Semantic Neural Machine Translation using AMR

Linfeng Song\textsuperscript{1}, Daniel Gildea\textsuperscript{1}, Yue Zhang\textsuperscript{2}, Zhiguo Wang\textsuperscript{3} and Jinsong Su\textsuperscript{4}
\textsuperscript{1}Department of Computer Science, University of Rochester, Rochester, NY 14627
\textsuperscript{2}School of Engineering, Westlake University, China
\textsuperscript{3}IBM T.J. Watson Research Center, Yorktown Heights, NY 10598
\textsuperscript{4}Xiamen University, Xiamen, China
\textsuperscript{1}\{lsong10,gildea\}cs.rochester.edu \textsuperscript{2}yue.zhang@wias.org.cn
\textsuperscript{3}zgw.tomorrow@gmail.com \textsuperscript{4}jssu@xmu.edu.cn

Abstract

It is intuitive that semantic representations can be useful for machine translation, mainly because they can help in enforcing meaning preservation and handling data sparsity (many sentences correspond to one meaning) of machine translation models. On the other hand, little work has been done on leveraging semantics for neural machine translation (NMT). In this work, we study the usefulness of AMR (short for abstract meaning representation) on NMT. Experiments on a standard English-to-German dataset show that incorporating AMR as additional knowledge can significantly improve a strong attention-based sequence-to-sequence neural machine translation model.

1 Introduction

It is intuitive that semantic representations ought to be relevant to machine translation, given that the task is to produce a target language sentence with the same meaning as the source language input. Semantic representations formed the core of the earliest symbolic machine translation systems, and have been applied to statistical but non-neural systems as well.

Leveraging syntax for neural machine translation (NMT) has been an active research topic (Stahlberg et al., 2016; Aharoni and Goldberg, 2017; Li et al., 2017; Chen et al., 2017; Bastings et al., 2017; Wu et al., 2017; Chen et al., 2018). On the other hand, exploring semantics for NMT has so far received relatively little attention. Recently, Marcheggiani et al. (2018) exploited semantic role labeling (SRL) for NMT, showing that the predicate-argument information from SRL can improve the performance of an attention-based sequence-to-sequence model by alleviating the “argument switching” problem,\textsuperscript{1} one frequent and severe issue faced by NMT systems (Isabelle et al., 2017). Figure 1 (a) shows one example of semantic role information, which only captures the relations between a predicate (\textit{give}) and its arguments (\textit{John}, \textit{wife} and \textit{present}). Other important information, such as the relation between \textit{John} and \textit{wife}, can not be incorporated.

In this paper, we explore the usefulness of abstract meaning representation (AMR) (Banarescu et al., 2013) as a semantic representation for NMT. AMR is a semantic formalism that encodes the meaning of a sentence as a rooted, directed graph. Figure 1 (b) shows an AMR graph, in which the nodes (such as \textit{give-01} and \textit{John}) represent the concepts, and edges (such as :ARG0 and :ARG1) represent the relations between concepts they connect. Comparing with semantic roles, AMRs capture more relations, such as the relation between \textit{John} and \textit{wife} (represented by the subgraph within dotted lines). In addition, AMRs directly capture entity relations and abstract away inflections and function words. As a result, they can serve as a source of knowledge for machine translation that is orthogonal to the textual input. Furthermore,
structural information from AMR graphs can help reduce data sparsity, when training data is not sufficient for large-scale training.

Recent advances in AMR parsing keep pushing the boundary of state-of-the-art performance (Flanigan et al., 2014; Artzi et al., 2015; Pust et al., 2015; Peng et al., 2015; Flanigan et al., 2016; Buys and Blunsom, 2017; Konstas et al., 2017; Wang and Xue, 2017; Lyu and Titov, 2018; Peng et al., 2018; Groschwitz et al., 2018; Guo and Lu, 2018), and have made it possible for automatically-generated AMRs to benefit downstream tasks, such as question answering (Mitra and Baral, 2015), summarization (Takase et al., 2016), and event detection (Li et al., 2015a). However, to our knowledge, no existing work has exploited AMR for enhancing NMT.

We fill in this gap, taking an attention-based sequence-to-sequence system as our baseline, which is similar to Bahdanau et al. (2015). To leverage knowledge within an AMR graph, we adopt a graph recurrent network (GRN) (Song et al., 2018; Zhang et al., 2018) as the AMR encoder. In particular, a full AMR graph is considered as a single state, with nodes in the graph being its substates. State transitions are performed on the graph recurrently, allowing substates to exchange information through edges. At each recurrent step, each node advances its current state by receiving information from the current states of its adjacent nodes. Thus, with increasing numbers of recurrent steps, each word receives information from a larger context. Figure 3 shows the recurrent transition, where each node works simultaneously. Compared with other methods for encoding AMRs (Konstas et al., 2017), GRN keeps the original graph structure, and thus no information is lost (Song et al., 2018). For the decoding stage, two separate attention mechanisms are adopted in the AMR encoder and sequential encoder, respectively.

Experiments on WMT16 English-German data (4.17M) show that adopting AMR significantly improves a strong attention-based sequence-to-sequence baseline (25.5 vs 23.7 BLEU). When trained with small-scale (226K) data, the improvement increases (19.2 vs 16.0 BLEU), which shows that the structural information from AMR can alleviate data sparsity when training data are not sufficient. To our knowledge, we are the first to investigate AMR for NMT.

Our code and parallel data (training/dev/test) with automatically parsed AMRs are available at https://github.com/freesunshine0316/semantic-nmt.

2 Related work

Most previous work on exploring semantics for statistical machine translation (SMT) studies the usefulness of predicate-argument structure from semantic role labeling (Wong and Mooney, 2006; Wu and Fung, 2009; Liu and Gildea, 2010; Baker et al., 2012). Jones et al. (2012) first convert Prolog expressions into graphical meaning representations, leveraging synchronous hyperedge replacement grammar to parse the input graphs while generating the outputs. Their graphical meaning representation is different from AMR under a strict definition, and their experimental data are limited to 880 sentences. We are the first to investigate AMR on a large-scale machine translation task.

Recently, Marcheggiani et al. (2018) investigate semantic role labeling (SRL) on neural machine translation (NMT). The predicate-argument structures are encoded via graph convolutional network (GCN) layers (Kipf and Welling, 2017), which are laid on top of regular BiRNN or CNN layers. Our work is in line with exploring semantic information, but different in exploiting AMR rather than SRL for NMT. In addition, we leverage a graph recurrent network (GRN) (Song et al., 2018; Zhang et al., 2018) for modeling AMRs rather than GCN, which is formally consistent with the RNN sentence encoder. Since there is no one-to-one correspondence between AMR nodes and source words, we adopt a doubly-attentive LSTM decoder, which is another major difference from Marcheggiani et al. (2018).

GRNs have recently been used to model graph structures in NLP tasks. In particular, Zhang et al. (2018) use a GRN model to represent raw sentences by building a graph structure of neighboring words and a sentence-level node, showing that the encoder outperforms BiLSTMs and Transformer (Vaswani et al., 2017) on classification and sequence labeling tasks; Song et al. (2018) build a GRN for encoding AMR graphs for text generation, showing that the representation is superior compared to BiLSTM on serialized AMR. We extend Song et al. (2018) by investigating the usefulness of AMR for neural machine translation. To
our knowledge, we are the first to use GRN for machine translation.

In addition to GRNs and GCNs, there have been other graph neural networks, such as graph gated neural network (GGNN) (Li et al., 2015b; Beck et al., 2018). Since our main concern is to empirically investigate the effectiveness of AMR for NMT, we leave it to future work to compare GCN, GGNN, and GRN for our task.

3 Baseline: attention-based BiLSTM

We take the attention-based sequence-to-sequence model of Bahdanau et al. (2015) as the baseline, but use LSTM cells (Hochreiter and Schmidhuber, 1997) instead of GRU cells (Cho et al., 2014).

3.1 BiLSTM encoder

The encoder is a bi-directional LSTM on the source side. Given a sentence, two sequences of states $\hat{h}_1, \hat{h}_2, ..., \hat{h}_N$ and $\hat{h}_1, \hat{h}_2, ..., \hat{h}_N$ are generated for representing the input word sequence $x_1, x_2, ..., x_N$ in the right-to-left and left-to-right directions, respectively, where for each word $x_i$,

$$\hat{h}_i = LSTM(\hat{h}_{i+1}, e_{x_i})$$

$$\hat{h}_i = LSTM(\hat{h}_{i-1}, e_{x_i})$$

$e_{x_i}$ is the embedding of word $x_i$.

3.2 Attention-based decoder

The decoder yields a word sequence in the target language $y_1, y_2, ..., y_M$ by calculating a sequence of hidden states $s_1, s_2, ..., s_M$ recurrently. We use an attention-based LSTM decoder (Bahdanau et al., 2015), where the attention memory $H$ is the concatenation of the attention vectors among all source words. Each attention vector $h_i$ is the concatenation of the encoder states of an input token in both directions ($\hat{h}_i$ and $\hat{h}_i$):

$$h_i = [\hat{h}_i; \hat{h}_i]$$

$$H = [h_1; h_2; ...; h_N]$$

$N$ is the number of source words.

While generating the $m$-th word, the decoder considers four factors: (1) the attention memory $H$; (2) the previous hidden state of the LSTM model $s_{m-1}$; (3) the embedding of the current input (previously generated word) $e_{y_m}$; and (4) the previous context vector $\zeta_{m-1}$ from attention memory $H$. When $m = 1$, we initialize $\zeta_0$ as a zero vector, set $e_{y_1}$ to the embedding of sentence start token “<s>”, and calculate $s_0$ from the last step of the encoder states via a dense layer:

$$s_0 = W_1[\hat{h}_0; \hat{h}_N] + b_1$$

where $W_1$ and $b_1$ are model parameters.

For each decoding step $m$, the decoder feeds the concatenation of the embedding of the current input $e_{y_m}$ and the previous context vector $\zeta_{m-1}$ into the LSTM model to update its hidden state:

$$s_m = LSTM(s_{m-1}, [e_{y_m}; \zeta_{m-1}])$$

Then the attention probability $\alpha_{m,i}$ on the attention vector $h_i \in H$ for the current decode step is calculated as:

$$\epsilon_{m,i} = v_1^T \tanh(W_h h_i + W_s s_m + b_2)$$

$$\alpha_{m,i} = \frac{\exp(\epsilon_{m,i})}{\sum_{j=1}^{N}\exp(\epsilon_{m,j})}$$

$W_h, W_s, v_2$ and $b_2$ are model parameters. The new context vector $\zeta_m$ is calculated via

$$\zeta_m = \sum_{i=1}^{N} \alpha_{m,i} h_i$$

The output probability distribution over the target vocabulary at the current state is calculated by:

$$P_{vocab} = \text{softmax}(V_3[s_m, \zeta_m] + b_3), \quad (1)$$

where $V_3$ and $b_3$ are learnable parameters.

4 Incorporating AMR

Figure 2 shows the overall architecture of our model, which adopts a BiLSTM (bottom left) and our graph recurrent network (GRN) (bottom right) for encoding the source sentence and AMR, respectively. An attention-based LSTM decoder is used to generate the output sequence in the target language, with attention models over both the sequential encoder and the graph encoder. The attention memory for the graph encoder is from the last step of the graph state transition process, which is shown in Figure 3.

---

2 We show the advantage of our graph encoder by comparing with another popular way for encoding AMRs in Section 6.3.
4.1 Encoding AMR with GRN

Figure 3 shows the overall structure of our graph recurrent network for encoding AMR graphs, which follows Song et al. (2018). Formally, given an AMR graph \( G = (V, E) \), we use a hidden state vector \( a^t \) to represent each node \( v_j \in V \). The state of the graph can thus be represented as:

\[
g = \{a^t\}_{v_j \in V}
\]

In order to capture non-local interaction between nodes, information exchange between nodes is executed through a sequence of state transitions, leading to a sequence of states \( g_0, g_1, \ldots, g_T \), where \( g_t = \{a^t\}_{v_j \in V} \), and \( T \) is the number of state transitions, which is a hyperparameter. The initial state \( g_0 \) consists of a set of initial node states \( a^0 = a_0 \), where \( a_0 \) is a vector of all zeros.

A recurrent neural network is used to model the state transition process. In particular, the transition from \( g_{t-1} \) to \( g_t \) consists of a hidden state transition for each node (such as from \( a_{t-1}^j \) to \( a^t \)), as shown in Figure 3. At each state transition step \( t \), our model conducts direct communication between a node and all nodes that are directly connected to the node. To avoid gradient diminishing or bursting, LSTM (Hochreiter and Schmidhuber, 1997) is adopted, where a cell \( c_t^j \) is taken to record memory for \( a_t^j \). We use an input gate \( i_t^j \), an output gate \( o_t^j \) and a forget gate \( f_t^j \) to control information flow from the inputs and to the output \( a_t^j \).

The inputs include representations of edges that are connected to \( v_j \), where \( v_j \) can be either the source or the target of the edge. We define each edge as a triple \((i, j, l)\), where \( i \) and \( j \) are indices of the source and target nodes, respectively, and \( l \) is the edge label. \( x^i_{j,l} \) is the representation of edge \((i, j, l)\), detailed in Section 4.1.1. The inputs for \( v_j \) are grouped into incoming and outgoing edges,

\[
\begin{align*}
\phi_j &= \sum_{(i,j,l) \in E_{in}(j)} x^i_{j,l} \\
\hat{\phi}_j &= \sum_{(j,k,l) \in E_{out}(j)} x^j_{k,l}
\end{align*}
\]

where \( E_{in}(j) \) and \( E_{out}(j) \) are the sets of incoming and outgoing edges of \( v_j \), respectively.

In addition to edge inputs, our model also takes the hidden states of the incoming and outgoing neighbors of each node during a state transition. Taking \( v_j \) as an example, the states of its incoming and outgoing neighbors are summed up before being passed to the cell and gate nodes:

\[
\begin{align*}
\psi_j &= \sum_{(i,j,l) \in E_{in}(j)} a^i_{t-1} \\
\hat{\psi}_j &= \sum_{(j,k,l) \in E_{out}(j)} a^k_{t-1}
\end{align*}
\]

Based on the above definitions of \( \phi_j, \hat{\phi}_j, \psi_j \) and \( \hat{\psi}_j \), the state transition from \( g_{t-1} \) to \( g_t \), as represented by \( a_t^j \), can be defined as:

\[
\begin{align*}
i_t^j &= \sigma(\hat{W}_i \phi_j + \hat{W}_i \hat{\phi}_j + U_i \psi_j + \hat{U}_i \hat{\psi}_j + b_i) \\
o_t^j &= \sigma(\hat{W}_o \phi_j + \hat{W}_o \hat{\phi}_j + U_o \psi_j + \hat{U}_o \hat{\psi}_j + b_o) \\
f_t^j &= \sigma(\hat{W}_f \phi_j + \hat{W}_f \hat{\phi}_j + U_f \psi_j + \hat{U}_f \hat{\psi}_j + b_f) \\
u_t^j &= \sigma(\hat{W}_u \phi_j + \hat{W}_u \hat{\phi}_j + U_u \psi_j + \hat{U}_u \hat{\psi}_j + b_u) \\
c_t^j &= f_t^j \odot c_{t-1}^j + i_t^j \odot u_t^j \\
a_t^j &= o_t^j \odot \tanh(c_t^j)
\end{align*}
\]

where \( i_t^j, o_t^j \) and \( f_t^j \) are the input, output and forget gates mentioned earlier. \( \hat{W}_x, \hat{W}_x, U_x, \hat{U}_x, b_x \), where \( x \in \{i, o, f, u\} \), are model parameters.

With this state transition mechanism, information of each node is propagated to all its neighboring nodes after each step. So after several transi-
tion steps, each node state contains the information of a large context, including its ancestors, descendants, and siblings. For the worst case where the input graph is a chain of nodes, the maximum number of steps necessary for information from one arbitrary node to reach another is equal to the size of the graph. We experiment with different numbers of transition steps to study the effectiveness of global encoding.

4.1.1 Input Representation

The edges of an AMR graph contain labels, which represent relations between the nodes they connect, and are thus important for modeling the graphs. The representation for each edge \( (i, j, l) \) is defined as:

\[
x_{i,j}^l = W_4 [e_l; e_{v_i}] + b_4,
\]

where \( e_l \) and \( e_{v_i} \) are the embeddings of edge label \( l \) and source node \( v_i \), and \( W_4 \) and \( b_4 \) are model parameters.

4.2 Incorporating AMR information with a doubly-attentive decoder

There is no one-to-one correspondence between AMR nodes and source words. To incorporate additional knowledge from an AMR graph, an external attention model is adopted over the baseline model. In particular, the attention memory from the AMR graph is the last graph state \( \hat{g}_T = \{ \hat{a}_T^{|v_e|} \} \). In addition, the contextual vector based on the graph state is calculated as:

\[
\hat{c}_{m,j} = \hat{v}_2^T \tanh(W_a \hat{a}_T^{|v_e|} + \hat{W}_s s_m + \hat{b}_2)
\]

\[
\hat{a}_{m,j} = \frac{\exp(\hat{c}_{m,j})}{\sum_{j=1}^M \exp(\hat{c}_{m,j})}
\]

\( W_a, \hat{W}_s, \hat{v}_2 \) and \( \hat{b}_2 \) are model parameters. The new context vector \( \hat{\xi}_m \) is calculated via \( \sum_{i=1}^N \hat{\alpha}_{m,i} \hat{a}_T^{|v_e|} \). Finally, \( \hat{\xi}_m \) is incorporated into the calculation of the output probability distribution over the target vocabulary (previously defined in Equation 1):

\[
P_{\text{vocab}} = \text{softmax}(V_3 [s_m, \hat{\xi}_m, \hat{\xi}_m] + b_3)
\]

5 Training

Given a set of training instances \( \{(X^{(1)}, Y^{(1)}), (X^{(2)}, Y^{(2)}), \ldots\} \), we train our models using the cross-entropy loss over each gold-standard target sequence \( Y^{(j)} = y_1^{(j)}, y_2^{(j)}, \ldots, y_M^{(j)} \):

\[
l = - \sum_{m=1}^M \log p(y_m^{(j)} | y_{m-1}^{(j)}, \ldots, y_1^{(j)}, X^{(j)}; \theta)
\]

\( X^{(j)} \) represents the inputs for the \( j \)-th instance, which is a source sentence for our baseline, or a source sentence paired with an automatically parsed AMR graph for our model. \( \theta \) represents the model parameters.

6 Experiments

We empirically investigate the effectiveness of AMR for English-to-German translation.

6.1 Setup

Data We use the WMT16\(^3\) English-to-German dataset, which contains around 4.5 million sentence pairs for training. In addition, we use a subset of the full dataset (News Commentary v11 (NC-v11), containing around 243 thousand sentence pairs) for development and additional experiments. For all experiments, we use newstest2013 and newstest2016 as the development and test sets, respectively.

To preprocess the data, the tokenizer from Moses\(^4\) is used to tokenize both the English and German sides. The training sentence pairs where either side is longer than 50 words are filtered out after tokenization. To deal with rare and compound words, byte-pair encoding (BPE)\(^5\) (Sennrich et al., 2016) is applied to both sides. In particular, 8000 and 16000 BPE merges are used

\( \text{http://www.statmt.org/wmt16/translation-task.html} \)

\( \text{http://www.statmt.org/moses/} \)

\( \text{https://github.com/rsennrich/subword-nmt} \)

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#Sent.</th>
<th>#Tok. (EN)</th>
<th>#Tok. (DE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NC-v11</td>
<td>226K</td>
<td>6.4M</td>
<td>7.3M</td>
</tr>
<tr>
<td>Full</td>
<td>4.17M</td>
<td>109M</td>
<td>118M</td>
</tr>
<tr>
<td>News2013</td>
<td>3000</td>
<td>84.7K</td>
<td>95.6K</td>
</tr>
<tr>
<td>News2016</td>
<td>2999</td>
<td>88.1K</td>
<td>98.8K</td>
</tr>
</tbody>
</table>

Table 1: Statistics of the dataset. Numbers of tokens are after BPE processing.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>EN-ori</th>
<th>EN</th>
<th>AMR</th>
<th>DE</th>
</tr>
</thead>
<tbody>
<tr>
<td>NC-v11</td>
<td>79.8K</td>
<td>8.4K</td>
<td>36.6K</td>
<td>8.3K</td>
</tr>
<tr>
<td>Full</td>
<td>874K</td>
<td>19.3K</td>
<td>403K</td>
<td>19.1K</td>
</tr>
</tbody>
</table>

Table 2: Sizes of vocabularies. EN-ori represents original English sentences without BPE.

\( ^3 \text{https://github.com/rsennrich/subword-nmt} \)
on the News Commentary v11 subset and the full training set, respectively. On the other hand, JAMR\textsuperscript{6} (Flanigan et al., 2016) is adopted to parse the English sentences into AMRs before BPE is applied. The statistics of the training data and vocabularies after preprocessing are shown in Table 1 and 2, respectively. For the experiments with the full training set, we used the top 40K of the AMR vocabulary, which covers more than 99.6% of the training set.

For our dependency-based and SRL-based baselines (which will be introduced in Baseline systems), we choose Stanford CoreNLP (Manning et al., 2014) and IBM SIRE to generate dependency trees and semantic roles, respectively. Since both dependency trees and semantic roles are based on the original English sentences without BPE, we used the top 100K frequent English words, which cover roughly 99.0% of the training set.

Hyperparameters We use the Adam optimizer (Kingma and Ba, 2014) with a learning rate of 0.0005. The batch size is set to 128. Between layers, we apply dropout with a probability of 0.2. The best model is picked based on the cross-entropy loss on the development set. For model hyperparameters, we set the graph state transition number to 10 according to development experiments. Each node takes information from at most 6 neighbors. BLEU (Papineni et al., 2002), TER (Snover et al., 2006) and Meteor (Denkowski and Lavie, 2014) are used as the metrics on cased and tokenized results.

For experiments with the NC-v11 subset, both word embedding and hidden vector sizes are set to 500, and the models are trained for at most 30 epochs. For experiments with full training set, the word embedding and hidden state sizes are set to 800, and our models are trained for at most 10 epochs. For all systems, the word embeddings are randomly initialized and updated during training.

Baseline systems We compare our model with the following systems. \textit{Seq2seq} represents our attention-based LSTM baseline (§3), and \textit{Dual2seq} is our model, which takes both a sequential and a graph encoder and adopts a doubly-attentive decoder (§4). To show the merit of AMR, we further contrast our model with the following baselines, all of which adopt the same doubly-attentive framework with a BiLSTM for encoding BPE-segmented source sentences: \textit{Dual2seq-LinAMR} uses another BiLSTM for encoding linearized AMRs. \textit{Dual2seq-Dep} and \textit{Dual2seq-SRL} adopt our graph recurrent network to encode original source sentences with dependency and semantic role annotations, respectively. The three baselines are useful for contrasting different methods of encoding AMRs and for comparing AMRs with other popular structural information for NMT.

We also compare with Transformer (Vaswani et al., 2017) and OpenNMT (Klein et al., 2017), trained on the same dataset and with the same set of hyperparameters as our systems. In particular, we compare with \textit{Transformer-tf}, one popular implementation\textsuperscript{7} of Transformer based on TensorFlow, and we choose \textit{OpenNMT-tf}, an official release\textsuperscript{8} of OpenNMT implemented with TensorFlow. For a fair comparison, \textit{OpenNMT-tf} has 1 layer for both the encoder and the decoder, and \textit{Transformer-tf} has the default configuration (N=6), but with parameters being shared among different blocks.

6.2 Development experiments

Figure 4 shows the system performances as a function of the number of graph state transitions on the development set. \textit{Dual2seq (self)} represents our dual-attentive model, but its graph encoder encodes the source sentence, which is treated as a chain graph, instead of an AMR graph. Compared with \textit{Dual2seq}, \textit{Dual2seq (self)} has the same number of parameters, but without semantic information from AMR. Due to hardware limitations, we

\textsuperscript{6}https://github.com/jflanigan/jamr
\textsuperscript{7}https://github/Kyubyong/transformer
\textsuperscript{8}https://github/OpenNMT/OpenNMT-tf

![Figure 4: DEV BLEU scores against transition steps for the graph encoders. The state transition is not applicable to \textit{Seq2seq}, so we draw a dashed line to represent its performance.](image-url)
Table 3: Test performance. NC-v11 represents training only with the NC-v11 data, while Full means using the full training data. * represents significant (Koehn, 2004) result ($p < 0.01$) over Seq2seq. ↓ indicates the lower the better.

Our Dual2seq shows consistent performance improvement by increasing the transition number both from 1 to 5 (roughly +1.3 BLEU points) and from 5 to 10 (roughly 0.2 BLEU points). The former shows greater improvement than the latter, showing that the performance starts to converge after 5 transition steps. Further increasing transition steps from 10 to 12 gives a slight performance drop. We set the number of state transition steps to 10 for all experiments according to these observations.

On the other hand, Dual2seq (self) shows only small improvements by increasing the state transition number, and it does not perform better than Seq2seq. Both results show that the performance gains of Dual2seq are not due to an increased number of parameters.

6.3 Main results

Table 3 shows the test BLEU, TER and Meteor scores of all systems trained on the small-scale News Commentary v11 subset or the large-scale full set. Dual2seq is consistently better than the other systems under all three metrics, showing the effectiveness of the semantic information provided by AMR. Especially, Dual2seq is better than both OpenNMT-tf and Transformer-tf. The recurrent graph state transition of Dual2seq is similar to Transformer in that it iteratively incorporates global information. The improvement of Dual2seq over Transformer-tf undoubtedly comes from the use of AMRs, which provide complementary information to the textual inputs of the source language.

In terms of BLEU score, Dual2seq is significantly better than Seq2seq in both settings, which shows the effectiveness of incorporating AMR information. In particular, the improvement is much larger under the small-scale setting (+3.2 BLEU) than that under the large-scale setting (+1.7 BLEU). This is an evidence that structural and coarse-grained semantic information encoded in AMRs can be more helpful when training data are limited.

When trained on the NC-v11 subset, the gap between Seq2seq and Dual2seq under Meteor (around 5 points) is greater than that under BLEU (around 3 points). Since Meteor gives partial credit to outputs that are synonyms to the reference or share identical stems, one possible explanation is that the structural information within AMRs helps to better translate the concepts from the source language, which may be synonyms or paronyms of reference words.

As shown in the second group of Table 3, we further compare our model with other methods of leveraging syntactic or semantic information. Dual2seq-LinAMR shows much worse performance than our model and only slightly outperforms the Seq2seq baseline. Both results show that simply taking advantage of the AMR concepts without their relations does not help very much. One reason may be that AMR concepts, such as John and Mary, also appear in the textual input, and thus are also encoded by the other (sequential) encoder. The gap between Dual2seq and Dual2seq-LinAMR comes from modeling the relations between concepts, which can be helpful for deciding target word order by enhancing the relations in source sentences. We conclude that properly encoding AMRs is necessary to make them

---

9AMRs can contain multi-word concepts, such as New York City, but they are in the textual input.
Encoding dependency trees instead of AMRs, Dual2seq-Dep shows a larger performance gap with our model (17.8 vs 19.2) on small-scale training data than on large-scale training data (25.0 vs 25.5). It is likely because AMRs are more useful on alleviating data sparsity than dependency trees, since words are lemmatized into unified concepts when parsing sentences into AMRs. For modeling long-range dependencies, AMRs have one crucial advantage over dependency trees by modeling concept-concept relations more directly. It is because AMRs drop function words, thus the distances between concepts are generally closer in AMRs than in dependency trees. Finally, Dual2seq-SRL is less effective than our model, because the annotations labeled by SRL are a subset of AMRs.

We outperform Marcheggiani et al. (2018) on the same datasets, although our systems vary in a number of respects. When trained on the NC-v11 data, they show BLEU scores of 14.9 only with their BiLSTM baseline, 16.1 using additional dependency information, 15.6 using additional semantic roles and 15.8 taking both as additional knowledge. Using Full as the training data, the scores become 23.3, 23.9, 24.5 and 24.9, respectively. In addition to the different semantic representation being used (AMR vs SRL), Marcheggiani et al. (2018) laid graph convolutional network (GCN) (Kipf and Welling, 2017) layers on top of a bidirectional LSTM (BiLSTM) layer, and then concatenated layer outputs as the attention memory. GCN layers encode the semantic role information, while BiLSTM layers encode the input sentence in the source language, and the concatenated hidden states of both layers contain information from both semantic role and source sentence. For incorporating AMR, since there is no one-to-one word-to-node correspondence between a sentence and the corresponding AMR graph, we adopt separate attention models. Our BLEU scores are higher than theirs, but we cannot conclude that the advantage primarily comes from AMR.

6.4 Analysis

Influence of AMR parsing accuracy To analyze the influence of AMR parsing on our model performance, we further evaluate on a test set where the gold AMRs for the English side are available. In particular, we choose the Little Prince corpus, which contains 1562 sentences with gold AMR annotations. Since there are no parallel German sentences, we take a German-version Little Prince novel, and then perform manual sentence alignment. Taking the whole Little Prince corpus as the test set, we measure the influence of AMR parsing accuracy by evaluating on the test set when gold or automatically-parsed AMRs are available. The automatic AMRs are generated by parsing the English sentences with JAMR.

Table 4 shows the BLEU scores of our Dual2seq model taking gold or automatic AMRs as inputs. Not listed in Table 4, Seq2seq achieves a BLEU score of 15.6, which is 1.2 BLEU points lower than using automatic AMR information. The improvement from automatic AMR to gold AMR (+0.7 BLEU) is significant, which shows that the translation quality of our model can be further improved with an increase of AMR parsing accuracy. However, the BLEU score with gold AMR does not indicate the potentially best performance that our model can achieve. The primary reason is that even though the test set is coupled with gold AMRs, the training set is not. Trained with automatic AMRs, our model can learn to selectively trust the AMR structure. An additional reason is the domain difference: the Little Prince data are in the literary domain while our training data are in the news domain. There can be a further performance gain if the accuracy of the automatic AMRs on the training set is improved.

<table>
<thead>
<tr>
<th>AMR Anno.</th>
<th>BLEU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automatic</td>
<td>16.8</td>
</tr>
<tr>
<td>Gold</td>
<td>17.5*</td>
</tr>
</tbody>
</table>

Table 4: BLEU scores of Dual2seq on the little prince data, when gold or automatic AMRs are available.

Figure 5: Test BLEU score of various sentence lengths available. In particular, we choose the Little Prince corpus, which contains 1562 sentences with gold AMR annotations. Since there are no parallel German sentences, we take a German-version Little Prince novel, and then perform manual sentence alignment. Taking the whole Little Prince corpus as the test set, we measure the influence of AMR parsing accuracy by evaluating on the test set when gold or automatically-parsed AMRs are available. The automatic AMRs are generated by parsing the English sentences with JAMR.

Table 4 shows the BLEU scores of our Dual2seq model taking gold or automatic AMRs as inputs. Not listed in Table 4, Seq2seq achieves a BLEU score of 15.6, which is 1.2 BLEU points lower than using automatic AMR information. The improvement from automatic AMR to gold AMR (+0.7 BLEU) is significant, which shows that the translation quality of our model can be further improved with an increase of AMR parsing accuracy. However, the BLEU score with gold AMR does not indicate the potentially best performance that our model can achieve. The primary reason is that even though the test set is coupled with gold AMRs, the training set is not. Trained with automatic AMRs, our model can learn to selectively trust the AMR structure. An additional reason is the domain difference: the Little Prince data are in the literary domain while our training data are in the news domain. There can be a further performance gain if the accuracy of the automatic AMRs on the training set is improved.

10https://amr.isi.edu/download.html
Performance based on sentence length We hypothesize that AMRs should be more beneficial for longer sentences: those are likely to contain long-distance dependencies (such as discourse information and predicate-argument structures) which may not be adequately captured by linear chain RNNs but are directly encoded in AMRs. To test this, we partition the test data into four buckets by length and calculate BLEU for each of them. Figure 5 shows the performances of our model along with Dual2seq-Dep and Seq2seq. Our model outperforms the Seq2seq baseline rather uniformly across all buckets, except for the first one where they are roughly equal. This may be surprising. On the one hand, Seq2seq fails to capture some dependencies for medium-length instances; on the other hand AMR parses are more noisy for longer sentences, which prevents us from obtaining extra improvements with AMRs.

Dependency trees have been proved useful in capturing long-range dependencies. Figure 5 shows that AMRs are comparatively better than dependency trees, especially on medium-length (21-30) sentences. The reason may be that the AMRs of medium-length sentences are much more accurate than longer sentences, thus are better at capturing the relations between concepts. On the other hand, even though dependency trees are more accurate than AMRs, they still fail to represent relations for long sentences. It is likely because relations for longer sentences are more difficult to detect. Another possible reason is that dependency trees do not incorporate coreferences, which AMRs consider.

Human evaluation We further study the translation quality of predicate-argument structures by conducting a human evaluation on 100 instances from the testset. In the evaluation, translations of both Dual2seq and Seq2seq, together with the source English sentence, the German reference, and an AMR are provided to a German-speaking annotator to decide which translation better captures the predicate-argument structures in the source sentence. To avoid annotation bias, translation results of both models are swapped for some instances, and the German annotator does
not know which model each translation belongs to. The annotator either selects a “winner” or makes a “tie” decision, meaning that both results are equally good.

Out of the 100 instances, Dual2seq wins on 46, Seq2seq wins on 23, and there is a tie on the remaining 31. Dual2seq wins on almost half of the instances, about twice as often as Seq2seq wins, indicating that AMRs help in translating the predicate-argument structures on the source side.

Case study The outputs of the baseline system (Seq2seq) and our final system (Dual2seq) are shown in Figure 6. In the first sentence, the AMR-based Dual2seq system correctly produces the reflexive pronoun sich as an argument of the verb trafen (meet), despite the distance between the words in the system output, and despite the fact that the equivalent English words each other do not appear in the system output. This is facilitated by the argument structure in the AMR analysis.

In the second sentence, the AMR-based Dual2seq system produces an overly literal translation for the English phrasal verb come across. The Seq2seq translation, however, incorrectly states that the police vehicles are refugees. The difficulty for the Seq2seq probably derives in part from the fact that are and coming are separated by the word constantly in the input, while the main predicate is clear in the AMR representation.

In the third sentence, the Dual2seq system correctly translates the object of breed as worms, while the Seq2seq translation incorrectly states that the scientists breed themselves. Here the difficulty is likely the distance between the object and the verb in the German output, which causes the Seq2seq system to lose track of the correct input position to translate.

7 Conclusion

We showed that AMRs can improve neural machine translation. In particular, the structural semantic information from AMRs can be complementary to the source textual input by introducing a higher level of information abstraction. A graph recurrent network (GRN) is leveraged to encode AMR graphs without breaking the original graph structure, and a sequential LSTM is used to encode the source input. The decoder is a doubly-attentive LSTM, taking the encoding results of both the graph encoder and the sequential encoder as attention memories. Experiments on a standard benchmark showed that AMRs are helpful regardless of the sentence length, and are more effective than other more popular choices, such as dependency trees and semantic roles.

Acknowledgments

We would like to thank the action editor and the anonymous reviewers for their insightful comments. We also thank Kai Song from Alibaba for suggestions on large-scale training, Parker Riley for comments on the draft, and Rochester’s CIRC for computational resources.

References


