Computational Linguistics in the Netherlands Journal 7 (2017) 93-108 Submitted 05/2017; Published 12/2017

Neural Semantic Parsing by Character-based Translation: Experiments with Abstract Meaning Representations

Rik van Noord[*] R.I.K.VAN.NOORD@RUG.NL
Johan Bos[*] JOHAN.BOS@RUG.NL

[*] *Center for Language and Cognition Groningen (CLCG), University of Groningen, The Netherlands*

Abstract

We evaluate the character-level translation method for neural semantic parsing on a large corpus of sentences annotated with Abstract Meaning Representations (AMRs). Using a sequence-to-sequence model, and some trivial preprocessing and postprocessing of AMRs, we obtain a baseline accuracy of 53.1 (F-score on AMR-triples). We examine five different approaches to improve this baseline result: (i) reordering AMR branches to match the word order of the input sentence increases performance to 58.3; (ii) adding part-of-speech tags (automatically produced) to the input shows improvement as well (57.2); (iii) So does the introduction of super characters (conflating frequent sequences of characters to a single character), reaching 57.4; (iv) optimizing the training process by using pre-training and averaging a set of models increases performance to 58.7; (v) adding silver-standard training data obtained by an off-the-shelf parser yields the biggest improvement, resulting in an F-score of 64.0. Combining all five techniques leads to an F-score of 71.0 on holdout data, which is state-of-the-art in AMR parsing. This is remarkable because of the relative simplicity of the approach.

1. Introduction

Various approaches to open-domain semantic parsing have been proposed in the last years. What we now could refer to as "traditional" approaches are semantic parsers that use supervised learning to create a syntactic analysis on which the meaning representations are constructed, usually in a compositional way. Research in this area comprises Bos et al. (2004), Copestake et al. (2005), Butler (2010), Le and Zuidema (2012), Lewis and Steedman (2013), Bos (2015), Artzi et al. (2015), and many others. Efforts to create datasets of sentences paired with meaning representations have stimulated research in semantic parsing (Banarescu et al. 2013a, Bos et al. 2017), especially those using the formalism of Abstract Meaning Representation (AMR), for which also shared tasks have been organized (May 2016). In this article, therefore, we concentrate on semantic parsing of AMRs, because large gold-standard datasets are available and various different approaches can be compared.

In contrast to the traditional approaches mentioned above, there have been interesting attempts recently to view semantic parsing as a translation task, mapping English expressions to logical forms under supervision of some deep learning method. Dong and Lapata (2016) used sequence-to-sequence (seq2seq) and sequence-to-tree (seq2tree) neural translation models to produce logical forms from sentences for four different datasets (but not AMRs). Barzdins and Gosko (2016) used a similar method to produce AMRs in the context of the previously mentioned shared task, but the performance of their neural parser was still far below the state-of-the-art. Despite this, their method inspired other researchers to adopt this seq2seq approach (Peng et al. 2017, Konstas et al. 2017). But, even though they got substantial improvements over Barzdins and Gosko (2016), their systems still did not come close to state-of-the-art. The neural approach of Foland and Martin (2017) did reach state-of-the-art performance, but they used five bi-LSTM networks instead of a single seq2seq model.

What all these attempts have in common, and why they are fascinating, is that they completely avoid complex models of the syntactic and semantic parsing process and therefore do not rely on heavily engineered features. However, except for Barzdins and Gosko (2016), they also only use word-level input. This is interesting, because Barzdins and Gosko (2016) obtained a substantial improvement for their character-level model over their word-level model. Character-embeddings, since they were introduced by Sutskever et al. (2011), have also shown improvements in a number of areas, such as POS-tagging (Santos and Zadrozny 2014, Plank et al. 2016), text classification (Zhang et al. 2015), and, most importantly, Neural Machine Translation (Chung et al. 2016).

The aim of this article is to find out how far we can push character-level neural semantic parsing: can we reach accuracy scores comparable with traditional approaches to semantic parsing? More specifically, our objectives are (1) try to reproduce the results of Barzdins and Gosko (2016); (2) improve on their results by employing several novel techniques; and (3) investigate whether injecting linguistic knowledge can improve neural semantic parsing.

We make three main contributions. First, we introduce novel techniques to improve neural AMR parsing. Second, we show that linguistic knowledge can still contribute to neural semantic parsing. Third, we show that adding silver standard to the training data makes a considerable (positive) difference in terms of performance. Our final model reaches an F-score of 71.0, which is the current state-of-the-art in AMR parsing.

2. Method and Data

We first give a bit of background on AMRs. Then we outline the basic ideas of the character-based translation model with English sentences as input and AMRs as output. We then establish a baseline system with the aim to improve it in the next section.

2.1 Abstract Meaning Representations

In our experiments utilizing neural semantic parsing we will focus on parsing Abstract Meaning Representations (AMRs). AMRs were introduced by Banarescu et al. (2013b) and are acyclic, directed graphs that represent the meaning of a sentence. There are, in fact, three ways to display an AMR: as a graph, as a set of triples, or as a tree. An example of an AMR is shown in Figure 1, here displayed as a tree, the format that is used in the annotated corpora. The corresponding triple representation is shown in Table 1.

```
(a / affect-01
   :ARG0 (w / wave-04
            :ARG1 (h2 / heat)
            :location (c / country :wiki "France" :name (n / name :op1 "France")))
   :ARG1 (p / person
            :ARG0-of (s / strike-02
                        :mod (h / hunger-01
                                :ARG0 p))))
```

Figure 1: AMR representing the meaning of *Hunger strikers were affected by France's heat wave.*

An AMR consists of concepts that are linked to variable names with a slash. In the example above we have that `a` is an instance of the concept `affect-01`, and `p` is an instance of the concept `person` (note that the names of the variables are not important). Concepts can be related to each other by using two-place predicates, which are indicated by a colon. So, the first `:ARG0` is an ordered relation between `a` and `w`. Inverse relations are denoted by the suffix `-of`. Note that, if one concept relates to more than one other concept (for instance, in the example above, the node `a` is related to `w` via `:ARG0`, and to `p` via `:ARG1`), the order of these relations within the AMR is not important.

Table 1: The AMR of *Hunger strikers were affected by France's heat wave.* displayed as the set of instance, attribute and relation triples.

Instance	**Attribute**	**Relation**
(instance, a, affect-01)	(TOP, a, affect-01)	(ARG0, a, w)
(instance, w, wave-04)	(wiki, c, France)	(ARG1, a, p)
(instance, h2, heat)	(op1, n, France)	(location, w, c)
(instance, c, country)		(ARG1, w, h2)
(instance, n, name)		(name, c, n)
(instance, p, person)		(ARG0, s, p)
(instance, s, strike-02)		(mod, s, h)
(instance, h, hunger-01)		(ARG0, h, p)

AMRs also allow for a re-occurrence of variables: the concept `person` with variable `p` stands in a relation with `affect-01` as well as with `hunger-01`. The brackets are important, because they signal which relations belong to which concepts (the spacing used in Figure 1 is optional and is only used to increase readability). Some of the concepts have a number as suffix that indicate a specific word sense. AMRs also include proper name reference resolution by including a link to a wikipedia entry (wikification).

For evaluation purposes, AMRs are converted into triples. The triples of the AMR in Figure 1 are shown in Table 1.

The accuracy of an AMR parser is computed by precision and recall on matching triples between gold standard AMRs and system-produced AMRs, using the `SMATCH` system (Cai and Knight 2013).

For the evaluation of our experiments we use the sentences annotated with AMRs from LDC release LDC2016E25[1], consisting of 36,521 training AMRs, 1,368 development AMRs and 1,371 test AMRs.

This release also includes the PropBank frameset and comes with pre-aligned AMRs and sentences. In all results shown in this article, the models are trained on the training data. As development and test data we use the designated dev and test set from LDC2016E25, which are the exact same sets that are used in LDC2015E89. We remove HTML-tags from the input sentences, but URLs are kept in.

2.2 The Basic Translation Model

To create our sequence-to-sequence translation model, we use the OpenNMT system (Klein et al. 2017). In contrast to Peng et al. (2017) and Konstas et al. (2017), who use word-level input, we use character-level input .[2]

We train a model with bidirectional encoding and general attention (Luong et al. 2015). Since training a full model takes two to three days on a GPU, we perform a heuristic parameter search instead of an exhaustive one. We started out with a default model and changed only one parameter value in separate experiments. If we improved over the default, the setting was kept and combined with other parameter settings that improved performance. All models were only tested on the development set. Ultimately, we arrived at the settings shown in Table 2. All our described models in this paper are trained with these settings. Training is stopped 3 epochs after there is no improve-

[1] https://catalog.ldc.upenn.edu/LDC2017T10

[2] We did experiment with word-based models, but they never obtained F-scores higher than 30.0. This is in line with Peng et al. (2017) and Konstas et al. (2017), who only arrived at their final F-scores by applying extensive anonymization methods.

ment in validation perplexity on the development set anymore. The best performing model on the development set is then used to decode the test set.

Table 2: Parameter settings of the seq2seq model.

Parameter	**Value**	**Parameter**	**Value**
Layers	2	RNN type	brnn
Nodes	500	Dropout	0.3
Epochs	20–25	Vocabulary	100–200
Optimizer	sgd	Max length	750
Learning rate	0.1	Beam size	5
Decay	0.7	Replace unk	true

Following Barzdins and Gosko (2016), we do not want our model to learn the arbitrary characters that are used to represent variables. The characters itself do not carry any semantic information and are only necessary to indicate co-referring nodes. Therefore we remove all variables from the AMRs and simply duplicate co-referring nodes from the input. An example of such a preprocessed AMR is shown in Figure 2. Note that this means that we lose information, since the variables cannot be put back perfectly. We describe an approach to restore the co-referring nodes in the output in section 2.3.3. All wikification relations present in AMRs in the training set are also removed and restored in a post-processing step. Newlines present in an AMR are replaced by spaces, and multiple spaces are squeezed into single ones (so the input AMR is represented on a single line).[3]

```
(m / material
   :mod (r / raw)
   :domain (o / opium)
   :ARG1-of (u / use-01
               :ARG2 (p / make-01
                        :ARG1 (h / heroin)
                        :ARG2 o)))
```

```
(material
   :mod (raw)
   :domain (opium)
   :ARG1-of (use-01
               :ARG2 (make-01
                        :ARG1 (heroin)
                        :ARG2 (opium))))
```

Figure 2: Example of the original AMR (left) and the variable-free AMR (right) displaying the meaning of *Opium is the raw material used to make heroin.*

2.3 Postprocessing and Restoring Information

The output of the seq2seq model is, of course, an AMR without variables, without wiki-links, and without co-occurrent variables. Furthermore, because of the character-based seq2seq model, it could well be that there are brackets in the output that do not match, or that some nodes representing concepts are incomplete. This, obviously, needs to be fixed.

First, the variables in the AMRs are restored by assigning a unique variable to each concept. We also try to fix invalidly produced AMRs by applying a few heuristics, such as inserting parentheses and quotes, or by removing unfinished nodes. This is done by using the restoring script from Barzdins and Gosko (2016).[4] Then, we apply three methods to increase the quality of the AMRs. They are described below.

[3] All pre- and post-processing scripts are available at https://github.com/RikVN/AMR

[4] Taken from https://github.com/didzis/tensorflowAMR/

2.3.1 Pruning

A problem with many deep learning approaches is the fact that the decoder does not keep track of what it has already produced. As a consequence, we sometimes end up with duplicated, redundant material in our generated AMRs. This hurts precision. We propose four different methods to remove this redundant material. This is done on node level, where nodes are defined as relation-concept pairs without children, e.g. `:mod (raw)` and `:domain (opium)`.

Table 3: Statistics of the different pruning methods. Methods were applied on the output of our baseline model on the dev set.

Model	**Nodes pruned**	**AMRs changed**	**F-score**
Baseline	0	0	54.8
Removing all re-occurrent nodes	1426	689	55.4
Removing re-occurrent nodes with same parent	135	95	55.0
Removing re-occurrent nodes with frequency >2	427	249	55.3
Removing all re-occurrent nodes with same parent, but also nodes with frequency >2	496	302	55.5

The statistics of applying these four methods on our baseline model (dev set) are shown in Table 3. Note that all these processes are trade-offs: usually duplicates are correctly recognized as redundant and can be removed, but sometimes we erroneously remove actual re-occurrent nodes.

The first method simply removes all re-occurrent nodes and is already quite effective: F-score increases by 0.6. The second method is more careful and only removes duplicate nodes if they have the same parent. This helps, but only by a small margin. The third method does not consider parent nodes, but removes nodes if they occur more than twice in the full AMR. This method also increases the F-score, but does not outperform the first method yet. The fourth method is a combination of the second and third method. All re-occurrent nodes with the same parent are removed, but also nodes occurring more than twice are removed. This results in the best F-score, an increase of 0.7 over the baseline. Two example AMRs whose branches are pruned using the fourth method are shown in Figure 3.

```
(material
       :mod (raw)
       ~~:mod (raw)~~
       :domain (opium)
       :ARG1-of (use-01
              :ARG2 (make-01
                     :ARG1 (heroin)
                     :ARG2 (opium))))
```

```
(material
       :mod (raw)
       :domain (opium)
              :mod (raw)
       :ARG1-of (use-01
              :ARG2 (make-01
                     :ARG1 (heroin)
                     ~~:mod (raw)~~
                     :ARG2 (opium))))
```

Figure 3: Example of pruned branches for the produced AMRs of *Opium is the raw material used to make heroin.* In the left AMR, the second occurrence of `:mod (raw)` is already removed, because both branches are children of `material`. However, in the right AMR, none of the `:mod (raw)` branches share the same parent, so only the third occurrence is removed.

2.3.2 Wikification

Since we removed wikification relations in preprocessing, our model will never output such a link. We restore wiki links in the output AMR by using an off-the-shelf system (Daiber et al. 2013), following the method presented by Bjerva et al. (2016). They look at the `:name` relations in an AMR and try to find this name on Wikipedia. If it has a page, the corresponding link gets added; otherwise the AMR remains unaltered.

2.3.3 Restoring co-referring nodes

Our system also tries to restore co-referring nodes. If we output a duplicate node (a node already produced for this AMR), it replaces the node by the variable name of the node encountered first. This can only happen once per unique node, since the third instance of such a node is already removed in the pruning phase. An example of how the co-referring nodes are restored is shown in Figure 4.

```
(m / material
       :mod (r / raw)
       :domain (o / opium)
              :mod (r2 / raw)
       :ARG1-of (u / use-01
              :ARG2 (m2 / make-01
                     :ARG1 (h / heroin)
                     :ARG2 (o2 / opium))))
```

```
(m / material
       :mod (r / raw)
       :domain (o / opium)
              :mod r
       :ARG1-of (u / use-01
              :ARG2 (m2 / make-01
                     :ARG1 (h / heroin)
                     :ARG2 o)))
```

Figure 4: Example of how co-referring nodes are restored. On the left an example of a produced AMR, on the right the AMR with co-reference restored.

2.4 Baseline Results

Our first objective was to reproduce the results obtained by Barzdins and Gosko (2016). We did so, arriving at an F-score of 53.1 (see Table 4). Compared to the F-score of 43.0 by Barzdins and Gosko (2016), our score is significantly higher. This is probably due to the higher amount of training data and the fact that they used Tensorflow instead of OpenNMT. We also reproduced their results by using the exact same data, software and parameter settings as they did, obtaining an F1-score of 42.3.[5]

Table 4: Baseline Results Semantic Parsing on LDC2016E25.

	Type	**Dev**	**Diff**	**Test**	**Diff**
Baseline	seq2seq	54.8		53.1	
Post-processing	Pruning	55.5	+ 0.7	53.7	+ 0.6
	Restoring Co-reference	55.7	+ 0.9	54.2	+ 1.1
	Wikification	55.8	+ 1.0	54.1	+ 1.0
All post-processing		57.3	+2.5	55.5	+ 2.4

As is shown in Table 4, concept pruning, restoring co-reference variables, and wikification all increase the F-score by about a percentage point each. This small gain of performance is what one could expect as each single operation has only a small impact on the overall contents of an AMR.

[5] We did not possess their Wikification and coreference restoring scripts, so differences might be attributed to that.

3. Improving the Basic Translation Model

In the previous section we outlined our basic method of producing AMRs using a seq2seq model based on characters. In this section, we look at five different techniques to move beyond the F-score that we obtain with our basic method, that we will consider in this section as baseline. Some of the techniques were already (briefly) introduced in van Noord and Bos (2017).

3.1 AMR Re-ordering

Although AMRs are unordered by definition, in our textual representation of the AMRs there is an order of the branches. However, these branches do not necessarily follow the word order in the corresponding English sentence. It has been shown that for (statistical) machine translation reordering improves translation quality (Collins et al. 2005). We use the provided alignments to permute the AMR in such a way that it best matches the word order. We do this both on sub-tree level and on individual node level. The best matching AMR is defined as the AMR in which the order of the nodes (when traversing over the AMR depth-first) is the closest to the order of the words in the English sentence, following the alignments. An example of an AMR with a branch order best matching the input sentence is shown in Figure 5.

```
(material                                   (material
   :mod (raw)                                  :domain (opium)
   :domain (opium)                             :mod (raw)
   :ARG1-of (use-01                            :ARG1-of (use-01
               :ARG2 (make-01                              :ARG2 (make-01
                          :ARG1 (heroin)                           :ARG2 (opium)
                          :ARG2 (opium))))                         :ARG1 (heroin))))
```

Figure 5: Example of a variable-free AMR before (left) and after re-ordering (right) for the sentence *Opium is the raw material used to make heroin.*

We are also able to use this approach to augment the training data, since each reordering of the AMR provides us with a new AMR-sentence pair. Due to the exponential increase, large AMRs often have thousands of possible orders. We performed a number of experiments to find out how we could best exploit this surplus of data. Ultimately, we found that it is most beneficial to "double" the training data by adding the best matching AMR to the existing data set.[6]

3.2 Introducing Super Characters

We are not necessarily restricted to only using characters as input. For example, we can view the AMR relations (e.g. `:ARG0`, `:mod`) as atomic instead of a set of characters. This ensures that the characters for relations (e.g. *m*, *o* and *d* for `:mod`) do not influence the general character embeddings of the concepts, which might improve performance. This way, we create a hybrid model that is a combination of word and character level input. An example of the AMR and sentence level input using super characters is shown in Figure 6 and Figure 7.

We also tried various ways to explicitly encode the tree structure by using super characters. In our basic model, the parentheses *'('* and *')'* are simply characters. This means that the model cannot

[6] Instead of ordering the AMR nodes reflected by the word order of sentence, we also tried two different experiments based on consistency. The first experiment simply ordered the nodes alphabetically, without any other influence. This decreased the result of our baseline model by 2.0. Our second experiment was focused on fixing irregularities: if two nodes occur in a different order than they usually do (based on the full training set), we simply switch them around. This method did not change the order as considerably as the alphabetical ordering, but the result of the baseline model still decreased by 1.0. Hence we discarded both reordering techniques.

AMR, chars:	(	t	h	i	n	g	+	:	q	u	a	n	t	+	1	+	:	p	o	l	a	r	i	t	y	+	-	)
AMR, super chars:	(	t	h	i	n	g	+	:quant	+	1	+	:polarity	+	-	)													

Figure 6: Input for the AMR `(t / thing :quant 1 :polarity -)` representing the sentence *Not one thing*, with and without super characters. The +-symbols represent spaces.

sentence:	I	+	a	m	+	n	o	t	+	t	h	a	t	+	r	i	c	h	+	.					
sentence + POS:	I	PRP	+	a	m	VBP	+	n	o	t	RB	+	t	h	a	t	IN	+	r	i	c	h	JJ	+	.

Figure 7: Input for the sentence *I am not that rich*, without and with POS-tags. POS-tags are inserted as super characters. The +-symbols represent spaces (word boundaries).

differentiate between a parenthesis that opens the full AMR and a parenthesis that opens, say, the fifth subtree of the AMR. One would expect it would help the model if it has this information explicitly encoded in the input. For example, in an experiment we replaced each parenthesis in the structure by a super character that also provides the subtree information (e.g., an opening parenthesis on the fifth level becomes `*5*(`, while a closing bracket on the third level becomes `*3*)`. However this resulted in an F-score lower than the baseline and we discarded the technique.

3.3 Adding Part-of-Speech Information

We might still be able to benefit from syntactic information, even though we use a character-level neural semantic parser. To show this, we parse the sentences with the POS-tagger of the C&C tools (Clark et al. 2003), employing the Penn POS tagset. Each tag is represented as a single character and placed after the last character representation of the word that matches the tag (see Figure 7). Put differently, we create a new super character for each unique tag and add this to the input sentence. On the one hand, this will increase the size of the input. On the other hand, just a single character will add a lot of general, potentially useful, information. For example, proper nouns correlate with the `:name` relation, while adjectives correlate with the `:mod` relation.

3.4 Adding Silver Standard Data

A problem with neural parsing approaches is data sparsity, since a lot of manual effort is required to create gold standard data. Peng et al. (2017) tried to overcome this by extensive generalization of the training data, but did not get near state-of-the-art results. Konstas et al. (2017) applied a similar method, but also used the GigaWord corpus to self-train their system. They use their own pre-trained parser to parse the previously unseen sentences and add those to the training data in a series of iterations. Ultimately, their system is trained on 20 million additional data AMR-sentence pairs and obtains an F-score of 62.1. Without this additional data, they obtain a score of 55.5, which is better than Peng et al. (2017), but not close to state-of-the-art performance.

Our method of obtaining new training data mainly differs from Konstas et al. (2017) in two ways: (i) we use two off-the-shelf parsers to create the training data instead of self-training; (ii) we employ a method to exclude lower-quality AMRs instead of using all available data. We therefore refer to this data as "silver standard" data, by which we mean something in between unchecked automatically produced data and gold standard data.

Instead of self-training our parser, we use the off-the-shelf AMR parsers CAMR (Wang et al. 2015) and JAMR (Flanigan et al. 2014) to create silver standard data for our system. Both are non-neural, syntax-based parsers. CAMR works by first generating a dependency tree for the English sentence, after which it uses a transition-based algorithm to create the AMR graph. JAMR is the first published AMR parser and does the parsing in two stages: first identifying the concepts by using a semi-Markov model, and then identifying the relations between these concepts by searching for the maximum spanning connected subgraph.

Both systems are trained on the LDC2015E86 AMR corpus, which contains 16,833 training instances. We parse 1,303,419 sentences from the Groningen Meaning Bank (Basile et al. 2012), which mainly consists of newswire text. AMRs that are either invalid or include `null-tag` or `null-edge` (this is what the CAMR parser outputs when it is not able to find a suitable candidate parse) are removed.

We do not simply add the other AMRs to our data set. To ensure that the AMRs are at least of decent quality, we compare the produced AMRs with each other using `SMATCH` (Cai and Knight 2013). If their pairwise score does not exceed 55.0, the AMRs are not considered for adding to our training set. This value was picked to filter out AMRs that would only hurt the training process, but to also still include a large variety of AMRs and sentences. Our final set contained 530,450 sentences, that have both a CAMR and JAMR parse.

We now have to determine which AMR to add to our silver data set. CAMR produces higher quality AMRs in general (64.0 vs 55.0 on the test set), but it might be beneficial to introduce some variety by also adding JAMR-parsed AMRs. We never add both CAMR and JAMR for the same sentence. We performed five experiments in which we added 100k silver AMRs, either containing 100%, 75%, 67%, 50% or 0% CAMR-parsed AMRs. The results of testing on the development set are shown in Table 5.

Table 5: F-scores on the dev set for adding different ratios of CAMR and JAMR parsed AMRs to our initial data set. All scores are without postprocessing improvement methods.

# CAMR AMRs	# JAMR AMRs	F-score
100,000	0	65.8
75,000	25,000	65.8
66,667	33,333	65.7
50,000	50,000	65.3
0	100,000	61.4

As would be expected, we see that only adding CAMR scores considerably better than only adding JAMR. However, the scores for adding 67% and 75% CAMR are very similar to adding 100% CAMR. But, since this does not indicate that adding JAMR actually helps performance, we only add the CAMR-parsed AMRs in our silver data experiments. We randomly selected 20k, 50k, 75k, 100k and 500k instances for these experiments.

3.5 Optimizing training

Aside from the pre- and post-processing methods described, we can also optimize the training process itself. The first method we employ is pre-training on our full data set including silver AMRs, after which the model is fine-tuned on the gold data only. Both phases use the same parameter settings, as experiments with different learning rates resulted in lower performance. A similar procedure was used by Konstas et al. (2017) and in general this is a method widely used in Neural Machine Translation (Denkowski and Neubig 2017).

The second method is averaging a set of models to decode the test set, instead of using a single model. This was first applied by Junczys-Dowmunt et al. (2016) as an alternative to the usual ensembling of models, which is known to give substantial improvements in Neural Machine

Translation (Sutskever et al. 2014). Ensembling, however, is very resource intensive, since the predictions of different models are averaged at decoding time. This as opposed to averaging, where the parameters of models are averaged to create a single model. This means that averaging, say, four models is four times faster than ensembling four models, while also using only a quarter of the memory the ensemble method uses. We tested with both ensembling and averaging and obtained similar results on the development set, thus opting to only use averaging in our experiments.

4. Results and Discussion

Table 6 shows the results of our improvement methods in isolation, meaning that only that individual method is added to our baseline model. Re-ordering has a clear positive effect, both for using the best re-ordering (+2.0) and adding that re-ordering to the existing data set (+5.2). Constructing super characters and adding POS-tags both lead to a similar increase in performance. Pre-training and subsequently fine-tuning also results in a substantial improvement, but creating an average model only has a slight positive effect. The biggest improvement comes from adding silver standard data to our training set, reaching a maximum of 65.8 on the dev set. However, there is a limit with regards to adding silver data, since adding 500k silver AMRs performed worse than adding 50k, 75k or 100k silver AMRs. Finding the optimal number of silver AMRs is difficult due to the long training times and is therefore left for future work.

Table 6: Results of the improvements in isolation (without post-processing).

	Type	**Dev**	**Diff**	**Test**	**Diff**
Baseline	seq2seq	54.8		53.1	
AMR Re-ordering	Best	56.8	+ 2.0	55.1	+ 2.0
	Doubling	60.0	+ 5.2	58.3	+ 5.2
Introducing Super Characters	Relations	58.3	+ 3.5	57.4	+ 4.3
Adding POS Tags	PTB	58.2	+ 3.4	57.2	+ 4.1
Training optimization	Averaging	54.9	+ 0.1	53.4	+ 0.3
	Pre-training	59.4	+ 4.6	58.6	+ 5.5
	Both	59.5	+ 4.7	58.7	+ 5.6
Adding Silver Standard Data	Adding 20k	62.2	+ 7.4	60.0	+ 6.9
	Adding 50k	64.7	+ 9.9	62.9	+ 9.8
	Adding 75k	65.7	+ 10.9	63.7	+ 10.6
	Adding 100k	65.8	+ 11.0	64.0	+ 10.9
	Adding 500k	63.8	+ 9.0	62.1	+ 9.0

Since the previous experiments were all in isolation, we now test whether a combination of our methods still increases performance. The tested combinations are shown in Table 7. Even after adding the silver data, the addition of POS-tags and super characters still increased the performance, albeit by a smaller margin. Interestingly, the best result (71.0) was not obtained by combining all improvement methods, since re-ordering the AMRs does not show an increase anymore after adding POS-tags and super characters. The best model without using any silver data obtains an F-score of 64.0, which is considerably higher than the AMR-only score (55.5) of Konstas et al. (2017).

Table 8 shows the results of the most notable previous AMR parsing systems. Our best model outperforms all these previous parsers and reaches state-of-the-art results. However, we are also the first approach that uses the LDC2016E25 data set, which contains slightly more than double the

Table 7: F-scores for our neural models, combining the different improvement methods.

Post-proc	Adding 100k Silver	POS tags	Super Chars	Re-ordering Best	Optimize Training	Dev	Test
✖	✖	✖	✖	✖	✖	54.8	53.1
✔	✖	✖	✖	✖	✖	57.3	55.5
✔	✖	✔	✔	✔	✔	65.1	64.0
✔	✔	✖	✖	✖	✖	68.0	66.4
✔	✔	✔	✖	✖	✖	68.9	67.3
✔	✔	✔	✔	✖	✖	70.4	69.0
✔	✔	✔	✔	✔	✖	69.0	68.0
✔	✔	✔	✔	✖	✔	71.9	71.0

number of gold standard training instances compared to the LDC2015E86 data set.[7] Therefore, we also trained the best performing model in Table 7 on the LDC2015E86 data set, while still applying all our improvement methods. This model still obtains an F-score of 68.5, outperforming all previous AMR parsers, except for the parser of Foland and Martin (2017).

Table 8: F-scores for AMR parsing. Comparison with previously published results on the test set.

Authors	Model	Train set (gold)	F-score
Flanigan et al. (2014)	JAMR-14	LDC2013E117	58.0
Damonte et al. (2017)	AMR-eager	LDC2015E86	64.0
Artzi et al. (2015)	CCG parsing	LDC2014T12	66.3
Wang et al. (2015)	CAMR	LDC2015E86	66.5
Flanigan et al. (2016)	JAMR-16	LDC2015E86	67.0
Pust et al. (2015)	SBMT	LDC2015E86	67.1
Barzdins and Gosko (2016)	char-based seq2seq	LDC2015E86	43.0
Peng et al. (2017)	word-based seq2seq	LDC2015E86	52.0
Konstas et al. (2017)	word-based seq2seq	LDC2015E86	55.5
Konstas et al. (2017)	word-based seq2seq + giga	LDC2015E86	62.1
Foland and Martin (2017)	5 bi-LSTM networks (word-based)	LDC2015E86	70.7
This article	char-based seq2seq model + silver	LDC2015E86	68.5
This article	char-based seq2seq model + silver	LDC2016E25	71.0

Damonte et al. (2017) presented a way to evaluate system output in a more detailed way, by focussing on various aspects that are present in an AMR: the role labelling, word sense disambiguation, named entity recognition, wikification, detecting negation, and so on. These detailed results of our best system are shown in Table 9, in which the results of the other parsers are taken from Damonte et al. (2017). Unfortunately, Foland and Martin (2017) did not publish these specific scores. As the table shows, our system scores higher than the other parsers on five of the eight metrics other than Smatch. In general, our system is quite conservative, obtaining a higher precision than recall for each metric. Given the results in Table 9, one would think that detecting negation and reentrancy would be ways to get an improvement in accuracy. Note that the other parsers score also relatively bad at these metrics. Compared to the other systems, our system scores worse on concepts, named entities, and wikification. A possible method to increase performance in the first two of those metrics

[7] LDC2015E86 only contains 16,833 instances, as opposed to the 36,521 of LDC2016E25.

is to adopt an anonymization or generalization approach for named entities and concepts, similar to Peng et al. (2017) or Konstas et al. (2017).

Table 9: Comparison with previous parsers using the evaluation script of Damonte et al. (2017). We also included precision and recall scores for our system.

	CAMR	**JAMR-16**	**AMR-eager**	**Our system**		
Metric	**F**	**F**	**F**	**Pr.**	**Rec.**	**F**
Smatch	63	67	64	76	67	71
Unlabeled	69	69	69	79	70	**74**
No WSD	64	68	65	76	67	**72**
Reentrancy	41	42	41	57	48	**52**
Concepts	80	**83**	**83**	87	78	82
Named entities	75	79	**83**	83	76	79
Wikification	0	**75**	64	82	54	65
Negations	18	45	48	67	58	**62**
SRL	60	60	56	70	62	**66**

5. Conclusion and Future Work

Applying re-ordering of AMR branches, introducing super characters, and adding POS-tags are techniques that substantially improve neural AMR parsing using a character-based seq2seq model. However, the biggest increase of performance is triggered by adding a large quantity of silver standard AMRs produced by existing (traditional) parsers. This is in line with the findings of Konstas et al. (2017), who used the Gigaword corpus to get extra training data, although their training method is different from ours.

The obtained results are promising. Our best model, with an F-score of 71.0, outperformed any known previously published result on AMR parsing. This is remarkable, for traditional approaches are often based on extensive, manually crafted lexicons using linguistic knowledge. It should be noted, of course, that we use some linguistic knowledge in the form of POS-tags in our best models, and that we employ existing parsers trained on extensive linguistics annotations. In fact, one could consider the use of silver standard AMR data as a disadvantage, as there is still a need of an existing high-quality AMR parser to get the silver data in the first place. In our approach we rely even on two different off-the-shelf parsers. It would therefore be interesting to explore other opportunities, such as self-learning, as proposed by Konstas et al. (2017).

We have the feeling that there are still a lot of techniques that one could try to increase the performance of neural AMR parsing. From a more esthetical perspective, it would be nice if one could eliminate the AMR repair strategies that are used to resolve unbalanced brackets. An interesting candidate that could master this problem would be the seq2tree model presented by Dong and Lapata (2016). Similarly, a more principled approach to deal with co-occurring variables would be desirable.

Another possible next step in semantic parsing is to change the target meaning representation. AMRs are unscoped meaning representations, and have no quantifiers. It would be challenging to transfer the techniques of neural semantic parsing to scoped meaning representations, such as those used in the Groningen Meaning Bank (Basile et al. 2012) or the Parallel Meaning Bank (Abzianidze et al. 2017).

Acknowledgements

First of all we would like to thank Antonio Toral and Lasha Abzianidze for helpful discussion on neural AMR parsing and machine translation. We thank the three anonymous reviewers for their comments. We would also like to thank the Center for Information Technology of the University of Groningen for their support and for providing access to the Peregrine high performance computing cluster. We also used a Tesla K40 GPU, which was kindly donated to us by the NVIDIA Corporation. This work was funded by the NWO-VICI grant "Lost in Translation Found in Meaning" (288-89-003).

References

Abzianidze, Lasha, Johannes Bjerva, Kilian Evang, Hessel Haagsma, Rik van Noord, Pierre Ludmann, Duc-Duy Nguyen, and Johan Bos (2017), The Parallel Meaning Bank: Towards a Multilingual Corpus of Translations Annotated with Compositional Meaning Representations, *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, Association for Computational Linguistics, Valencia, Spain, pp. 242–247.

Artzi, Yoav, Kenton Lee, and Luke Zettlemoyer (2015), Broad-coverage CCG Semantic Parsing with AMR, *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, Lisbon, Portugal, pp. 1699–1710.

Banarescu, Laura, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf H̃ermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider (2013a), Abstract Meaning Representation for Sembanking, *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, Sofia, Bulgaria, pp. 178–186. http://www.aclweb.org/anthology/W13-2322.

Banarescu, Laura, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider (2013b), Abstract Meaning Representation for Sembanking, *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, Association for Computational Linguistics, Sofia, Bulgaria, pp. 178–186.

Barzdins, Guntis and Didzis Gosko (2016), RIGA at SemEval-2016 Task 8: Impact of Smatch Extensions and Character-Level Neural Translation on AMR Parsing Accuracy, *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, Association for Computational Linguistics, San Diego, California.

Basile, Valerio, Johan Bos, Kilian Evang, and Noortje Venhuizen (2012), Developing a large semantically annotated corpus, *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC 2012)*, Istanbul, Turkey, pp. 3196–3200.

Bjerva, Johannes, Johan Bos, and Hessel Haagsma (2016), The Meaning Factory at SemEval-2016 Task 8: Producing AMRs with Boxer, *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, Association for Computational Linguistics, San Diego, California, pp. 1179–1184.

Bos, Johan (2015), Open-Domain Semantic Parsing with Boxer, *in* Megyesi, Beáta, editor, *Proceedings of the 20th Nordic Conference of Computational Linguistics (NODALIDA 2015)*, pp. 301–304.

Bos, Johan, Stephen Clark, Mark Steedman, James R. Curran, and Julia Hockenmaier (2004), Wide-Coverage Semantic Representations from a CCG Parser, *Proceedings of the 20th International Conference on Computational Linguistics (COLING '04)*, Geneva, Switzerland, pp. 1240–1246.

Bos, Johan, Valerio Basile, Kilian Evang, Noortje Venhuizen, and Johannes Bjerva (2017), The Groningen Meaning Bank, *in* Ide, Nancy and James Pustejovsky, editors, *Handbook of Linguistic Annotation*, Vol. 2, Springer, pp. 463–496.

Butler, Alastair (2010), *The Semantics of Grammatical Dependencies*, Vol. 23, Emerald Group Publishing Limited.

Cai, Shu and Kevin Knight (2013), Smatch: an Evaluation Metric for Semantic Feature Structures, *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, Association for Computational Linguistics, Sofia, Bulgaria, pp. 748–752.

Chung, Junyoung, Kyunghyun Cho, and Yoshua Bengio (2016), NYU-MILA Neural Machine Translation Systems for WMT16, *Proceedings of the First Conference on Machine Translation*, Association for Computational Linguistics, Berlin, Germany, pp. 268–271.

Clark, Stephen, James R Curran, and Miles Osborne (2003), Bootstrapping POS taggers using unlabelled data, *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, Association for Computational Linguistics, pp. 49–55.

Collins, Michael, Philipp Koehn, and Ivona Kučerová (2005), Clause restructuring for statistical machine translation, *Proceedings of the 43rd annual meeting on association for computational linguistics*, Association for Computational Linguistics, pp. 531–540.

Copestake, Ann, Dan Flickinger, Ivan Sag, and Carl Pollard (2005), Minimal Recursion Semantics: An introduction, *Journal of Research on Language and Computation* **3** (2–3), pp. 281–332.

Daiber, Joachim, Max Jakob, Chris Hokamp, and Pablo N. Mendes (2013), Improving Efficiency and Accuracy in Multilingual Entity Extraction, *Proceedings of the 9th International Conference on Semantic Systems (I-Semantics)*.

Damonte, Marco, Shay B. Cohen, and Giorgio Satta (2017), An Incremental Parser for Abstract Meaning Representation, *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, Association for Computational Linguistics, Valencia, Spain, pp. 536–546.

Denkowski, Michael and Graham Neubig (2017), Stronger Baselines for Trustable Results in Neural Machine Translation, *Proceedings of the First Workshop on Neural Machine Translation*, Association for Computational Linguistics, Vancouver, pp. 18–27.

Dong, Li and Mirella Lapata (2016), Language to Logical Form with Neural Attention, *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Association for Computational Linguistics, Berlin, Germany, pp. 33–43.

Flanigan, Jeffrey, Chris Dyer, Noah A Smith, and Jaime Carbonell (2016), CMU at SemEval-2016 task 8: Graph-based AMR parsing with infinite ramp loss, *Proceedings of SemEval* pp. 1202–1206.

Flanigan, Jeffrey, Sam Thomson, Jaime Carbonell, Chris Dyer, and Noah A. Smith (2014), A Discriminative Graph-Based Parser for the Abstract Meaning Representation, *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Association for Computational Linguistics, Baltimore, Maryland, pp. 1426–1436.

Foland, William and James H. Martin (2017), Abstract Meaning Representation Parsing using LSTM Recurrent Neural Networks, *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Association for Computational Linguistics, Vancouver, Canada, pp. 463–472.

Junczys-Dowmunt, Marcin, Tomasz Dwojak, and Rico Sennrich (2016), The AMU-UEDIN Submission to the WMT16 News Translation Task: Attention-based NMT Models as Feature Functions in Phrase-based SMT, *Proceedings of the First Conference on Machine Translation*, Association for Computational Linguistics, Berlin, Germany, pp. 319–325. http://www.aclweb.org/anthology/W16-2316.

Klein, Guillaume, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M Rush (2017), OpenNMT: Open-Source Toolkit for Neural Machine Translation, *arXiv preprint arXiv:1701.02810*.

Konstas, Ioannis, Srinivasan Iyer, Mark Yatskar, Yejin Choi, and Luke Zettlemoyer (2017), Neural AMR: Sequence-to-Sequence Models for Parsing and Generation, *arXiv preprint (accepted in ACL-2017) arXiv:1704.08381*.

Le, Phong and Willem Zuidema (2012), Learning Compositional Semantics for Open Domain Semantic Parsing, *Proceedings of COLING 2012*, Vol. 1 of *COLING 2012*, Association for Computational Linguistics, p. 3350.

Lewis, Mike and Mark Steedman (2013), Combined distributional and logical semantics, *Transactions of the Association of Computational Linguistics – Volume 1* pp. 179–192.

Luong, Thang, Hieu Pham, and Christopher D. Manning (2015), Effective Approaches to Attention-based Neural Machine Translation, *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, Lisbon, Portugal, pp. 1412–1421.

May, Jonathan (2016), SemEval-2016 Task 8: Meaning Representation Parsing, *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, San Diego, California, pp. 1063–1073.

Peng, Xiaochang, Chuan Wang, Daniel Gildea, and Nianwen Xue (2017), Addressing the Data Sparsity Issue in Neural AMR Parsing, *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, Association for Computational Linguistics, Valencia, Spain, pp. 366–375.

Plank, Barbara, Anders Søgaard, and Yoav Goldberg (2016), Multilingual Part-of-Speech Tagging with Bidirectional Long Short-Term Memory Models and Auxiliary Loss, *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, Association for Computational Linguistics, Berlin, Germany, pp. 412–418. http://anthology.aclweb.org/P16-2067.

Pust, Michael, Ulf Hermjakob, Kevin Knight, Daniel Marcu, and Jonathan May (2015), Parsing English into abstract meaning representation using syntax-based machine translation, *Training* **10**, pp. 218–021.

Santos, Cicero D and Bianca Zadrozny (2014), Learning character-level representations for part-of-speech tagging, *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pp. 1818–1826.

Sutskever, Ilya, James Martens, and Geoffrey E Hinton (2011), Generating text with recurrent neural networks, *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 1017–1024.

Sutskever, Ilya, Oriol Vinyals, and Quoc V Le (2014), Sequence to sequence learning with neural networks, *Advances in neural information processing systems*, pp. 3104–3112.

van Noord, Rik and Johan Bos (2017), The Meaning Factory at SemEval-2017 Task 9: Producing AMRs with Neural Semantic Parsing, *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, Association for Computational Linguistics, Vancouver, Canada, pp. 929–933.

Wang, Chuan, Nianwen Xue, and Sameer Pradhan (2015), A Transition-based Algorithm for AMR Parsing, *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Association for Computational Linguistics, Denver, Colorado, pp. 366–375.

Zhang, Xiang, Junbo Zhao, and Yann LeCun (2015), Character-level convolutional networks for text classification, *Advances in neural information processing systems*, pp. 649–657.