1).*

The set of all possible trees τ for given input string z we denote as $\mathcal{T}(z)$.

III. Parsing Algorithm

A. *Minimum Bayes-Risk Decoding*

According to previous definition, parsing of string z is choosing the "best" parsing tree τ from all possible parsing trees $\mathcal{T}(z)$. In our project parsing aims to find the highest scoring tree τ^* for z according to the underlying PCFG, also called "Viterbi parse":

$$\tau^* = \arg\max_{\tau\in\mathcal{T}(z)} p(\tau). \tag{2}$$

Goodman suggested dynamic programming approach, called "Labelled Recall Algorithm" [2], which aims to find the highest scoring tree (2).

This algorithm has two phases. At the first phase it computes marginals which are defined as

$$\mu(a,i,j) = \sum_{\tau\in\mathcal{T}(z):(a,i,j)\in\tau} p(\tau),$$

where $(a,i,j) \in \tau$ denotes that nonterminal a spans words $x_i \dots x_j$ in the parse tree τ.

The second phase includes a dynamic programming algorithm which finds a tree τ^* that maximized the sum over marginals in that tree:

$$\tau^* = \arg\max_{\tau\in\mathcal{T}(z)} \sum_{(a,i,j)\in\tau} \mu(a,i,j).$$

Goodmans algorithm is described in Algorithm 1.

Algorithm 1 Labelled Recall Algorithm

```
**Inputs:** Sentence $x_1 \dots x_N$, PCFG$(\mathcal{N},\mathcal{L},\mathcal{R})$, parameters $T \in \mathbb{R}^{m\times m\times m}$, $Q \in \mathbb{R}^{m\times m}$, $\pi \in \mathbb{R}^{m\times 1}$.
**Marginals:** $\forall a \in \mathcal{N}$, $\forall i,j \in [N], i \le j$ compute the marginals $\mu(a,i,j)$ using the inside-outside algorithm.
**Base case:** $\forall i \in [N]$,
    $\gamma^{i,i} = \max_{(a\to x_i)\in\mathcal{R}} \mu(a,i,i).$
**Maximize Labelled Recall:** $\forall i,j \in [N], i < j$,
    $\gamma^{i,j} = \max_{a\in\mathcal{N}} \mu(a,i,j) + \max_{i\le k<j} (\gamma^{i,k} + \gamma^{k+1,j}).$
```

B. *Inside-Outside Algorithm*

The representation of the inside-outside algorithm in tensor form for calculation of marginal terms $\mu(a,i,j)$.

Algorithm 2 Inside-Outside Algorithm

```
**Inputs:** Sentence $x_1 \dots x_N$, PCFG$(\mathcal{N},\mathcal{L},\mathcal{R})$, parameters $T \in \mathbb{R}^{m\times m\times m}$, $Q \in \mathbb{R}^{m\times m}$, $\pi \in \mathbb{R}^{m\times 1}$.
**Data Structures:**
    • $\alpha^{i,j} \in \mathbb{R}^{1\times m}$, $i,j \in [N]$, $i \le j$, is a row vector representing inside terms ranging over $a \in \mathcal{N}$.
    • $\beta^{i,j} \in \mathbb{R}^{m\times 1}$, $i,j \in [N]$, $i \le j$, is a column vector representing outside terms ranging over $a \in \mathcal{N}$.
**Inside Base Case:** $\forall i \in [N], \forall (a \to x_i) \in \mathcal{R}$:
$[\alpha^{i,i}]_a = Q_{a,x}$
**Inside Recursion:** $\forall i,j \in [N]$, $i < j$:
$[\alpha^{i,j}]_a = \sum_{k=i}^{j-1} T_{(2,3)}(\alpha^{i,k}, \alpha^{k+1,j})$
**Outside Base Case:** $\forall a \in \mathcal{N}$:
$[\beta^{1,N}]_a = \pi_a$
**Outside Recursion:** $\forall i,j \in [N]$, $i \le j$:
$\beta^{i,j} = \sum_{k=1}^{i-1} T_{(1,2)}(\beta^{k,j}, \alpha^{k,i-1}$
$+ \sum_{k=j+1}^{N} T_{(1,3)}(\beta^{i,k}, \alpha^{j+1,k})$
**Marginals:** $\forall a \in \mathcal{N}$, $\forall i,j \in [N]$, $i \le j$:
$\mu(a,i,j) = [\alpha^{i,j}]_a \cdot [\beta^{i,j}]_a$
```

Time complexity of the inside-outside algorithm is $\mathcal{O}(m^3N^3)$, since each naive tensor contraction takes time $\mathcal{O}(m^3)$. To reduce the complexity we approximate parsing using different tensor decompostions.

IV. Tensor Decompositions

In this section we provide an overview of three tensor decomposition methods: canonical decomposition, Tucker decomposition and Tensor Train decomposition. Using any of these decompositions we can multiply tensor-by-vectors is linear in m time instead of cubic complexity for naive multiplication.

A. Canonical Decomposition

The canonical polyadic decomposition (CPD), also known as tensor rank decomposition, is basically a generalization of SVD to tensors. In that way, we can represent a order d tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times \ldots n_d}$ as a linear combination of r pure rank tensors as follows,

$$\mathcal{A} = \sum_{i=1}^{r} \lambda_i \mathbf{a}_i^1 \otimes \mathbf{a}_i^2 \otimes \ldots \mathbf{a}_i^d$$

where $\lambda_i \in \mathbb{R}$ and $\mathbf{a}_i^j \in \mathbb{R}^{n_j}$. In practice this decomposition is computed by using different optimization methods such as Levenber - Marquardt (LM), nonlinear conjugate gradient (NCG) or Broyden-Fletcher-Goldfarb-Shanno (BFGS); or by using alternating least squares (ALS).

Remark 4 (Complexity of Tensor Contraction)**.** *Let $\mathcal{A} \in \mathbb{R}^{m \times m \times m}$ be a 3-dimensional tensor represented in the canonical decomposition form, $v_1, v_2 \in \mathbb{R}^m$. The complexity of computing contraction of $\mathcal{A}$ and v_1, v_2 is $\mathcal{O}(rm)$.*

B. Tucker Decomposition

Tucker decomposition decomposes a tensor into a set of matrices and one small core tensor. That is, for a tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$, its Tucker decomposition is given by

$$\text{vec}(\mathcal{A}) = (W \otimes V \otimes U) \cdot \text{vec}(\mathcal{C})$$

where, vec is the vectorization operator, with $U \in \mathbb{R}^{n_1 \times r_1}$, $V \in \mathbb{R}^{n_2 \times r_2}$, $W \in \mathbb{R}^{n_3 \times r_3}$ and the core tensor $\mathcal{C} \in \mathbb{R}^{r_1 \times r_2 \times r_3}$.

Remark 5 (Complexity of Tensor Contraction)**.** *Let $\mathcal{A} \in \mathbb{R}^{m \times m \times m}$ be a 3-dimensional tensor in the Tucker decomposition form, $v_1, v_2 \in \mathbb{R}^m$. The complexity of computing contraction of $\mathcal{A}$ and v_1, v_2 is $\mathcal{O}(rm + r^3)$.*

C. Tensor Train Decomposition

Tensor Train (TT) decomposition is a representation of d-tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times \ldots n_d}$ in the following way:

$$\mathcal{A}(i_1, \ldots, i_d) = \\ \sum g_1(\alpha_0, i_1, \alpha_1) g_2(\alpha_1, i_2, \alpha_2) \times \ldots \times \\ g_{d-1}(\alpha_{d-2}, i_{d-1}, \alpha_{d-1}) g_d(\alpha_{d-1}, i_d, \alpha_d)$$

where summation is on the indices α_k, which are repeated exactly twice. Let α_k varies from 1 to r_k, then $r_1, \ldots, r_{d-1}$ are ranks of TT decomposition.

Remark 6 (Complexity of Tensor Contraction)**.** *Let $\mathcal{A} \in \mathbb{R}^{m \times m \times m}$ be a 3-dimensional tensor in the TT form, $v_1, v_2 \in \mathbb{R}^m$. The complexity of computing contraction of $\mathcal{A}$ and v_1, v_2 is $\mathcal{O}(mr^2)$.*

V. Methodology

The full implementation of the project is open source available in the following GitHub repository. All the scripts and notebooks reproducing the results can also be found there.

VI. Experiments

A. Data

We used a section of the Penn TreeBank (PTB) [3] obtained from Wall Street Journal (WSJ). The Natural Language Toolkit (NLTK) let us access to a 5% fragment of the PTB corpus (about 1,600 sentences), which we used to familiarize with the task and train simple latent-variable parsing model using the expectation-maximization algorithm. In order to test our methodology, we considered the Multilingual L-PCFG Models dataset. We focused our analysis on the French grammar section to run our experiments.

B. Evaluation

The most common parsing evaluation metrics examine the conceptual ”distance” between the candidate parse generated by the parser, and the correctly annotated solution (the *gold standard*). Gold standards are sometimes annotated by hand, but more often they are inferred from existing corpora.

The current standard comparison is the ParseEval metric [7]. It defines a set of values that focus primarily on the constituency differences between the two trees. The idea of this metric is straightforward: measure the *exact match*, i.e. proportion of gold standard trees that parser got right.

ParseEval metric computes two values: **precision** and **recall**. To compute precision it finds proportion of constituents in *parse tree* which are also present in *gold tree*. To compute recall it finds proportion of constituents in *gold tree* which are also present in *parse tree*.

You can see example on the fig. 2

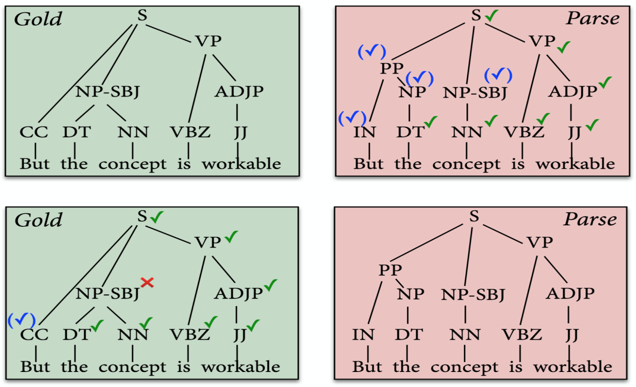

Fig. 2. Computing Precision (top) and Recall (bottom).

After computing precision p and recall r, we can easily compute the f_1 score by

$$f_1 = \frac{2\ p\ r}{p + r},$$

C. Results

We performed a sequence of tests to benchmark, with respect to time and accuracy, our proposed methodology in comparison to the base line (inside outside with no tensors). We begin by comparing both Tucker and Tensor Train decomposition algorithms in the approximating syntactic parse task. We run both approaches fixing the ranks of the tensor decomposition to be 10, 50 and 100. We observe the results plotted in figures 3, 4, 5 and 6.

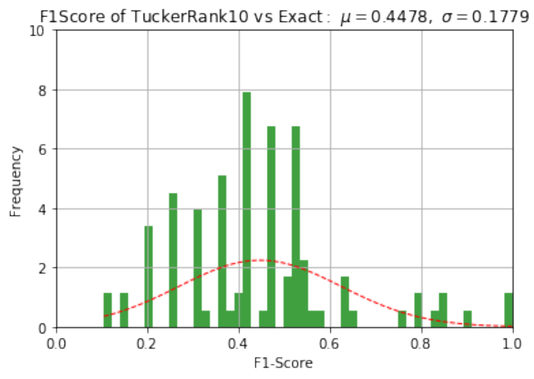

Fig. 3. Distribution of F1 scores for rank 10 - Tucker decomposition

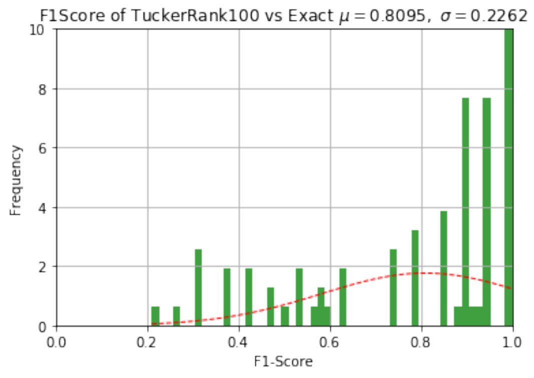

Fig. 4. Distribution of F1 scores for rank 100 - Tucker decomposition

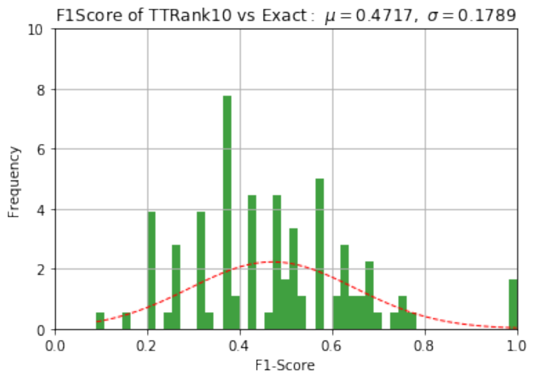

Fig. 5. Distribution of F1 scores for rank 10 - Tensor Train decomposition

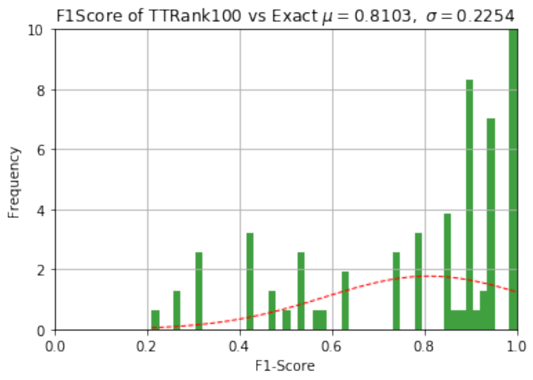

Fig. 6. Distribution of F1 scores for rank 100 - Tensor Train decomposition

We can observe a slightly better performance of Tensor Train decomposition approach compared to Tucker, however both seem to significantly obtain a remarkable F1-score performance for parsing the sentences in our french dataset considered. We obtained for rank 100 an average F1-score of 0.8095 for Tucker decomposition and 0.8103 for Tensor Train decomposition. For TT decompositions with rank over 150 the F1-score is equal to 1.0, meaning that the parsing is exact, however we still observe a significant speed improvement for such ranks.

In addition, we run several experiments to find how time complexity of our methodology relates to the size of the rank we will use to approximate our tensor. In fig. 7 we observe a substantial improvement in time performance. The gap in performance time maintains for different rank approximations used, which proves the main hypothesis of our research, without compromising accuracy.

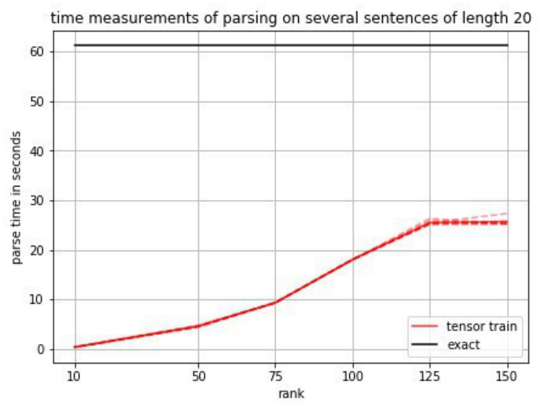

Fig. 7. Time benchmark - Tensor Train decomposition vs baseline method.

VII. Conclusion

We described and implemented a tensor decomposition approach for probabilistic context-free parsing. The scheme presented leads to a speed-up in PCFG parsing with minimal loss in accuracy using ParseEval metric.

Acknowledgment

The authors would like to thank the guidance and patience of the teaching assistants assigned to contribute in our group.

Contribution

Sergey Divakov
Idea, architecture and uniting all parts of the parser code. Implementing Labelled Recall Algorithm and Inside-Outside algorithm with tensor train. Also implemented em-algorithm for training the pcfgs. Conducting numerical experiments, contribution to the presentation and report (content for numerical results section).

Anastasia Koloskova
Writing helper functions for parser (in algorithmic part and for converting algorithm output to convenient format to get ParseEval scores). Worked with theory (complexity of tensor contractions, etc.). Writing the report and creating presentation.

Alfredo De la Fuente
The development of jupyter notebook for the use of functions to deal with PCFGs by using NLTK libraries tools, for the development of functions to measure ParseEval metric that was used in the accuracy tests and the implementation of routines to work with Penn Treebank dataset. In addition, active participation in writing distinct sections of the project report.

Vladislav Pimanov
Working with theory of tensors decompositions. Implementing effective tensor contractions using tensor decompositions for the inside-outside algorithm speed-up. Writing python3 wrappers for the tensor decompositions. Working with scikit tensor open source python library. The canonical polyadic tensor decomposition (also known as CANDECOMP/PARAFAC, same as in the underlying paper, which we based on) and Tucker decompositions were applied. Also ttpy package was used for the Tensor Train decomposition.

References

[1] S. B. Cohen and M. Collins., *Tensor decomposition for fast parsing with latent-variable PCFGs*, 2012, In Proceedings of NIPS.

[2] J. Goodman., *Parsing algorithms and metrics*, 1996, In Proceedings of ACL.

[3] M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz., *Building a large annotated corpus of English: The Penn treebank.*, 1993, Computational Linguistics 19:313330.

[4] S. Narayan and S. B. Cohen, *Optimizing Spectral Learning for Parsing.*, 2016, In Proceedings of ACL.

[5] R. Socher, C. D. Manning, and A. Y. Ng. , *Learning continuous phrase representations and syntactic parsing with recursive neural networks.*, 2010, In Proceedings of the NIPS Deep Learning and Unsupervised Feature Learning Workshop.

[6] G. Strang. , *Introduction to linear algebra - volume 3*, 1993, Wellesley-Cambridge Press Wellesley, MA.

[7] Black, D.S., A.J. Kelly, M.J. Mardis, H.S. Moyed 1991. *Structure and organization of hip, an operon that affects lethality due to inhibition of peptidoglycan or DNA synthesis.* J.Bacteriol.