M2L at SemEval-2016 Task 8: AMR Parsing with Neural Networks

Yevgeniy Puzikov, Daisuke Kawahara, Sadao Kurohashi
Graduate School of Informatics, Kyoto University
Yoshida-honmachi, Sakyo-ku
Kyoto, 606-8501, Japan
puzikov@nlp.ist.i.kyoto-u.ac.jp
{kuro,dk}@i.kyoto-u.ac.jp

Abstract

This paper describes our contribution to the SemEval 2016 Workshop. We participated in the Shared Task 8 on Meaning Representation parsing using a transition-based approach, which builds upon the system of Wang et al. (2015a) and Wang et al. (2015b), with additions that utilize a Feedforward Neural Network classifier and an enriched feature set. We observed that exploiting Neural Networks in Abstract Meaning Representation parsing is challenging and we could not benefit from it, while the feature enhancements yielded an improved performance over the baseline model.

1 Introduction

Abstract Meaning Representation (AMR) (Banarescu et al., 2013; Dorr et al., 1998) is a semantic formalism which represents sentence meaning in a form of a rooted directed acyclic graph. AMR graph nodes represent concepts, labelled directed edges between the nodes show the relationships between concepts. The AMR formalism was created in order to explore the semantics behind natural language units for further analysis and application in various tasks.

At the time of writing this paper two AMR parsers are publicly available: graph-based JAMR (Flanigan et al., 2014) and transition-based CAMR (Wang et al., 2015a; Wang et al., 2015b). The latter has served as our baseline model, which we tried to improve by incorporating additional features defined for a wider conditioning context and a neural network (NN) classifier.

Inspired by the results of Chen and Manning (2014) and Weiss et al. (2015), who obtained state-of-the-art results in transition-based dependency parsing using Feedforward Neural Networks (FFNN), and taking into account the transition nature of the CAMR model, we performed a series of experiments in the same direction. Neural networks have been successfully applied to many NLP fields and we were curious to examine their potential in the task of AMR parsing. Specifically, we investigated the possibility of constraining the averaged perceptron algorithm (Collins, 2002) predictions by those of an FFNN at the initial step of the inference process.

2 Preprocessing

We used the Stanford CoreNLP v3.6.0 toolkit (Manning et al., 2014) to get named entity (NE) and dependency information, the latter in the form of Stanford dependencies was obtained from the NN dependency parser (Chen and Manning, 2014).

We used a publicly available semantic role labelling (SRL) system with a predicate disambiguation module (Björkelund et al., 2009). The system is a part of MATE tools[1], which also include a lemmatizer, a part of speech (POS) tagger, and a dependency parser. We use them to obtain lemmas and POS tags. All the tools were used with the pretrained models.

Using MALT dependencies instead of or in tandem with Stanford dependencies did not change the

[1] https://storage.googleapis.com/google-code-archive-downloads/v2/code.google.com/mate-tools

Proceedings of SemEval-2016, pages 1154–1159,
San Diego, California, June 16-17, 2016.

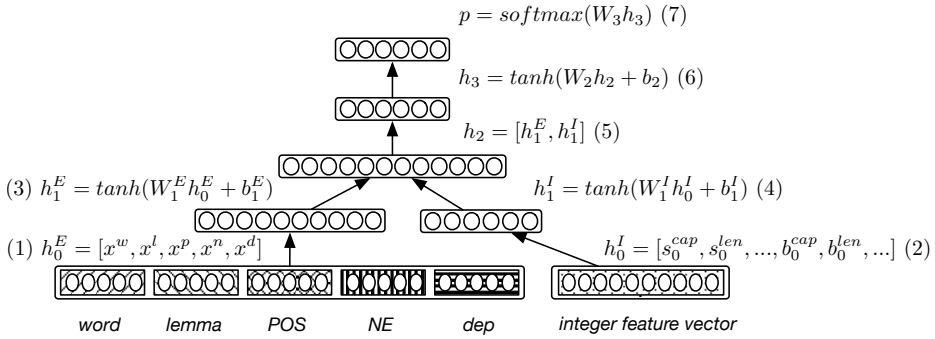

Figure 1: Schematic view of our NN model.

overall performance of the AMR parser. Due to time limitations, we did not perform a full analysis of the accuracy of both tools, which would be an interesting and important point to investigate further.

All information extracted after the preprocessing step was combined into one CoNLL-format file (Hajič et al., 2009). Finally, we used an AMR graph-to-sentence aligner of Flanigan et al. (2014) to map word spans to concept fragments in the AMR graph.

3 Parsing Algorithm

We use the same set of transitions and a parsing algorithm as Wang et al. (2015b). We skip the full description due to space limitations and refer to the original paper. It forms a quadruple $Q = (S, T, s_0, S_t)$, where S is a set of parsing states (or configurations), T is a set of parsing transitions (or actions), s_0 is the initial state and S_t is a set of terminal states of the parser. Each state is a triple (σ, β, G), where σ denotes a stack, storing indices of the nodes which have not been processed yet; it's top element is σ_0. β is a buffer, storing the children of σ_0. Finally, G is a partially built AMR graph aligned with the sentence tokens.

At the beginning of the parsing procedure, σ is initialized with a post-order traversal of the input dependency tree with topmost element σ_0; β is initialized with σ_0's children or set to null if σ_0 is a leaf node. G is initialized with the nodes and edges of the dependency tree, but all node and edge labels are set to null. The parser processes all nodes and their outgoing edges in the tree in a bottom-up left-right manner, applying some transition to the current node or edge. Parsing terminates when both σ and β are empty.

There are nine basic transitions (*NEXT-EDGE, SWAP, REATTACH, REPLACE-HEAD, REENTRANCE, MERGE, NEXT-NODE, DELETE-NODE, INFER*), some of which result in assigning either a concept label or an edge label. The sets of concept labels for aligned nodes, unaligned nodes and edge labels ($S_{aligned\_tag}$, $S_{unaligned\_tag}$, S_{edge}, respectively) are constructed during the preprocessing stage and are later used to provide candidate concept tags or edge labels for the respective transitions. Let's also introduce a set $S_{total} = S_{aligned\_tag} \bigcup S_{unaligned\_tag} \bigcup S_{edge}$. $|S_{total}|$ determines the number of classes for our classification algorithm to choose from.

We propose a parsing strategy which first uses an NN classifier to constrain the space of candidate transitions by choosing an unlabelled version of a transition, and then forces the perceptron algorithm to make a prediction only on the label of the chosen transition. We have also tried to completely substitute the averaged perceptron algorithm with an NN, experimenting with various types and architectures of the latter, but experimental results were unsatisfactory. This question is yet to be analysed in full, but presumably we failed to provide our NN classifiers with a sufficiently rich input representation: we used a very simple technique of concatenating embedding vectors (Section 4), which apparently did not capture enough information for an NN to make accurate predictions.

4 Neural Network Architecture

During preprocessing, we create a vocabulary V which stores all unique words, lemmas, POS tags, NE tags and dependency labels from the training set.

Perceptron model features	
Singular features	**Feature combination**
$\{\sigma_0, \beta_0, \beta_0^{head+}\}.\{w, l, n, p, d, brown4, brown6, brown10, brown20\}$	$\{\sigma_0.w, \sigma_0.l, \sigma_0.p, \sigma_0.n, \sigma_0.d, \sigma_0.len\} + t.tag$
$\sigma_0^{head}.\{w, l, p, n, d\}, \sigma_0.tag$	$\sigma_0.d + \beta_0.brown20, \sigma_0.brown4 + \beta_0.brown20, \sigma_0.brown20 + \beta_0.brown4, \sigma_0.brown20 + \beta_0.d$
$\sigma_0.isnom, \beta_0.len, \beta_0.rph$	$\sigma_0.n + \beta_0.n, \sigma_0.d + \beta_0.l, \sigma_0.p + \beta_0.l, \sigma_0.l + \beta_0.d, \sigma_0.l + \beta_0.p$
$dist_{\sigma_0,\beta_0}, dist_{\beta_0,\beta_0^{head+}}$	$\beta_0.l + \beta_0^{rsb}.d, \beta_0.l + \beta_0.numswp, \beta_0.l + \beta_0^{head+}.d, \beta_0.l + \beta_0^{head+}.p$
$\{\beta_0, \beta_0^{head+}\}.\{arglabel, prdlabel\}, \sigma_0.verb\_sense$ ⋆	$\beta_0.n + \beta_0^{head+}.n, \beta_0.tag + \beta_0^{head+}.tag, \beta_0.n + \beta_0^{head+}.tag, \sigma_0.l + \beta_0.tag + \beta_0^{head+}.tag$
$\sigma_0.eq\_verb\_sense, \{\beta_0, \beta_0^{head+}\}.\{isarg, isprd\}$ ⋆	$path_{\beta_0,\sigma_0} + \beta_0.l + \sigma_0.l, path_{\beta_0,\beta_0^{head+}} + \beta_0.l + \beta_0^{head+}.l$
$\{\sigma_0, \sigma_0^p, \sigma_0^{lsb}, \sigma_0^{rsb}, \sigma_0^{rsb2}, \sigma_0^{prs1}, \sigma_0^{prs2}\}.\{w, l, p, n, d\}$ •	$dist_{\sigma_0,\beta_0} + path_{\sigma_0,\beta_0}, dist_{\sigma_0,\beta_0} + path_{\beta_0^{head+},\beta_0}$
$\{\beta_0, \beta_0^p, \beta_0^{lsb}, \beta_0^{rsb}, \beta_0^{rsb2}, \beta_0^{prs1}, \beta_0^{prs2}\}.\{w, l, p, n, d\}$ •	$\sigma_0.p + t_{is\_verb}, \beta_0.d + \beta_0.arglabel$ ⋆
NN model features	
Embedding features: $\{\sigma_0, \sigma_0^p, \sigma_0^{lsb}, \sigma_0^{rsb}, \sigma_0^{rsb2}, \sigma_0^{prs1}, \sigma_0^{prs2}\}.\{w, l, p, n, d\}$	**Numerical features:** $\{\sigma_0, \beta_0\}.\{cap, arg0, arg1, arg2, len, nech, isnom, isleaf\}$
$\{\beta_0, \beta_0^p, \beta_0^{lsb}, \beta_0^{rsb}, \beta_0^{rsb2}, \beta_0^{prs1}, \beta_0^{prs2}\}.\{w, l, p, n, d\}$	

Table 1: Feature sets. σ_0^{head} is the head of σ_0 in the dependency tree; β_0^{head+} is the node, to which β_0 could be attached to as a result of either reentrance or reattachment transition; t is a transition under consideration – during feature extraction we consider the tag which might be assigned as a result of transition ($t.tag$) or whether this candidate is a verb sense tag ($t.tag_{is\_verb}$). The $isnom$ feature checks whether the lemma of a corresponding element is in the NomBank dictionary. $nech$ is the number of an element's children which have an NE label from the set {"PERSON", "LOCATION", "ORGANIZATION", "MISC"}. Other features are self-explanatory.

Each of them is mapped to the same D-dimensional vector space ($e_i^w, e_i^l, e_i^p, e_i^n, e_i^d \in \mathbb{R}^D$). We create one embedding matrix $E \in \mathbb{R}^{|V| \times D}$, where $|V|$ is the total size of the vocabulary and D is the dimensionality of dense embedding vectors.

In each parsing configuration we consider a set of elements which might be useful in the prediction task. These elements are σ_0, β_0 and their neighbouring nodes, which for σ_0 include σ_0^p (the parent of σ_0 in the dependency tree), σ_0^{lsb} (the left sibling), $\sigma_0^{rsb}, \sigma_0^{rsb2}$ (the first and the second right siblings), $\sigma_0^{prs1}, \sigma_0^{prs2}$ (the first and the second previous tokens in the sentence); neighbouring nodes for β_0 are defined in a similar manner. Thus, the total number of relevant elements is $n_{elem} = 14$.

The overall architecture of the network is depicted in Figure 1. The input layer of the network consists of two components. The first one is formed by concatenating all the corresponding embedding vectors for each element's feature (Figure 1 (1)). Each of the x components is, in turn, a concatenation of the embeddings of the configuration elements for a particular type of annotation. For example, x^w is an $\mathbb{R}^N$ vector, where $N = d \times n_{elem}$. We form x^l, x^p, x^n, x^d in a similar manner and concatenate x^w, x^l, x^p, x^n, x^d to form the input vector h_0^E. The second component of the input vector is h_0^I (Figure 1 (2)), a concatenation of vectors, representing non-embedding numerical features (see Table 1).

We separately map two parts of the input layer to hidden layers using the $tanh$ activation function: h_0^E to h_1^E, h_0^I to h_1^I (Figure 1 (3) and (4)) . We then concatenate layers h_1^E and h_1^I (Figure 1 (5)) and pass the resultant vector to the last hidden layer h_3, applying the $tanh$ function again (Figure 1 (6)). Finally, a softmax layer is added on top of h_3 in order to calculate probabilities of the output classes (Figure 1 (7)).

5 Feature Sets

We have designed two separate feature sets for the NN and perceptron classifiers. The feature set for the latter is roughly the same as in (Wang et al., 2015b) (Table 1). Following the authors of CAMR, we also make use of the NomBank 1.0 dictionary (Meyers et al., 2004) [2]. Unfortunately, we could not obtain the copy of the same SRL system which was used by the authors. Therefore, we also measure accuracy improvement from incorporating the semantic features defined in the original paper but extracted after processing the data with a different SRL system (marked with a ⋆).

We also measure the improvement from incorporating the features extracted from a wider configuration context – they were not included into the baseline model and are marked with a •.

In the case of the NN classifier we follow a standard feature extraction procedure and discard transition-specific features. Apart from the embedding features, we also include a number of numerical features, which proved to be useful in our exper-

[2] http://nlp.cs.nyu.edu/meyers/NomBank.html

iments.

6 Training Procedure and Parsing Policy

We trained the perceptron with a weight-averaging procedure, described in (Collins, 2002).

For the NN classifier we first prepared a training set $\{(x_i, y_i)\}_{i=1}^{n}$, where x_i denotes a feature representation of configuration i and y_i is the unlabelled version of the correct transition. The training objective is to minimize L2-regularized negative log-likelihood of the model:

$$L(\theta) = -\sum_{i=1}^{n} \log P(y_i|x_i, \theta) + \lambda||\theta||_2^2,$$
$$\theta = [E, W_1^E, W_1^I, W_2, W_3].$$

We randomly initialized the embeddings within $(-0.01, 0.01)$ and fixed their dimensionality at 32. All weight matrices were initialized using the normalized initialization technique of Glorot and Bengio (2010). Hidden layer sizes were fixed as follows: $h_0^E = 2,240, h_0^I = 32, h_1^E = 100, h_1^I = 16, h_2 = 116, h_3 = 64$. All the biases were set to 0.02.

We train our network using mini-batches of size 200 under the early stopping settings with RMSprop[3] as an optimization algorithm. The learning rate of 5×10^{-4} and $\lambda = 1.5 \times 10^{-5}$ were found to perform best on the validation set.

Given a feature representation of the current configuration s, the parsing algorithm first provides a pool of legal transitions $\mathcal{T} \in S_{total}$, $|\mathcal{T}| \ll |S_{total}|$. We compute the probabilities of nine unlabelled transitions using the NN. If the network is confident about its prediction (we set an empirically chosen probability threshold of 0.9), we choose the highest scoring candidate and force the perceptron to predict the label of the transition, if it is a transition which assigns a concept tag or edge label. Each candidate $t \in \mathcal{T}$ is being scored by a linear scoring function $score(t, s) = \omega \cdot \phi(t, s)$, where ω denotes a weight vector for a particular candidate transition and $\phi(t, s)$ is a feature function mapping a (t, s) pair to a real-valued feature vector. The best scoring transition is chosen and applied to the configuration. If the NN chooses a transition which does not assign labels, we apply this transition without asking for the perceptron prediction. Finally, if the network is not confident about the prediction – that is, if the probability for the highest scoring candidate is lower than 0.9 – we disregard the NN prediction and choose the prediction given by the perceptron algorithm.

[3] http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

Model	P	R	F_1
Perceptron (baseline)	0.56	0.66	0.60
Perceptron + SRL	0.57	0.68	0.62
Perceptron + wide context	**0.58**	**0.69**	**0.63**
Perceptron + SRL + wide context	0.58	0.68	0.63
NN + Perceptron	0.56	0.66	0.60
NN + Perceptron + SRL + wide context	0.58	0.68	0.63

Table 2: Experimental results.

7 Experiments

All the experiments were performed on the LDC2015E86 dataset, provided by the organizers. In our experiments we followed the standard train/dev/test split ($16,833$, $1,368$ and $1,371$ sentences, respectively). Parser performance was evaluated with the Smatch (Cai and Knight, 2013) scoring script v2.0.2 [4] (Table 2).

As expected, using SRL features resulted in better performance compared to the baseline model (roughly a 2 F_1 points gain). Conditioning on a wider context was also beneficial – widening the context to include more configuration elements is often a good feature expansion technique (Toutanova et al., 2003; Chen et al., 2014). In contrast to our expectations, the NN classifier did not improve the parser performance. This might be due to the higher complexity of the AMR parsing task or the peculiarities of the underlying parsing algorithm (as mentioned in Section 3, we discarded some action-specific features due to the difficulty of their integration into the NN model). Further investigation on this matter is required to draw ground conclusions.

8 Conclusion

We have performed a range of experiments which resulted in improving the performance of the baseline AMR parsing system. The results show that a richer feature set is very likely to lead to more accurate predictions. Unfortunately, our attempts to further im-

[4] http://alt.qcri.org/semeval2016/task8/index.php?id=data-and-tools

prove the system using NN were not that successful. This goes against the hypothesis that a small number of dense vector embedding features are sufficient to capture the information necessary for accurate inference, which in traditional approaches is achieved by using a large amount of sparse hand-crafted features (Chen and Manning, 2014). The obtained results will be used in our further investigation on this matter.

Acknowledgments

We thank the anonymous reviewers for their insightful comments and suggestions. The first author is supported by a Japanese Government (MEXT) research scholarship.

References

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract meaning representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 178–186.

Anders Björkelund, Love Hafdell, and Pierre Nugues. 2009. Multilingual semantic role labeling. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task*, CoNLL '09, pages 43–48, Stroudsburg, PA, USA. Association for Computational Linguistics.

Shu Cai and Kevin Knight. 2013. Smatch: an evaluation metric for semantic feature structures. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 748–752, Sofia, Bulgaria, August. Association for Computational Linguistics.

Danqi Chen and Christopher Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 740–750, Doha, Qatar, October. Association for Computational Linguistics.

Wenliang Chen, Yue Zhang, and Min Zhang. 2014. Feature embedding for dependency parsing. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 816–826, Dublin, Ireland, August. Dublin City University and Association for Computational Linguistics.

Michael Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10*, EMNLP '02, pages 1–8, Stroudsburg, PA, USA. Association for Computational Linguistics.

Bonnie Dorr, Nizar Habash, and David Traum. 1998. A thematic hierarchy for efficient generation from lexical-conceptual structure. In *Proceedings of the Third Conference of the Association for Machine Translation in the Americas, AMTA-98, in Lecture Notes in Artificial Intelligence*, pages 333–343.

Jeffrey Flanigan, Sam Thomson, Jaime Carbonell, Chris Dyer, and A. Noah Smith. 2014. A discriminative graph-based parser for the abstract meaning representation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1426–1436.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS10). Society for Artificial Intelligence and Statistics*.

Jan Hajič, Massimiliano Ciaramita, Richard Johansson, Daisuke Kawahara, Maria Antònia Martí, Lluís Màrquez, Adam Meyers, Joakim Nivre, Sebastian Padó, Jan Štěpánek, Pavel Straňák, Mihai Surdeanu, Nianwen Xue, and Yi Zhang. 2009. The conll-2009 shared task: Syntactic and semantic dependencies in multiple languages. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task*, CoNLL '09, pages 1–18, Stroudsburg, PA, USA. Association for Computational Linguistics.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60.

A. Meyers, R. Reeves, C. Macleod, R. Szekely, V. Zielinska, B. Young, and R. Grishman. 2004. The nombank project: An interim report. In A. Meyers, editor, *HLT-NAACL 2004 Workshop: Frontiers in Corpus Annotation*, pages 24–31, Boston, Massachusetts, USA, May 2 - May 7. Association for Computational Linguistics.

Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL '03, pages 173–180, Stroudsburg, PA, USA. Association for Computational Linguistics.

Chuan Wang, Nianwen Xue, and Sameer Pradhan. 2015a. A transition-based algorithm for amr parsing. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 366–375, Denver, Colorado, May–June. Association for Computational Linguistics.

Chuan Wang, Nianwen Xue, and Sameer Pradhan. 2015b. Boosting transition-based amr parsing with refined actions and auxiliary analyzers. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 857–862, Beijing, China, July. Association for Computational Linguistics.

David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. 2015. Structured training for neural network transition-based parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 323–333, Beijing, China, July. Association for Computational Linguistics.