Structural Realization with GGNNs

Jinman Zhao Gerald Penn Huan Ling
Department of Computer Science
University of Toronto
{jzhao,gpenn,linghuan}@cs.toronto.edu

Abstract

In this paper, we define an abstract task called structural realization that generates words given a prefix of words and a partial representation of a parse tree. We also present a method for solving instances of this task using a Gated Graph Neural Network (GGNN). We evaluate it with standard accuracy measures, as well as with respect to perplexity, in which its comparison to previous work on language modelling serves to quantify the information added to a lexical selection task by the presence of syntactic knowledge. That the addition of parse-tree-internal nodes to this neural model should improve the model, with respect both to accuracy and to more conventional measures such as perplexity, may seem unsurprising, but previous attempts have not met with nearly as much success. We have also learned that transverse links through the parse tree compromise the model's accuracy at generating adjectival and nominal parts of speech.

1 Introduction

We conjecture that this may be an opportune time to reassess the extent to which syntax is capable of contributing to a word prediction task. *Structured realization* is a generalization of language modelling in which we receive $n - j$ words as input, together with a syntactic structure that has a yield of n word positions and spans the input, plus an "overhang" of j unrealized word positions. Our task is to fill in the most likely missing j words. Language modelling generally possesses only the trivial annotation that consists of the words themselves and has historically assumed that $j = 1$, constituting an n-gram. Notable exceptions date back to the work of Chelba (2000) on *structured language modelling*, in which the syntactic annotation is partial, in that there is no overhang ($j = 0$), but structurally non-trivial, although often sparing relative to corpora that parsers are trained upon.[1] The most thorough exploration of this direction is probably that of Köhn and Baumann (2016), who equip a variety of language models with a pretrained dependency parser, which they use to predict the part of speech (POS) of the next word and some overarching syntactic structure, and then predict the next word from its POS plus an n-gram word history. They report a roughly 6% perplexity reduction across the different models.

In the specific case where a complete, spanning, syntactic representation is provided, but the model is evaluated solely from a zero-prefix initialization (i.e., $n = j$), this generalization can be viewed as a simple purely syntactic surface-realization problem, as one would find in a generation task.

With no fanfare whatsoever in CL circles, the machine learning community proposed an evaluation task seven years ago called "MadLibs" Kiros et al. (2014). In our terminology, the syntactic annotation provided is merely $n - j$ words followed by a string of j POS tags. While it may be difficult to imagine that someone would be in possession of this POS information without also knowing how the POS tags connected together, the authors were interested in testing a new multiplicative neural language model, in which attributes (such as POS tags) can be attached to input words.

In a neural setting, parse trees can be encoded with a generalization of recurrent neural networks (RNNs) called Graph Neural Networks (GNNs). GNNs have been used as encoders to deal with a variety of different NLP problems (see related work section later). Gated GNNs (GGNNs) are an improvement over GNNs that is analogous to that of GRUs over RNNs. They train faster, and they address problems with vanishing gradients.

[1] Chelba (2000) proposes that, in order to iteratively predict one word at a time, a structured language model should predict syntactic structure over every word that it has predicted, but in his evaluation, it is very clear that he is more concerned with the first stage of word prediction.

We shall compare two modes of our model here using GGNN-encoded parse trees: one with parse trees from OntoNotes 5.0 (Hovy et al., 2006; Pradhan et al., 2013; Weischedel et al., 2013), and one with vestigial transitions between pre-terminal categories in sequence, which resembles the syntactic annotation selected by Kiros et al. (2014), although here the word prefix is also POS-annotated. We also test the combination of the two: a syntactic tree augmented by a linear pipeline of transitions between pre-terminals. We compute sentence-level accuracy by measuring how many words in the generated strings legitimately belong to their assigned POS categories, and compute word-level accuracy scores in three ways: accuracy at choosing a word of the appropriate part of speech (this time with the prefix of words corrected to what the corpus says, as necessary), rank of the corpus sentences by data likelihood, and word-guessing accuracy, relative to what appears in the corpus.

2 Method

In this paper, we exploit a Gated Graph Neural Network (GGNN) (Li et al., 2016) as a parse tree encoder. GNN encoders have been shown to be efficient for neural machine translation (Beck et al., 2018; Bastings et al., 2017) whereas in our case, we focus on structured realization. GGNNs define a propagation model that extends RNNs to arbitrary graphs and learn propagation rules between nodes. We aim to encode syntactic trees by propagating category labels throughout the tree's structure.

2.1 Gated Graph Neural Networks

For completeness, we briefly summarize the GGNN model (Li et al., 2016). A GGNN uses a directed graph $\{V, E\}$ where V and E are the sets of nodes and edges. We represent the initial state of a node v as s_v and the hidden state of node v at propagation time step t as h_v^t. The adjacency matrix $A \in \mathbb{R}^{|V| \times N|V|}$ determines how the nodes in the graph propagate information to each other, where N represents the number of different edge types. Figure 1 is the visual representation of a GGNN; it starts with $h_v^0 = s_v$, then follows a propagation model which unrolls T steps and generates h_v^T at the end. Each unroll step follows the same rule to compute h_v^t from $h_v^{(t-1)}$ and A:

$$
\begin{aligned}
a_v^t &= A_v^\top \, [h_1^{t-1^\top}, ..., h_{|V|}^{t-1^\top}]^\top + b \\
r_v^t &= \sigma(W^r a_v^t + U^r h_v^{(t-1)}) \\
z_v^t &= \sigma(W^z a_v^t + U^z h_v^{(t-1)}) \\
\widetilde{h_v^t} &= tanh(W a_v^t + U(r_v^t \odot h_v^{(t-1)})) \\
h_v^t &= (1 - z_v^t) \odot h_v^{(t-1)} + z_v^t \odot \widetilde{h_v^t}.
\end{aligned}
\tag{1}
$$

$b, W, W^r, W^z, U, U^r, U^z$ above are trainable parameters.

After information is propagated for T time steps, each node's hidden state collectively represents a message about itself and its neighbourhood, which is then passed to its neighbours. Finally there is the output model. For example, Acuna et al. (2018) implemented their output model by:

$$
\begin{aligned}
h_v &= tanh(FC_1(h_v^T)) \\
\text{out}_v &= FC_2(h_v)
\end{aligned}
\tag{2}
$$

where FC_1 and FC_2 are two fully connected layers.

2.2 Gated Graph Neural Network Models

In this part, we will describe how we use GGNNs and parse trees to build our three experimental models. Figure 2 depicts example trees for these models.

2.2.1 Input Tree Construction

Since we are using GGNNs, we first need to construct the graph by giving the parse tree. We build three different models:

Model 1: For a given parse tree, let N be the number of nodes in the parse tree. Then the adjacency matrix of the tree, denoted as **A**, is an N $\times$ 2N matrix, concatenating two N $\times$ N matrices. **A**[:N,:] is the forward adjacency matrix of the tree and **A**[N:,:] is the backward adjacency matrix.

Model 2: The input does not consider interior parse tree nodes, but instead works more like a conventional language model. For each parse tree, and given a sequence of words $(w_1, w_2, ..., w_{n-1})$, we retain all and only the pre-terminal parse tree nodes, and then attempt to predict the next word w_n. This is the model of Kiros et al. (2014). Note that, while it is essentially a language model, the nodes of this Model are a subset of the nodes of Model 1, although the edges are completely different, encoding only transitions between the pre-terminals

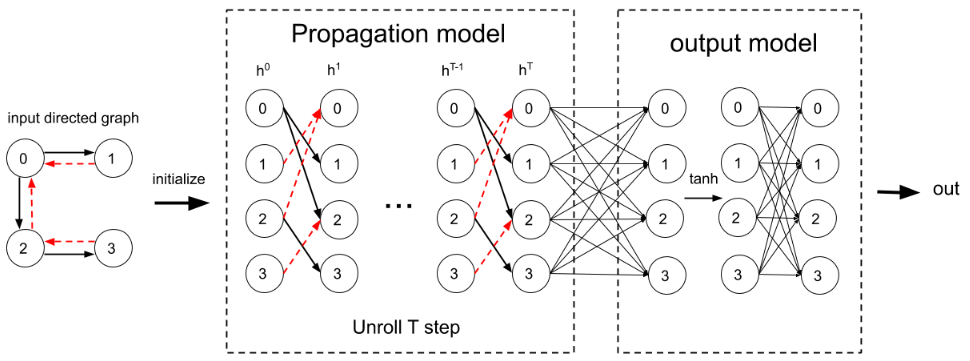


Figure 1: An example GGNN. The GGNN generates output from a directed graph. It consists of a propagation model and an output model. During the propagation step, there are two different edge types in this graph. Black arrows are the OUT edges while red arrows are the IN edges.

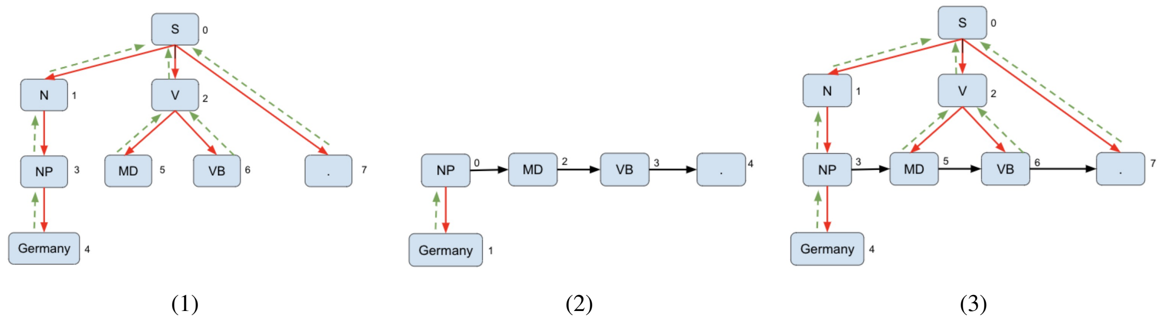


Figure 2: Example of an input parse tree with a given input word prefix (w_1 = *Germany*) and a completion consisting of POS/pre-terminal categories. The number near each node represents its index within the adjacency matrix of the tree (Figure 3). Red arrows are forward edges, green dashed arrows are backward edges and black arrows represent a transverse edge between pre-terminals. The partial trees each contain 1 terminal("Germany"), 4 pre-terminals ("NP","MD","VB",".") and, in Models (1) and (3), 3 other interior categories ("S","N","V"). We want to predict the word after *Germany*, which will be the child of pre-terminal "MD". The input of Model 1 considers tree nodes and forward/backward edges, but not transverse pre-terminal edges. The input of Model 2 does not include other parts of the tree except pre-terminals and the given terminals, yet it contains all three kinds of edges. The input of Model 3 which contains all tree nodes and all edges.

in sequence. This time, the adjacency matrix A is N × 3N which is a concatenation of three N × N matrices: $A^{forward}$, $A^{backward}$ and $A^{pre-terminal}$.

Model 3: This one is the combination of the above two. The number of nodes is the same as for Model 1. The adjacency matrix is the addition of each respective pair of $A^{forward}$, $A^{backward}$ and $A^{pre-terminal}$, concatenated together.

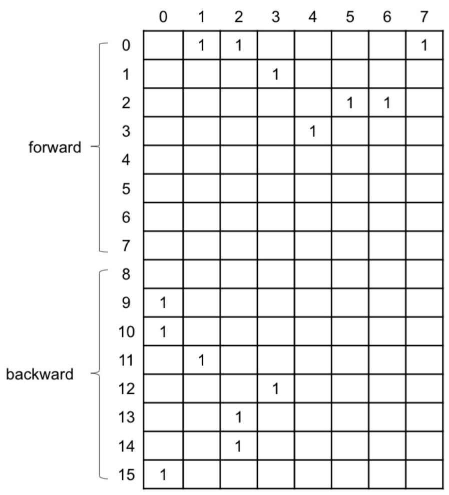
Figure 2 depicts an example for each model. By comparing the results for different models later, we will understand how essential inner nodes and edges between pre-terminals are for word prediction. Also note that Model 1 and Model 3 have the same number of nodes, but the number of nodes in Model 2 is smaller. Nevertheless, in all three models, the input may contain a prefix of $n - j$ words. As mentioned above, when this prefix is zero-length, we have three classical surface realization models. But we can also view all three models as generalizations of language models, in which:

$$\begin{aligned} P(W) &= P(w_1 w_2 ... w_n) \\ &= \prod_{i=1}^{n} P(w_i | tree_{i-1}) \end{aligned} \tag{3}$$

and $tree_{i-1}$ is the parse tree with the $1^{th}, 2^{th} ... (i-1)^{th}$ word tokens in place.

2.2.2 Terminal, Pre-terminal and Interior Tags

Once we have constructed the graph, we need to construct input for the model. Let $D = 100$ be the dimension of a set of word-embedding vectors over a fixed lexicon. The input to each model is an N × D matrix. All three types of nodes need



		0	1	2	3	4	5	6	7
forward	0		1	1					1
	1				1				
	2						1	1	
	3					1			
	4								
	5								
	6								
	7								
backward	8								
	9	1							
	10	1							
	11		1						
	12				1				
	13			1					
	14			1					
	15	1							

Figure 3: The adjacency matrix of the above input example (1) in Figure 2. Blank slots represent 0, meaning no edge between two nodes. A = $[A^{forward}, A^{backward}]$.

to be represented in same-dimensional vectors: 1) terminals, i.e. words, 2) pre-terminals (nodes that only appear as a parent of a leaf), and 3) interior node tags (nodes that are neither leaves nor pre-terminals). We then need to normalize all vectors.

We associate each terminal (word) with its GloVe (Pennington et al., 2014) pre-trained word vector (trained from Wikipedia 2014 + Gigaword 5, containing a 400K-word vocabulary).

For pre-terminals, we gather sequences (e.g., "NP MD VB ." in Figure 1) for each input sentence, prepare a corpus consisting only of these tags, and train embedding vectors directly on the POS tags by using the GloVe algorithms (Pennington et al., 2014). We then associate each pre-terminal with its corresponding vector.

The number of interior tags is larger than D, however, so one-hot is not appropriate in this case. For each interior node, we randomly generate a D-dimensional vector, sampling each entry of the vector from a standard Gaussian distribution.

2.2.3 Predict Words

After the input presentation, the propagation step and a fully connected layer, the model will generate an N $\times$ D output matrix. In other words, all N nodes in the parse tree will have D-dimensional output. In language modelling mode, we would not care about any output except the one generated by the pre-terminal dominating the position of w_{n-j+1}. Let $\hat{v}$ denote this normalized D-dimensional output. The probability of w_{n-j+1} given the tree, $P(w_{n-j+1} = i | tree_{n-j}) =:$

$$\frac{exp(c^2 \times (\hat{v}^T \cdot v_i))}{\sum_{j=0}^{V} exp(c^2 \times (\hat{v}^T \cdot v_j)))} \quad (4)$$

where V is the size of the pre-trained lexicon, v_i and v_j are vector representations for the ith and jth word types. We choose i with maximum conditional probability. This is equivalent to choosing the i for which v_i is the closest word vector $\hat{v}$.

When $c = 1$ and $tree_{n-j}$ consists only of the sequence of input words $(w_1, w_2, ..., w_{n-j})$, Eq 4 would correspond to a standard language model. The interval $[e^{-1}, e^1]$ is too small as the range of the numerator to distinguish between good predictions and bad predictions. So instead of only normalizing them, we also multiply by a constant c. Thus the range of the numerator becomes $[e^{-c^2}, e^{c^2}]$. We tuned c manually from 1 to 15 based on model 1. Figure 4 shows that $c = 6$ is the best, as it has the lowest cross entropy compared with other values. We will assume $c = 6$ in Section 3.

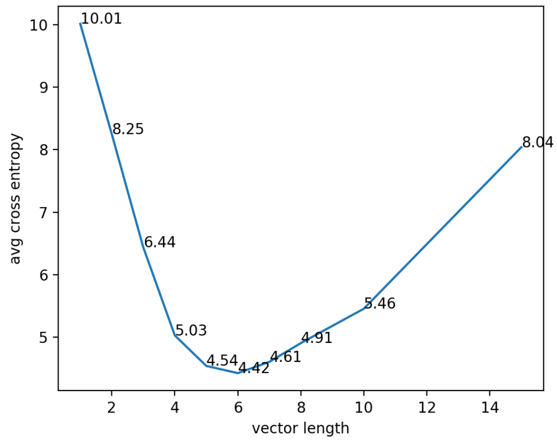


Figure 4: Average cross entropy loss of validation set using Model 1 with different magnitude of vectors.

3 Results

3.1 Datasets

We train and test all models on OntoNotes 5.0, which contains 110,000+ English sentences from print publications. We also train and evaluate the perplexity of all models on the Penn Treebank (Marcus et al., 1993), as this has become a standard among syntax-driven language models. PTB \$2-21 are used as training data, \$24 is for validation, and \$23 is used for testing.

We excluded those trees/sentences with words that are not in GloVe's (Pennington et al., 2014) pre-trained vocabulary from both the training and validation data. The test set and validation set were excluded from our development cycle. Dataset statistics are provided in Table 1.

3.2 Training Details

We used 100-dimensional pre-trained GloVe vectors to represent different kinds of leaves in the tree. As the loss function, we use cross entropy loss which is calculated based on Equation 4. For each complete parse tree in the training set, let n be the number of leaves/terminals in this tree. So this tree has n different possible prefixes of known words($w_1...w_i$, $0 \leq i \leq n-1$), each with a parse tree as training input. In addition, since the number of nodes in different graphs is distinct, we use stochastic gradient descent with a learning rate of 0.01 to train the model (i.e. batch size = 1).

Perplexity relates to cross-entropy loss:

$$PPL = e^{-\frac{1}{N}\sum_i p(x)\log y(x)} \tag{5}$$

This is corpus-level perplexity, where x is an arbitrary word and $p(x)$ is the function we discussed in Eq 4 and N is the number of predictions. The magnitude $c = 6$ also has lowest perplexity.

3.3 Realization Accuracy

A simple way to evaluate the accuracy of the models as implementations of the structured realization task is to consider their sentence output in terms of POS accuracy. If we simply remove the yields of corpus trees and attempt to regenerate them from the trees, the resulting strings will often differ from the original yields, but they may still be grammatical in the sense of the first j tokens having the appropriate POS tag sequence. Table 2 shows the sentence-level word and POS accuracies on the OntoNotes test set. Both OntoNotes and PTB provide gold-standard (human labeled) syntactic constituency parse trees. We trained our model on these trees. These trees are expensive, however, so we also evaluated on trees obtained from the Berkeley neural parser (Kitaev and Klein, 2018) a state-of-the-art constituency parser with an $F_1 = 95$ score on the PTB.

3.4 Continuity of Latent Spaces

Some trees have the same unlabelled tree structure, although they may have different nodes. We can randomly pick two such isomorphic constituency trees T_1 and T_2, delete their leaves then linearly interpolate between their corresponding nodes and generate. For an arbitrary node of the i^{th} intermediate tree, the vector representation would be:

$$\begin{aligned} node = (1-\lambda) \times T_1(node) \\ + \lambda \times T_2(node), \end{aligned} \tag{6}$$

for some value of $\lambda \in [0, 1]$. Table 3 demonstrates sentences generated from trees for various values of λ. This kind of "semantic continuity" has been demonstrated before on vector encodings, but, to our knowledge, not on structured spaces such as parallel trees of vectors.

3.5 Perplexity

Perplexity is perhaps the most common evaluation measure in the language modelling literature. The formula of perplexity was shown in Eq 5.

We trained and evaluated our Models on the different datasets listed in Table 1. The perplexities of the test data sets are listed in Table 4. RNNG (Dyer et al., 2016) is a state-of-the-art syntax-aware model. LSTM-256 LM is our self implemented language model using 2-layer LSTM cell with sequence length 20 and hidden state size 256. Our three models have lower perplexities across the board compared with RNNG on both OntoNotes and PTB. Model 3 on gold parse trees has the lowest perplexity overall, although it is important to remember that our models benefit from distributions from Wikipedia that are implicitly encoded in the GloVe vectors. LSTMs that use GloVe perform worse than the LSTMs with trainable word embeddings shown here.[2]

In addition, for comparion, we trained our models on PTB $2–21 excluding those trees that contain words that are not in GloVe, but tested on the entire PTB $23 with gold syntactic constituency parsing trees. For those words not in GloVe, we followed the method in RNNG (Dyer et al., 2016). First, we replace them by <UNK-XXX>(e.g. <UNK-DASH>,<UNK-NUM>). Then, for each UNK to-

[2]Kruskal-Wallis and post-hoc Mann-Whitney tests with Bonferroni correction reveal that M1–3 with benepar trees are statistically significantly different ($p < 10^{-10}$) from RNNG at the sentence level (H=56.84 PTB; 65.54 OntoNotes), and from LSTM at the word level (H=1485.94 PTB; 2561.44 OntoNotes), on both corpora, except that there was no significant difference found between M1 and RNNG with OntoNotes. All effect sizes were small (df=3, V=0.05). With OntoNotes, no significance was found between M1 and M2; with PTB, none was found between M2 and M3.

Dataset	Train	Test	Valid	Vocabulary	Max_s	Ave_s	Max_t	Ave_t
OntoNotes	102254	5430	5625	45408	210	19	570	54
PTB	31680	1945	1114	30924	116	23	308	68

Table 1: Statistics of the datasets used in this project. Max_s/Ave_s are the maximum/average lengths of sentences. Max_t/Ave_t are the maximum/average numbers of nodes in trees.

Accuracy	M1	M2	M3
word gold	32.34	32.09	**34.64**
word benepar	31.47	31.93	**33.96**
POS gold	**94.39**	89.47	92.3
POS benepar	**93.6**	89.19	91.9

Table 2: Sentence-level accuracies of the models on the OntoNotes test set. "Benepar" is the Berkeley neural parser (Kitaev and Klein, 2018).

T_1	god was very good to me .
	jesus was very happy for him .
	god said accusatory ones while it .
	i made my people talking alone .
	he had their people talking again .
T_2	he told their people coming again .

Table 3: Sentences generated between two random trees that have the same unlabelled tree structure. T_1 = "(S (NP (NNP)) (VP (VBD) (ADJP (ADJP (RB) (JJ)) (PP (IN) (NP (PRP))))) (. .))", T_2 = "(S (NP (PRP)) (VP (VBD) (S (NP (PRP$) (NNS)) (VP (VBG) (ADVP (RB))))) (. .))."

Model			Test ppl
KN-5-gram (Kneser and Ney, 1995)			169.3
LSTM-128 (Zaremba et al., 2014)			113.4
GRU-256			112.3
RNNG (Dyer et al., 2016)			102.4
SO-RNNG (Kuncoro et al., 2017)			101.2
GA-RNNG (Kuncoro et al., 2017)			100.9
NVLM (Zhang and Song, 2019)			91.6
Model 1 (gold)	81.05		
Model 2 (gold)	72.09		
Model 3 (gold)	71.07	(benepar)	84.44
Transformer-XL			54.52

Table 5: Perplexities of the PTB test set (entire $23). RNNG, SO-RNNG, GA-RNNG and NVLM use the same method to preprocess data, keeping only vocabulary that appear more than once in the training set. For hapaxes in the training set and words in the validation/test sets that occur once in the training set, they replace them with <UNK-POS> tokens. Their models only contain 24 000 word types, whereas ours contain 31 000. In some other language modelling settings, the vocabulary size can be as small as 10 000.

ken, we use the average of the vector representations of words labelled as XXX in the training set to obtain the vector representation of this token. The perplexity of the entire PTB $23 is listed in Table 5. RNNG, SO-RNNG, GA-RNNG and NVLM are all syntax-aware models. Our Models achieve very good perplexity. Note that while Transformer-XL does perform better, it uses roughly 2.4×10^7 parameters whereas ours uses 9×10^5. We have a larger vocabulary size because we retain words that appear in GloVe regardless of frequency. Larger vocabulary sizes generally increase perplexity.

Model	OntoNotes	PTB	Tree type
LSTM-256	125.8	126.43	
RNNG	116.7	119.66	
Model 1	92.86	75.14	gold
Model 2	90.10	66.78	gold
Model 3	**73.65**	**65.75**	gold
Model 1	101.4	75.56	benepar
Model 2	95.13	69.02	benepar
Model 3	**80.18**	**68.06**	benepar

Table 4: Perplexities of the OntoNotes/PTB test trees in which all words have GloVe vectors.

3.6 Word-prediction Accuracy and Rank

Given a parse along with the prefix $w_1, ...w_{n-j}$, we can remove the leaves $(w_{n-j+1}, w_{n+1}, ..., w_n)$ from the parse tree, and predict w_{n-j+1}, where $1 \leq j \leq n$. Thus, for a tree with n word positions, we can perform word prediction up to n times. Unlike the structured realization accuracies above, conventional practice in language modelling evaluation is to restore the integrity of w_{n-j} according to the corpus before predicting w_{n-j+1} when the previous prediction step was unsuccessful. Word accuracies according to this regimen are given in Table 8, along with accuracy at predicting any word with the required part of speech.

To better evaluate the results, we also compute the rank for each predicted word. Let $\mathbf{v\prime}$ be the vector representation of the true w_{n-j+1} and $\hat{\mathbf{v}}$ denote the output vector as discussed earlier. For each vector representation of a word in the pre-trained GloVe vocabulary set, compute the Euclidean distance between it and $\hat{\mathbf{v}}$. Rank r means $||\mathbf{v\prime} - \hat{\mathbf{v}}||$ is the r^{th} smallest distance in comparison to the other words in the vocabulary set. If the rank is small, then the model is capable of finding a close prediction. Small rank also means the model is

model	≤ 10	≤ 100	≤ 1000	≤ 10000	>10000	Med	Mean
Model 1	**55.5%**	11.9%	16.7%	13.1%	2.8%	5	1057
Model 2	**55.2%**	12.8%	17.1%	12.3%	2.6%	5	965
Model 3	**57.8%**	12.2%	15.8%	11.9%	2.4%	**4**	907
LSTM-256 LM	**50.7%**	23.7%	16.1%	8.2%	1.4%	10	**564**

Table 6: Rank distributions for models on the OntoNotes Test Set.

model	≤ 10	≤ 100	≤ 1000	≤ 10000	>10000	Med	Mean
Model 1	**55.7%**	13.8%	17.6%	11.3%	1.4%	4	678
Model 2	**57.0%**	14.5%	16.5%	10.7%	1.3%	4	614
Model 3	**57.2%**	13.9%	16.7%	10.8%	1.3%	4	624
LSTM-256 LM	**50.9%**	22.3%	17.1%	8.8%	0.9%	10	**470**

Table 7: Rank distributions for models on the PTB Test Set.

able to learn the relation between the next word and the given partial parse tree.

Table 6 and Table 7 show the overall, median, and mean rank distributions of the different models, compared to LSTM-256 within the ranges 10^{ϕ} to $10^{\phi+1}$, $0 \leq \phi \leq 4$. Most of the ranks are ≤ 10 and the median ranks for all models are less than 5. Our GGNN based models have more predictions that rank less than or equal to 10 compared with LSTM-256. Model 1 and Model 2 have similar ranks; Model 3's are slightly better. Model 3 has the lowest median rank. Although LSTM-256 has the lowest mean rank, LSTM-256's vocabulary size is much smaller than our GGNN based models.'

3.7 Generating words of a specific POS

Sometimes a model has an output vector located very far from the vector representation of the true word (i.e. its rank is very large), but the predicted word can at least be assigned the correct pre-terminal POS. This means the prediction is in some sense correct, because it is more likely to be grammatically and semantically acceptable. For example, given a sequence "within three days she had," and a gold-standard next word of "worked," with parent "VBN," "turned" could be a good prediction even though it is far from "worked", because "turned" also belongs to "VBN."

Since we train terminals and pre-terminals separately, there is no prior connection defined between them. For example, given a tag "NN," we do not know which words belong to "NN" when training the vectors for the words, or when choosing the vector for "NN." So this is a learned ability. Let us denote the true i^{th} word as t and the predicted i^{th} word as p. To evaluate this capability, every time the model predicts a word p, we count it as a correct prediction if: (1) p occurs somewhere in the training data, dominated by a category c, and (2) c also dominates this occurrence of t.

In Table 8, we present this accuracy rate in the second column for each of the different models. On the OntoNotes test data, Models 1 and 3 have higher rates than Model 2, while Model 2 has the highest POS accuracy on the PTB test data. Alongside this, we also compute the overall accuracy of selecting the correct word (i.e., when the true word has rank 1), as well as the macro-averaged and macro-median accuracy of selecting the correct word, broken down by the pre-terminal dominating the position to be predicted.

All three models have high POS accuracies in general (medians: 99.90, 99.93 and 99.7, respectively), but Models 2 and 3 have very bad accuracies for some POSs such as 'NN' (60.68–67.45), 'NNS' (32.32–68.31) and 'VBN' (38.5–49.1).

4 Related Work

Graph Neural Networks as Graph Encoders GNNs were first proposed by Scarselli et al. (2009). Li et al. (2016) added gating mechanisms for recurrent networks on graphs. In parallel, (Bruna et al., 2013) proposed Graph Convolutional Networks. GCNs differ from GGNNs in their graph propagation model. GGNNs exploit recurrent neural networks to learn propagation weights through time steps. Each step shares the same set of parameters. On the other hand, GCNs train unshared CNN layers through time steps. In this paper, we employed GGNNs as a design choice. Similar to our model architecture, Bastings et al. (2017); Beck et al. (2018) used graphs to incorporate syntax into neural machine translation and Marcheggiani and Titov (2017) used ERS graph convolutional networks as dependency tree encoders for semantic role labelling. Even before graph neural networks

Model	OntoNotes				PTB			
		Word acc				Word acc		
	POS acc	**μavg**	**macavg**	**med**	**POS acc**	**μavg**	**macavg**	**med**
Model 1	**94.35**	32.4	26.6	38.6	94.28	35.47	41.74	47.46
Model 2	89.4	32.0	37.39	43.52	**97.66**	37.12	**43.74**	**49.22**
Model 3	92.33	**34.7**	**37.9**	**44.3**	95.73	**37.27**	41.69	44.37
LSTM-256 LM		22.4				23.57		

Table 8: Percentage POS prediction accuracies and word prediction accuracies, for each model.

become popular, there were attempts akin to graph encoders. Dyer et al. (2015); Socher et al. (2013); Tai et al. (2015); Zhu et al. (2015); Le and Zuidema (2014) encoded tree structure with recursive neural networks or Tree-LSTMs.

Surface Realization Song et al. (2018) introduced a graph-to-sequence LSTM for AMR-to-text generation that can encode AMR structures directly. The model takes multiple recurrent transition steps in order to propagate information beyond local neighbourhoods. But this method must maintain the entire graph state at each time step. Our models also simultaneously update every node in the tree at every time step. The encoder of Trisedya et al. (2018) takes input RDF triples rendered as a graph and builds a dynamic recurrent structure that traverses the adjacency matrix of the graph one node at a time. Marcheggiani and Perez-Beltrachini (2018), again using a GCN, take only the nodes of the RDF graph as input, using the edges directly as a weight matrix. They, too, must update the entire graph at every time step.

Language Modelling The task of language modelling has a long and distinguished history. Although the term itself was not coined until Jelinek et al. (1975), the earliest work of Shannon (1948) on entropy presents what are effectively character-level language models as a motivating example. In both cases, given a prefix of characters/words or classes (Brown et al., 1992), the aim of the task is to predict the next such event. n-gram language models factor any dependency of the next event on the prefix through its dependency on the final $n-1$ events in the prefix. This long remained the dominant type of language model, but the advent of neural language models (Bengio et al., 2003), and particularly vector-space embeddings of certain lexical-semantic relations, has drastically changed that landscape. See, e.g., models using recurrent networks (Mikolov et al., 2010), year (Mikolov et al., 2011), LSTMs (Sundermeyer et al., 2012), sequence-to-sequence LSTMs models (Sutskever et al., 2014), and convolutional networks (Gehring et al., 2017) and transformers (Devlin et al., 2019).

An earlier, but ultimately unsuccessful attempt at dislodging n-gram language models was that of Chelba (2000), who augmented this prefix with syntactic information. Chelba (2000) did not use conventional parse trees from any of the then-common parse-annotated corpora, nor from linguistic theory, because these degraded rather than enhanced language modelling performance. Instead, he had to remain very sparing in order to realize an empirical improvement. The present model not only shares information at the dimensional level, but projects syntactic structure over the words to be predicted. While this makes structured realization a very different task from structured language modelling, this not only appears to improve perplexity, but does so without having to change the conventional representation of trees found in syntactic corpora. The present model could therefore be used to evaluate competing syntactic representations in a controlled way that quantifies their ability to assist with word prediction, as we have here.

5 Conclusion

GGNNs have proved to be effective as encoders of constituent parse trees from a variety of perspectives, including realization accuracy, perplexity, word-level prediction accuracy, categorical cohesion of predictions, and novel lexical selection. A limitation of this study is the comparatively modest size of its corpora, which is due to the requirement for properly curated parse-annotated data. Finding ways to scale up to larger training and test sets without the bias introduced by automated parsers remains an important issue to investigate.

References

David Acuna, Huan Ling, Amlan Kar, and Sanja Fidler. 2018. Efficient interactive annotation of segmentation datasets with polygon-rnn++. In *CVPR*.

Joost Bastings, Ivan Titov, Wilker Aziz, Diego

Marcheggiani, and Khalil Sima'an. 2017. Graph convolutional encoders for syntax-aware neural machine translation. In *EMNLP*.

Daniel Edward Robert Beck, Gholamreza Haffari, and Trevor Cohn. 2018. Graph-to-sequence learning using gated graph neural networks. In *ACL*.

Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. 2003. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155.

Peter F Brown, Peter V Desouza, Robert L Mercer, Vincent J Della Pietra, and Jenifer C Lai. 1992. Class-based n-gram models of natural language. *Computational linguistics*, 18(4):467–479.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2013. Spectral networks and locally connected networks on graphs. *CoRR*, abs/1312.6203.

C. Chelba. 2000. *Exploiting Syntactic Structure for Natural Language Modeling*. Ph.D. thesis, Johns Hopkins University.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-based dependency parsing with stack long short-term memory. In *ACL*.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. 2016. Recurrent neural network grammars. In *Proc. of NAACL*.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. 2017. Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1243–1252. JMLR. org.

Eduard Hovy, Mitchell Marcus, Martha Palmer, Lance Ramshaw, and Ralph Weischedel. 2006. Ontonotes: The 90% solution. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*, NAACL-Short '06, pages 57–60, Stroudsburg, PA, USA. Association for Computational Linguistics.

F. Jelinek, L.R. Bahl, and R.L. Mercer. 1975. Design of a linguistic statistical decoder for the recognition of continuous speech. *IEEE Transactions on Information Theory*, IT-21(3).

Ryan Kiros, Richard Zemel, and Ruslan R Salakhutdinov. 2014. A multiplicative model for learning distributed text-based attribute representations. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2348–2356. Curran Associates, Inc.

Nikita Kitaev and Dan Klein. 2018. Constituency parsing with a self-attentive encoder. *arXiv preprint arXiv:1805.01052*.

R. Kneser and H. Ney. 1995. Improved backing-off for m-gram language modeling. In *1995 International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pages 181–184 vol.1.

A. Köhn and T. Baumann. 2016. Predictive incremental parsing helps language modeling. In *Proc. 26th COLING*, pages 268–277.

Adhiguna Kuncoro, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, Graham Neubig, and Noah A. Smith. 2017. What do recurrent neural network grammars learn about syntax? In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 1249–1258, Valencia, Spain. Association for Computational Linguistics.

Phong Le and Willem Zuidema. 2014. The inside-outside recursive neural network model for dependency parsing. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 729–739. Association for Computational Linguistics.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard S. Zemel. 2016. Gated graph sequence neural networks. *CoRR*, abs/1511.05493.

Diego Marcheggiani and Laura Perez-Beltrachini. 2018. Deep graph convolutional encoders for structured data to text generation. In *Proceedings of the 11th International Conference on Natural Language Generation*, pages 1–9, Tilburg University, The Netherlands. Association for Computational Linguistics.

Diego Marcheggiani and Ivan Titov. 2017. Encoding sentences with graph convolutional networks for semantic role labeling. In *EMNLP*.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*.

Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. 2011. Extensions of recurrent neural network language model. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5528–5531. IEEE.

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.

Sameer Pradhan, Alessandro Moschitti, Nianwen Xue, Hwee Tou Ng, Anders Björkelund, Olga Uryupina, Yuchen Zhang, and Zhi Zhong. 2013. Towards robust linguistic analysis using OntoNotes. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 143–152, Sofia, Bulgaria. Association for Computational Linguistics.

F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. 2009. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80.

C.E. Shannon. 1948. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423 (July), 623–656 (October).

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*.

Linfeng Song, Yue Zhang, Zhiguo Wang, and Daniel Gildea. 2018. A graph-to-sequence model for AMR-to-text generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1616–1626, Melbourne, Australia. Association for Computational Linguistics.

Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. 2012. Lstm neural networks for language modeling. In *Thirteenth annual conference of the international speech communication association*.

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

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. In *ACL*.

Bayu Distiawan Trisedya, Jianzhong Qi, Rui Zhang, and Wei Wang. 2018. GTR-LSTM: A triple encoder for sentence generation from RDF data. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1627–1637, Melbourne, Australia. Association for Computational Linguistics.

Ralph Weischedel, Martha Palmer, Mitchell Marcus, Eduard Hovy, Sameer Pradhan, Lance Ramshaw, Nianwen Xue, Ann Taylor, Jeff Kaufman, Michelle Franchini, et al. 2013. Ontonotes release 5.0 ldc2013t19. *Linguistic Data Consortium, Philadelphia, PA*, 23.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2014. Recurrent neural network regularization.

Yuyu Zhang and Le Song. 2019. Language modeling with shared grammar. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4442–4453, Florence, Italy. Association for Computational Linguistics.

Xiao-Dan Zhu, Parinaz Sobhani, and Hongyu Guo. 2015. Long short-term memory over tree structures. *CoRR*, abs/1503.04881.