UCL+Sheffield at SemEval-2016 Task 8: Imitation learning for AMR parsing with an α-bound

James Goodman[*] **Andreas Vlachos**[#] **Jason Naradowsky**[*]
[*] Computer Science Department, University College London
james@janigo.co.uk, jason.narad@gmail.com
[#] Department of Computer Science, University of Sheffield
a.vlachos@sheffield.ac.uk

Abstract

We develop a novel transition-based parsing algorithm for the abstract meaning representation parsing task using exact imitation learning, in which the parser learns a statistical model by imitating the actions of an expert on the training data. We then use the imitation learning algorithm DAGGER to improve the performance, and apply an α-bound as a simple noise reduction technique. Our performance on the test set was 60% in F-score, and the performance gains on the development set due to DAGGER was up to 1.1 points of F-score. The α-bound improved performance by up to 1.8 points.

1 Introduction

In abstract meaning representation parsing (Banarescu et al., 2013), the goal is to parse natural language in a domain-independent graph-based meaning representation (AMR). In the first AMR parsing work, Flanigan et al. (2014) split the task into two sub-tasks; *concept identification* and *graph creation*. The sub-tasks are learned independently, and exact inference is used to find highest-scoring maximum spanning connected acyclic graph that contains all the concepts identified in the first stage. Later work by Wang et al. (2015b) adopted a different strategy based on the similarity between the dependency parse of a sentence and the semantic AMR graph. They start from the dependency parse and learn a transition-based parser that converts it into an AMR graph. To learn the parser, Wang et al. (2015b) define an algorithm that for each instance in the training data infers the action sequence that convert the input dependency tree into the corresponding AMR graph and train a classifier to predict the actions to be taken during testing. This strategy is also referred to as exact imitation learning, while the algorithm that infers the action sequence in the training instances is commonly referred to as the expert policy.

In our submission to SemEval Task 8 on AMR parsing, we follow the transition-based paradigm of Wang et al. (2015b) with modifications to the parsing algorithm, and also use the DAGGER imitation learning algorithm (Ross et al., 2011) to generalise better to unseen data. The central idea of DAGGER is that the distribution of states encountered by the expert policy during training may not be a good approximation to those seen in testing by the trained policy. Previous work by Rao et al. (2015) used SEARN, a similar imitation learning algorithm, on the AMR problem, with an algorithm that constructs the AMR graph directly from the sentence tokens. Imitation learning has also been used successfully in other semantic parsing tasks (Vlachos and Clark, 2014; Berant and Liang, 2015).

In imitation learning approaches such as DAGGER the previous actions become features for classification learning. However the partial graphs in AMR parsing are rather complex to represent in this way, and combined with the finite amount of training data different actions can be chosen by the expert even though the feature representations for them can be very similar. These decisions appear as noisy outliers in classification learning. To control noise we experiment with the α-bound discussed by Khardon and Wachman (2007), which excludes a training example from future training once it has been misclas-

Action Name	Param.	Pre-conditions	Outcome of action
NextEdge	l_r	β non-empty	Set label of edge (σ_0, β_0) to l_r. Pop β_0.
NextNode	l_c	β empty	Set concept of node σ_0 to l_c. Pop σ_0, and initialise β.
Swap		β non-empty	Make β_0 parent of σ_0 (reverse edge) and its sub-graph. Pop β_0 and insert β_0 as σ_1.
ReplaceHead		β non-empty	Pop σ_0 and delete it from the graph. Parents of σ_0 become parents of β_0. Other children of σ_0 become children of β_0. Insert β_0 at the head of σ and re-initialise β.
Reattach	κ	β non-empty	Pop β_0 and delete edge (σ_0, β_0). Attach β_0 as a child of κ. If κ has already been popped from σ then re-insert it as σ_1.
DeleteNode		β empty; leaf σ_0	Pop σ_0 and delete it from the graph.
Insert	l_c		Insert a new node δ with AMR concept l_c as the parent of σ_0, and insert δ into σ.
InsertBelow	l_c		Insert a new node δ with AMR concept l_c as a child of σ_0.
Reentrance	κ	Phase 2	Insert edge (σ_0, κ). Then apply NextEdge action.
Wikify	ω	Phase 2	See main text

Table 1: Action Space for the transition-based graph parsing algorithm

sified α times in training. Khardon and Wachman (2007) do not report any experimental results for this, and we are not aware of any previous use of this method.

2 System description

In the following subsections we focus on the differences from previous work and in particular that of Wang et al. (2015b) who introduced the transition-based dependency-to-AMR paradigm we follow. We initialise the main algorithm with a stack of the nodes in the dependency tree, root node first. This stack is termed σ. A second stack, β is initialised with all children of the top node in σ. The state at any time is described by σ, β, and the current graph (which starts as the dependency tree). Each action manipulates the top nodes in each stack, σ_0 and β_0. We reach a terminal state when σ is empty. [1]

2.1 The Expert Policy

The expert policy used in training applies heuristic rules to determine the next action from a given state. It uses the training alignments to construct a mapping between nodes in the dependency tree, and nodes in the target AMR. Any unmapped nodes in the dependency tree will be deleted by the expert, and any unmapped nodes in the AMR graph will be inserted. All our experiments use node alignments from the system of Pourdamghani et al. (2014).

2.2 Action Space

Flanigan et al. (2014) and Wang et al. (2015b), both use AMR fragments as their smallest unit, which may consist of more than one AMR concept. Instead, we always work with the individual AMR nodes, and rely on Insert actions to learn how to build common fragments, such as country names. The main adaptations to the actions, summarised in Table 1, stem from this. NextNode and NextEdge form the core action set, labelling nodes and edges respectively without changing the graph structure. Swap, Reattach and ReplaceHead change this structure, but always retain a tree structure. ReplaceHead covers two distinct actions in Wang et al. (2015b); ReplaceHead and Merge. Their Merge action merges σ_0 and β_0 into a composite node; this is not required without composite nodes and retention of a 1:1 mapping between nodes and AMR concept. Unlike Wang et al. (2015b) we do not parameterise Swap or Reattach actions with a label. We leave that decision to a later NextEdge action. We permit a Reattach action to use parameter κ equal to any node within six edges from σ_0, excluding any that would disconnect the graph or creating a cycle.

The Insert action inserts a new node as a parent of the current σ_0. Wang et al. (2015a) later introduced an 'Infer' action similar to our Insert action. Infer inserts an AMR concept node above the current node

[1]Code available at `https://github.com/hopshackle/dagger-AMR`. SemevalSubmission tag bookmarks the version used.

as Insert does, but is restricted to nodes that occur outside of AMR 'fragments', which continue to be the base building block.

Reentrance is the one action that will turn a Tree into a non-Tree. We only consider Reentrance actions during a second pass ("Phase Two") through the AMR graph once the first pass has reached a terminal state. In this second pass we consider each node as σ_0 in turn, and each nearby node as a possible κ to insert a new edge (σ_0, κ). We follow any Reentrance action with a NextEdge action to label the new arc. This approach simplifies the first pass, during which the graph is guaranteed to be a Tree. Reentrance makes only a small difference in the final F-Score, and was turned off for our final submission.

Also during Phase Two we decide whether to Wikify σ_0. This adds a new leaf node with a relation of `wiki`. There are three parameter values for ω that determine the wiki concept. In turn these use "-"; a concatenation of all child concepts in original word order; a dictionary look-up keyed on a concatenation of the child nodes if these were seen in the training data. An example of the third option is if a `name` node with a single child node of "Michael" is seen in the training data with a `wiki` relation of "Michael_Jackson". This wikification is held in the dictionary, and will be used for any `name` node with a single child node of "Michael" in test. If instead in test the `name` node had two children; "Michael", and "Jackson", then during ω would either add a wiki node of "-", or one of "Michael_Jackson" by concatenating the two child concepts in the order they appear in the sentence.

Figure 1 shows a parse of a sentence fragment. The current σ_0 node is shown dashed and in red. From the top the actions are Insert(date-entity); NextNode(WORD); NextEdge(year); *second diagram*; NextNode(WORD); ReplaceHead to remove "in"; *third diagram*; NextNode(WORD); NextEdge(mod); Reattach to move "date-entity"; *fourth diagram*; NextNode(VERB); ReplaceHead to remove "by"; NextEdge(ARG0); NextEdge(time); NextNode(strike-01).

Wang et al. (2015b) use all AMR concepts and relations that appear in the training set as possible parameters (l_c and l_r) if they appear in any sentence containing the same lemma as σ_0 and β. We reduce this to just concepts that have been aligned to the

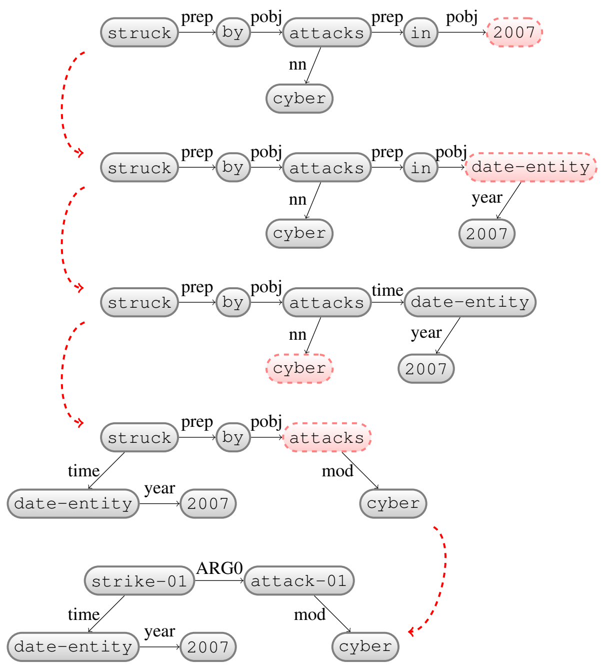

Figure 1: Example parse from dependency tree to AMR of the sentence fragment "…struck by cyber attacks in 2007."

current lemma. We initially run the expert policy over the training set, and track the AMR concept assigned for each lemma. These provide the possible l_c that will be used for NextNode actions. Similarly we track the lemmas at head and tail of each expert-assigned AMR relation, and compile possible l_r from these. There is no direct generalisation between different concepts and relations; so `ARG0` and `ARG0-of` are independently learned relations for example, although they represent the same semantic relationship.

AMR concepts/relations will never be considered during test if they were not aligned to that lemma in the training data. To relax this restriction we allow l_c to take the values `WORD`, `LEMMA`, `VERB`, which respectively use the word, lemma, or the lemma concatenated with '-01' as the AMR concept. This is inspired by Werling et al. (2015), who use a similar set of actions in a concept identification phase. These options improve performance (by 0.5 to 1.0 points on a validation set) by generalising to unseen tokens in test data, which otherwise would have no mapped AMR concepts. For the l_c parameters on Insert (InsertBelow) actions, we use all AMR concepts that the expert inserted above (below) any node in the training set with the same lemma as σ_0.

2.3 Additional action constraints

Transition-based parsing algorithms have classically relied on a fixed length of trajectory T for guarantees on performance, or at least a bounded T (Goldberg and Elhadad, 2010; Honnibal et al., 2013; Sartorio et al., 2013; McDonald and Nivre, 2007). In our approach T is theoretically unbounded and the algorithm could Insert, or Reattach *ad infinitum*.

We impose constraints to prevent these situations. A Swap action cannot be applied to a previously Swapped edge; once a node has been moved by Reattach, then it cannot be Reattached again; an Insert action is only permissible if no previous Insert action has been used with that node as σ_0; an Insert action is not permissible if it would insert an AMR concept already in use as any of the parent, children, grand-parents or grand-children of σ_0.

Any action that would create a cycle is prohibited. We do not prevent duplication of argument relations so that a concept could have two outgoing `ARG1` edges. We start with a fully connected graph (the dependency tree), and preserve full connectivity as none of the actions will disconnect a graph.

2.4 Imitation Learning

In the first iteration we only use the expert policy to generate a trajectory for each training sentence, and train a classifier π_1 from the collected data. At each step in the ith iteration we randomly choose the expert (with probability β_i) or else π_{i-1}. The full set of collected data from all iterations is then used to train π_{i-1} to obtain π_i. $\beta_1 = 1$ and we set $\beta_i = 0.7\beta_{i-1}$. Hence each iteration uses the expert policy less and less, with the training trajectories increasingly approximating the states the classifier would encounter without expert knowledge of the target. Formally the DAGGER algorithm is online (Ross et al., 2011), while we use it in a batch mode as described above. We use an averaged AROW classifier for all our experiments, with parameter $r = 100$ (Crammer et al., 2009). After each batch DAGGER iteration we use 3 iterations of (on-line) AROW training using all the collected data.

As well as the α-bound for noise reduction, we tried two additional modifications to DAGGER and AROW. Firstly, we considered reducing the number of actions explored at each stage. The currently trained classifier evaluates all possible actions, and discards any that exceed the best scoring action by some threshold. Only this smaller set, plus the expert action, are included in the training example. This speeds up classifier training. In the first iteration we choose three to five actions randomly. Ross and Bagnell (2014) use a random set of exploratory actions in their AGGREVATE algorithm, but do not use the classifier to focus the exploration.

Secondly, we used only the smallest training sentences in the first iteration, as measured by number of AMR nodes. At each further iteration the AMR size threshold for the training set was increased. The motivation was to train the classifier on 'easy' sentences, before introducing more complex ones. We start with up to 30 nodes, and increase this by 10 each iteration. Rao et al. (2015) similarly use the smallest sentences for training, but do not increase the size threshold as training proceeds.

2.5 Features

All features used are detailed in Table 2, largely based on Wang et al. (2015b). All are 0-1 indicator functions. *inserted* is 1 if the node was inserted by the parser; *dl* is the dependency label in the original dependency tree; *ner* the named entity tag; *POS* the part-of-speech tag; *prefix* is the string before the hyphen if word is hyphenated; *suffix* is the string after the hyphen; *brown* is the 100-class Brown cluster id with cuts at 4, 6, 10 and 20 [2]; *deleted* is the lemma of any child node previously deleted by the parser; *merged* is the lemma of any node merged into this node by a ReplaceHead action; *distance* is the distance between the tokens in the sentence; *path* concatenates lemmas and dls between the tokens in the dependency tree; *POSpath* concatenates POS tags between the tokens; *NERpath* concatenates NER tags between the tokens.

The key differences to Wang et al. (2015b) are the inclusion of the brown, POSpath, NERpath, prefix and suffix feature types.

2.6 Pre-processing

Pre-processing steps on the training sentences were to: pass the full sentence through the Stanford De-

[2]From `http://metaoptimize.com/projects/wordreprs/` and the code of Liang (2005)

Context	Features
σ_0	lemma, dl, ner, POS, inserted, prefix, suffix, brown, deleted, lemma-dl
σ_{0P}	inserted, lemma, brown
σ_{0C}	label, ner, label-brown
β_0	inserted, POS, lemma, brown, ner, dl, prefix, suffix, merged
κ	ner, POS, lemma, brown, label
$\sigma_0 \rightarrow \beta_0$	label, path, lemma-path-lemma, POSpath, inserted-inserted, lemma-POS, POS-lemma, dl-lemma, lemma-dl, lemma-label, label-lemma, ner-ner, distance
$\beta_0 \rightarrow \kappa$	path, lemma-path-lemma, NERpath, POSpath, distance, lemma-POS, dl-lemma, ner-ner
$\sigma_0 \rightarrow \kappa$	distance, lemma-path-lemma, brown-brown, NERpath, POSpath, lemma-dl, lemma-label
$\sigma_{0P} \rightarrow \sigma_0$	label, POS-lemma, dl-lemma, ner-ner
$\sigma_{0PP} \rightarrow \sigma_{0P} \rightarrow \sigma_0$	lemma-lemma-lemma
$\sigma_0 \rightarrow \sigma_{0C}$	POS-lemma, lemma-POS, dl-lemma, ner-ner

Table 2: Features used by context. σ_{0P} is the parent of σ_0, σ_{0PP} the parent of σ_{0P}, and σ_{0C} a child of σ_0.

pendency Parser v3.3.1 to construct a dependency tree (Manning et al., 2014); remove punctuation tokens; "/" characters are treated as token separators, but hyphenated words are kept as single tokens; simple regex expressions are applied to find common date formats, and convert these to numeric sequences (e.g. 03-Jan-72 to 3 1 1972); similar regex conversions of common numeric expressions e.g. "two thousand" becomes "2000". The parser was then able to learn to construct `date-entity`, `temporal-quantity` and similar AMR moieties.

3 Results

In all experiments, the training data was the union of all training and dev sets in the task data, and the union of all test sets was used for validation. Table 3 shows the F-Score of the validation set with and without the Reentrance action (Reent, NoR), and with combinations of reduced search (Red), and incremental data (Inc). All of these give the same result of 0.65 to 2 dp, and the bold entry was submitted. The Baseline entry uses only a single iteration, and the DAGGER experiments report the highest F-Score achieved over 10 iterations (usually reached between the 3rd to 5th iterations).

The reduced information in the first iterations for incremental data and reduced search lead to lower initial performance here, but they still achieve the 0.65 result in time and each iteration is faster. DAGGER provides a gain of 0.6 and 1.1 points of F-Score for experiments with and without Reentrance.

Table 4 shows that the α-bound helps significantly, with a gain of 1.8 points of F-Score in this example. We found consistently that the most extreme setting of α=1 worked best.

Table 3: F-Score results on validation set ($\alpha = 1$).

Parameter settings	NoR	Reent.
Baseline	0.642	0.640
DAGGER	0.648	0.651
DAGGER & Inc	0.653	0.655
DAGGER & Red	0.648	0.653
DAGGER & Inc & Red	**0.651**	0.649

Table 4: Alpha-bound results with DAGGER, Inc and Red

Alpha-bound	F-Score
1	0.655
2	0.649
None	0.637

4 Conclusion

Imitation Learning algorithms like DAGGER help in the AMR task, as in other structured prediction problems. Performance is improved using the α-bound to reduce the impact of noise in the training examples, and future work could investigate the impact in similar tasks.

The reduction in search space and incremental growth of the training set do not have a significant impact on the results. The speed improvements they provide make little difference here, but could benefit more computationally demanding loss functions than the 0-1 expert loss used in DAGGER.

Acknowledgments

Andreas Vlachos is supported by the EPSRC grant Diligent (EP/M005429/1) and Jason Naradowsky by a Google Focused Research award.

References

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Phillip Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract Meaning Representation for Sembanking. *Proceedings of the 7th Lingustic Annotation Workshop & Interoperability with Discourse*, pages 178–186.

Jonathan Berant and Percy Liang. 2015. Imitation learning of agenda-based semantic parsers. *Transactions of the Association for Computational Linguistics (TACL)*, 3:545–558.

Koby Crammer, Alex Kulesza, and Mark Dredze. 2009. Adaptive regularization of weight vectors. In *Advances in neural information processing systems*, pages 414–422.

Jeffrey Flanigan, Sam Thomson, Jaime Carbonell, Chris Dyer, and Noah A Smith. 2014. A discriminative graph-based parser for the abstract meaning representation.

Yoav Goldberg and Michael Elhadad. 2010. An efficient algorithm for easy-first non-directional dependency parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 742–750. Association for Computational Linguistics.

Matthew Honnibal, Yoav Goldberg, and Mark Johnson. 2013. A non-monotonic arc-eager transition system for dependency parsing. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 163–172. Citeseer.

Roni Khardon and Gabriel Wachman. 2007. Noise tolerant variants of the perceptron algorithm. *The journal of machine learning research*, 8:227–248.

Percy Liang. 2005. *Semi-supervised learning for natural language*. Ph.D. thesis, Citeseer.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60.

Ryan T McDonald and Joakim Nivre. 2007. Characterizing the errors of data-driven dependency parsing models. In *EMNLP-CoNLL*, pages 122–131.

Nima Pourdamghani, Yang Gao, Ulf Hermjakob, and Kevin Knight. 2014. Aligning english strings with abstract meaning representation graphs. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 425–429.

Sudha Rao, Yogarshi Vyas, Hal Daume III, and Philip Resnik. 2015. Parser for abstract meaning representation using learning to search. *arXiv preprint arXiv:1510.07586*.

Stephane Ross and J Andrew Bagnell. 2014. Reinforcement and imitation learning via interactive no-regret learning. *arXiv preprint arXiv:1406.5979*.

Stéphane Ross, Geoffrey J Gordon, and J Andrew Bagnell. 2011. A reduction of imitation learning and structured prediction to no-regret online learning.

Francesco Sartorio, Giorgio Satta, and Joakim Nivre. 2013. A transition-based dependency parser using a dynamic parsing strategy. In *ACL (1)*, pages 135–144.

Andreas Vlachos and Stephen Clark. 2014. A new corpus and imitation learning framework for context-dependent semantic parsing. *Transactions of the Association for Computational Linguistics*, 2:547–559.

Chuan Wang, Nianwen Xue, and Sameer Pradhan. 2015a. 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.

Chuan Wang, Nianwen Xue, and Sameer Pradhan. 2015b. A transition-based algorithm for amr parsing. *North American Association for Computational Linguistics, Denver, Colorado*.

Keenon Werling, Gabor Angeli, and Christopher D. Manning. 2015. Robust subgraph generation improves abstract meaning representation 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 982–991, Beijing, China, July. Association for Computational Linguistics.