Noise reduction and targeted exploration in imitation learning for Abstract Meaning Representation parsing

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

Semantic parsers map natural language statements into meaning representations, and must abstract over syntactic phenomena, resolve anaphora, and identify word senses to eliminate ambiguous interpretations. Abstract meaning representation (AMR) is a recent example of one such semantic formalism which, similar to a dependency parse, utilizes a graph to represent relationships between concepts (Banarescu et al., 2013). As with dependency parsing, transition-based approaches are a common approach to this problem. However, when trained in the traditional manner these systems are susceptible to the accumulation of errors when they find undesirable states during greedy decoding. Imitation learning algorithms have been shown to help these systems recover from such errors. To effectively use these methods for AMR parsing we find it highly beneficial to introduce two novel extensions: noise reduction and targeted exploration. The former mitigates the noise in the feature representation, a result of the complexity of the task. The latter targets the exploration steps of imitation learning towards areas which are likely to provide the most information in the context of a large action-space. We achieve state-of-the art results, and improve upon standard transition-based parsing by 4.7 F_1 points.

1 Introduction

Meaning representation languages and systems have been devised for specific domains, such as ATIS for air-travel bookings (Dahl et al., 1994) and database queries (Zelle and Mooney, 1996; Liang et al., 2013). Such machine-interpretable representations enable many applications relying on natural language understanding. The ambition of Abstract Meaning Representation (AMR) is that it is domain-independent and useful in a variety of applications (Banarescu et al., 2013).

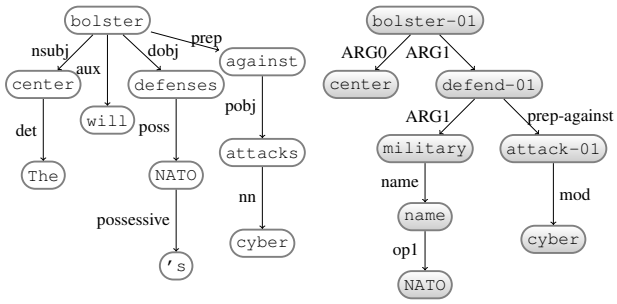

Figure 1: Dependency (left) and AMR graph (right) for: "The center will bolster NATO's defenses against cyber-attacks.'

The first AMR parser by Flanigan et al. (2014) used graph-based inference to find a highest-scoring maximum spanning connected acyclic graph. Later work by Wang et al. (2015b) was inspired by the similarity between the dependency parse of a sentence and its semantic AMR graph (Figure 1). Wang et al. (2015b) start from the dependency parse and learn a transition-based parser that converts it incrementally into an AMR graph using greedy decoding. An advantage of this approach is that the initial stage of dependency parsing is well-studied and trained using larger corpora than that for which AMR annotations exist.

Greedy decoding, where the parser builds the parse while maintaining only the best hypothesis at each step, has a well-documented disadvantage: error propagation (McDonald and Nivre, 2007). When the parser encounters states during parsing that are unlike those found during training, it is more likely to make mistakes, leading to states which are increasingly more foreign and causing errors to accumulate.

Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, pages 1–11,
Berlin, Germany, August 7-12, 2016.

One way to ameliorate this problem is to employ imitation learning algorithms for structured prediction. Algorithms such as SEARN (Daumé III et al., 2009), DAGGER (Ross et al., 2011), and LOLS (Chang et al., 2015) address the problem of error propagation by iteratively adjusting the training data to increasingly expose the model to training instances it is likely to encounter during test. Such algorithms have been shown to improve performance in a variety of tasks including information extraction(Vlachos and Craven, 2011), dependency parsing (Goldberg and Nivre, 2013), and feature selection (He et al., 2013). In this work we build on the transition-based parsing approach of Wang et al. (2015b) and explore the applicability of different imitation algorithms to AMR parsing, which has a more complex output space than those considered previously.

The complexity of AMR parsing affects transition-based methods that rely on features to represent structure, since these often cannot capture the information necessary to predict the correct transition according to the gold standard. In other words, the features defined are not sufficient to "explain" why different actions should preferred by the model. Such instances become noise during training, resulting in lower accuracy. To address this issue, we show that the α-bound Khardon and Wachman (2007), which drops consistently misclassified training instances, provides a simple and effective way of reducing noise and raising performance in perceptron-style classification training, and does so reliably across a range of parameter settings. This noise reduction is essential for imitation learning to gain traction in this task, and we gain 1.8 points of F_1-Score using the DAGGER imitation learning algorithm.

DAGGER relies on an externally specified expert (oracle) to define the correct action in each state; this defines a simple 0-1 loss function for each action. Other imitation learning algorithms (such as LOLS, SEARN) and the variant of DAGGER proposed by Vlachos and Clark (2014) (henceforth V-DAGGER) can leverage a task level loss function that does not decompose over the actions taken to construct the AMR graph. However these require extra computations to roll-out to an end-state AMR graph for each possible action *not* taken. The large action-space of our transition system makes these algorithms computationally infeasible, and roll-outs to an end-state for many of the possible actions will provide little additional information. Hence we modify the algorithms to target this exploration to actions where the classifier being trained is uncertain of the correct response, or disagrees with the expert. This provides a further gain of 2.7 F_1 points.

This paper extends imitation learning to structured prediction tasks more complex than previously attempted. In the process, we review and compare recently proposed algorithms and show how their components can be recombined and adjusted to construct a variant appropriate to the task in hand. Hence we invest some effort reviewing these algorithms and their common elements.

Overall, we obtain a final F-Score of 0.70 on the newswire corpus of LDC2013E117 (Knight et al., 2014). This is identical to the score obtained by Wang et al. (2015a), the highest so far published. Our gain of 4.5 F_1 points from imitation learning over standard transition-based parsing is orthogonal to that of Wang et al. (2015a) from additional trained analysers, including co-reference and semantic role labellers, incorporated in the feature set. We further test on five other corpora of AMR graphs, including weblog domains, and show a consistent improvement in all cases with the application of imitation learning using DAGGER and the targeted V-DAGGER we propose here.

2 Transition-based AMR parsing

AMR parsing is an example of the wider family of structured prediction problems, in which we seek a mapping from an input $\mathbf{x} \in \mathcal{X}$ to a structured output $\mathbf{y} \in \mathcal{Y}$. Here $\mathbf{x}$ is the dependency tree, and $\mathbf{y}$ the AMR graph; both are graphs and we notationally replace $\mathbf{x}$ with s_1 and $\mathbf{y}$ with s_T, with $s_{1\ldots T} \in \mathcal{S}$. s_i are the intermediate graph configurations (states) that the system transitions through.

A transition-based parser starts with an input s_1, and selects an action $a_1 \in \mathcal{A}$, using a classifier. a_i converts s_i into s_{i+1}, i.e. $s_{i+1} = a_i(s_i)$. We term the set of states and actions $\langle s_1, a_1, \ldots a_{T-1}, s_T \rangle$ a trajectory of length T. The classifier $\hat{\pi}$ is trained to predict a_i from s_i, with $\hat{\pi}(s) = \arg\max_{a \in \mathcal{A}} \mathbf{w_a} \cdot \Phi(s)$, assuming a linear classifier and a feature function $\Phi(s)$.

We require an expert, π^*, that can indicate what actions should be taken on each s_i to reach the target (gold) end state. In problems like POS-tagging these are directly inferable from gold, as the number of actions (T) equals the number of

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			Insert a new node δ with AMR concept l_c as a child of σ_0.

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

Algorithm 1: Greedy transition-based parsing

Data: policy π, start state s_1
Result: terminal state s_T

$s_{current} \leftarrow s_1$;
while $s_{current}$ *not terminal* **do**
  $a_{next} \leftarrow \pi(s_{current})$
  $s_{current} \leftarrow a_{next}(s_{current})$
$s_T \leftarrow s_{current}$

tokens with a 1:1 correspondence between them. In dependency parsing and AMR parsing this is not straightforward and dedicated transition systems are devised.

Given a labeled training dataset $\mathcal{D}$, algorithm 1 is first used to generate a trajectory for each of the inputs ($d \in D$) with $\pi = \pi^*$, the expert from which we wish to generalise. The data produced from all expert trajectories (i.e. $\langle s_{i,d}, a_{i,d} \rangle$ for all $i \in 1 \ldots T$ and all $d \in 1 \ldots D$), are used to train the classifier $\hat{\pi}$, the learned classifier, using standard supervised learning techniques. Algorithm 1 is reused to apply $\hat{\pi}$ to unseen data. Our transition system (defining $\mathcal{A}$, $\mathcal{S}$), and feature sets are based on Wang et al. (2015b), and are not the main focus of this paper. We introduce the key concepts here, with more details in the supplemental material.

We initialise the state with the stack of the nodes in the dependency tree, root node at the bottom. 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 with one node per token). At any stage before termination some of the nodes will be labelled with words from the sentence, and others with AMR concepts. Each action manipulates the top nodes in each stack, σ_0 and β_0. We reach a terminal state when σ is empty. The objective function to maximise is the Smatch score (Cai and Knight, 2013), which calculates an F_1-Score between the predicted and gold-target AMR graphs.

Table 1 summarises the actions in $\mathcal{A}$. NextNode and NextEdge form the core action set, labelling nodes and edges respectively without changing the graph structure. Swap, Reattach and ReplaceHead change graph structure, keeping it a tree. 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 create a cycle.

The Insert/InsertBelow actions insert a new node as a parent/child of σ_0. These actions are not used in Wang et al. (2015b), but Insert is very similar to the Infer action of Wang et al. (2015a). We do not use the Reentrance action of Wang et al. (2015b), as we found it not to add any benefit. This means that the output AMR is always a tree.

Our transition system has two characteristics which provide a particular challenge: given a sentence, the trajectory length T is theoretically unbounded; and $|\mathcal{A}|$ can be of the order 10^3 to 10^4. Commonly used transition-based systems have a fixed trajectory length T, which often arises naturally from the nature of the problem. In PoS-tagging each token requires a single action, and in syntactic parsing the total size of the graph is limited to the number of tokens in the input. The lack of a bound in T here is due to Insert actions that can grow the the graph, potentially *ad infinitum*, and actions like Reattach, which can move a sub-graph repeatedly back-and-forth. The action space size is due to the size of the AMR vocabulary, which for relations (edge-labels) is restricted to about 100 possible values, but for concepts (node-labels) is almost as broad as an En-

Algorithm 2: Generic Imitation Learning

Data: data D, expert π^*, Loss function $F(s)$
Result: learned classifier C, trained policy $\hat{\pi}$

1 Initialise C_0; **for** $n = 1$ **to** N **do**
2 Initialise $E_n = \phi$;
3 $\pi_{Rollin} = RollInPolicy(\pi^*, C_{0...n-1}, n)$;
4 $\pi_{Rollout} = RollOutPolicy(\pi^*, C_{0...n-1}, n)$;
5 **for** $d \in D$ **do**
6 Predict trajectory $\hat{s}_{1:T}$ with π_{Rollin};
7 **for** $\hat{s}_t \in \hat{s}_{1:T}$ **do**
8 **foreach** $a_t^j \in Explore(\hat{s}_t, \pi^*, \pi_{Rollin})$ **do**
9 $\Phi_t^j = \Phi(d, a_t^j, \hat{s}_{1:t})$;
10 Predict $\hat{s}'_{t+1:T}$ with $\pi_{Rollout}$;
11 $L_t^j = F(\hat{s}'_T)$;
12 **foreach** j **do**
13 $ActionCost_t^j = L_t^j - \min_k L_t^k$
14 Add $(\Phi_t, ActionCost_t)$ to E_n;
15 $\hat{\pi}_n, C_n = Train(C_{1...n-1}, E_1 \dots E_n)$;

glish dictionary. The large action space and unbounded T also make beam search difficult to apply since it relies on a fixed length T with commensurability of actions at the same index on different search trajectories.

3 Imitation Learning for Structured Prediction

Imitation learning originated in robotics, training a robot to follow the actions of a human expert (Schaal, 1999; Silver et al., 2008). The robot moves from state to state via actions, generating a trajectory in the same manner as the transition-based parser of Algorithm 1.

In the imitation learning literature, the learning of a policy $\hat{\pi}$ from just the expert generated trajectories is termed "exact imitation".As discussed, it is prone to error propagation, which arises because the implicit assumption of i.i.d. inputs (s_i) during training does not hold. The states in any trajectory are dependent on previous states, and on the policy used. A number of imitation learning algorithms have been proposed to mitigate error propagation, and share a common structure shown in Algorithm 2. Table 2 highlights some key differences between them.

The general algorithm firstly applies a policy π_{RollIn} (usually the expert, π^*, to start) to the data instances to generate a set of 'RollIn' trajectories in line 6 (we adopt the terminology of 'RollIn' and 'RollOut' trajectories from Chang et al. (2015)). Secondly a number of 'what if' scenarios are considered, in which a different action a_t^j is taken from a given s_t instead of the actual a_t in the RollIn trajectory (line 8). Each of these exploratory actions generates a RollOut trajectory (line 10) to a terminal state, for which a loss (L) is calculated using a loss function, $F(s_T^j)$, defined on the terminal states. For a number of different exploratory actions taken from a state s_t on a RollIn trajectory, the action cost (or relative loss) of each is calculated (line 13). Finally the generated $\langle s_t, a_t^j, ActionCost_t^j \rangle$ data are used to train a classifier, using any cost-sensitive classification (CSC) method (line 15). New π_{RollIn} and $\pi_{RollOut}$ are generated, and the process repeated over a number of iterations. In general the starting expert policy is progressively removed in each iteration, so that the training data moves closer and closer to the distribution encountered by just the trained classifier. This is required to reduce error propagation. For a general imitation learning algorithm we need to specify:

- the policy to generate the RollIn trajectory (the *RollInPolicy*)
- the policy to generate RollOut trajectories, including rules for interpolation of learned and expert policies (the *RollOutPolicy*)
- which one-step deviations to explore with a RollOut (the *Explore* function)
- how RollOut data are used in the classification learning algorithm to generate $\hat{\pi}_i$. (within the *Train* function)

Exact Imitation can be considered a single iteration of this algorithm, with π_{RollIn} equal to the expert policy, and a 0-1 binary loss for F (0 loss for $\pi^*(s_t)$, the expert action, and a loss of 1 for any other action); all one-step deviations from the expert trajectory are considered without explicit RollOut to a terminal state.

In **SEARN** (Daumé III et al., 2009), one of the first imitation learning algorithms in this framework, the π_{RollIn} and $\pi_{RollOut}$ policies are identical within each iteration, and are a stochastic blend of the expert and all classifiers trained in previous iterations. The $Explore$ function considers every possible one-step deviation from the RollIn trajectories, with a full RollOut to a terminal state. The

Algorithm	$\hat{\pi}$	RollIn	RollOut	Explore	Train
Exact Imitation	Deterministic	Expert only	None. 0/1 expert loss	All 1-step	E_1 only
SEARN	Stochastic	Mixture	Mixture, step-level stochastic	All 1-step	E_n only
LOLS	Deterministic	Learned only	Mixture, trajectory-level stoch.	All 1-step	$E_1 \dots E_n$
SCB-LOLS	Deterministic	Learned only	Mixture, trajectory-level stoch.	Random	$E_1 \dots E_n$
SMILE	Stochastic	Mixture	None. 0/1 expert loss	All 1-step	$E_1 \dots E_n$
DAGGER	Deterministic	Mixture	None. 0/1 expert loss	All 1-step	$E_1 \dots E_n$
V-DAGGER	Deterministic	Mixture	Mixture, step-level stochastic	All 1-step	$E_1 \dots E_n$
AGGREVATE	Deterministic	Learned only	Expert only	Random	$E_1 \dots E_n$

Table 2: Comparison of selected aspects of Imitation Learning algorithms.

$Train$ function uses only the training data from the most recent iteration (E_n) to train C_n.

LOLS extends this work to provide a deterministic learned policy (Chang et al., 2015), with $\hat{\pi}_n = C_n$. At each iteration $\hat{\pi}_n$ is trained on *all* previously gathered data $E_{1\dots n}$; π_{RollIn} uses the latest classifier $\hat{\pi}_{n-1}$, and each RollOut uses the same policy for all actions in the trajectory; either π^* with probability β, or $\hat{\pi}_{n-1}$ otherwise. Both LOLS and SEARN use an exhaustive search of alternative actions as an $Explore$ function. Chang et al. (2015) consider Structured Contextual Bandits (SCB) as a partial information case, the SCB modification of LOLS permits only one cost function call per RollIn (received from the external environment), so exhaustive RollOut exploration at each step is not possible. SCB-LOLS $Explore$ picks a single step $t \in \{1 \dots T\}$ at *random* at which to make a *random* single-step deviation.

Another strand of work uses only the expert policy when calculating the action cost. Ross and Bagnell (2010) introduce **SMILE**, and later **DAGGER** (Ross et al., 2011). These do not RollOut as such, but as in exact imitation consider all one-step deviations from the RollIn trajectory and obtain a $0/1$ action cost for each by asking the expert what it would do in that state. At the nth iteration the training trajectories are generated from an interpolation of π^* and $\hat{\pi}_{n-1}$, with the latter progressively increasing in importance; π^* is used with probability (1-δ)$^{n-1}$ for some decay rate δ. $\hat{\pi}_n$ is trained using all $E_{1\dots n}$. Ross et al. (2011) discuss and reject calculating an action cost by completing a RollOut from each one-step deviation to a terminal state. Three reasons given are:

1. Lack of real-world applicability, for example in robotic control.
2. Lack of knowledge of the final loss function, if we just have the expert's actions.
3. Time spent calculating RollOuts and calling the expert.

Ross and Bagnell (2014) *do* incorporate RollOuts to calculate an action cost in their **AGGREVATE** algorithm. These RollOuts use the expert policy only, and allow a cost-sensitive classifier to be trained that can learn that some mistakes are more serious than others. As with DAGGER, the trained policy cannot become better than the expert.

V-DAGGER is the variant proposed by Vlachos and Clark (2014) in a semantic parsing task. It is the same as DAGGER, but with RollOuts using the same policy as RollIn. For both V-DAGGER and SEARN, the stochasticity of the RollOut means that a number of independent samples are taken for each one-step deviation to reduce the variance of the action cost, and noise in the training data. This noise reduction comes at the expense of the time needed to compute additional RollOuts.

4 Adapting imitation learning to AMR

Algorithms with full RollOuts have particular value in the absence of an optimal (or near-optimal) expert able to pick the best action from any state. If we have a suitable loss function, then the benefit of RollOuts may become worth the computation expended on them. For AMR parsing we have both a loss function in Smatch, and the ability to generate arbitrary RollOuts.

We therefore use a heuristic expert. This reduces the computational cost at the expense of not always predicting the best action. An expert needs an alignment between gold AMR nodes and tokens in the parse-tree or sentence to determine the actions to convert to one from the other. These alignments are not provided in the gold AMR, and our expert uses the AMR node to token alignments of JAMR (Flanigan et al., 2014). These alignments are not trained, but generated using regex and string matching rules. However, trajectories are in the range 50-200 actions for most training sentences, which combined with the size of $|\mathcal{A}|$ makes an exhaustive search of all one-step deviations expensive. Compare this to unlabeled shift-

reduce parsers with 4 actions, or POS tagging with $|\mathcal{A}| \sim 30$.

4.1 Targeted exploration

To reduce this cost we note that exploring RollOuts for all possible alternative actions can be uninformative when the learned and expert policies agree on an action and none of the other actions score highly with the learned policy. Extending this insight we modify the $Explore$ function in Algorithm 2 to only consider the expert action, plus all actions scored by the current learned policy that are within a threshold τ of the score for the best rated action. In the first iteration, when there is no current learned policy, we pick a number of actions (usually 10) at random for exploration. Both SCB-LOLS and AGGREVATE use partial exploration, but select the step $t \in 1 \dots T$, and the action a_t at random. Here we optimise computational resources by directing the search to areas for which the trained policy is least sure of the optimal action, or disagrees with the expert.

Using imitation learning to address error propagation of transition-based parsing provides theoretical benefit from ensuring the distribution of s_t, a_t in the training data is consistent with the distribution on unseen test data. Using RollOuts that mix expert and learned policies additionally permits the learned policy to exceed the performance of a poor expert. Incorporating targeted exploration strategies in the $Explore$ function makes this computationally feasible.

4.2 Noise Reduction

Different samples for a RollOut trajectory using V-DAGGER or SEARN can give very different terminal states s_T (the final AMR graph) from the same starting s_t and a_t due to the step-level stochasticity. The resultant high variance in the reward signal hinders effective learning. Daumé III et al. (2009) have a similar problem, and note that an approximate cost function outperforms single Monte Carlo sampling, "likely due to the noise induced following a single sample".

To control noise we use the α-bound discussed by Khardon and Wachman (2007). This excludes a training example (i.e. an individual tuple s_i, a_i) from future training once it has been misclassified α times in training. We find that this simple idea avoids the need for multiple RollOut samples.

An attraction of LOLS is that it randomly selects either expert or learned policy for each RollOut, and then applies this consistently to the whole trajectory. Using LOLS should reduce noise without increasing the sample size. Unfortunately the unbounded T of our transition system leads to problems if we drop the expert from the RollIn or RollOut policy mix too quickly, with many trajectories never terminating. Ultimately $\hat{\pi}$ learns to stop doing this, but even with targeted exploration training time is prohibitive and our LOLS experiments failed to provide results. We find that V-DAGGER with an α-bound works as a good compromise, keeping the expert involved in RollIn, and speeding up learning overall.

Another approach we try is a form of *focused costing* (Vlachos and Craven, 2011). Instead of using the learned policy for $\beta\%$ of steps in the RollOut, we use it for the first b steps, and then revert to the expert. This has several potential advantages: the heuristic expert is faster than scoring all possible actions; it focuses the impact of the exploratory step on immediate actions/effects so that mistakes $\hat{\pi}$ makes on a distant part of the graph do not affect the action cost; it reduces noise for the same reason. We increase b in each iteration so that the expert is asymptotically removed from RollOuts, a function otherwise supported by the decay parameter, δ.

4.3 Transition System adaptations

Applying imitation learning to a transition system with unbounded T can and does cause problems in early iterations, with RollIn or RollOut trajectories failing to complete while the learned policy, $\hat{\pi}$, is still relatively poor. To ensure every trajectory completes we add action constraints to the system. These avoid the most pathological scenarios, such as disallowing a Reattach of a previously Reattached sub-graph. These constraints are only needed in the first few iterations until $\hat{\pi}$ learns, via the action costs, to avoid these scenarios. They are listed in the Supplemental Material. As a final failsafe we insert a hard-stop on any trajectory once $T > 300$.

To address the size of $|A|$, we only consider a subset of AMR concepts when labelling a node. Wang et al. (2015b) use all concepts that occur in the training data in the same sentence as the lemma of the node, leading to hundreds or thousands of possible actions from some states. We use the smaller set of concepts that were assigned by the expert to the lemma of the current node any-

	Exact Imitation			Imitation Learning				
Experiment	No α	α=1	α-Gain	No α	α=1	IL Gain (α)	IL Gain (No α)	Total Gain
AROW, C=10	65.5	66.8	1.3	65.5	67.4	0.6	0.0	1.9
AROW, C=100	66.4	66.6	0.2	66.4	67.7	1.1	0.0	1.3
AROW, C=1000	66.4	67.0	0.6	66.5	68.2	1.2	0.1	1.8
PA, C=100	66.7	66.5	-0.2	67.2	68.7	2.2	0.5	2.0
Perceptron	65.5	65.3	-0.2	66.6	68.6	3.3	1.1	3.1

Table 3: DAGGER with α-bound. All figures are F-Scores on the validation set. 5 iterations of classifier training take place after each DAgger iteration. A decay rate (δ) for π^* of 0.3 was used.

where in the training data. We obtain these assignments from an initial application of the expert to the full training data.

We add actions to use the actual word or lemma of the current node to increase generalisation, plus an action to append '-01' to 'verbify' an unseen word. This is similar to the work of Werling et al. (2015) in word to AMR concept mapping, and is useful since 38% of the test AMR concepts do not exist in the training data (Flanigan et al., 2014).

Full details of the heuristics of the expert policy, features used and pre-processing are in Supplemental Material. All code is available at `https://github.com/hopshackle/dagger-AMR`.

4.4 Naïve Smatch as Loss Function

Smatch (Cai and Knight, 2013) uses heuristics to control the combinatorial explosion of possible mappings between the input and output graphs, but is still too computationally expensive to be calculated for every RollOut during training. We retain Smatch for reporting all final results, but use 'Naïve Smatch' as an approximation during training. This skips the combinatorial mapping of nodes between predicted and target AMR graphs. Instead, for each graph we compile a list of:

- Node labels, e.g. `name`
- Node-Edge-Node label concatenations, e.g. `leave-01:ARG0:room`
- Node-Edge label concatenations, e.g. `leave-01:ARG0`, `ARG0:room`

The loss is the number of entries that appear in only one of the lists. We do not convert to an F_1 score, as retaining the absolute number of mistakes is proportional to the size of the graph.

The flexibility of the transition system means multiple different actions from a given state s_i can lead, via different RollOut trajectories, to the same target s_T. This can result in many actions having the best action cost, reducing the signal in the training data and giving poor learning. To encourage short trajectories we break these ties with a penalty of $T/5$ to Naïve Smatch. Multiple routes of the same length still exist, and are preferred equally. Note that the ordering of the stack of dependency tree nodes in the transition system means we start at leaf nodes and move up the tree. This prevents sub-components of the output AMR graph being produced in an arbitrary order.

5 Experiments

The main dataset used is the newswire (proxy) section of LDC2014T12 (Knight et al., 2014). The data from years 1995-2006 form the training data, with 2007 as the validation set and 2008 as the test set. The data split is the same as that used by Flanigan et al. (2014) and Wang et al. (2015b). [1]

We first assess the impact of noise reduction using the alpha bound, and report these experiments without Rollouts (i.e. using DAGGER) to isolate the effect of noise reduction. Table 3 summarises results using exact imitation and DAGGER with the α-bound set to discard a training instance after one misclassification. This is the most extreme setting, and the one that gave best results. We try AROW (Crammer et al., 2013), Passive-Aggressive (PA) (Crammer et al., 2006), and perceptron (Collins, 2002) classifiers, with averaging in all cases. We see a benefit from the α-bound for exact imitation only with AROW, which is more noise-sensitive than PA or the simple perceptron. With DAGGER there is a benefit for all classifiers. In all cases the α-bound and DAGGER are synergistic; without the α-bound imitation learning works less well, if at all. α=1 was the optimal setting, with lesser benefit observed for larger values.

We now turn our attention to targeted exploration and focused costing, for which we use V-DAGGER as explained in section 4. For all V-

[1] Formally Flanigan et al. (2014; Wang et al. (2015b) use the pre-release version of this dataset (LDC2013E117). Werling et al. (2015) conducted comparative tests on the two versions, and found only a very minor changes of 0.1 to 0.2 points of F-score when using the final release.

Authors	Algorithmic Approach	R	P	F
Flanigan et al. (2014)	Concept identification with semi-markov model followed by optimisation of constrained graph that contains all of these.	0.52	0.66	0.58
Werling et al. (2015)	As Flanigan et al. (2014), with enhanced concept identification	0.59	0.66	0.62
Wang et al. (2015b)	Single stage using transition-based parsing algorithm	0.62	0.64	0.63
Pust et al. (2015)	Single stage System-Based Machine Translation	-	-	0.66
Peng et al. (2015)	Hyperedge replacement grammar	0.57	0.59	0.58
Artzi et al. (2015)	Combinatory Categorial Grammar induction	0.66	0.67	0.66
Wang et al. (2015a)	Extensions to action space and features in Wang et al. (2015b)	0.69	0.71	0.70
This work	Imitation Learning with transition-based parsing	0.68	0.73	0.70

Table 4: Comparison of previous work on the AMR task. R, P and F are Recall, Precision and F-Score.

DAGGER experiments we use AROW with regularisation parameter C=1000, and δ=0.3.

Figure 2 shows results by iteration of reducing the number of RollOuts explored. Only the expert action, plus actions that score close to the best-scoring action (defined by the threshold) are used for RollOuts. Using the action cost information from RollOuts does surpass simple DAGGER, and unsurprisingly more exploration is better.

Figure 3 shows the same data, but by total computational time spent[2]. This adjusts the picture, as small amounts of exploration give a faster benefit, albeit not always reaching the same peak performance. As a baseline, three iterations of V-DAGGER without targeted exploration (threshold = ∞) takes 9600 minutes on the same hardware to give an F-Score of 0.652 on the validation set.

Figure 4 shows the improvement using focused costing. The 'n/m' setting sets b, the number of initial actions taken by $\hat{\pi}$ in a RollOut to n, and then increases this by m at each iteration. We gain an increase of 2.9 points from 0.682 to 0.711. In all the settings tried, focused costing improves the results, and requires progressive removal of the expert to achieve the best score.

We use the classifier from the Focused Costing 5/5 run to achieve an F-Score on the held-out test set of 0.70, equal to the best published result so far (Wang et al., 2015a). Our gain of 4.7 points from imitation learning over standard transition-based parsing is orthogonal to that of Wang et al. (2015a) using exact imitation with additional trained analysers; they experience a gain of 2 points from using a Charniak parser (Charniak and Johnson, 2005) trained on the full OntoNotes corpus instead of the Stanford parser used here and in Wang et al. (2015b), and a further gain of 2 points from a semantic role labeller. Table 4 lists previous AMR work on the same dataset.

[2]experiments were run on 8-core Google Cloud n1-highmem-8 machines.

	Validation F-Score			Test F-Score	
Dataset	EI	D	V-D	V-D	Rao et al
proxy	0.670	0.686	0.704	0.70	0.61
dfa	0.495	0.532	0.546	0.50	0.44
bolt	0.456	0.468	0.524	0.52	0.46
xinhua	0.598	0.623	0.683	0.62	0.52
lpp	0.540	0.546	0.564	0.55	0.52

Table 5: Comparison of Exact Imitation (EI), DAGGER (D), V-DAGGER (V-D) on all components of the LDC2014T12 corpus.

Using DAGGER with this system we obtained an F-Score of 0.60 in the Semeval 2016 task on AMR parsing, one standard deviation above the mean of all entries. (Goodman et al., 2016)

Finally we test on all components of the LDC2014T12 corpus as shown in Table 5, which include both newswire and weblog data, as well as the freely available AMRs for *The Little Prince*, (lpp)[3]. For each we use exact imitation, DAGGER, and V-DAGGER on the train/validation/splits specified in the corpus. In all cases, imitation learning without RollOuts (DAGGER) improves on exact imitation, and incorporating RollOuts (V-DAGGER) provides an additional benefit. Rao et al. (2015) use SEARN on the same datasets, but with a very different transition system. We show their results for comparison.

Our expert achieves a Smatch F-Score of 0.94 on the training data. This explains why DAGGER, which assumes a good expert, is effective. Introducing RollOuts provides additional theoretical benefits from a non-decomposable loss function that can take into account longer-term impacts of an action. This provides much more information than the 0/1 binary action cost in DAGGER, and we can use Naïve Smatch as an approximation to our actual objective function during training. This informational benefit comes at the cost of increased noise and computational expense, which we control with targeted exploration and focused

[3]http://amr.isi.edu/download.html

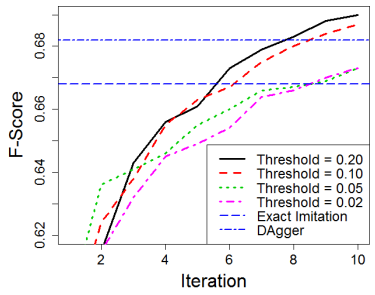

Figure 2: Targeted exploration with V-DAGGER by iteration.

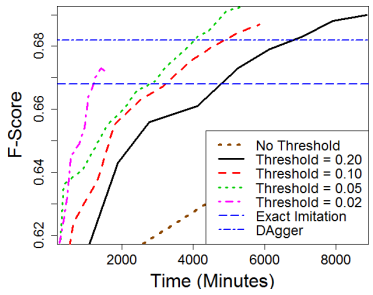

Figure 3: Targeted exploration with V-DAGGER by time.

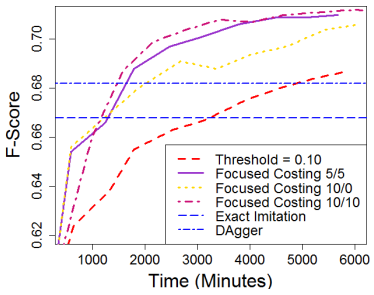

Figure 4: Focused costing with V-DAGGER. All runs use threshold of 0.10.

costing. We gain 2.7 points in F-Score, at the cost of 80-100x more computation. In problems with a less good expert, the gain from exploration could be much greater. Similarly, if designing an expert for a task is time-consuming, then it may be a better investment to rely on exploration with a poor expert to achieve the same result.

6 Related Work

Other strategies have been used to mitigate the error propagation problem in transition-based parsing. A common approach is to use beam search through state-space for each action choice to find a better approximation of the long-term score of the action, e.g. Zhang and Clark (2008). Goldberg and Elhadad (2010) remove the determinism of the sequence of actions to create *easy-first* parsers, which postpone uncertain, error-prone decisions until more information is available. This contrasts with working inflexibly left-to-right along a sentence, or bottom-to-top up a tree.

Goldberg and Nivre (2012) introduce *dynamic* experts that are *complete* in that they will respond from any state, not just those on the perfect trajectory assuming no earlier mistakes; any expert used with an imitation learning algorithm needs to be *complete* in this sense. Their algorithm takes exploratory steps off the expert trajectory to augment the training data collected in a fashion very similar to DAGGER.

Honnibal et al. (2013) use a *non-monotonic* parser that allows actions that are inconsistent with previous actions. When such an action is taken it amends the results of previous actions to ensure post-hoc consistency. Our parser is non-monotonic, and we have the same problem encountered by Honnibal et al. (2013) with many different actions from a state s_i able to reach the target s_T, following different "paths up the mountain". This leads to poor learning. To resolve this with fixed T they break ties with a monotonic parser, so that actions that do not require later correction are scored higher in the training data. In our variable T environment, adding a penalty to the size of T is sufficient (section 4.4).

Vlachos and Clark (2014) use V-DAGGER to give a benefit of 4.8 points of F-Score in a domain-specific semantic parsing problem similar to AMR. Their expert is sub-optimal, with no information on alignment between words in the input sentence, and nodes in the target graph. The parser learns to link words in the input to one of the 35 node types, with the 'expert' policy aligning completely at random. This is infeasible with AMR parsing due to the much larger vocabulary.

7 Conclusions

Imitation learning provides a total benefit of 4.5 points with our AMR transition-based parser over exact imitation. This is a more complex task than many previous applications of imitation learning, and we found that noise reduction was an essential pre-requisite. Using a simple 0/1 binary action cost using a heuristic expert provided a benefit of 1.8, with the remaining 2.7 points coming from RollOuts with targeted exploration, focused costing and a non-decomposable loss function that was a better approximation to our objective.

We have considered imitation learning algorithms as a toolbox that can be tailored to fit the characteristics of the task. An unbounded T meant that the LOLS RollIn was not ideal, but this could be modified to slow the loss of influence of the expert policy. We anticipate the approaches that we have found useful in the case of AMR to reduce the impact of noise, efficiently support large action spaces with targeted exploration, and cope with unbounded trajectories in the transition system will be of relevance to other structured prediction tasks.

Acknowledgments

Andreas Vlachos is supported by the EPSRC grant Diligent (EP/M005429/1) and Jason Naradowsky by a Google Focused Research award. We would also like to thank our anonymous reviewers for many comments that helped improve this paper.

References

Yoav Artzi, Kenton Lee, and Luke Zettlemoyer. 2015. Broad-coverage ccg semantic parsing with amr. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1699–1710, Lisbon, Portugal, September. Association for Computational Linguistics.

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, Sofia, Bulgaria, August. Association for Computational Linguistics.

Shu Cai and Kevin Knight. 2013. Smatch: an evaluation metric for semantic feature structures. In *ACL (2)*, pages 748–752.

Kai-wei Chang, Akshay Krishnamurthy, Alekh Agarwal, Hal Daumé III, and John Langford. 2015. Learning to search better than your teacher. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 2058–2066.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 173–180. 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*, pages 1–8. Association for Computational Linguistics.

Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. 2006. Online passive-aggressive algorithms. *The Journal of Machine Learning Research*, 7:551–585.

Koby Crammer, Alex Kulesza, and Mark Dredze. 2013. Adaptive regularization of weight vectors. *Mach Learn*, 91:155–187.

Deborah A Dahl, Madeleine Bates, Michael Brown, William Fisher, Kate Hunicke-Smith, David Pallett, Christine Pao, Alexander Rudnicky, and Elizabeth Shriberg. 1994. Expanding the scope of the atis task: The atis-3 corpus. In *Proceedings of the workshop on Human Language Technology*, pages 43–48. Association for Computational Linguistics.

Hal Daumé III, John Langford, and Daniel Marcu. 2009. Search-based structured prediction. *Machine learning*, 75(3):297–325.

Jeffrey Flanigan, Sam Thomson, Jaime Carbonell, Chris Dyer, and Noah A 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*, pages 1426–1436. Association for Computational Linguistics.

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.

Yoav Goldberg and Joakim Nivre. 2012. A dynamic oracle for arc-eager dependency parsing. In *COLING*, pages 959–976.

Yoav Goldberg and Joakim Nivre. 2013. Training deterministic parsers with non-deterministic oracles. *Transactions of the association for Computational Linguistics*, 1:403–414.

James Goodman, Andreas Vlachos, and Jason Naradowsky. 2016. Ucl+sheffield at semeval-2016 task 8: Imitation learning for amr parsing with an alpha-bound. In *Proceedings of the 10th International Workshop on Semantic Evaluation*.

He He, Hal Daumé III, and Jason Eisner. 2013. Dynamic feature selection for dependency parsing. In *Empirical Methods in Natural Language Processing*.

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.

Kevin Knight, Laura Baranescu, Claire Bonial, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Daniel Marcu, Martha Palmer, and Nathan Schneider. 2014. Abstract meaning representation (amr) annotation release 1.0. *Linguistic Data Consortium Catalog. LDC2014T12*.

Percy Liang, Michael I Jordan, and Dan Klein. 2013. Learning dependency-based compositional semantics. *Computational Linguistics*, 39(2):389–446.

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

Xiaochang Peng, Linfeng Song, and Daniel Gildea. 2015. A synchronous hyperedge replacement grammar based approach for amr parsing. *CoNLL 2015*, page 32.

Michael Pust, Ulf Hermjakob, Kevin Knight, Daniel Marcu, and Jonathan May. 2015. Using syntax-based machine translation to parse english into abstract meaning representation. *arXiv preprint arXiv:1504.06665*.

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*.

Stéphane Ross and Drew Bagnell. 2010. Efficient reductions for imitation learning. In *13th International Conference on Artificial Intelligence and Statistics*, pages 661–668.

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. In *14th International Conference on Artificial Intelligence and Statistics*, volume 15, pages 627–635.

Stefan Schaal. 1999. Is imitation learning the route to humanoid robots? *Trends in cognitive sciences*, 3(6):233–242.

David Silver, James Bagnell, and Anthony Stentz. 2008. High performance outdoor navigation from overhead data using imitation learning. *Robotics: Science and Systems IV, Zurich, Switzerland*.

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.

Andreas Vlachos and Mark Craven. 2011. Search-based structured prediction applied to biomedical event extraction. In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning*, pages 49–57. Association for Computational Linguistics.

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.

John M Zelle and Raymond J Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1050–1055.

Yue Zhang and Stephen Clark. 2008. A tale of two parsers: investigating and combining graph-based and transition-based dependency parsing using beam-search. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 562–571. Association for Computational Linguistics.