Perplexity of n-Gram and Dependency Language Models*

Martin Popel and David Mareček

Charles University in Prague, Institute of Formal and Applied Linguistics
{popel,marecek}@ufal.mff.cuni.cz

Abstract. Language models (LMs) are essential components of many applications such as speech recognition or machine translation. LMs factorize the probability of a string of words into a product of $P(w_i|\mathbf{h}_i)$, where $\mathbf{h}_i$ is the context (history) of word w_i. Most LMs use previous words as the context. The paper presents two alternative approaches: *post-ngram LMs* (which use following words as context) and *dependency LMs* (which exploit dependency structure of a sentence and can use e.g. the governing word as context). Dependency LMs could be useful whenever a topology of a dependency tree is available, but its lexical labels are unknown, e.g. in tree-to-tree machine translation. In comparison with baseline interpolated trigram LM both of the approaches achieve significantly lower perplexity for all seven tested languages (Arabic, Catalan, Czech, English, Hungarian, Italian, Turkish).

1 Introduction

Language models (LMs) are essential components of many applications such as speech recognition or machine translation (MT). LM is a statistical model that assigns a probability to every sentence s which is represented as a sequence of m words, i.e. $P(s) = P(w_1, \ldots, w_m)$. LMs factorize this joint probability into a product of conditional probabilities in form $P(w_i|\mathbf{h}_i)$, where h_i is the context (traditionally called history) of word w_i. Most LMs (e.g. standard n-gram LMs, maximum entropy LMs [1], factored LMs [2] and even some grammar-based LMs [3]) use previous words as the context, so the joint probability can be computed in a left-to-right manner:

$$P(s) = \prod_{i=1\ldots m} P(w_i|w_1, \ldots, w_{i-1}). \tag{1}$$

N-gram LMs consider only the last $n-1$ words using so-called $(n-1)^{\text{th}}$ order Markov property, i.e. $\mathbf{h}_i \equiv (w_{i-n}, \ldots, w_{i-1})$; [1]

$$P_{ngram}(s) = \prod_{i=1\ldots m} P(w_i|\mathbf{h}_i) = \prod_{i=1\ldots m} P(w_i|w_{i-n}, \ldots, w_{i-1}). \tag{2}$$

However, we can use other factorization orderings instead of left-to-right. For example, in right-to-left ordering we use so-called *post-ngrams* (post-bigram, post-trigram, etc.) as the context, i.e. $\mathbf{h}_i \equiv (w_{i+1}, \ldots, w_{i+n-1})$;

* The work on this project was supported by the grants GAUK 116310, GA201/09/H057, FP7-ICT-2009-4-247762, and FP7-ICT-2007-3-231720.

[1] For simplicity, artificial start-of-sentence tokens are usually inserted before each sentence, in other words, for $i < 1$ we define $w_i \equiv$ <NONE>.

P. Sojka et al. (Eds.): TSD 2010, LNAI 6231, pp. 173–180, 2010.

$$P_{post\text{-}ngram}(s) = \prod_{i=1\ldots m} P(w_i|\mathbf{h}_i) = \prod_{i=1\ldots m} P(w_i|w_{i+1}, \ldots, w_{i+n-1}). \quad (3)$$

Generally, we can define a directed acyclic graph (DAG) on words of the given sentence, and define $\mathbf{h}_i \equiv (w_j : (j,i) \in Edges(DAG))$.[2]

The key topic of this paper is a comparison of LMs based on different DAGs.[3] We are particularly interested in DAGs which are based on dependency parsing; we call the resulting language models *dependency LMs*.

In Sect. 2, we briefly summarize related work on dependency LMs. Afterwards, we describe possible ways of designing dependency LMs (Sect. 3) and exploiting additional context attributes (Sect. 4). Experiments are reported in Sect. 5 and concluding discussion is given in Sect. 6.

2 Related Work on Dependency LMs

There are papers (e.g. [3,4]) that use the term *dependency language model* (with different meanings though), but we are not aware of any universal definition of the term. Nevertheless, the common idea behind the term is to use some kind of dependency trees (such as those used in CoNLL shared tasks [5]) for language modeling. Hereinafter, *parent* denotes the governing word (and the corresponding tree node), similarly *children* denotes the dependent words (modifiers).

There are many possible ways how to exploit dependency trees in dependency LMs. Chelba et al. [3] use them in conventional left-to-right factorization ordering – briefly, the context considered for a word comprises the preceding bigram and a *link stack*, which is a list of words that precede the current word, but their parent does not. Shen et al. [4] compute the probability of a tree (which represents the given sentence) using probabilistic distributions P_L and P_R for left and right side generative probabilities respectively and P_T for a probability of a word being the root. The probability of a word is conditioned by its parent and also siblings that lie between the word and the parent. See Fig. 1 and Formula 4 for illustration.

$$\begin{aligned} Prob = & P_L(the|boy\text{-as-parent}) \\ & \times P_L(boy|will,\ find\text{-as-parent}) \times P_L(will|find\text{-as-parent}) \\ & \times P_T(find) \\ & \times P_R(it|find\text{-as-parent}) \times P_R(interesting|it, find\text{-as-parent}) \end{aligned} \quad (4)$$

Most related to our experiments is a research by Charniak [6], who defines two LMs based on his immediate-head parser. The first LM is called *bihead* – the probability of a

[2] Performing the factorization in DAG's topological ordering ensures, we condition always only on words whose probabilities have been already computed. Note that n-gram LMs are a special case of this generalized DAG-based LM – $Edges(DAG_{ngram}) = \{(j,i) : j \in \{i-n+1, \ldots, i-1\}\}$. Likewise, the before-mentioned post-ngram LMs are a special case of DAG-based LM – $Edges(DAG_{post\text{-}ngram}) = \{(j,i) : j \in \{i+1, \ldots, i+n-1\}\}$.

[3] The quality of LMs is measured by *cross-entropy* $H = -(1/|T|)\sum_{i=1}^{|T|} log_2 P(w_i|\mathbf{h}_i)$, where T is test data. For convenience, we report *perplexity* ($PP = 2^H$), as it is usual in literature. Lower perplexity implies better LM.

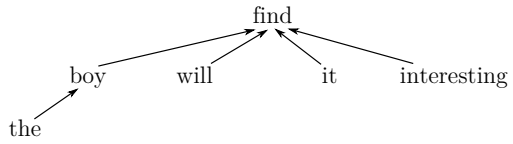

Fig. 1. Dependency tree of sentence "the boy will find it interesting"

word is conditioned by its parent, similarly to our model `wp` as it is defined in Sect. 3.1. The second LM is called *trihead* – it conditions by word's parent and grandparent, similarly to our model `wp,wq`. Charniak reports 22% improvement in perplexity over trigram LM.[4]

3 Designing Dependency LMs

In Sect. 1 we introduced DAG-based LMs. DAGs of dependency LMs can be constructed from the dependency tree of a given sentence s using several methods. Actually, we need just the topology T of the dependency tree.[5]

3.1 Model `wp` (Word Form of Parent)

The simplest method for constructing dependency LMs is to use T itself with edges directed from parent to child as the DAG, which means that each word is conditioned by its parent. For example, the probability of the sentence from Fig. 1 is factorized as follows: $P_{\texttt{wp}}(s|T) = P(the|boy)P(boy|find)P(will|find)$ $P(find|\texttt{<NONE>})P(it|find)P(interesting|find)$.

3.2 Model `wp,wg` (Word Forms of Parent and Grandparent)

Inspired by [6], we also define model in which each word is conditioned by word forms of its parent and grandparent. For example, in phrase "listen to news" we have $P(news|\text{parent} = to, \text{grandparent} = listen)$.

3.3 Other Possible Models

Both the models, `wp` and `wp,wg`, are to be applied in *bottom-up factorization ordering*. Alternatively, we could define models in which a word is conditioned by its children, so the models would be applied in *top-down factorization ordering*. Similarly to [4], we could enhance the models with conditioning also on siblings.

[4] Charniak reports perplexity of trigram LM = 167, perplexity of trihead LM = 130, but the absolute values are not comparable to our results, because he uses a special "speech-like" corpus with reduced vocabulary, see [6] for details.

[5] Formally, $T = (V, r, \rho, ord)$, where V are nodes, $r \in V$ is the root, $\rho : V \setminus \{r\} \to V$ is a function which assigns parent nodes and $ord \subset V \times V$ is total ordering of nodes.

4 Additional Context Information

All LMs in our experiments (in Sect. 5) are computed using distribution of form $P(w|\mathbf{h})$, so they can be described by the context $\mathbf{h} = h_1, \ldots, h_F$ they use. Performance of LMs can be improved by supplying additional context factors h_f that can be used either for enlarging the context or for better smoothing. In addition to using various word-positions (e.g. preceding word, parent,...) in the context, we can extract various attributes from each word-position (word form, POS tag, number of children,...). Every context factor h_f has form $[attribute][word\text{-}position]$. For example, context `tp,wp,tg` means POS tag of parent, word form of parent and POS tag of grandparent. Possible attributes and word-positions are:

attributes	
`w`	word form
`l`	lemma
`t`	POS tag
`T`	coarse-grained POS tag
`N`	the word is $\mathtt{N}^{th}$ child of its parent
`C`	number of children
`E`	edge direction (left or right)

word-positions	
`-1,-2,...`	preceding words (1^{st}, 2^{nd},...)
`+1,+2,...`	following words (1^{st}, 2^{nd},...)
`p`	parent
`g`	grandparent
default	current word (applicable only for attributes `N`,`C` and `E`)

Attributes `N` and `C` are quantized – possible values are 0,1,2,3 and `more`.

5 Experiments

5.1 Data

For experiments, we used the data from CoNLL 2007 shared task [5], and we choose following seven languages: Arabic (ar), Catalan (ca), Czech (cs), English (en), Hungarian (hu), Italian (it), and Turkish (tr). Properties of this data are summarized in Tab. 1. The data are divided into two parts: `train` which was used for training the LMs, and `test` which was used solely for computing perplexity.

5.2 Four Experimental Settings

We consider four experimental settings which correspond to resources that may be available when using LMs in real applications:

Table 1. CoNLL data statistics. (OOV = percentage of `test` words not seen in `train`)

language	ar	ca	cs	en	hu	it	tr
sentences (train)	2,912	14,957	25,363	18,576	6,033	3,109	5,634
tokens (train)	111,669	430,844	432,296	446,573	131,799	71,199	65,182
unique words (train)	21,058	35,213	63,151	26,599	33,754	13,003	18,181
sentences (test)	130	166	285	213	389	248	299
tokens (test)	5,124	5,016	4,724	5,003	7,344	5,095	4,513
OOV	11.7%	3.8%	10.2%	2.6%	22.1%	12.0%	26.0%

- PLAIN: no additional information available, just word forms (Sect. 5.4),
- TAGS: part-of-speech (POS) tags and lemmata available (Sect. 5.5),
- DEP: topology of dependency trees available (Sect. 5.6),
- DEP+TAGS: topology, POS tags and lemmata available (Sect. 5.7).

In Sections 5.4 – 5.7 we compare perplexity of several LMs conforming the given setting. Note that the perplexity values are not directly comparable across the languages, because of different training data size and domain.

5.3 Smoothing

We use linear interpolation of models:[6]

$$P_{smoothed}(w|h_1,...,h_F) = \lambda_0 P_0(w) + \lambda_1 P_{ML}(w) + \sum_{f=1...F} \lambda_{f+1} P_{ML}(w|h_1,...,h_f),$$

where P_{ML} is the maximum likelihood estimate and P_0 is so-called probability of unseen words. $P_0(w) = 0$ if word w was observed in training data and $P_0(w) = 1/(dictionary\_upper\_bound - train\_vocabulary\_size)$ otherwise (we set $dictionary\_upper\_bound = 10^6$). Weights $\lambda_0 ... \lambda_F$ are trained using EM algorithm, so that they sum to one. We used 20% of the training data as "held-out data" for estimating the lambda weights. Note that the probability of unseen words always obtains the same weight λ_0 for all models – it is the ratio of held-out words that were not seen in the rest of training data.

5.4 PLAIN: Just Word Forms

As expected, enlarging the context lowers perplexity, i.e. bigrams have higher perplexity than trigrams, post-bigrams than post-trigrams, etc. However, for the training data sizes used in our experiments the improvement for 4-grams compared to trigrams is negligible.

Most surprising outcome of Tab. 2 is that post-ngram LMs have significantly better perplexity than standard ngram LMs. The difference is so prominent, that for five of the seven languages it is even better to use one following word as the context than two preceding words and for all the languages it is better to use two following words than three preceding words.

5.5 TAGS: POS Tags and Lemmata Available

In the TAGS setting (Tab. 3), we still do not exploit dependency structure of sentences – we try to lower the perplexity as much as possible just by enriching the context with additional attributes of following words.[7] The attributes are: POS tag (`t`), coarse-grained POS tag (`T`) and lemma (`l`). We trained a naïve tagger on the `train` data,

[6] It would be beneficial to compare also other smoothing techniques (see [7] for an overview), especially Generalized Parallel Backoff [2], but it is beyond the scope of this paper.

[7] We performed experiments also with additional attributes of preceding words, but similarly to the PLAIN setting, preceding words gave higher perplexity than following words. For clarity and space reasons, we do not show the results in Tab. 3.

Table 2. Perplexity of PLAIN models

Model	Perplexity						
	ar	ca	cs	en	hu	it	tr
`w-1` (bigram)	2,052	368	3,632	387	5,203	1,606	4,034
`w+1` (post-bigram)	2,006	337	3,391	355	4,735	1,440	3,720
`w-1,w-2` (trigram)	1,988	325	3,530	356	5,183	1,552	4,015
`w+1,w+2` (post-trigram)	1,950	301	3,298	328	4,721	1,399	3,712
`w-1,w-2,w-3`	1,989	324	3,531	355	5,183	1,553	4,015
`w+1,w+2,w+3`	1,951	299	3,299	327	4,721	1,400	3,712

which assigns the most frequent (full/coarse-grained) POS tag for a given word or the overall most frequent tag if the word was unseen in training data. Similarly, we created a naïve lemmatizer and tagged and lemmatized the `test` data.

We confirm the well-known finding (see e.g. [2]) that additional attributes improve the perplexity. In our experiments, for six of the seven languages it is even better to use one following word & its POS tag than two following words. In Tab. 3 we can see that adding coarse-grained POS tags to POS tags helps a little and adding lemmata helps only for some languages (it helps for morphologically rich languages such as Czech and Hungarian).[8]

Table 3. Perplexity of TAGS models

Model	Perplexity						
	ar	ca	cs	en	hu	it	tr
`w-1,w-2` (trigram baseline)	1,988	325	3,530	356	5,183	1,552	4,015
`t+1,w+1`	1,706	310	2,999	316	4,091	1,277	3,346
`t+1,w+1,t+2,w+2`	1,641	276	2,909	287	4,067	1,246	3,340
`T+1,t+1,w+1,T+2,t+2,w+2`	1,641	271	2,901	286	4,059	1,243	3,315
`T+1,t+1,l+1,w+1,T+2,t+2,l+2,w+2`	1,611	271	2,884	286	3,924	1,242	3,060

5.6 DEP: Topology of Dependency Trees Available

We used Malt parser [8] and trained it on the `train` data with manual annotation of dependency structure, but automatic annotation of POS tags and lemmatization (using naïve tagger and lemmatizer from the previous section). Subsequently, we parsed both the `train` and `test` data. We trained and tested our dependency LMs on this new data.

Results in Tab. 4 show perplexity just for selected models (because possible configurations are numerous). Note that there is no single best ordering of context factors, e.g. for Arabic, Hungarian and Turkish it is better to use `N` before `wp`, while other languages have lower perplexity with the opposite ordering. The improvement of the DEP setting against PLAIN is even greater than with the TAGS setting.

[8] Actually, for English there are no lemmata in CoNLL 2007 data.

Table 4. Perplexity of DEP models

Model	Perplexity						
	ar	ca	cs	en	hu	it	tr
w-1,w-2 (trigram baseline)	1,988	325	3,530	356	5,183	1,552	4,015
wp	2,160	456	3,500	506	5,706	1,767	3,850
wp,wg	2,128	424	3,482	492	5,711	1,740	3,847
E,wp	1,993	327	3,070	391	4,891	1,393	3,386
E,C,N,wp	1,533	239	2,529	287	3,746	1,177	2,977
E,C,wp,N	1,570	232	2,410	277	3,879	1,137	2,988
E,C,N,wp,wg	1,525	229	2,522	283	3,746	1,174	2,976
E,C,wp,N,wg	1,561	222	2,403	272	3,879	1,135	2,986

5.7 DEP+TAGS: Both POS Tags and Topology Available

Tab. 5 shows that overall best results (i.e. lowest perplexity) for all the languages were reached by combination of both types of additional context information (DEP+TAGS).

Table 5. Perplexity of DEP+TAGS models

Model	Perplexity						
	ar	ca	cs	en	hu	it	tr
w-1,w-2 (trigram baseline)	1,988	325	3,530	356	5,183	1,552	4,015
E,C,Tp,tp,N,wp,Tg,tg,wg	1,303	202	2,077	242	3,392	1,051	2,659
E,C,Tp,tp,N,lp,wp,Tg,tg	1,280	209	2,034	244	3,346	1,052	2,555
E,C,Tp,tp,N,lp,wp,Tg,tg,lg	1,270	200	2,028	244	3,346	1,050	2,555

6 Discussion

Perplexity is traditionally considered as a good indication of LM performance in an intrinsic evaluation.[9] Fig. 2 illustrates that we achieved significant improvements in perplexity over the baseline (trigram LM) for all seven tested languages and all four settings (e.g. for English, PLAIN $\approx$ 8% improvement, TAGS $\approx$ 20% improvement, DEP $\approx$ 24% improvement, DEP+TAGS $\approx$ 31% improvement). We conclude with three main outcomes:

- In contrast to the common practice of using preceding words as context in language modeling, we observed better perplexity when using following words (i.e. post-ngram LMs).
- Dependency LMs achieved even better perplexity than post-ngram LMs.
- Using additional context information (e.g. POS tag and lemma) improved the perplexity both for post-ngram LMs and for dependency LMs.

[9] Of course, if we are interested in performance of a particular application, it is better to use extrinsic evaluation (e.g. WER for speech recognition and BLEU for MT).

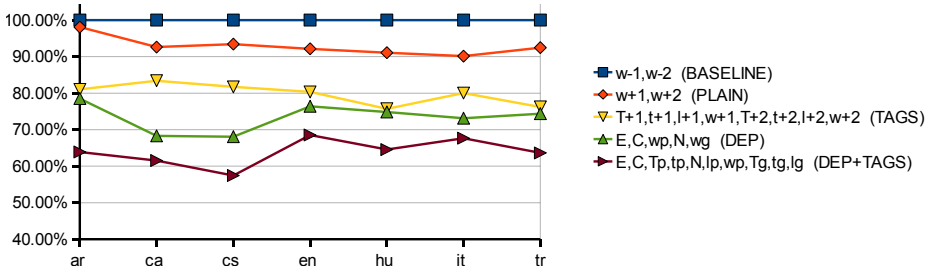

Fig. 2. Percentage comparison of perplexity of new LMs with the baseline (trigram LM = 100%). The best LM was chosen for each of settings: PLAIN, TAGS, DEP and DEP+TAGS.

The drawback of our dependency LMs is that they cannot be easily combined with ngram LMs nor translation/acoustic models at the word level. Unless we know the parse topology from another source, we must use a dependency parser that needs to see the whole sentence in advance, so the dependency LMs can be used merely for re-ranking of n-best lists of whole sentences. According to [6], this drawback of dependency LMs is not necessarily compelling. Similarly, we hope that the superior perplexity of our new LMs will result in improvements in real applications. In future, we would like to perform experiments with post-ngram LMs in speech recognition and with dependency LMs in tree-to-tree MT.

References

1. Rosenfeld, R.: A Maximum Entropy Approach to Adaptive Statistical Language Modelling. Computer speech and language 10, 187 (1996)
2. Bilmes, J., Kirchhoff, K.: Factored Language Models and Generalized Parallel Backoff. In: HLT/NAACL-2003, Edmonton, Alberta (2003)
3. Chelba, C., Engle, D., Jelinek, F., Jimenez, V., Khudanpur, S., Mangu, L., Printz, H., Ristad, E., Rosenfeld, R., Stolcke, A., et al.: et al.: Structure and Performance of a Dependency Language Model. In: Proceedings of Eurospeech (1997)
4. Shen, L., Xu, J., Weischedel, R.: A New String-to-Dependency Machine Translation Algorithm with a Target Dependency Language Model, pp. 577–585 (2008)
5. Nivre, J., Hall, J., Kubler, S., McDonald, R., Yuret, D., Nilsson, J., Riedel, S., Yuret, D.: The CoNLL 2007 Shared Task on Dependency Parsing. In: Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007, Prague, pp. 915–932 (2007)
6. Charniak, E.: Immediate-Head Parsing for Language Models. In: Processing of ACL 2001, vol. 39, pp. 116–123 (2001)
7. Chen, S., Goodman, J.: An Empirical Study of Smoothing Techniques for Language Modeling. Computer Speech and Language 13, 359–394 (1999)
8. Nivre, J., Hall, J., Nilsson, J., Chanev, A., Eryigit, G., Kübler, S., Marinov, S., Marsi, E.: MaltParser: A Language-Independent System for Data-Driven Dependency Parsing. Natural Language Engineering 13, 95–135 (2007)