Semantics as a Foreign Language

Gabriel Stanovsky*[2,3] and Ido Dagan[1]

[1]Bar-Ilan University Computer Science Department, Ramat Gan, Israel
[2]Paul G. Allen School of Computer Science & Engineering, University of Washington, Seattle, WA
[3]Allen Institute for Artificial Intelligence, Seattle, WA
gabis@cs.washington.edu
dagan@cs.biu.ac.il

Abstract

We propose a novel approach to semantic dependency parsing (SDP) by casting the task as an instance of multi-lingual machine translation, where each semantic representation is a different foreign dialect. To that end, we first generalize syntactic linearization techniques to account for the richer semantic dependency graph structure. Following, we design a neural sequence-to-sequence framework which can effectively recover our graph linearizations, performing almost on-par with previous SDP state-of-the-art while requiring less parallel training annotations. Beyond SDP, our linearization technique opens the door to integration of graph-based semantic representations as features in neural models for downstream applications.

1 Introduction

Many sentence-level representations were developed with the goal of capturing the sentence's proposition structure and making it accessible for downstream applications (Montague, 1973; Carreras and Màrquez, 2005; Banarescu et al., 2013; Abend and Rappoport, 2013). See Abend and Rappoport (2017), for a recent survey.

While syntactic grammars (Marcus et al., 1993; Nivre, 2005) induce a rooted tree structure over the sentence by connecting verbal predicates to their arguments, these semantic representations often take the form of the more general labeled graph structure, and aim to capture a wider notion of propositions (e.g, nominalizations, adjectivals, or appositives). In particular, we will focus on the three graph-based semantic representations collected in the Broad-Coverage Semantic Dependency Parsing SemEval shared task (SDP) (Oepen et al., 2015): (1) DELPH-IN Bi-Lexical Dependencies (DM) (Flickinger, 2000),[1] (2) Enju Predicate-Argument Structures (PAS) (Miyao et al., 2014), and (3) Prague Semantic Dependencies (PSD) (Hajic et al., 2012). These annotations have garnered recent attention (e.g., (Buys and Blunsom, 2017; Peng et al., 2017a)), and were consistently annotated in parallel on over more than 30K sentences of the Wall Street Journal corpus (Charniak et al., 2000).

In this work we take a novel approach to graph parsing, casting sentence-level semantic parsing as a multilingual machine-translation task (MT). We deviate from current graph-parsing approaches to SDP (Peng et al., 2017a) by treating the different semantic formalisms as foreign target dialects, while having English a as a common source language (Section 3). Subsequently, we devise a neural MT sequence-to-sequence framework that is suited for the task.

In order to apply sequence-to-sequence models for structured prediction, a linearization function is required to interpret the model's sequential input and output. Initial work on structured prediction sequence-to-sequence modeling has focused on tree structures (Vinyals et al., 2015; Aharoni and Goldberg, 2017), as these are quite easy to linearize using the bracketed representation (as employed in the Penn TreeBank (Marcus et al., 1993)). Following, various efforts were made to port the attractiveness of sequence-to-sequence modeling to the more general graph structure of semantic representations, such as AMR or MRS (Peng et al., 2017b; Barzdins and Gosko, 2016; Konstas et al., 2017; Buys and Blunsom, 2017). However, to the best of our knowledge, all such current methods actually sidestep the challenge of graph linearization – they reduce the input graph to a tree using lossy heuristics, which are specifi-

*Work performed while at Bar-Ilan University.

[1] DM is automatically derived from Minimal Recursion Semantics (MRS) (Copestake et al., 1999).

cally tailored for their target representation.

In contrast, we design a novel deterministic and lossless linearization (Section 4), which is applicable to any graph with ordered nodes (e.g., sentence word order). To that end, we devise solutions for the various obstacles for linearizing a graph structure, such as reentrancies (or multiple heads), non-connected components, and non-projective relations. This lineariztion allows us to follow the spirit of Johnson et al. (2017) in training all source-target combinations in a multi-task approach (Section 5). These combinations include the three traditional text to semantic parsing tasks, as well as six additional inter-representation translation tasks, constituting of all binary combinations of the target representations (e.g., PSD to PAS, or DM to PSD).

Following, we design an encoder-decoder model which has two shared encoders, one for raw English sentences and another for linearized graphs, and a single global graph decoder. Interestingly, we show that training on the auxiliary inter-representation translation tasks greatly improves the performance on the original SDP tasks, without requiring any additional manual annotation effort (Section 6).

Our contributions are two-fold. First, we show that novel sequence-to-sequence models are able to effectively capture and recover general graph structures, making them a viable and easily extensible approach towards the SDP task. Second, beyond SDP, as the inclusion of syntactic linearization was shown beneficial in various tasks (Aharoni and Goldberg, 2017; Le et al., 2017) so does our approach prompt easy integration of graph-based representations as complementary semantic signal in various downstream applications.

2 Background

We begin this section by presenting the corpus we use to train and test our model (the SDP corpus) and the current state-of-the-art in predicting semantic dependencies. Then, we discuss previous work on sequence-to-sequence models for tree prediction, which this work extends to general graph structures. Finally, we briefly describe the multilingual translation approach, which we borrow and adapt to the semantic parsing task.

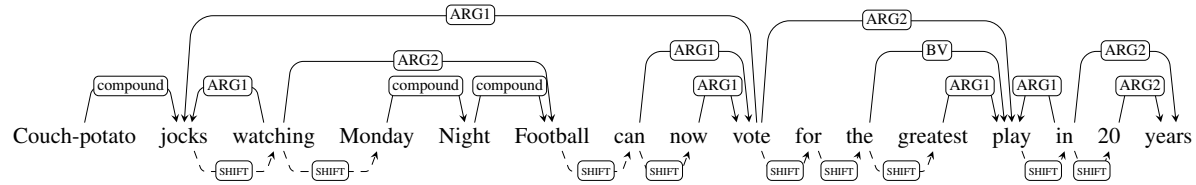
	DM	PAS	PSD
#Train sentences	35,657	35,657	35,657
#Test sentences	1,410	1,410	1,410
#Labels	59	42	91
%Trees	2.30	1.22	42.19
%Projective	2.91	1.64	41.92

Table 1: SDP corpus statistics. Numbers taken from Oepen et al. (2015).

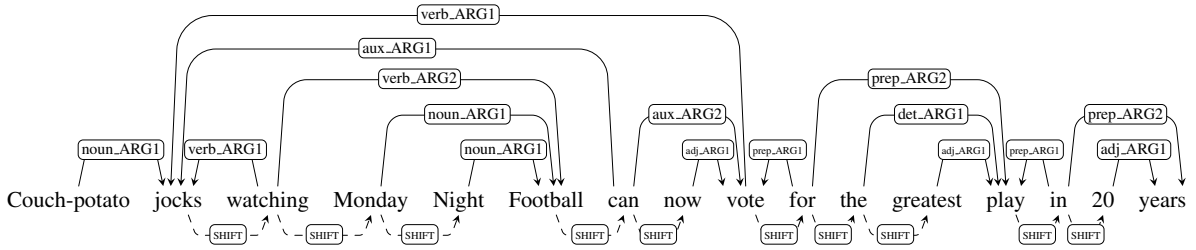
2.1 Semantic Dependencies

In general, the development of most semantic formalisms was carried out by disjoint and independent efforts. However, the 2014 and 2015 SemEval shared tasks (Oepen et al., 2014, 2015) have culminated in the Semantic Dependency Parsing (SDP) resource, a consistent and large corpus (roughly 39K sentences), annotated in parallel with three well-established formalisms: DELPH-IN MRS-Derived Bi-Lexical Dependencies (DM) (Flickinger, 2000), Enju Predicate-Argument Structures (PAS) (Miyao et al., 2014), and Prague Semantic Dependencies (PSD) (Hajic et al., 2012). While varying in their labels and annotation guidelines, all three representations induce a graph structure, where each node corresponds to a single word in the sentence. See Table 1 for more details on this corpus, and Figure 1 for examples of the three SDP formalisms. SDP has enabled the application of machine learning models for the task. Peng et al. (2017a) have set the state-of-the-art results on all three tasks, using techniques inspired by graph-based dependency parsing models (Kiperwasser and Goldberg, 2016; Dozat and Manning, 2016; Kuncoro et al., 2016). Their best results were obtained by leveraging the fact that SDP was annotated on parallel texts. They reached 88% average labeled F1 score across the SDP representations on an in-domain test set, via joint prediction of the three representations using higher-order cross-representation features. The first row in Table 4 summarizes their performance for the three prediction tasks.

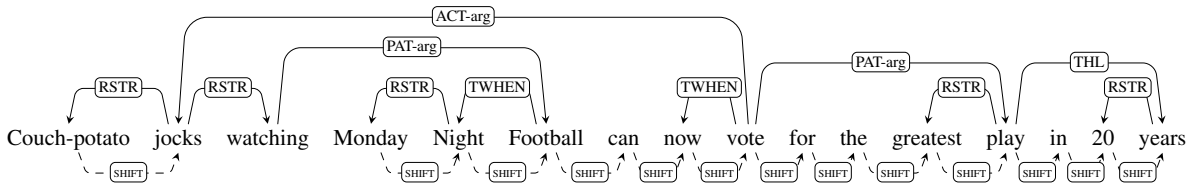
In this work we will take a different approach to structured prediction of the SDP corpus. We will design a novel sequence-to-sequence model, not necessitating parallel annotations, which are often unavailable for multi-task learning.



(a) DELPH-IN Minimal Recursion Semantics-derived bi-lexical dependencies (DM).



(b) Enju Predicate-Argument Structures (PAS).



(c) Prague Semantic Dependencies (PSD).

Figure 1: Example of gold annotations for the three sentence-level representations in the SDP corpus (DM, PAS, and PSD) on the same sentence, which was slightly shortened for presentation. Arcs in each of the representations appear above the sentence. Our "SHIFT" edges, which appear dashed below it, were introduced in Section 4 to ensure that all nodes are reachable from the first word.

2.2 Structured Prediction using Sequence-to-Sequence Models

In contrast to the graph-parsing algorithms discussed in Section 2.1, a recent line of work has explored the usage of more general sequence-to-sequence models to perform structured prediction, focusing specifically on predicting tree structures. These approaches devise a task-specific linearization function which converts the structured representation to a sequential string, which is then used to train the recurrent neural network. During inference, the inverted linearization function is applied to the output to recover the desired structure.

Vinyals et al. (2015) showed that sequence-to-sequence phrase-based constituency parsing can be achieved using a tree depth-first search (DFS) traversal as a linearization function.[2] Following this work, several recent efforts have employed a similar DFS approach to AMR and MRS parsing (Barzdins and Gosko, 2016; Konstas et al., 2017; Peng et al., 2017b; Buys and Blunsom, 2017), after reducing AMR to trees by removing re-entrencies.

Several recent works have found syntactic linearization useful outside of neural parsers. For example, in neural machine translation, Aharoni and Goldberg (2017) showed that predicting target-side linearized syntactic trees can improve the quality and grammatically of the predicted translations. In Section 4, we show for the first time that the DFS approach is a viable linearization function also for semantic dependencies, by extending it to account for the challenges introduced by the richer graph structures in SDP.

2.3 Multi-lingual Machine Translation

Multi-Task Learning (MTL) is a modeling approach which shares and tunes the model parameters across several tasks. In some instances of MTL, a subset of the tasks may be defined as the "main tasks", while the other tasks are treated as auxiliaries which improve performance on the main tasks by contributing to their training signal. MTL had regained popularity in recent years thanks to its easy and wide-spread applicability in neural networks (Collobert et al., 2011; Sogaard

[2] DFS for rooted trees is equivalent to the bracketed notation of the Penn Treebank.

and Goldberg, 2016).

Perhaps most relevant to this work is Google's neural machine translation system by Johnson et al. (2017), which trained a single sequence-to-sequence model to translate between multiple languages. They introduced the usage of a special tag in the source sentence to specify the desired target language. For example, *<2es>* indicates that the model should translate the input sentence to Spanish.

In Section 4, we adapt the MTL strategy to train a single model for all SDP formalisms. We use a similar "to" and "from" tags to indicate source and desired target representations, and show that introducing auxiliary inter-task translations can improve performance on the main target tasks, namely parsing semantic representations for raw input text.

3 Task Definition

We define the task of **semantic translation**, as converting to, and between, different sentence-level semantic representations. Formally, a sentence-level semantic representation according to formalism R is a tuple, $\mathcal{M}_{\mathcal{R}} = (S, G)$, where $S = \{w_1, ..., w_n\}$ is a raw sentence, and $G = (V, E \mid V = \{v_1, ..., v_n\}, E \subseteq V^2)$ is a labeled graph whose vertices have a one-to-one correspondence with the words in S,[3] while its edges represent binary semantic relations, adhering to R's specifications.

Using these notations, our input is defined as a triplet $(source, target, \mathcal{M}_{source})$. Preceding the input semantic representation are identifiers for source and target representation schemes (e.g., "PAS", "DM" or "PSD"). The semantic translation task is then to produce $\mathcal{M}_{target}$. I.e., the sentence's representation under the $target$ formalism.

This definition is broad enough to encapsulate many sentence-level representations, and in this work we will use the three SDP representations, as well as an empty "RAW" representation (where $E(G) = \emptyset$ for all sentences) to allow for translations from raw input sentences. We note that future work may extend this framework with other graph-based sentence representations.

[3] The one-to-one node-to-word correspondence follows SDP's formulation, but can be relaxed to adjust for other graph structures.

4 Graph Linearization

As discussed in Section 2, structured prediction in a sequence-to-sequence framework requires a linearization function, from the desired structure to a linear sequence, and vice versa.

Oftentimes, such linearization consists of node traversal along the edges of the input graph. While previous work have had certain structural constraints on their input (e.g., imposing tree or non-cyclic constructions), in this work, we construct a *lossless* function which allows us to feed the sequence-to-sequence network with a linearized *general* graph representation and expect a linearized graph in its output.

In this section, we describe our linearization traversal order, which generalizes the DFS traversal applied previously only for trees. We do this by converting an SDP graph such that all nodes are reachable from node v_1. We then outline the challenging aspects of graph properties (which do not exist in trees), show that they are prevalent in the SDP corpus, and describe our proposed solutions. To the best of our knowledge, this is the first work which tackles the task of general graph linearization.

While our linearization can be predicted with good accuracy (as we show in following sections), there is ample room to experiment with representational variations, which we start exploring in Section 6. Our conversion code is made publicly available,[4] allowing further experimentation with general graph linearization for SDP and other related tasks.

4.1 Traversing Graphs with Non-Connected Components

The DFS approach is an applicable linearization of trees since a recursive traversal, which starts at the root and explores all outgoing edges, is guaranteed to visit all of the graph's nodes. However, DFS linearization is not directly applicable to SDP, as its graphs often consist of several non-connected components.

For such graphs, there exists no starting node from which all of the nodes are reachable via DFS traversal, and certain nodes are bound to be left out of the traditional DFS encoding. For example, the words "can" and "greatest" in Figure 1a reside in different components, and therefore no single path

[4] https://github.com/gabrielStanovsky/semantics-as-foreign-language

(which traverses along the graph's edge direction) will discover both of them.

To overcome this limitation, we make sure that all nodes are reachable from node v_1, corresponding to the first word in the sentence, from which we start our traversal. This is achieved by introducing an artificial `SHIFT` edge between any two consecutive nodes v_i, v_{i+1} for which there is no directed path already connecting them. Following, it is easy to see, by induction, that all nodes are reachable from v_1, as for every node v_i there exists a directed path $(v_1, v_2, ..., v_{i-1}, v_i)$. For example, revisit the previously mentioned "can" and "greatest" nodes in Figure 1a, which are connected using "SHIFT" edges.

4.2 Linearizing a DFS Graph Traversal

Intuitively, our linearization is a pre-order DFS, generalizing Vinyals et al. (2015)'s approach to syntactic linearization. We start from v_1 and explore all paths from it, in a depth-first manner. Once a path is exhausted, either by reaching a node with no outgoing edges[5] or by reaching an already visited node, we use special backtracking edges to form a path backwards "up" the graph, until we hit a node which still has unexplored outgoing edges.

Formally, our linearization of a given DFS traversal is composed of 3 types of elements (see Figure 2 for example):

First, a **Node reference** identifies a node in the graph, which in turn corresponds to word in the SDP formalism. We identify nodes using two tokens: (1) Their position in the sentence, relative to the previous node in the path (while the first position in the linearization is written in absolute terms, as "0"), and (2) Explicitly writing the word corresponding to the node.

For example, in Figure 2, traversing the *ARG1* edge from "easy" lands at "`ind/2` **`understand`**", whose outgoing *ARG2* edge arrives at "`ind/-4` **`success`**".

Second, an **Edge reference**, identifies an edge label. These are denoted by a single token, composed of 2 parts: (1) The edge's formalism (in our case, the SDP representation to which it pertains), and (2) The edge label. Traversing an edge (u, v) with label L will be encoded by placing the edge reference between the node references of u and v. For instance, in Figure 2, moving from node 0 to node 1 through the edge labeled "poss" is encoded with the following string: "`ind/0` **`Their`** `PAS/poss ind/1` **`success`**".

Finally, **Backtracking edges**, signify a step "backward" in the traversal. These are denoted with a single token, similarly to edge references, with the addition of a "BACK" suffix. For example, in Figure 2, we backtrack from the already visited node "understand" by writing: "*`PAS/ARG1/BACK`*".

This linearization can be deterministically and efficiently inverted back to the graph structure. This is done by building the graph while reading the linearization, adding to it nodes and edges when they first appear, and omitting possible node recurrences in the linearization (due to cycles or backtracking edges), such as "success", which appears twice in the Figure 2.

Redundancy in encoding We note that certain items in our proposed linearization are redundant. First, writing down the explicit word in the traversal is not necessary, as the positional index is sufficient to uniquely identify a node. Second, a single backtracking tag would have been enough to identify the specific edge which is currently being backtracked (e.g., `BACK` instead of `PAS/verb_ARG1/BACK)`. The latter is similar to the redundancy in the syntactic linearization of Vinyals et al. (2015), who specify the type of closing bracket, e.g., `NP(...)NP` instead of `NP(...)`.

In Section 6 we show empirically that our model benefits from explicitly generating these redundancies during decoding.

4.3 DFS Traversal Order

A graph DFS traversal does not dictate an order in which to explore the different outgoing paths at each branching point. Consider, as a recurring example, the branching point at the word "vote" in Figure 1c, in which we need to choose an order amongst its four neighbors.

While syntactic linearization conveniently follows the ordering of the words in the sentence, Konstas et al. (2017) have noted that different child visiting linearization orders affect the performance of text generation from AMR. In particular, they found that following the order of annotation of a human expert worked best.

Intuitively, since different graph traversals affect the sequence of encoded nodes during train-

[5]Note that after introducing the artificial "SHIFT" edges, only v_n may have no outgoing edges.

DFS order type	Example (*vote*'s PSD neighbors)
Random permutation	*(play, for, jocks, now)**
Sentence order Neighbor's index in the sentence	*(jocks, now, for, play)*
Closest words Neighbor's absolute distance	*(now, for, play, jocks)*
Smaller-first # nodes reachable from the neighbor	*(now, play, for, jocks)*

Table 2: Different neighbor exploration orders. Under the name of each order type, we list the key by which we sort each node's neighbors. The "Example" column shows the corresponding ordering of "vote"'s neighbors in Figure 1c. *An example of one possible random permutation.

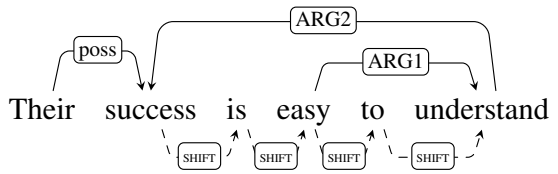
ing, the network will inevitably have to learn different weights and attention when presented with different orderings. Therefore, some traversal orderings may be easier to learn than others, leading to better (hopefully more semantic) abstractions.

To the best of our knowledge, the human annotation order is not available for the SDP annotations, and there is no clear a priori optimal ordering. We therefore experiment with several visiting orders, as described in Table 2. Notably, **Sentence order** is equivalent to the ordering used by Vinyals et al. (2015) for syntactic linearization, while **Closest words** orders child nodes from short to longer range-dependencies (commonly associated with syntactic versus semantic relations), and **Smaller first** is motivated by the easy-first approach (Goldberg and Elhadad, 2010), first encoding paths which are shorter (and easier to memorize), before longer, more complicated sequences.

In Section 6 we evaluate the effect of these variations on the SDP parsing task.

5 Model

We start by describing our model architecture, inspired by recent MT architectures, while allowing for different types of inputs, namely English sentences and linearized graphs. Following, we present our methods for training and testing, and specific hyper-parameter configuration and implementation details.



(a) Gold PAS representation from the SDP corpus. Original gold edges appear above the words, while our introduced edges appear below them.

```
ind/0 Their PAS/poss ind/1 success
SHIFT ind/1 is SHIFT ind/1 easy
PAS/ARG1 ind/2 understand PAS/ARG2
ind/-4 success PAS/ARG2/BACK ind/4
understand PAS/ARG1/BACK ind/-2 easy
SHIFT ind/1 to SHIFT ind/1 understand
```

(b) Our linearization scheme for the sentence in 2a. Each node is represented by its relative index and surface form. Backwards traversing edges (marked with *BACK*) appear in italics.

Figure 2: Example of gold PAS representation from the development partition of the SDP corpus (top), and our corresponding linearization (bottom).

5.1 Architecture

Our architecture, depicted in Figure 3, consists of a sequence-to-sequence model using a bi-LSTM encoder-decoder with attention on input and output tokens, similar to that used by Johnson et al. (2017) for multi-lingual MT. As described in Section 3, it is trained on 9 translation tasks in parallel. We split these into two groups, consisting of 3 primary tasks and 6 auxiliary tasks, as follows:

$$\texttt{PRIMARY} = \{(\text{RAW}, tgt) \mid tgt \in (\text{DM, PAS, PSD})\}$$

$$\texttt{AUXILIARY} = \{(src, tgt) \in \{\text{DM, PAS, PSD}\}^2 \mid src \neq tgt\}$$

The PRIMARY tasks deal with converting raw sentences to linearized graph structures, which we can compare to previous published baselines and are therefore our main interest. Conversely, while the AUXILIARY tasks provide additional training signal to tune our model, they are also interesting from an analytic point-of-view, which we examine in depth in Section 6.

To allow the model to differentiate between the different tasks, we prefix each input sample with two tags (see example in Figure 3). First, similarly to Johnson et al. (2017), we add a tag indicating the desired target representation, e.g., <to:DM>. Second, In contrast to multi-lingual MT which omits the source language (to allow for

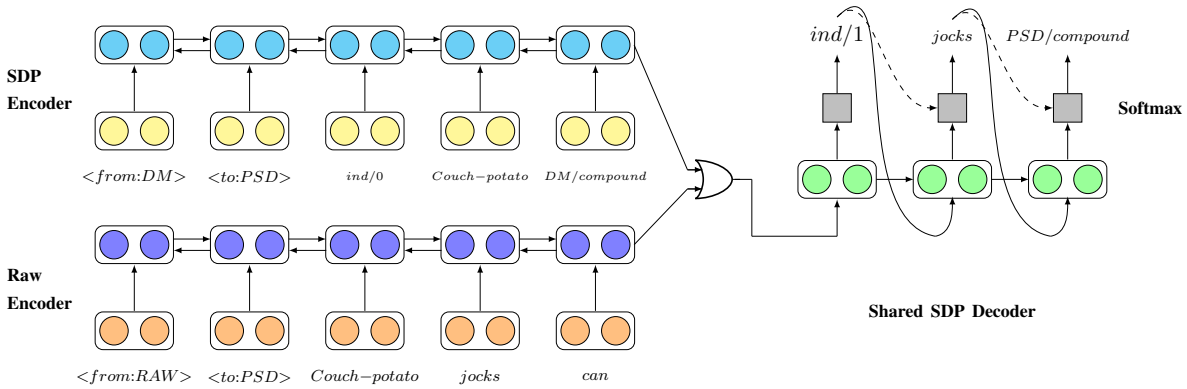


Figure 3: **Simplified sketch of our sequence-to-sequence architecture.** The figure depicts encoding and decoding of two input training samples, one from raw text to PSD (lower left), and the other from DM to PSD (top left). The OR gate denotes choosing only one sample to encode at each training step. While the two samples use different encoders, they share a single global SDP decoder (right) which outputs a the graph structure. As denoted by dashed edges, at every decode step we can deterministically interject and override the softmax probabilities for redundant elements, based on previous predictions. For simplicity sake, a small number of units is showed for encoders and decoder and the attention and deep encoder-decoder layers are omitted.

code switching), we explicitly denote the source representation, e.g., `<from:PSD>`. This addition further strengthens the correlation between inputs from the same representation.[6]

Further deviating from the current practice in MT, our architecture uses two encoders and a single decoder (while common MT regards the encoder-decoder as a single unit). The first shared encoder specializes in encoding raw text for all `PRIMARY` tasks, while a second encodes linearized graph structures for the `AUXILIARY` tasks. Both encoders are linked to a single decoder which converts their output representations to a linearized graph.

Intuitively, the two encoders correspond to the different nature of input to the `PRIMARY` tasks (an English sentence) versus that of the `AUXILIARY` tasks (a linearized graph), while a single decoder allows for a common linearized graphs output format. Since the decoder is trained across all 9 tasks, both encoders are optimized to arrive at similar latent representations which are geared towards graph prediction.

5.2 Training and Inference

The overall size of multi-task training data is $320,913$ samples. This constitutes a 9-fold increase over a single-model for SDP ($35,657$ sentences in the SDP corpus) and a 3-fold increase over a standard MTL approach to SDP (without the `AUXILIARY` tasks). During training, we penalize the model on all predicted elements, including the redundant elements discussed in Section 4.2. During inference, however, these redundancies may cause contradictions leading to incoherent sequences. Namely, a word may not conform to the previous word index, and a backtracking edge may point to a different relation. To overcome this we artificially increase the softmax probabilities (dashed edges in Figure 3) so that they reflect the DFS path decoded up until that point. Specifically, we override the predicted word according to the previous index, and backtrack "up" the corresponding edge.

5.3 Implementation Details

All of our hyper-parameters were tuned on a held out partition of 1000 sentences in the training set. In particular, we use 3 hidden layers for both of the encoders, and 2 hidden layers for the decoder. English word embeddings were fixed with 300-dimensional GloVe embeddings (Pennington et al., 2014), while the graph elements, which consist of a lexicon of roughly 400 tokens across three representations, were randomly initialized. We trained the model until convergence, roughly 20

[6] Moreover, "code-switching" between semantic representations is inherently undesired.

	DM	PAS	PSD	Avg.
Random	86.1	87.7	78.4	84.1
Sentence order	87.2	90.3	79.9	85.8
Closest words	87.5	89.8	79.7	85.8
Smaller-first	**87.9**	**90.9**	**80.3**	**86.2**

Table 3: Evaluation of different DFS orderings, in labeled F1 score, across the different tasks.

epochs, in about 12 hours on a GPU (NVIDIA GeForce GTX 1080 Ti), in batches of 50 sentences. All of these sentences belong to the same task, which is chosen at random before each batch.

Finally, our models were developed using the OpenNMT-py library (Klein et al., 2017), and are made available.[7]

6 Evaluation

We perform several evaluations, testing the impact of alternative configurations, including the different DFS traversal orders and MTL versus single-task approach, as well as our model's performance against current state-of-the-art on each of the PRIMARY tasks.

6.1 Results

The results of our different analyses are reported in Tables 3-6, as elaborated below. For all evaluations, we use the in-domain test partition of the SDP corpus, containing $1,410$ sentences. Following Peng et al. (2017a) we report performance using labeled F1 scores as well as average scores across representations. We compare the produced graphs, after applying the inverted linearization function, rather than comparing the DFS path directly, as there may be several DFS graph traversals encoding the same relations.

DFS order matters - Table 3 depicts our model's performance when linearizing the graphs according to the different traversal orders discussed and exemplified in Table 2. Overall, we find that the "smaller-first" approach performs best across all datasets, and that imposing one of our orders is always preferable over random permutations. Intuitively, the "smaller-first" approach presents shorter, and likely easier, paths first, thus minimizing the amount of error-propagation for following decoding steps. Due to its better performance, we will report only the smaller-first's performance in all following evaluations.

From English to SDP - Table 4 presents the performance of our complete model ("MTL PRIMARY+AUX") versus Peng et al. (2017a). On average, our model performs within 1% F1 point from the state-of-the art (outperforming it on the harder PSD task), despite using the more general sequence-to-sequence approach instead of a dedicated graph-parsing algorithm. In addition, an ablation study shows that multi-tasking the PRIMARY tasks is beneficial over a single task setting, which in turn is outperformed by the inclusion of the AUXILIARY tasks.

Simulating disjoint annotations - In contrast with SDP's complete overlap of annotated sentences, multi-task learning often deals with disjoint training data. To simulate such scenario, we retrained the models on a randomly selected set of 33% of the train sentences for each representation ($11,886$ sentences), such that the three representations overlap on only 10% ($3,565$ sentences). The results in Table 5 show that our approach is more resilient to the decrease in annotation overlap, outperforming the state-of-the-art model on the DM and PSD task, as well as on the average score. We hypothesize that this is in part thanks to our ability to use the inter-task translations, even when these exist only for part of the annotations.

6.2 Translating Between Representations

As a byproduct of training on the AUXILIARY tasks, our model can also be tested on translating *between* the different representations. This is done by presenting it with a linearized graph of one representation and asking it to translate it to another. To the best of our knowledge, this is the first work which tries to accomplish this.

We report the performance of all source-target combinations in Table 6. These evaluations provide several interesting comparisons between the representations: (1) For all representations, translating from any of the other two is easier than parsing from raw text, (2) The PAS and DM representations can be converted between them with high accuracy (95.7% and 96.1%, respectively). This can be due to their structural resemblance, noted in previous work (Peng et al., 2017a; Oepen et al., 2015), and (3) While PSD serves as a viable input for conversion to DM and PAS (92.1% F1 on

[7] https://github.com/gabrielStanovsky/semantics-as-foreign-language

	DM	PAS	PSD	Avg.
Peng et al. (2017a)	**90.4**	**92.7**	78.5	**87.2**
Single	70.1	73.6	63.6	69.1
MTL PRIMARY	82.4	87.2	71.4	80.3
MTL PRIMARY+AUX	87.9	90.9	**80.3**	86.2

Table 4: Evaluation of our model (labeled F1 score) versus the current state of the art. "Single" denotes training a different encoder-decoder for each task. "MTL PRIMARY" reports the performance of multi-task learning on only the PRIMARY tasks. "MTL PRIMARY+AUX" shows the performance of our full model, including MTL with the AUXILIARY tasks.

	DM	PAS	PSD	Avg.
Peng et al. (2017a)	86.8	**90.5**	77.3	84.9
MTL PRIMARY+AUX	**87.1**	89.6	**79.1**	**85.3**

Table 5: Performance (labeled F1 score) of our model versus the state of the art, when reducing the amount of overlap in the training data to 10%.

To \From	DM	PAS	PSD	Avg.
DM		96.1	92.4	94.3
PAS	95.7		91.7	93.7
PSD	89.5	87.6		88.6
Avg.	92.6	91.9	92.1	

Table 6: Performance (labeled F1 score) of inter-task translations. Each column depicts the performance converting from a specific source representation, while each row denotes the corresponding target representation.

average), it is relatively harder to convert either of them to PSD (88.6%). This might indicate that PSD subsumes some of the information in DM and PAS.

7 Conclusions and Future Work

We presented a novel sequence-to-sequence approach to the task of semantic dependency parsing, by casting the problem as multi-lingual machine translation. To that end, we introduced a DFS-based graph linearization function which generalizes several previous works on tree linearization. Following, we showed that our model, inspired by neural MT, benefits from the inter-task training signal, reaching performance almost on-par with current state of the art in several scenarios.

Future work can employ this linearization function within downstream applications, as was done with syntactic linearization, or extend this framework with other graph-based representations, such as universal dependencies (Nivre et al., 2016) or AMR (Banarescu et al., 2013).

Acknowledgements

This work was supported in part by grants from the MAGNET program of the Israeli Office of the Chief Scientist (OCS); the German Research Foundation through the German-Israeli Project Cooperation (DIP, grant DA 1600/1-1) and the Israel Science Foundation (grant No. 1157/16).

References

Omri Abend and Ari Rappoport. 2013. Universal conceptual cognitive annotation (UCCA). In *Proceedings of the 2013 conference of the Association for Computational Linguistics*.

Omri Abend and Ari Rappoport. 2017. The state of the art in semantic representation. In *ACL*.

Roee Aharoni and Yoav Goldberg. 2017. Towards string-to-tree neural machine translation. In *ACL*.

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.

Guntis Barzdins and Didzis Gosko. 2016. Riga at semeval-2016 task 8: Impact of smatch extensions and character-level neural translation on amr parsing accuracy. In *SemEval@NAACL-HLT*.

Jan Buys and Phil Blunsom. 2017. Robust incremental neural semantic graph parsing. In *ACL*.

Xavier Carreras and Lluís Màrquez. 2005. Introduction to the conll-2005 shared task: Semantic role labeling. In *Proceedings of CONLL*, pages 152–164.

Eugene Charniak, Don Blaheta, Niyu Ge, Keith Hall, John Hale, and Mark Johnson. 2000. Bllip 1987-89 wsj corpus release 1. *Linguistic Data Consortium, Philadelphia*, 36.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537.

Ann. Copestake, Dan Flickinger, and Ivan A. Sag. 1999. Minimal recursion semantics: An introduction.

Timothy Dozat and Christopher D. Manning. 2016. Deep biaffine attention for neural dependency parsing. *CoRR*, abs/1611.01734.

Dan Flickinger. 2000. On building a more effcient grammar by exploiting types. *Natural Language Engineering*, 6(1):15–28.

Yoav Goldberg and Michael Elhadad. 2010. An efficient algorithm for easy-first non-directional dependency parsing. In *HLT-NAACL*.

Jan Hajic, Eva Hajicová, Jarmila Panevová, Petr Sgall, Ondrej Bojar, Silvie Cinková, Eva Fucíková, Marie Mikulová, Petr Pajas, Jan Popelka, et al. 2012. Announcing prague czech-english dependency treebank 2.0. In *LREC*, pages 3153–3160.

Melvin Johnson, Mike Schuster, Quoc V. Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda B. Viégas, Martin Wattenberg, Gregory S. Corrado, Macduff Hughes, and Jeffrey Dean. 2017. Google's multilingual neural machine translation system: Enabling zero-shot translation. *TACL*, 5:339–351.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional lstm feature representations. *TACL*, 4:313–327.

G. Klein, Y. Kim, Y. Deng, J. Senellart, and A. M. Rush. 2017. OpenNMT: Open-Source Toolkit for Neural Machine Translation. *ArXiv e-prints*.

Ioannis Konstas, Srinivasan Iyer, Mark Yatskar, Yejin Choi, and Luke S. Zettlemoyer. 2017. Neural amr: Sequence-to-sequence models for parsing and generation. In *ACL*.

Adhiguna Kuncoro, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, and Noah A. Smith. 2016. Distilling an ensemble of greedy dependency parsers into one mst parser. In *EMNLP*.

An Nguyen Le, Ander Martinez, Akifumi Yoshimoto, and Yuji Matsumoto. 2017. Improving sequence to sequence neural machine translation by utilizing syntactic dependency information. In *IJCNLP*.

Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330.

Yusuke Miyao, Stephan Oepen, and Daniel Zeman. 2014. In-house: An ensemble of pre-existing off-the-shelf parsers. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 335–340.

Richard Montague. 1973. The proper treatment of quantification in ordinary english. In *Approaches to natural language*, pages 221–242. Springer.

Joakim Nivre. 2005. Dependency grammar and dependency parsing.

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajic, Christopher D. Manning, Ryan T. McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, Reut Tsarfaty, and Daniel Zeman. 2016. Universal dependencies v1: A multilingual treebank collection. In *LREC*.

Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Silvie Cinková, Dan Flickinger, Jan Hajic, and Zdenka Uresová. 2015. Semeval 2015 task 18: Broad-coverage semantic dependency parsing. In *SemEval@NAACL-HLT*.

Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Dan Flickinger, Jan Hajic, Angelina Ivanova, and Yi Zhang. 2014. Semeval 2014 task 8: Broad-coverage semantic dependency parsing. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 63–72.

Hao Peng, Sam Thomson, and Noah A. Smith. 2017a. Deep multitask learning for semantic dependency parsing. In *Proceedings of the Association for Computational Linguistics*, Vancouver, Canada. Association for Computational Linguistics.

Xiaochang Peng, Chuan Wang, Daniel Gildea, and Nianwen Xue. 2017b. Addressing the data sparsity issue in neural amr parsing. In *EACL*.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543.

Anders Sogaard and Yoav Goldberg. 2016. Deep multi-task learning with low level tasks supervised at lower layers. In *ACL*.

Oriol Vinyals, Lukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey E. Hinton. 2015. Grammar as a foreign language. In *NIPS*.