Abstract

Abstract Meaning Representations (AMR) are a broad-coverage semantic formalism which represents sentence meaning as a directed acyclic graph. To train most AMR parsers, one needs to segment the graph into subgraphs and align each such subgraph to a word in a sentence; this is normally done at preprocessing, relying on hand-crafted rules. In contrast, we treat both alignment and segmentation as latent variables in our model and induce them as part of end-to-end training. As marginalizing over the structured latent variables is infeasible, we use the variational autoencoding framework. To ensure end-to-end differentiable optimization, we introduce a continuous differentiable relaxation of the segmentation and alignment problems. We observe that inducing segmentation yields substantial gains over using a ‘greedy’ segmentation heuristic. The performance of our method also approaches that of a model that relies on Lyu and Titov (2018)’s segmentation rules, which were hand-crafted to handle individual AMR constructions.

1 Introduction

Abstract Meaning Representations (AMR) (Banarescu et al., 2013) are a broad-coverage semantic formalism which represents sentence meaning as rooted labeled directed acyclic graphs. The representations have been exploited in a wide range of tasks, including text summarization (Liu et al., 2015; Dohare and Karnick, 2017; Hardy and Vlachos, 2018), machine translation (Jones et al., 2012; Song et al., 2019), paraphrase detection (Issa et al., 2018) and question answering (Mitra and Baral, 2016).

An AMR graph can be regarded as consisting of multiple concept subgraphs, which can be individually aligned to sentence tokens (Flanigan et al., 2014). In Figure 1, each dashed box represents the boundary of a single semantic subgraph. Red arrows represent the alignment between subgraphs and tokens. For example, ‘(o / opine-01: ARG1 (t / thing))’ signifies something that triggers the frame ‘opine-01’ and fills its semantic role ARG1. Intuitively, this subgraph needs to be aligned to the token ‘opinion’. Similarly, ‘(b / boy)’ should be aligned to the token ‘boy’. Given such an alignment and segmentation, it can be straightforward to construct a simple graph-based parser; for example, parsing could be framed as tagging input tokens with subgraphs (including empty subgraphs), followed by predicting relations between the subgraphs. The key obstacle to training an AMR parser is that the segmentation and alignment between AMR subgraphs and words are latent, i.e. not annotated in the data.

Most previous work adopts a pipeline approach to handling the obstacle. They rely on a pre-learned aligner (Pourdamghani et al., 2014) to produce the alignment, and apply a rule system to segment the AMR subgraph (Flanigan et al., 2014; Werling et al., 2015; Damonte et al., 2017; Ballesteros and Al-Onaizan, 2017; Peng et al., 2015; Artzi et al., 2015; Groschwitz et al., 2018). Notably, Lyu and Titov (2018) jointly optimize the parser and alignment in the context of a graph-based approach. Yet, rules, handling specific constructions, still need to be crafted to segment the graph. As more construc-
tions are getting introduced to AMRs (Bonial et al., 2018) and AMR sembanks in languages other than English are being developed (Anchiêta and Pardo, 2018; Migueles-Abraira et al., 2018; Song et al., 2020), getting rid of the rules and learning graph segmentation from scratch becomes a compelling problem to tackle.

We propose to optimize a graph-based parser while treating alignment and graph segmentation as latent variables. A graph-based parser consists of two parts: concept identification and relation identification. The concept identification model generates the AMR nodes, and relation identification component decides on the labeled edges. During training, both components rely on latent alignment and segmentation which is being induced at the same time. Importantly, at test time, the parser simply tags the input with the subgraphs and predicts the relations, so there is no test-time over-head from using the latent-structure apparatus. An extra benefit of this approach, in contrast to encoder-decoder AMR models (Konstas et al., 2017; van Noord and Bos, 2017; Cai and Lam, 2020) is its transparency, as one can readily see which input token triggered each subgraph.

To achieve our goal, we frame the alignment and segmentation problems as choosing a generation order of concept nodes, as we explain in section 2.2. As marginalization over the latent generation orders is infeasible, we adopt the variational autoencoder (VAE) framework (Kingma and Welling, 2014). Loosely speaking, a trainable neural module (an encoder in the VAE) is used to sample a plausible generation order (i.e. a segmentation plus an alignment), which is then used to train the parser (a decoder in the VAE). However, one cannot ‘differentiate through’ a sample of discrete variables to train the encoder. We adopt the stochastic softmax (Paulus et al., 2020) technique to address this challenge. Furthermore, to efficiently apply the stochastic softmax, we derive an inference algorithm for our problem; it can be regarded as an instance of Bregman’s method (Bregman, 1967).

Our method makes very few assumptions about the nature of the graphs, so it may be effective in other tasks that can be framed as graph prediction (e.g., executable semantic parsing (Liang, 2016) or scene graph prediction (Xu et al., 2017)), but we leave this for future work.

2 Casting Alignment and Segmentation as Choosing a Generation Order

2.1 Preliminaries

We introduce the basic concepts and notation here. We refer to words in the sentences as \( x = (x_0, \ldots, x_{n-1}) \), where \( n \) is sentence length. The concepts (i.e. labeled nodes) are \( v = (v_0, v_1, \ldots, v_m) \), where \( m \) is the number of concepts. In particular, \( v_m = \emptyset \) denotes a dummy terminal node; we refer to all nodes, except for the terminal node (\( \emptyset \)), as concept nodes.

A relation between ‘predicate concept’ \( i \) and ‘argument concept’ \( j \) is denoted by \( E_{ij} \in \mathcal{E} \); it

---

1This is the graph-recategorization system first developed by Lyu and Titov (2018). The rules are crafted for individual AMR constructions. Then, the subsequent work extends the rules (Zhang et al., 2019a; Cai and Lam, 2020). We adopt the code from Zhang et al. (2019a).
Figure 2: AMR concept identification model generates nodes following latent generation order at training time.

Figure 3: At test time, the AMR concept identification model generates nodes autoregressively starting from each token.

is set to $\emptyset$ if $j$ is not an argument of $i$. We will use $E$ to denote all edges (i.e. relations) in the graph. In addition, we refer to the whole graph as $G = (v, E)$.

Our goal is to associate each input token to a (potentially empty) subset of the graph’s nodes, while making sure that we get a partition of the node set, i.e. each node in the original graph belongs to exactly one subset. In that way we deal with both segmentation and alignment. Each subset uniquely corresponds to a vertex-induced subgraph (i.e. the subset of nodes together with any edges whose both endpoints are in this subset). For this reason, we will refer to the problem as graph decomposition and to each subset as a subgraph. We will explain how we deal with edges in subsequent sections.

2.2 Generation Order

Instead of only selecting a subset of nodes for each token (and doing it in one shot), we also select the order in which nodes in the subset are chosen. In Figure 2, dashed red arrows point from every node to the subsequent node to be selected. For example, for word ‘opinion’, the node ‘opine-01’ is chosen first, and then it is followed by another node (‘thing’). After this node, we have an arrow pointing to the node $\emptyset$, signifying the termination.

We refer to the set of red arrows as a generation order. To recover a subgraph structure from a generation order, we assign connected nodes (excluding the terminal node) to the same subgraph. Then, a subgraph will be aligned to the token that generated those nodes. In our example, ‘opine-01’ and ‘thing’ are connected, and those are both aligned to the word ‘opinion’. The alignment is encoded by arrows between tokens and concept nodes, while the segmentation is represented with arrows in-between concept nodes.

From the modeling perspective, the nodes will be generated with an autoregressive model, thus, we do not have to explicitly represent and store all potential subgraphs in our model. It is easy to see how to apply such autoregressive concept identification model at test time (Figure 3). Starting from each token, a chain of nodes are generated until a $\emptyset$ node is predicted. It is more challenging to see how to induce the order and train the autoregressive model at the same time; we will discuss this in Sections 3 and 4.

While in Figure 2 the red arrows yield a valid generation order, the arrows need to obey certain constraints. Formally, we denote alignment by $A \in \{0, 1\}^n \times (m+1)$, where $A_{ki} = 1$ ensures that for token $k$ we start by generating node $i$. As the token can only point to one node, we have a constraint $\sum_i A_{ki} = 1$. Similarly, we have segmentation $S \in \{0, 1\}^{m \times (m+1)}$ with a constraint $\sum_j S_{ij} = 1$. Here, $S_{ij} = 1$ encodes that node $i$ is followed by node $j$. In Figure 2, we have $A_{03} = A_{10} = A_{23} = A_{33} = A_{42} = 1$ and $S_{01} = S_{13} = S_{25} = 1$; the rest is 0. Now, we have the full generation order as their concatenation $O = A \circ S \in \{0, 1\}^{(n+m) \times (m+1)}$. As one node can only be generated once (except for $\emptyset$), we have a joint constraint: $\forall j \neq m, \sum_i O_{ij} = 1$. Furthermore, the graph defined by $O$ should be acyclic, as it represents the generative process. We denote the set of all valid generation orders as $O$. In the following sections, we will discuss how this generation order is used in the model and how to infer it as a latent variable while enforcing the above constraints.

3 Our Model

Formally, we aim at estimating $P_\theta(v, E | x)$, the likelihood of AMR graph given the sentence. Our graph-based parser is composed of two parts: concept identification $P_\theta(v | x, O)$ and relation identi-
The opinion of the boy

LSTM

As shown in Figure 4, our neural model first en-
well-defined for real-valued
which will also be explained in Section 4. One

variational inference and also
stochastic softmax
context which one is used.

identification and one for relation identification. For simplicity,
this objective end-to-end, we will use tools from
discuss it in Section 4.2). To efficiently optimize

Formally, the probability of concept identification step can be decomposed into probability of generating $m$ concepts nodes and $n$ terminal nodes (one for each token):

$$
P_{\theta}(v, O, v) = \log \sum_{O} P_{\theta}(O) P_{\theta}(v|x, O) P_{\theta}(E|x, O, v),$$

where $P_{\theta}(O)$ is a prior on the generation order (we
discuss it in Section 4.2). To efficiently optimize
this objective end-to-end, we will use tools from
variational inference and also stochastic softmax,
which will also be explained in Section 4. One
important requirement for applying the stochastic
softmax is that both concept and relation identification
models admit relaxation, i.e. they should be
well-defined for real-valued $O$. In the following
subsections, we go through concept identification,
relation identification, and their corresponding relaxations.

3.1 Concept Identification

As shown in Figure 4, our neural model first en-
codes the sentence with BiLSTM, producing token representations $h_k^{\text{token}}$ ($k \in [0, \ldots, n-1]$), then
generates nodes autoregressively at each token with
another LSTM.

For training, we need to be able to run the models
with any potential generation order and compute $P_{\theta}(v|x, O)$. If we take the order defined in
Figure 2, the node 1 (‘thing’) is predicted relying on
the corresponding hidden representation;

we refer to this representation as $h_1^{\text{node}}$ where is 1
is the node index. With the discrete generation
order defined by red arrows in Figure 2, $h_1^{\text{node}}$
is just the LSTM state of its parent (i.e. ‘opine-
01’). The caveat is that our computation should
be well-defined when the generation order $O$ is
soft (i.e. attention-like instead of pointer-like). In
that case $h_1^{\text{node}}$ will be a weighted sum of LSTM
representations of other nodes and input tokens,
where the weights are defined by $O$. Similarly,
the termination symbol $\emptyset$ for the token ‘opinion’
is predicted from its hidden representation; we refer
to this representation as $h_1^{\text{tail}}$, where 1 is the
position of ‘opine’ in the sentence. With the hard
generation order of Figure 2, $h_1^{\text{tail}}$ is just the LSTM
state computed after choosing the preceding node
(i.e. ‘thing’). In the relaxed case, it will again be a
weighted sum with the weights defined by $O$.

Formally, the probability of concept identification
step can be decomposed into probability of

$$
P_{\theta}(v|x, O) = \prod_{i=0}^{m-1} P_{\theta}(v_i|h_i^{\text{node}}) \prod_{k=0}^{n-1} P_{\theta}(\emptyset|h_k^{\text{tail}}).$$

$h_i^{\text{node}}$ is given by the weighted sum of the LSTM
states of preceding nodes as defined by $O$:

$$
h_i^{\text{node}} := \sum_{j=0}^{m-1} S_{ji} \text{LSTM}(h_j^{\text{node}}, v_j)
+ \sum_{k=0}^{n-1} A_{ki} h_k^{\text{token}}. $$

Note that the preceding node can be either a concept
node (then the output of the LSTM, consuming
the preceding node, is used) or a word (then we
use its contextualized encoding). Note that this
expression is ‘circular’ each node’s representation
$h_i^{\text{node}}$ is computed based on representations of all
the nodes $h_j^{\text{node}}$; $i, j \in [1, \ldots, m-1]$. Iterating
the assignment for a valid discrete generation order
(i.e. a DAG, like the one given in Figure 2), will
converge to a stationary point. The stationary point
will be equal to the result of applying the autore-
gressive model (i.e. Figure 4). It will be reached
after $T$ steps, where $T$ is the number of nodes in
the largest subgraph. This ‘message passing’ process
is fully differentiable and importantly well-defined for a relaxed (i.e., continuous) generation order. We use \( T = 4 \), as we do not expect subgraphs with more than 4 nodes.

The representations \( h_i^{\text{tail}} \), needed for the terms \( P_{\theta}(\emptyset|h_i^{\text{tail}}) \) in Equation (3), are computed similarly:

\[
h_i^{\text{tail}} = \sum_{j=0}^{m-1} B_{jk} \text{LSTM}(h_j^{\text{node}}, v_j) + \sum_{j=0}^{m-1} B_{jk} h_k^{\text{token}},
\]

where \( B_{jk} = 1 \) denotes that the concept node \( j \) is the last concept node before generating \( \emptyset \) for the token \( k \), else \( B_{jk} = 0 \). (e.g., in Figure 2, we have \( B_{11} = B_{42} = 1 \), and others being 0.) These \( B_{jk} \) can be computed from \( O = A \circ S \), as explained in Appendix C. Again, in the discrete case, the result will be exactly equivalent to what is obtained by running the corresponding autoregressive model (as in test time, Figure 4) but the computation is also well-defined and differentiable in the continuous case. Intuitively, if one to think of relaxed \( O \) as representing transition probabilities in a Markov process, then \( B_{jk} \) can thought of as the probability of having node \( j \) as the last concept node in the chain, when starting from token \( k \).

We specify \( P_{\emptyset}(v_i|h_i^{\text{node}}) \) and \( P_{\emptyset}(\emptyset|h_i^{\text{tail}}) \) in Appendix B. We use the copy mechanism to address the data sparsity (Peng et al., 2017), where the copy action will trigger the usage of nominalization dictionary (e.g., encoding that word ‘meeting’ can correspond to a node ‘meet’).

### 3.2 Relation Identification

Similarly to Lyu and Titov (2018), we use an arc-factored model for relation identification, where the score of an arc label between for an edge from \( i \) to \( j \) is a biaffine function of the node representations \( h_i^{\text{edge}} \) and \( h_j^{\text{edge}} \):

\[
P_{\theta}(E|x, O, v) = \prod_{i=1}^{m} P_{\theta}(E_{ij}|h_i^{\text{edge}}, h_j^{\text{edge}}),
\]

where \( \emptyset \) denotes concatenation, \( h_i^{\text{node}} \) is defined in section 3.1, and \( A_{ki}^{\infty} \) indicates whether \( i \) is in a subgraph aligned to token \( k \) or not. Not that this is different from \( A_{ki} \) which encodes that the node \( i \) is the first node in the subgraph (e.g., in Figure 2, \( A_{11} = 0 \) but \( A_{11}^{\infty} = 1 \)). In the continuous case, as used during training, \( A_{ki}^{\infty} \) can be thought of as the alignment probability that can be computed from \( O = A \circ S \) (see Appendix for details), and it is differentiable with respect to \( O \).

### 4 Estimating Latent Generation Order

We show how to estimate the latent generation order jointly with the parser.

#### 4.1 Variational Inference

In Equation (2), marginalization over \( O \) is intractable due to the use of neural parameterization in \( P_{\theta}(v|x, O) \) and \( P_{\theta}(E|x, O, v) \). Instead, we resort to the variational auto-encoder (VAE) framework (Kingma and Welling, 2014). Instead of using the original marginal likelihood objective, in VAE, its lower bound is used:

\[
\log \sum_{O} P_{\theta}(O) P_{\theta}(v|x, O) P_{\theta}(E|x, O, v) \\
\geq \log E_{O \sim Q_{\phi}(O|G, x)} P_{\theta}(v|x, O) P_{\theta}(E|x, O, v) \\
- KL(Q_{\phi}(O|G, x)||P_{\theta}(O)),
\]

where \( KL \) is the KL divergence, and the \( Q_{\phi}(O|G, x) \) (the encoder, aka the inference network) is a distribution parameterized with a neural network. The lower bound is maximized with respect to both the original parameters \( \theta \) and the variational parameters \( \phi \). The distribution \( Q_{\phi}(O|G, x) \) can be thought of as an approximation to the intractable posterior distribution \( P_{\theta}(O|G, x) \).

#### 4.2 Stochastic Softmax

In order to estimate the gradient with respect to the encoder parameters \( \phi \), we use the stochastic softmax (Paulus et al., 2020). The stochastic softmax is a generalization the Gumbel-softmax trick (Jang et al., 2016; Maddison et al., 2017) to the structured case, and belongs to the class of perturbation-MAP methods (Papandreou and Yuille, 2011; Hazan and Jaakkola, 2012). It lets use the reparameterization trick (Kingma and Welling, 2014) with structured discrete latent variables. Instead of sampling \( O \) directly, we sample a random variable \( \epsilon \) from a fixed distribution \( G \). Then, we apply
a deterministic parameterized function $\mathbf{O}_\phi$ to get $\mathbf{O} = \mathbf{O}_\phi(\epsilon, \mathbf{G}, \mathbf{x})$. More concretely, we independently compute logits $W \in \mathbb{R}^{(n+m) \times (m+1)}$ for all the potential edges in the generation order, and perturb them:

$$W = F_\phi(\mathbf{G}, \mathbf{x})$$  \hspace{1cm} (10)

$$\tilde{W} = W + \epsilon, \quad \text{where} \quad \epsilon_{ij} \sim \mathcal{G}(0, 1)$$  \hspace{1cm} (11)

where $F_\phi$ is a neural module that we will define later, $\mathcal{G}(0, 1)$ is the standard Gumbel distribution (Jang et al., 2016; Maddison et al., 2017), and $\epsilon \in \mathbb{R}^{(n+m) \times (m+1)}$. Then, those perturbed logits $W$ are fed into a constrained convex optimization problem that produce a soft generation order, which is differentiable with respect to the logits and satisfies the constraints:

$$\mathbf{O}(\tilde{W}, \tau) := \arg \max_{\mathbf{O} \geq 0} \langle \tilde{W}, \mathbf{O} \rangle - \tau \langle \mathbf{O}, \log \mathbf{O} \rangle$$

s.t. $\forall j < m$:

$$\sum_{i=0}^{n+m-1} O_{ij} = 1; \quad \forall i \sum_{j=0}^{m} O_{ij} = 1$$  \hspace{1cm} (12)

The linear equalities, as previously discussed in Section 2.2, need to be satisfied by a valid generation order. There is an extra acyclicity constraint, which is enforced by masking on $\mathbf{W}$ (see section 4.2.3). Importantly, we have introduced an entropic regularizer, weighted by $\tau > 0$ (‘the temperature’). This entropic regularizer makes the optimum $\mathbf{O}(\tilde{W}, \tau)$ differentiable with respect to $\mathbf{W}$ and, consequently, with respect to $\phi$. Now, we have

$$\mathbf{O}_\phi(\epsilon, \mathbf{G}, \mathbf{x}) = \mathbf{O}(\tilde{W}, \tau)$$  \hspace{1cm} (13)

We still need to handle the KL term in Equation (9). We define the prior probability $P_\theta(\mathbf{O})$ implicitly by having $\mathbf{W} = 0$ in the stochastic softmax framework. Even then, $KL(Q_\phi(\mathbf{O}|\mathbf{G}, \mathbf{x})|P_\theta(\mathbf{O}))$ cannot be easily computed. Following Mena et al. (2018), we upper bound it\(^4\) by replacing it with $KL(\mathcal{G}(\mathbf{W}, 1)||\mathcal{G}(0, 1))$, which is available in closed form. In addition, we apply the Free Bits trick to prevent ‘the posterior collapse’ (Kingma et al., 2017). In other words, we use $\max(\lambda, KL(\mathcal{G}(\mathbf{W}, 1)||\mathcal{G}(0, 1)))$.

\(^4\)Consequently, this results in a lower bound on Expression (9).

### 4.2.1 Bregman’s Method

To optimize objective (12) we iterate over the following steps of optimization:

$$\log \mathbf{O}(0) = \tilde{W}$$  \hspace{1cm} (14)

$$\forall j < m, \quad \log \mathbf{O}_{i,m} = \log \text{Softmax}(\mathbf{O}_{i,m}) \quad \text{(15)}$$

$$\forall i, \quad \log \mathbf{O}_{i,t+1} = \log \text{Softmax}(\mathbf{O}_{i,t+1}) \quad \text{(16)}$$

where $\{i, :\}$ index ith row, and $\{:, j\}$ index jth column.

**Proposition 1.** $\lim_{t \to \infty} \exp(\log \mathbf{O}(t)) = \mathbf{O}(\tilde{W}, \tau)$ where $\mathbf{O}(\tilde{W}, \tau)$ is defined in Equation (12).

See Appendix F for a proof based on the Bregman’s method (Bregman, 1967). In practice, we take $t = 500$, and have $\mathbf{O}_\phi(\epsilon, \mathbf{G}, \mathbf{x}) = \exp(\log \mathbf{O}(500))$. Importantly, this algorithm is highly parallelizable and amendable to batch implementation on GPU. We compute the gradient with unrolled optimization.\(^5\)

\(^5\)Alternatively, an exact backward pass can be computed using the implicit function theorem, as in Agrawal et al. (2019). The implementation is unfortunately too slow for our purposes.

### 4.2.2 Structured Straight-Through Estimator

One limitation of the differentiable sampling is that it creates a gap between training (where a continuous relaxation is used) and testing (where everything is discrete). One popular way to bridge this gap is to use the Straight-Through (ST) estimator (Bengio et al., 2013), i.e. use a discrete solution that it creates a gap between training (where a continuous relaxation is used) and testing (where everything is discrete). One popular way to bridge this gap is to use the Straight-Through (ST) estimator (Bengio et al., 2013), i.e. use a discrete solution in the forward computation pass, but backpropagate with the relaxed computation. In our ST estimator, we use $\mathbf{O}(\tilde{W}, 0)$ in the forward pass, and set $\nabla_{\tilde{W}} \mathbf{O}(\tilde{W}, 0) := \nabla_{\tilde{W}} \mathbf{O}(\tilde{W}, \tau)$. See Appendix G for a more extended discussion.

### 4.2.3 Neural Parameterization

We introduce the neural modules used for estimating logits $W = F_\phi(\mathbf{G}, \mathbf{x})$ and also the masking mechanism that both ensures acyclicity and enables the use of the copy mechanism. We have $W = W^{\text{raw}} + W^{\text{mask}}$. First, we define the un-
We experiment on LDC2016E25 (AMR2.0) and LDC2020T02 (AMR3.0). Pre-processing, post-

processing and hyper-parameter details are provided in Appendix. The evaluation is primarily based on Smatch (Cai and Knight, 2013), while more fine-grained information is produced with the evaluation tool of Damonte et al. (2017). We compare our generation-order induction framework to two ways of ‘pre-fixing’ the segmentation, i.e. producing the segmentation on a preprocessing step. We vary the segmentation methods while keeping the rest of the model identical to our full model (e.g., the same autoregressive model). We also evaluate against other recent parsers and provide ablation studies for our induction framework, as well as analysis of the induced segmentation.

6.1 Pre-fixed Segmentation

We introduce two kinds of pre-fixed segmentation for comparison: (1) a greedy segmentation that serves as a baseline and (2) a hand-crafted rule-based segmentation, which relies on rules designed to handle specific AMR constructions. Arguably, the latter can be thought of as an upper bound for how well an induction method can do.

The fixed segmentations can be incorporated into our latent-generation-order framework, so the alignment between concept nodes and the tokens will still be induced. This is achieved by having a pre-fixed S, then set \( S_{ij}^{\text{mask}} = \infty(S_{ij} - 0.5) \) if \( j \neq m \), and \( S_{im}^{\text{mask}} = \infty(0.5 - \sum_{j<m} S_{ij}) \).

Greedy Segmentation. Many nodes are aligned to tokens with the copy mechanism. We could force the unaligned nodes to join its neighbors. This is very similar to the forced alignment of unaligned nodes used in the transition parser of Naseem et al. (2019). We traversal the AMR graph the same way as we do when we produce the masking (Section 4.2.3). During the traversal, we greedily combine subgraphs until one of the constraints are violated: (1) the combined subgraph will have more than 4 nodes; (2) the combined subgraph will have more than 2 copy-able nodes. We present the algorithm recursively (see Algorithm 1).

Variable \( z_i \) indicates whether node \( i \) is copy-able and \( T = 4 \) represent the maximum subgraph size; \( n \) denotes the current subgraph size; \( z \) indicates whether the current subgraph contain a copy-able node; \( k \) is the last node in the current subgraph, which is used to generate to future nodes in a subgraph. The condition \( n + n' \leq T \land z' + z \leq 1 \) determines whether we combine the current subgraph rooted at node \( i \) and the subgraph rooted at
A few CEOs ....

We compare our model with recent AMR parsers in (2018), or more specifically its reimplementation. Overall, our models perform similarly to observed minor variations in scores; we selected the best one.

6.2 Results

We compare our model with recent AMR parsers in Table 1. Overall, our models perform similarly to recent parsers, except for Cai and Lam (2020). Importantly, both our VAE model and the rule-based segmentation achieve high concept identification scores. This suggests that the bottleneck of our graph-based parser is on the relation identification stage, which we largely borrowed from Lyu and Titov (2018) and kept relatively basic. For example, independent scoring of the edges may be too restrictive.

We also experimented on AMR 3.0, results are summarized in Table 2. Overall, our VAE segmentation beats the baseline and performs close to the rule-based system. In addition, the performance gap between the rule-based system and VAE is smaller on AMR 3.0, presumably because the rules were developed for AMR 2.0.

6.3 Ablations

Our full model uses the Structured Straight-Through (ST) gradient estimator and the Free Bits with $\lambda = 10$. We perform analysis of different variations of the stochastic softmax: (1) the soft stochastic softmax is the original one with the entropic regularizer (see section 4.2); (2) the rounded stochastic softmax, which applies a threshold function (at 0.5) to the soft prediction; (3) our full model with the ST estimator. As we can see in

We experimented with several potential orderings and observed minor variations in scores; we selected the best one.

**Algorithm 1:** Greedy Segmentation

```
Input: graph G, node index i
Result: segmentation S, n, z, k
S = 0, k = i, n = 1, z = z_i;
forall j ∈ Child[i] do
  if j notvisited then
    S', n', z', k' = Greedy(G, j);
    if n + n' ≤ T ∧ z' + z ≤ 1 then
      S_k_j = 1, n = n + n',
      z = z + z' k = k';
  end
end
```

Table 1: Scores with standard deviation on the AMR 2.0 test set. Integer number denotes 2 significant digits. Results are over 4 runs.

| Metric | Concept | SRL | Smatch |
|--------|---------|-----|--------|
| Cai and Lam | 88.1 | 74.2 | 80.2 |
| Zhang et al. | 86.0 | 71.0 | 77.0 |
| Naseem et al. | 86.0 | 72.0 | 75.5 |
| greedy | 87.5 ± 0.1 | 71.3 ± 0.1 | 75.2 ± 0.1 |
| rule | 88.7 ± 0.2 | 73.6 ± 0.2 | 76.8 ± 0.4 |
| full | 88.3 ± 0.3 | 73.0 ± 0.2 | 76.1 ± 0.2 |

Table 2: Scores with standard deviation on the AMR 3.0 test set, averaged over 2 runs.

| Metric | Concept | SRL | Smatch |
|--------|---------|-----|--------|
| greedy | 87.0 | 71.5 | 74.8 |
| rule | **88.0** | **72.6** | **75.8** |
| full | 87.8 | **72.9** | 75.6 |

---

8We experimented with several potential orderings and observed minor variations in scores; we selected the best one.

9To author’s knowledge, there is no published results on AMR 3.0 yet.

10Such rounding does not provide any guarantee of being a valid generation order, but serves as a baseline.
### Table 3: Scores with different latent segmentation on the AMR 2.0 test set. Scores are averaged over 2 runs

| Metric          | Concept | SRL | Smatch |
|-----------------|---------|-----|--------|
| soft            | 84.9    | 68.1| 70.3   |
| rounding        | 87.7    | 71.8| 74.5   |
| straight-through | **88.3**| **73.0**| **76.1** |

Table 3, there is a substantial gap between using structured ST and the two other versions. This illustrates the need for exposing the parsing model to discrete structures in training.

We examine the importance of the variational auto-encoder framework and the Free Bits trick. In our parameterization discussed in section 4.2.3, it is possible to set $A^\text{raw} = 0$ and $S^\text{raw} = 0$, which corresponds to sampling from the prior in training (i.e. quasi-uniformly while respecting the constraints defined by masking) rather than learning this importance sampling distribution with the VAE encoder. There are 4 potential options: (1) ‘full prior’, $A^\text{raw} = 0$ and $S^\text{raw} = 0$; (2) ‘alignment prior’, $A^\text{raw} = 0$ (while $S^\text{raw}$ is learned) (3) ‘segmentation prior’, $S^\text{raw} = 0$ (while $A^\text{raw}$ is learned); (4) our full model where both are learned, i.e. they constitute an output of a trained VAE encoder. All those models use Free Bits ($\lambda = 10$), while for ‘no free bits’ $\lambda = 0$. The results are summarized in Table 4. As expected, the full model performs the best, demonstrating that it is important to learn alignments and segmentation. Interesting, both ‘segmentation prior’ and ‘alignment prior’ obtain reasonable performance, but the ‘full prior’ model fails badly. One possible explanation is that given one of them being learned, the variations remaining in the other can be controlled, so the overall sampling variance will be small. The Free Bits trick appears crucial as it prevents the (partial) posterior collapse in our model. We inspected the logits after training, and observed that, without free-bits, the learned $W$ are very small, in the $[-0.01, +0.01]$ range.

### Table 4: Scores with different latent segmentation on the AMR 2.0 test set, averaged over 2 runs

| Metric          | Concept | SRL | Smatch |
|-----------------|---------|-----|--------|
| no free bits    | 83.5    | 66.3| 66.1   |
| full prior      | 81.7    | 62.6| 61.9   |
| alignment prior | 86.0    | 69.1| 70.5   |
| segmentation prior | 87.6    | 71.1| 74.4   |
| full            | **88.3**| **73.0**| **76.1** |

Table 4, there is a substantial gap between using structured ST and the two other versions. This illustrates the need for exposing the parsing model to discrete structures in training.

### 6.4 Analysis

We compare different segmentations and show their ‘segmentation density’. We define segmentation density as $\sum_{i,j \leq m} S_{ij}^m$. This is a number between 0 to 1, and is 0 if all subgraphs have only 1 node. To compare different segmentations, for a given model, for every pair of nodes we see if the pair belongs to the same subgraph. We then compute the F1 score for this ‘same-subgraph’ relation.

Results are shown in Table 5. We can see that the induced segmentation is in a slightly better agreement with the rule-based one than the greedy heuristic. It also, in some sense, in the ‘middle’, as it has better agreement with the other two. Overall, the hand-crafted rule-based segmentation is very different from the other two segmentations.

‘Greedy’, ‘rule’, and our learned segmentations have segmentation density of 29.0%, 17.7% and 30.0%, respectively. This again confirms that the rule-based segmentation has very different properties.

Figures 5-7 present an example from the development set. The VAE model segments the graph in the most intuitive way, while the greedy segmentation can be arbitrary, and the rule-based segmentation failed to handle all nodes.\(^{11}\)

### 7 Related Work

As AMR does not specify a procedure for parsing, a wide range of approaches for AMR parsing have been explored, including graph-based models (Flanigan et al., 2014; Werling et al., 2015; Lyu and Titov, 2018; Zhang et al., 2019a), transition-based models (Damonte et al., 2017; Ballesteros and Al-Onaizan, 2017), grammar-based models (Peng et al., 2015; Artzi et al., 2015; Groschwitz et al., 2018; Lindemann et al., 2020) and neural autoregressive models (Konstas et al., 2017; van Noord and Bos, 2017; Zhang et al., 2019b; Cai and Lam, 2020).

In terms of AMR subgraph segmentation, most strong parsers use subgraph segmentation. They

\(^{11}\)Non-aligned nodes are aligned to other tokens not shown in the figure.
typically rely on hand-crafted rules, with rule templates developed by studying training set statistics and ensuring the necessary level of coverage. Alternatively, Artzi et al. (2015) optimized the parser parameters and the subgraph segmentation with CCG (Steedman, 1997, 2004) lexicon induction. Groschwitz et al. (2017, 2018); Lindemann et al. (2020) use a decomposition automaton (Koller and Kuhlmann, 2011) to segment AMR subgraphs. Peng et al. (2015) apply y Markov Chain Monte Carlo to sample synchronous hyperedge replacement grammar rules for derivations of AMR graph. All those approaches rely on existing grammar formalisms that limit the space of possible segmentations. Importantly, the state-of-the-art neural autoregressive model still benefit from an explicit subgraph segmentation (Graph Recategorization) (Cai and Lam, 2020).

More generally, outside of AMR parsing, differentiable relaxations of latent structure representations have received attention in NLP (Kim et al., 2017; Liu and Lapata, 2018), including previous applications of the perturb-and-MAP framework (Corro and Titov, 2019). From the general goal perspective – inducing a segmentation of a linguistic structure – our work is related to tree-substitution grammar induction (Sima’an et al., 1995; Cohn et al., 2010), the DOP paradigm (Bod et al., 2003) and ‘unsupervised semantic parsing’ (Poon and Domingos, 2009; Titov and Klementiev, 2011), though the methods used in that previous work are very different from ours.

8 Conclusions

To get rid of hand-crafted segmentation systems, used in previous AMR parsers, we cast the alignment and segmentation as generation-order induction. We propose to treat this generation order as a latent variable in a variational auto-encoder framework. With the stochastic softmax, the model is end-to-end differentiable, and our method outperforms a technique relying on a simple segmentation heuristic and approaches the performance of a method using rules designed to handle specific AMR constructions.

Importantly, while the latent variable modeling machinery is used in training, the parser is very simple at test time. It tags the input words with AMR concept nodes with autoregressive models, and then predicts relations between the nodes with a feedforward network. One way to further improve the graph-based AMR parser is to replace the fully factorized relation identification component. For example, we could adopt the iterative inference strategy developed by Cai and Lam (2020) while keeping our segmentation induction component intact.

Acknowledgments

The project was supported by the European Research Council (ERC StG BroadSem 678254), the Dutch National Science Foundation (NWO VIDI 639.022.518) and Bloomberg L.P.

References

Akshay Agrawal, Brandon Amos, S. Barratt, Stephen P. Boyd, Steven Diamond, and J. Z. Kolter. 2019. Differentiable convex optimization layers. ArXiv, abs/1910.12430.

Rafael Anchiêta and Thiago Pardo. 2018. Towards AMR-BR: A SemBank for Brazilian Portuguese language. In Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018), Miyazaki, Japan. European Language Resources Association (ELRA).

Yoav Artzi, Kenton Lee, and Luke Zettlemoyer. 2015. Broad-coverage ccg semantic parsing with amr. In EMNLP.

Miguel Ballesteros and Yaser Al-Onaizan. 2017. Amr parsing using stack-lstms. ArXiv, abs/1707.07755.

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.

Yoshua Bengio, N. Léonard, and Aaron C. Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. ArXiv, abs/1308.3432.

Rens Bod, Remko Scha, Khalil Sima’an, et al. 2003. Data-oriented parsing. University of Chicago Press.

Claire Bonial, Bianca Badarau, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Tim O’Gorman, Martha Palmer, and Nathan Schneider. 2018. Abstract Meaning Representation of constructions: The more we include, the better the representation. In Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018), Miyazaki, Japan. European Language Resources Association (ELRA).

L. M. Bregman. 1967. The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. Ussr Computational Mathematics and Mathematical Physics, 7:200–217.
Deng Cai and Wai Lam. 2020. Amr parsing via graph-sequence iterative inference. In ACL.

Shu Cai and K. Knight. 2013. Smatch: an evaluation metric for semantic feature structures. In ACL.

Trevor Cohn, Phil Blunsom, and Sharon Goldwater. 2010. Inducing tree-substitution grammars. The Journal of Machine Learning Research, 11:3053–3096.

Michele Conforti, Gerard Cornuejols, and Giacomo Zambelli. 2014. Integer Programming. Springer Publishing Company, Incorporated.

Caio Corro and Ivan Titov. 2019. Differentiable perturb-and-parse: Semi-supervised parsing with a structured variational autoencoder. ArXiv, abs/1807.09875.

Marco Damonte, Shay B. Cohen, and Giorgio Satta. 2017. An incremental parser for abstract meaning representation. In EACL.

Shibhansh Dohare and Harish Karnick. 2017. Text Summarization using Abstract Meaning Representation. arXiv preprint arXiv:1706.01678.

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

Jonas Groschwitz, Meaghan Fowlie, Mark Johnson, and Alexander Koller. 2017. A constrained graph algebra for semantic parsing with AMRs. In IWCS 2017 - 12th International Conference on Computational Semantics - Long papers.

Jonas Groschwitz, Matthias Lindemann, Meaghan Fowlie, Mark Johnson, and Alexander Koller. 2018. Amr dependency parsing with a typed semantic algebra. In ACL.

Hardy and Andreas Vlachos. 2018. Guided neural language generation for abstractive summarization using abstract meaning representation. In EMNLP.

Tamir Hazan and Tommi Jaakkola. 2012. On the partition function and random maximum a-posteriori perturbations. arXiv preprint arXiv:1206.6410.

Fuad Issa, Marco Damonte, Shay B. Cohen, Xiaohui Yan, and Yi Chang. 2018. Abstract meaning representation for paraphrase detection. In NAACL-HLT.

Eric Jang, Shixiang Gu, and Ben Poole. 2016. Categorical reparameterization with gumbel-softmax. ArXiv, abs/1611.01144.

Bevan K. Jones, Jacob Andreas, Daniel Bauer, Karl Moritz Hermann, and Kevin Knight. 2012. Semantics-based machine translation with hyperedge replacement grammars. In COLING.

Yoon Kim, Carl Denton, Luong Hoang, and Alexander M Rush. 2017. Structured attention networks. arXiv preprint arXiv:1702.00887.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. CoRR, abs/1412.6980.

Diederik P. Kingma, Tim Salimans, and M. Welling. 2017. Improved variational inference with inverse autoregressive flow. ArXiv, abs/1606.04934.

Diederik P Kingma and Max Welling. 2014. Auto-encoding variational bayes. International Conference on Learning Representations.

Alexander Koller and Marco Kuhlmann. 2011. A generalized view on parsing and translation. In Proceedings of the 12th International Conference on Parsing Technologies, pages 2–13, Dublin, Ireland. Association for Computational Linguistics.

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

Percy Liang. 2016. Learning executable semantic parsers for natural language understanding. Communications of the ACM, 59(9):68–76.

Matthias Lindemann, Jonas Groshwitz, and Alexander Koller. 2020. Fast semantic parsing with well-typedness guarantees. ArXiv, abs/2009.07365.

Fei Liu, Jeffrey Flanigan, Sam Thomson, Norman M. Sadeh, and Noah A. Smith. 2015. Toward Abstractive Summarization Using Semantic Representations. In HLT-NAACL.

Yang Liu and Mirella Lapata. 2018. Learning structured text representations. Transactions of the Association for Computational Linguistics, 6:63–75.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. ArXiv, abs/1907.11692.

Chunchuan Lyu and Ivan Titov. 2018. Amr parsing as graph prediction with latent alignment. ArXiv, abs/1805.05286.

Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. 2017. The concrete distribution: A continuous relaxation of discrete random variables. ArXiv, abs/1611.00712.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McCallum. 2014. The Stanford CoreNLP natural language processing toolkit. In Association for Computational Linguistics (ACL) System Demonstrations, pages 55–60.
André FT Martins, Noah A Smith, and Eric Xing. 2009. Concise integer linear programming formulations for dependency parsing. In Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP, pages 342–350.

Gonzalo E. Mena, David Belanger, Scott W. Linderman, and Jasper Snoek. 2018. Learning latent permutations with gumbel-sinkhorn networks. ArXiv, abs/1802.08665.

Noelia Migueles-Abraira, Rodrigo Agerri, and Arantza Díaz de Ilarraza. 2018. Annotating Abstract Meaning Representations for Spanish. In Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018), Miyazaki, Japan. European Language Resources Association (ELRA).

Arindam Mitra and Chitta Baral. 2016. Addressing a question answering challenge by combining statistical methods with inductive rule learning and reasoning. In AAAI.

Tahira Nasir, Abhishek Shah, Hui Wan, Radu Florian, Salim Roukos, and Miguel Ballesteros. 2019. Rewarding smatch: Transition-based amr parsing with reinforcement learning. ArXiv, abs/1905.13370.

Rik van Noord and Johan Bos. 2017. Neural semantic parsing by character-based translation: Experiments with abstract meaning representations. ArXiv, abs/1705.09980.

George Papandreou and Alan L. Yuille. 2011. Perturb-and-map random fields: Using discrete optimization to learn and sample from energy models. 2011 International Conference on Computer Vision, pages 193–200.

Max B. Paulus, Dami Choi, Daniel Tarlow, Andreas Krause, and Chris J. Maddison. 2020. Gradient estimation with stochastic softmax tricks. ArXiv, abs/2006.08063.

Xiaochang Peng, Linfeng Song, and Daniel Gildea. 2015. A synchronous hyperedge replacement grammar based approach for AMR parsing. In Proceedings of the Nineteenth Conference on Computational Natural Language Learning, pages 32–41, Beijing, China. Association for Computational Linguistics.

Xiaochang Peng, Chuan Wang, Daniel Gildea, and Ni-anwen Xue. 2017. Addressing the data sparsity issue in neural AMR parsing. In Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers, pages 366–375, Valencia, Spain. Association for Computational Linguistics.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In Empirical Methods in Natural Language Processing (EMNLP), pages 1532–1543.

Hoifung Poon and Pedro Domingos. 2009. Unsupervised semantic parsing. In Proceedings of the 2009 conference on empirical methods in natural language processing, pages 1–10.

Nima Pourdamghani, Yang Gao, Ulf Hermjakob, and Kevin Knight. 2014. Aligning English strings with Abstract Meaning Representation graphs. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 425–429, Doha, Qatar. Association for Computational Linguistics.

M. Schlichtkrull, Thomas Kipf, P. Bloem, R. V. Berg, Ivan Titov, and M. Welling. 2018. Modeling relational data with graph convolutional networks. In ESWC.

Khalil Sima’an, Rens Bod, Steven Krauwer, and Remko Scha. 1995. Efficient disambiguation by means of stochastic tree substitution grammars. In Recent Advances in NLP, volume 136.

Li Song, Yuling Dai, Yihuan Liu, Bin Li, and Weiguang Qu. 2020. Construct a sense-frame aligned predicate lexicon for Chinese AMR corpus. In Proceedings of The 12th Language Resources and Evaluation Conference, pages 2962–2969, Marseille, France. European Language Resources Association.

Linfeng Song, Daniel Gildea, Yue Zhang, Zhiguo Wang, and Jinsong Su. 2019. Semantic neural machine translation using amr. Transactions of the Association for Computational Linguistics, 7:19–31.

Mark Steedman. 1997. Surface structure and interpretation. In Linguistic inquiry.

Mark Steedman. 2004. The syntactic process. In Language, speech, and communication.

Ida Szubert, Marco Damonte, Shay B. Cohen, and Mark Steedman. 2020. The role of reentrancies in abstract meaning representation parsing. In Findings of EMNLP.

Ivan Titov and Alexandre Klementiev. 2011. A Bayesian model for unsupervised semantic parsing. In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, pages 1445–1455.

Keenon Werling, Gabor Angeli, and Christopher D. Manning. 2015. Robust subgraph generation improves abstract meaning representation parsing. In ACL.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2019. Huggingface’s transformers: State-of-the-art natural language processing. ArXiv, abs/1910.03771.
We specify the root identification as:

\[ h^{\text{root}} \]

with copy mechanism. Formally, we have a small AMR graph is a rooted directed acyclic graph, as yet we are learning a latent alignment. We consider another root identifier that chooses the root, and a decoding algorithm to obtain a connected graph. Therefore, we need to specify the root identification as:

\[
P_\theta(i|\mathbf{x}, \mathbf{O}, \mathbf{v}) = \frac{\exp\left(\langle h^{\text{root}}, h^c_i \rangle \right)}{\sum_{j=0}^{m-1} \exp\left(\langle h^{\text{root}}, h^c_j \rangle \right)}
\]

where \( h^{\text{root}} \) is a trainable vector. Inspired by Zhang et al. (2019a), who utilizes the fact that AMR graph is very closely related to dependency tree, we first decode the AMR graph as a maximum spanning tree with log probability of most likely arc-label as edge weights. The reentrancy edges are added afterwards, if their probability is larger than 0.5. We add at most 5 reentrancy edges, based on the empirical founding of Szubert et al. (2020).

A Decoding

AMR graph is a rooted directed acyclic graph, as traversal on a connected graph from the root gives the directed acyclic graph. Therefore, we specify embedding of node \( i \) \( v_i \) tokens \( V \), which con- share \( S \) assigns a score to candidate nodes \( \mathbf{v} \), and \( \mathbf{h}^c \) takes the submatrix of \( \mathbf{S} \), excluding the diagonal entries, as we do not expect a subgraph containing more than 4 nodes. We observe \( \mathbf{A}^\infty \) should obey the following self-consistency equation:

\[
\mathbf{A}^\infty = \mathbf{A}^\infty \mathbf{S}_{:,m} + \mathbf{A}
\]

C Computing \( \mathbf{B} \) and \( \mathbf{A}^\infty \)

We obtain \( \mathbf{B} \) by having:

\[
\mathbf{B} = \mathbf{A}[\mathbf{S}_{:,m} + \operatorname{Diag}((\mathbf{S}_{:,m})^T)\mathbf{T}];
\]

where \( \mathbf{S}_{:,m} \) takes the submatrix of \( \mathbf{S} \), excluding the last column, and \( \operatorname{Diag}(\mathbf{S}_{:,m}) \) is the diagonal matrix whose diagonal entries are the last column of \( \mathbf{S} \). Intuitively, \( [\mathbf{S}_{:,m} + \operatorname{Diag}((\mathbf{S}_{:,m})^T)\mathbf{T}] \) can be thought as a Markov transition matrix that pass down the alignment along the generation order, and keep the alignment mass if the node will generate \( \emptyset \). We truncate the transition at \( T = 4 \), as we do not expect a subgraph containing more than 4 nodes.

This means, node \( i \) is generated from token \( k \) iff node \( i \) is generated from token \( k \) and node \( i \) generates node \( j \) or node \( j \) is directly generated from token \( k \). This \( \mathbf{A}^\infty \) can be computed by initializing \( \mathbf{A}^\infty = \mathbf{A} \), and repeating equation 27 as assignment for \( T = 4 \) times. Intuitively, the \( \mathbf{A}^\infty \) alignment is passed down the segmentation arrows, while keep getting alignment mass from the first node alignment. As a result, all nodes get assigned alignment. Alternatively, the algorithmic assignment defines a truncated power series expansion of self-consistency equation solution \( \mathbf{A}^\infty = [I - \mathbf{S}_{:,m}]^{-1} \mathbf{A} \).
D Hyper-Parameters

We use RoBERTa-large (Liu et al., 2019) from Wolf et al. (2019) as contextualised embedding before LSTMs. BiLSTM for concept identification has 1 layer and BiLSTM for relation identification has 2 layers. Both have hidden size 1024. Their averaged representation is used for alignment. Rel-GCN used 128 hidden units and 2 layers. Relation identification used 128 hidden units. The LSTM for the locally auto-regressive model is one layer with 1024 hidden units. Adam (Kingma and Ba, 2015) is used with learning rate $3e - 4$ and beta=0.9, 0.99. Early stopping is used with maximum 60 epochs of training. Dropout is set at 0.33.

E Pre-and-Post processing

We follow Lyu and Titov (2018) for pre-and-post processing. We use CoreNLP (Manning et al., 2014) for tokenization and lemmatization. The copy-able dictionary is built with the rules based on string matching between lemmas and concept node string as in Lyu and Titov (2018).

For post-processing, wiki tags are added after named entity being produced in graph via a look-up table built from the training set or provided by CoreNLP. We also collapse nodes that represent the same pronouns, as a heuristics for co-reference resolution.

F Proof of Proposition 1

Bregman’s method solves convex optimization with linear equality, the setting is as follows:

$$\min_{x \in \Omega} F(x) \quad \text{s.t.} \quad Ax = b, \quad \text{(28)}$$

where $F$ is strongly convex and continuously differentiable. Two important ingredients are Bregman’s divergence $D_F(x, y) = F(x) - F(y) - \langle \nabla F(y), x - y \rangle$, and Bregman’s projection: $P_{\omega,F}(y) = \min_{x \in \Omega} D_F(x, y)$, where $\omega$ represents constraint. Now, the Bregman’s method works as:

Theorem 1 ((Bregman, 1967)). $\lim_{t \to \infty} y^t$ solves the optimization problem 28.

Proof of Proposition 1. We show Proposition 1 by showing Algorithm defined by equations 14, 15, 16 and 17 implements Bregman’s method. Then, Proposition 1 follows from Theorem 1.

Now, we build Bregman’s method for our optimization problem 12. For simplicity, we focus on the linear algebraic structure, but do not strictly follow the standard matrix notation. Clearly, we have $O$ as variable, and $F(O) = -\langle W, O \rangle + \tau \log O \log O - 1$.

Algorithm 2: Bregman’s method for solving convex optimization over linear constraints

1. For initialization, we have $O(0) = \exp(\frac{W}{\tau})$. This corresponds to initialization step as in our equation 14. Then, we iterate through constraints to perform Bregman’s projection. First, the column normalization constraints $\forall j < m, \sum_{i=0}^{n+m-1} O_{ij} = 1$. Take a $j < m$, we need to compute $P_{ij} (O(j), 1, F(O(t)))$. A very important property is that our $F(O) = \sum_{ij} f_{ij} (O_{ij})$, where $f_{ij} (O_{ij}) = -\tilde{W}_{ij} O_{ij} + \gamma O_{ij} \log O_{ij} - 1$. Moreover, $D_F(x, y) = 0 \iff x = y$. It’s not hard to see that for variables that is not involved in the constraints, they are kept the same. Now, let’s focus on column $j$, we have:

$$\arg \min_{x_{\sum_i x_i = 1}} F(x) - F(O_{i,j}) - \langle \nabla F(O_{i,j}), x - O_{i,j} \rangle$$

$$= \arg \min_{x_{\sum_i x_i = 1}} -\langle W_{i,j}, x \rangle + \tau \langle x, \log x - 1 \rangle$$

$$= \arg \min_{x_{\sum_i x_i = 1}} \langle x_{\sum_i x_i = 1}, \nabla F(O_{i,j}) \rangle$$

$$= \arg \min_{x_{\sum_i x_i = 1}} \langle x_{\sum_i x_i = 1}, \nabla F(O_{i,j}) \rangle$$

$$= \arg \min_{x_{\sum_i x_i = 1}} x_{\sum_i x_i = 1} \langle x_{\sum_i x_i = 1}, \nabla F(O_{i,j}) \rangle$$

$$= \arg \min_{x_{\sum_i x_i = 1}} x_{\sum_i x_i = 1} \langle x_{\sum_i x_i = 1}, \nabla F(O_{i,j}) \rangle$$

$$= \text{Softmax}(\log O_{i,j})$$

As iterating over those mutually non-overlapping
The matrix $A$ is totally unimodular if every square submatrix has determinant 0, ±1.

We combined a few theorems and definitions.

14 A is totally unimodular if every square submatrix has determinant 0, ±1.

Proof. We show $A$ admits row-bicoloring. We color the first $m$ rows red, and remaining $n + m$ rows blue. The sum of red rows is: $R_{ij} = \sum_{k=0}^{m-1} A_{k,ij} = \sum_{k=0}^{m-1} \delta_{j,k} = [[j < m]]$ and the sum of blues is $B_{ij} = \sum_{k=m}^{2m+n-1} A_{k,ij} = \sum_{k=m}^{2m+n-1} \delta_{i,k-m} = 1$. Therefore, $R_{ij} - B_{ij} = [[j = m]] \in \{0, \pm1\}$. $A$ admits a row-bicoloring. Since $A$ has only 0, ±1 value, and one variable in $O$ at most participates in two constrains (in-coming and out-going), by theorem 3, $A$ is totally unimodular.

Now, we prove proposition 2.

Proof. We have $A$ being totally unimodular. We have $c = \tilde{W}$, $l = 0$, $u = 1$, by theorem 2, the LP solution contains integer vector. Without being overly formal, $\tilde{W}$ yields a unique solution with probability 1. Hence, the $O(W, 0)$ yields integer solution with probability 1.