

Deconstructing Supertagging into Multi-task Sequence Prediction

Zhenqi Zhu

School of Computing Science
Simon Fraser University
Burnaby, BC , Canada
zhenqiz@sfu.ca

Anoop Sarkar

School of Computing Science
Simon Fraser University
Burnaby, BC , Canada
anoop@sfu.ca

Abstract

Supertagging is a sequence prediction task where each word is assigned a piece of complex syntactic structure called a supertag. We provide a novel approach to multi-task learning for Tree Adjoining Grammar (TAG) supertagging by deconstructing these complex supertags in order to define a set of related but auxiliary sequence prediction tasks. Our multi-task prediction framework is trained over the exactly same training data used to train the original supertagger where each auxiliary task provides an alternative view on the original prediction task. Our experimental results show that our multi-task approach significantly improves TAG supertagging with a new state-of-the-art accuracy score of 91.39% on the Penn treebank supertagging dataset.

1 Introduction

A treebank for lexicalized tree-adjoining grammar (TAG) (Joshi and Schabes, 1997) consists of annotated sentences where each word is provided a complex tree structure called a supertag and the overall parse of the sentence combines these supertags into a parse tree. Supertagging is a task that learns a sequence prediction task from this annotated data and is able to then assign the most likely sequence of supertags to an input sequence of words (Bangalore and Joshi, 1999). Once the right supertag is assigned then parsing is a much easier task and may not even be needed for many applications where information about syntax is needed but a full parse is unnecessary.

Supertagging has been shown to be useful for both Tree Adjoining Grammar (TAG) (Bangalore and Joshi, 1999) and combinatory categorical grammar (CCG) (Hockenmaier and Steedman, 2007) parsing. In this paper we aim to improve the state-of-the-art for the task of learning a TAG supertagger from an annotated treebank (Kasai

et al., 2018). We observe that supertag prediction does not take full advantage of the complex structural information contained within each supertag. Neural models have been used to learn embeddings over these supertags and thereby share weights among similar supertags. Friedman et al. (2017) provide tree-structured neural models over supertags which can learn interesting relationships between supertags but the approach does not lead to higher supertagging accuracy.

Our main contribution is to provide several novel ways to deconstruct supertags to create multiple alternative auxiliary tasks, which we then combine using a multi-task prediction framework and we show that this can lead to a significant improvement in supertagging accuracy.

Multi-task learning (MTL) (Caruana, 1997) learns multiple heterogeneous tasks in parallel with a shared representation so that what is learned for one task can be shared for another task. In most cases the improvement is due to weight sharing between different tasks (Collobert and Weston, 2008; Luong et al., 2015). While some combinations may not provide any benefit in MTL (Bingel and Sogaard, 2017) and the improvements might be simply due to training on more data. However, MTL can be effective even when using large pre-trained models (Liu et al., 2019).

Unlike most other work in multi-task learning with neural models we do not use different annotated datasets for each task. Similar to the approach to combining different representations for phrase structure parsing in (Vilares et al., 2019) we also construct multiple tasks from exactly the same training data set. Our approach is also distinct in that we take advantage of the structure of the supertags by deconstructing the tree structure implicit in each supertag.

Our experimental results show that our novel multi-task learning framework leads to a new

state-of-the-art accuracy score of 91.39% for TAG supertagging on the Penn Treebank dataset (Marcus et al., 1993; Chen et al., 2006) which is a significant improvement over the previous multi-task result for supertagging that combines supertagging with graph-based parsing (Kasai et al., 2018).

2 The Supertagging Task

Supertagging assigns complex structural descriptions to each word in the sentence. The complex structural descriptions come from grammar formalisms that are more expressive than context-free grammars for phrase structure trees or dependency trees. In Tree Adjoining Grammar (TAG), the supertags are tree fragments that can express various syntactic facts such as transitive verb, wh-extraction, relative clauses, appositive clauses, light verbs, prepositional phrase attachment and many other syntactic phenomena. In combinatory categorial grammar (CCG) the supertags are types and their type-raised variants which also capture similar syntactic phenomena as in TAG supertags. Supertagging can be viewed as “almost parsing” (Bangalore and Joshi, 1999) and can provide the benefits of syntactic parsing without a full parser.

In this paper we focus on the TAG supertagging task, however, our proposed methods can likely be used to improve CCG supertagging as well. Supertagging is a relatively simple linear time sequence prediction task similar to part of speech tagging. Supertagging can be useful in many applications such as machine translation, grammatical error detection, disfluency prediction, and many others while being a much simpler task than full parsing.

In addition, for both TAG and CCG, supertagging is an essential first step to parsing so any improvements in supertag prediction will benefit parsing as well. For all these reasons, in this paper we focus on the supertagging task. TAG and CCG can be parsed using graph-parsing methods in $O(n^3)$ but the complexity of unrestricted parsing for both formalisms is $O(n^6)$ which is prohibitive on real-world data. Neural linear-time transition based parsers are still not accurate enough to compete with the state-of-the-art supertagging models or parsers that use supertagging as the initial step (Chung et al., 2016; Kasai et al., 2018).

An example of the supertagging task for Tree Adjoining Grammars (TAGs) is shown in Fig. 1. The \downarrow symbol on a leaf node represents a substitu-

tion node which can be expanded by a tree rooted in the same label, e.g. t3 rooted in NP substitutes into the NP \downarrow node in t46. The * symbol on the leaf node of a tree t represents an adjunction node (also called a footnode) and signifies that t can be inserted into an internal node of another tree with the same label, e.g. t103 adjoins into the AP node in t46. The \diamond node is called the *head* and represents the node where the word token is inserted into the tree. The table on the right shows how many different supertags are possible for each word in the sentence.

Three factors make supertagging a challenging task for sequence prediction: much more severe token level ambiguity when compared to other like part-of-speech tagging, a large number of distinct supertag types (4727 distinct supertags in our dataset, including an unknown supertag) and a complex internal structure for each supertag.

3 Baseline Supertagging Model

For our baseline supertagging model we use the state-of-the-art model that currently has the highest accuracy on the Penn treebank dataset (Kasai et al., 2018). For the supertagging model the main contribution of Kasai et al. (2018) was two-fold: the first was to add a character CNN for modeling word embeddings using subword features, and the second was to add highway connections to add more layers to a standard bidirectional LSTM. The output layer was a standard multi-layer perceptron that had a softmax output over the set of supertags. Another extension to the standard sequence prediction model in Kasai et al. (2018) was to combine supertagging with graph-based parsing.

In this paper, we focus on the supertagging model and compare only on supertagging accuracy. The neural model for supertagging that we use as a baseline uses graph-based parsing as an auxiliary task and has the current highest accuracy score on the Penn treebank (90.81%). The model has three main components: the input layer, the bidirectional LSTM component, and the output layer which computes a softmax over the set of supertags. The input to the model is a sequence of words and the output is a sequence of supertags, one per word, which makes it a standard tagging aka sequence prediction task.

3.1 Input Layer

Each word in the input sequence is converted into a word embedding in the input layer. Following

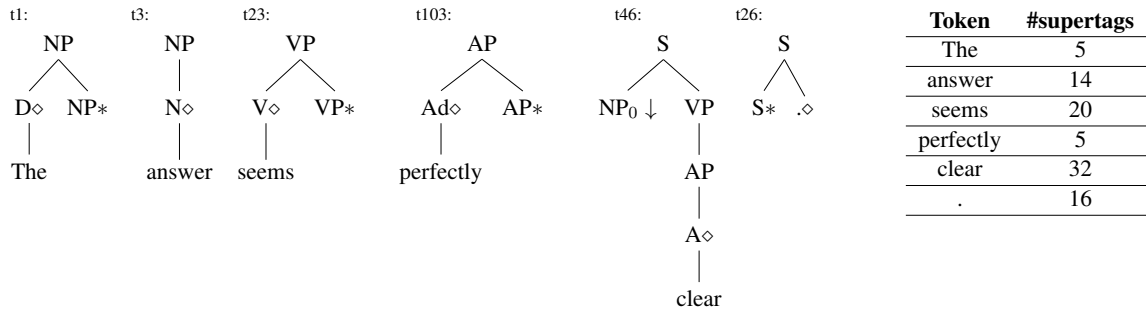


Figure 1: An example that explains the supertagging task for Tree Adjoining Grammars (TAGs). For the sentence “The answer seems perfectly clear.” the correct supertag for each word is shown above. The table on the right shows how many different supertags are possible for each word in the sentence. See Section 2 for more details on the notation used to define the supertags and how the supertags can be combined to form a parse tree.

(Kasai et al., 2018) we use two components in the word embedding:

- a 30-dimensional character level embedding vector computed using a char-CNN which captures the morphological information (Santos and Zadrozny, 2014; Chiu and Nichols, 2016; Ma and Hovy, 2016; Kasai et al., 2018). Each character is encoded as a 30-dimensional vector, and then we apply 30 convolutional filters with a window size of 5. This produces a 30-dimensional character embedding.
- a 100/200/300 size word embedding which is initialized using GloVe (Pennington et al., 2014). For words that do not appear in GloVe, we randomly initialized the word embedding.

A start of sentence token and an end of sentence token is added into the beginning and ending position of each sentence, but is not included in the computation of loss and accuracy.

Unlike (Kasai et al., 2018) we do not use predicted part of speech (POS) tags as part of the input sequence. In our experiments, the improvement was negligible and there was a significant overhead of having to do part of speech predictions at test time.

3.2 BiLSTM Layer

The core of this base model is a bidirectional recurrent neural network, in particular a Long Short-Term Memory neural network (Graves and Schmidhuber, 2005). For the hyperparameters, we use the settings in Kasai et al. (2018) in order to ensure a fair comparison.

Unlike (Kasai et al., 2018) we do not use highway connections in our model. We did experiment with the addition of highway connections but we found no improvement in accuracy over the baseline BiLSTM-only model with a significant increase in training time.

The bidirectional representation has 1024 units, a combination of the 512 forward and backward units each. Dropout layers (Gal and Ghahramani, 2016; Srivastava et al., 2014) are inserted between the input and BiLSTM layer, between BiLSTM layers, and between recurrent time steps. The dropout rate used was 0.5. We used 2-3 BiLSTM layers. Kasai et al. (2018) provide some reasons why > 3 layers do not provide any additional accuracy even with highway connections.

3.3 Output Layer

We concatenate hidden vectors from both directions of the last layer of BiLSTM and pass it into a multilayer perceptron (MLP). In practice a single layer perceptron performs just as well in this task. The number of input neurons of the single layer perceptron equals 1024 (2×512) and the output vector size equals the number of labels for each specific task: 4727 for the main supertagging task.

4 Deconstructing Supertags

The error analysis of our baseline BiLSTM model is shown in Fig. 1. We observed some consistent ways in which the baseline model confused the correct supertag with the incorrect one. We also observed that the baseline BiLSTM model can achieve over 97% 3-best accuracy on the supertagging task. This means it should be possible to boost the accuracy by rescoring the alternatives that already exist in the n -best output of the baseline supertagger. Rather than a re-ranking frame-




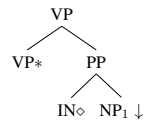
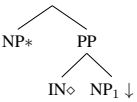
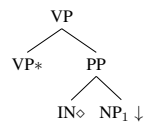
Prediction	Ground Truth	#times in dev
t2: 	t36: 	194
tCO: 	t13: 	156
t4: 	t13: 	144

Table 1: The top-3 errors made by the state-of-the-art Bi-LSTM supertagger. tCO stands for a co-head in the case where a supertag has multiple heads. One example is a sentence fragment like *pull it from the marketplace* which contains a multi-word predicate *pull ... from*; *pull* is the $V\blacklozenge$ head of tree t531 which has a IN^c node where tCO (headed by *from*) is inserted.

work we used a multi-task learning framework in order to boost the scores of correct supertags over the error-prone supertags. The auxiliary tasks we created based on our error analysis are as follows.

4.1 Auxiliary Tasks

4.1.1 HEAD

Consider the trees t2 and t36 in Table 1. t2 is headed by a noun head N and t36 is headed by an adjective A . The label of the head node is a useful auxiliary task for disambiguation. We define a function $HEAD(t)$ to get the head node (marked by a diamond) of supertag t . There are 29 distinct HEAD labels.

4.1.2 ROOT

Consider the trees t4 and t13 in Table 1. t4 modifies an NP node while t13 modifies a VP node. This is a case of preposition attachment ambiguity. The label of the root node is a useful auxiliary task for disambiguation. We define a function $ROOT(t)$ to get the root node of supertag t . There are 48 distinct ROOT labels.

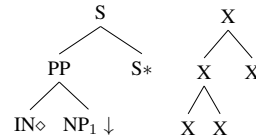
4.1.3 TYPE

Consider the trees tCO and t13 in Table 1. tCO is a supertag that does not use adjunction (this type of supertag is called an initial tree). In contrast, t13 modifies an internal VP node in another supertag (this type of supertag is called an auxiliary tree). In addition a left auxiliary tree modifies from the left

while a right auxiliary tree modifies from the right. To make this task more sensitive we also include the node label of the root (for initial trees) or footnote which is the node marked with $*$ (for left/right auxiliary trees). We define a function $TYPE(t)$ to obtain the type of each supertag. There are 67 distinct types.

4.1.4 SKETCH

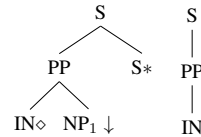
In many cases, the overall shape of the supertag is useful for disambiguation, ignoring the node labels. The following example keeps the tree structure of the supertag but removes the node labels:



Tree sketches help disambiguation (see t81 in Table 5). We define a function $SKETCH(t)$ that returns the sketch. There are 602 distinct supertag sketches.

4.1.5 SPINE

The spine of a supertag is the path from the root node to the head node (marked by \blacklozenge). The following example keeps only the path from root to head and produces a spine supertag:



Spine supertags are helpful for disambiguation as well (see t132 in Table 5). We use a function $SPINE(t)$ to return the spine of supertag t . There are 1372 distinct supertag spines.

4.2 Multi-task Framework

Unlike most other work in multi-task learning with neural models we do not use different datasets for each task. We use exactly the same training data set but we construct multiple tasks with alternate output labels by automatically deconstructing the supertags (the output labels in the original task). These alternate output labels are easier to predict than the full set of supertags, and these new output labels are related to the original supertag in a linguistically relevant way. As a result, we train on the same training set but with alternate output labels, each forming a different task. We then combine these multiple tasks in order to improve the performance in the original supertagging task.

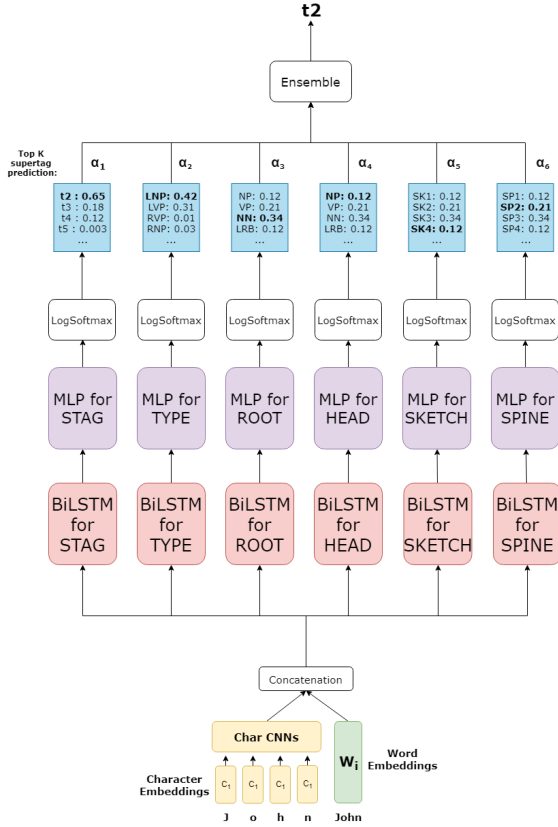


Figure 2: The prediction procedure of combining models trained on separate tasks

The usual criticism of a fair comparison between multi-task and single-task learning is that the multi-task setting simply uses more labeled data instances (typically with different data sources) and as a result a fair comparison between a multi-task and a single-task setting should involve large pre-trained models trained using a language modelling objective (such as ELMO (Peters et al., 2018) or BERT (Devlin et al., 2018)). In our case, because we re-use the same training set for multi-task learning, we have made sure our experimental settings exactly match the previous best state-of-the-art method for supertagging (Kasai et al., 2018) and we use the same pre-trained word embeddings to ensure a fair comparison.

We train six different neural sequence prediction models **independently** on the supertagging task, root node prediction (ROOT), head node prediction (HEAD), tree type prediction (TYPE), tree sketch prediction (SKETCH) and tree spine prediction (SPINE) tasks. For each task, we use the state-of-the-art baseline supertagging model as defined in Section 3. The only change is that the

output size for softmax is changed to reflect the number of output labels in each task. We obtain very high accuracies for each of the tasks. For example, on the dev set we obtain the following accuracies: ROOT = 97.04%, HEAD = 93.37%, TYPE = 93.14%, SKETCH = 93.74% and SPINE = 91.00%.

We train the model, including the word embedding (which is initialized using a pre-trained embedding) and character-level CNNs by optimizing the negative log-likelihood of the predicted sequences of output labels. The output labels for each task is different: supertag, root node, head node, tree type, sketch, spine. Training is done using minibatches. The main hyperparameters are as follows: we use the ADAM optimizer with a batch size of 100 and learning rate $\ell = 0.001$ (Kingma and Ba, 2015). After every training epoch, we evaluate the model on the dev set, if the accuracy on dev set has not been improved for five consecutive epochs, training stops. The maximum number of epochs is 70. After obtaining the best model trained with $\ell = 0.001$, we further fine-tune the best model using $\ell = 0.0001$ for at most 10 more epochs. By conducting this step, we have seen 0.1% to 0.2% accuracy improvement depending on the task.

After obtaining the best trained model on each of the multiple tasks we combine the multiple tasks together in order to create a decoder for the supertagging task.

We first run the baseline supertagger to obtain the distribution P_{STAG} and using this distribution we select the top-K output supertags for each word in each sentence in the dev or test data. We experiment with different values of K but we know that even $K=3$ gives 97% accuracy for the supertagging task. For each dev or test sentence we also compute the output softmax distributions for each task, $P_{\text{HEAD}}, P_{\text{ROOT}}, P_{\text{TYPE}}, P_{\text{SKETCH}}, P_{\text{SPINE}}$. Each of these probabilities are defined as a sequence prediction task over the auxiliary tasks using the functions defined in Section 4.1.

$$\begin{aligned}
 P_{\text{HEAD}}(t) &= P(\text{HEAD}(t)) \\
 P_{\text{ROOT}}(t) &= P(\text{ROOT}(t)) \\
 P_{\text{TYPE}}(t) &= P(\text{TYPE}(t)) \\
 P_{\text{SKETCH}}(t) &= P(\text{SKETCH}(t)) \\
 P_{\text{SPINE}}(t) &= P(\text{SPINE}(t))
 \end{aligned}$$

We compute the argmax sequence of supertags $t_1^*, t_2^*, \dots, t_T^*$ by scoring each supertag t_i^* individ-

ually from the top-K list by combining the probabilities from the different tasks as follows:

$$t_i^* = \arg \max_{t_i \in S} \quad (1)$$

$$\alpha_1 P_{\text{STAG}}(t_i) + \alpha_2 P_{\text{HEAD}}(t_i) \quad (2)$$

$$+ \alpha_3 P_{\text{ROOT}}(t_i) + \alpha_4 P_{\text{TYPE}}(t_i) \quad (3)$$

$$+ \alpha_5 P_{\text{SKETCH}}(t_i) + \alpha_6 P_{\text{SPINE}}(t_i) \quad (4)$$

S is the top-K set of supertags for each word in the input sequence. The hyperparameters α_i can be tuned. However we found in our experiments that the results were not very sensitive to the values, and the uniform distribution over all the tasks performed the best. The model and decoding step for our multi-task model is shown in Fig. 2. We also experiment with a commonly used multi-task model where some or all of the components are shared between the different (unlike our approach)..

5 Dataset

We use the dataset that has been widely used by previous work in supertagging and TAG parsing (Bangalore et al., 2009; Chung et al., 2016; Friedman et al., 2017; Kasai et al., 2017, 2018). We use the grammar and the TAG-annotated WSJ Penn Tree Bank extracted by Chen et al. (2006). As in previous work, we use Sections 01-22 as the training set, Section 00 as the dev set, and Section 23 as the test set. The training, dev, and test sets comprise 39832, 1921, and 2415 sentences; 950028, 46451, 56683 tokens, respectively.

The TAG-annotated version of Penn treebank (Chen and Shankar, 2001) includes 4727 distinct supertags (including an unknown supertag) and the grammar file of all supertags is downloaded from <http://mica.lif.univ-mrs.fr/>. There are 69 auxiliary tree TYPES, 40 distinct types of ROOT node and 30 different types of HEAD node, 602 tree SKETCHes and 1372 tree SPINES.

6 Results and Discussion

For our experiments, we implemented all of the models we discussed above in PyTorch (Paszke et al., 2017). We have various hyperparameters and Table 2 shows the results obtained from the different model configurations which were described in Section 3. The table also includes the results from the multi-task model and decoder described in Section 4. We experiment with pre-

trained GloVe word embeddings of three different sizes: 100, 200 and 300.

With our multi-task approach, all base models gain significant improvements compared to a single supertagging base model between 0.4% to 0.65%. We also varied the parameter K which picks the top-K supertags from the baseline model for use with the multi-task model. Table 3 that increasing K helps up to a point. After K=10 there is no further improvement.

We obtain a new state-of-the-art result of 91.39% which is significantly better than the 90.81% result which combines supertagging with the parsing task and so is using more labeled training information used by our supertagger models.

Table 4 shows the result of task ablation for each task. We can see that adding a new task always improves the results. The best result is obtained by using all five auxiliary tasks.

We computed a significance score on the accuracy of our best model BiLSTM3+CNN+GloVe200 with and without multi-task learning. On the dev set, using McNemar’s significance test we found that the multi-task model is significantly better than the baseline model with a p-value of 0.0062; on the test set, the p-value is 0.0064.

We evaluated our own implementation of the baseline BiLSTM-only model and even with highway connections we only obtained 89.25% on the dev set compared to the built-in BiLSTM implementation in Pytorch (without highway connections) which obtains 89.94%.

6.1 Task Contribution

Table 5 shows some examples about how each of auxiliary tasks can help in the correction of supertag prediction. Examples of each task are selected if a considerable number of predictions of each example are corrected after applying the multi-task model.

While the multi-task model can correct many wrong predictions made by the baseline model, the multi-task model may also override some correct predictions.

The first row is an example of the prediction of head node that helps differentiate two similar supertags, t2 and t36. In the dev set, there are 24 words of which ground truth supertags are t2, wrongly predicted as t36 by a single base model; 25 words of which ground truth supertags are t36, wrongly predicted as t2. All of those words are

Model	Multi-task	Dev	Test
BiLSTM3+HW+CNN+POS+GloVe100 (Kasai et al., 2018)	-	90.45	90.81
BiLSTM2+GloVe100	No	89.11	-
	Yes	89.67	-
BiLSTM2+CNN+GloVe100	No	89.45	-
	Yes	90.12	-
BiLSTM3+GloVe100	No	89.41	-
	Yes	90.02	-
BiLSTM3+CNN+GloVe100	No	89.83	-
	Yes	90.41	-
BiLSTM3+CNN+GloVe200	No	89.94	-
	Yes	90.55	91.37
BiLSTM3+CNN+GloVe300	No	89.91	-
	Yes	90.45	-
Shared BiLSTM layer (BiLSTM3+CNN+Glove200)	No	90.11	90.83
	Yes	90.11	90.83

Table 2: Supertagging task results. The number after BiLSTM represents the number of BiLSTM layers; CNN refers to the word embedding model using character-level CNN; the number immediately after GloVe represents the dimension of pre-trained GloVe word vectors. HW in Kasai et al. (2018) refers to highway connections, and POS refers to the use of predicted part-of-speech tags as inputs. We do not use HW or POS in our models as they do not provide any benefit.

Top-K	Dev	Test
Top-3	90.55	91.37
Top-5	90.58	91.38
Top-10	90.58	91.39
Top-20	90.58	91.39

Table 3: Change in accuracy as K is increased when choosing Top-K supertags for rescoring. The model used is BiLSTM+CNN+GloVe200.

correctly predicted by the multi-task model. The ROOT, TYPE, SKETCH and SPINE are all the same for t2 and t36, the only difference is the HEAD value, N for t2 and A for t36. The model for the HEAD task correctly predicts the head node of those words which is further improved using our multi-task approach.

The second row demonstrates how the tree sketch can help discriminate supertags. t81 and t27 have exactly the same ROOT, HEAD, SPINE (S-VP-V) and TYPE (Init), the only difference between these two supertags is the tree structure.

The third to fifth rows are examples of the effect of multiple auxiliary tasks in getting the prediction right. The third row is an example of the prediction of TYPE and SKETCH that can help differentiate supertags. The TYPE of t3 is **Init**, while t38 has TYPE **Left+NP**. They also have dif-

ferent tree sketches. There are 11 words of which supertags are wrongly predicted as t3 by a single supertagging model, but correctly predicted as t38 by the multi-task model; also, 3 words of which supertags are wrongly predicted as t38 by a single supertagging model, but correctly predicted as t3 by the multi-task model.

The forth row is an example of how the prediction of the ROOT can help differentiate supertags. The ROOT of t3 is **NP**, while t18 has ROOT **N** (N is also its head node). For the last row, t132 and t20 have the same root node(S), head node(Punct) and tree type (Right+S) but they are different in the tree spine (**S-Punct** for t20 and **S-PRN-Punct** for t132) and SKETCH. The joint effort of various models plays a significant role in getting the prediction right.

7 Related Work

Bangalore et al. (2009) and Chung et al. (2016) trained a feature based classification model for TAG supertags, that extract features using lexical, part-of-speech attributes from the left and right context in a 6-word window and the lexical, orthographic (e.g. capitalization, prefix, suffix, digit) and part-of-speech attributes of the word being supertagged. Neural network based supertagging models in TAG (Kasai et al., 2018) and CCG (Xu

Multi-task setting	Dev	Test
None	89.94	90.73
HEAD	90.00	90.79
ROOT	90.06	90.91
TYPE	90.15	91.07
SKETCH	90.25	90.99
SPINE	90.22	91.08
HEAD+ROOT	90.15	90.94
TYPE+HEAD+ROOT	90.27	91.10
TYPE+HEAD+ROOT+SKETCH	90.48	91.27
TYPE+HEAD+ROOT+SKETCH+SPINE	90.55	91.37

Table 4: Result of different multi-task combinations. The base model is BiLSTM+CNN+GloVe200.

Ground truth	Baseline	Multi-Task	Most Helpful Task
t2: 	t36: 	t2: 	HEAD
t81: 	t27: 	t81: 	SKETCH
t3: 	t38: 	t3: 	TYPE, SKETCH
t3: 	t18: 	t3: 	ROOT, SKETCH
t132: 	t20: 	t132: 	SPINE, SKETCH

Table 5: Some examples of how the deconstructing of base models correct the prediction made by the supertagging model.

et al., 2015; Lewis et al., 2016; Xu, 2016; Vaswani et al., 2016) have shown substantial improvement in performance, but the supertagging models are all quite similar as they all use a bi-directional RNN feeding into a prediction layer. Structural features of supertags are heavily used in pre-neural statistical parsing methods (Bangalore et al., 2009) and proved to be useful. The use of supertag structure was explored in (Friedman et al., 2017) where they adopt grammar features into a tree-structured

neural model over the supertags but this model was unable to beat the state-of-the-art. (Kasai et al., 2018) combines supertagging with parsing which does provide state-of-the-art accuracy but at the expense of computational complexity.

Kasai et al. (2017) extends the BiLSTM model with predicted part-of-speech tags and suffix embeddings as inputs, then Kasai et al. (2018) further extends the BiLSTM model with highway connection as well as character CNN as input, and jointly train the supertagging model with parsing model and this work had the state-of-the-art accuracy before our paper on the Penn treebank dataset. Friedman et al. (2017) investigated a recursive tree-based vector representation of TAG supertags, but while their model can learn useful facts about supertags, about how one can be related to another, there was no performance improvement as a result of their model on the supertagging task. Xu et al. (2015) uses RNN for the CCG supertagging task, Lewis et al. (2016) adopted the LSTM structure into this task, while Vaswani et al. (2016) also introduced another variation of Bi-LSTM into this task. Xu (2016) then proposed an attention-based Bi-LSTM supertagging model.

8 Conclusion

In this paper we have introduced a novel multi-task framework for the TAG supertagging task. The approach involved a novel multi-task learning framework which led to a new state-of-the-art accuracy score of 91.39% for TAG supertagging on the Penn treebank dataset.

Our multi-task prediction framework is trained over the exactly same training data used to train the original supertagger where each auxiliary task provides an alternative view on the original pre-

diction task.

In the future we would like to explore further tasks to integrate into our multi-task sequence prediction framework. We also believe that the idea of our multi-task framework can be applied into similar tasks such as CCG supertagging task of which the labels themselves contains the latent information. We would also like to investigate how to semi-automatically generate new tasks which can be of further help in the multi-task setting.

Acknowledgments

We would like to thank Logan Born and the anonymous reviewers for their helpful comments. The research was also partially supported by the Natural Sciences and Engineering Research Council of Canada grants NSERC RGPIN-2018-06437 and RGPAS-2018-522574 and a Department of National Defence (DND) and NSERC grant DGDND-2018-00025 to the second author.

References

- Srinivas Bangalore, Pierre Boullier, Alexis Nasr, Owen Rambow, and Benoît Sagot. 2009. Mica: A probabilistic dependency parser based on tree insertion grammars (application note). In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, pages 185–188.
- Srinivas Bangalore and Aravind K Joshi. 1999. Supertagging: An approach to almost parsing. *Computational linguistics*, 25(2):237–265.
- Joachim Bingel and Anders Søgaard. 2017. [Identifying beneficial task relations for multi-task learning in deep neural networks](#). In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 164–169, Valencia, Spain. Association for Computational Linguistics.
- Rich Caruana. 1997. Multitask learning. *Machine learning*, 28(1):41–75.
- John Chen, Srinivas Bangalore, and K Vijay-Shankar. 2006. Automated extraction of tree-adjoining grammars from treebanks. *Natural Language Engineering*, 12(3):251–299.
- John Chen and Vijay Shankar. 2001. *Towards efficient statistical parsing using lexicalized grammatical information*. Ph.D. thesis, Citeseer.
- Jason PC Chiu and Eric Nichols. 2016. Named entity recognition with bidirectional lstm-cnns. *Transactions of the Association for Computational Linguistics*, 4:357–370.
- Wonchang Chung, Siddhesh Suhas Mhatre, Alexis Nasr, Owen Rambow, and Srinivas Bangalore. 2016. Revisiting supertagging and parsing: How to use supertags in transition-based parsing. In *12th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+ 12)*, pages 85–92.
- Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM.
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.
- Dan Friedman, Jungo Kasai, R Thomas McCoy, Robert Frank, Forrest Davis, and Owen Rambow. 2017. Linguistically rich vector representations of supertags for tag parsing. In *Proceedings of the 13th International Workshop on Tree Adjoining Grammars and Related Formalisms*, pages 122–131.
- Yarin Gal and Zoubin Ghahramani. 2016. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in neural information processing systems*, pages 1019–1027.
- Alex Graves and Jürgen Schmidhuber. 2005. Frame-wise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5-6):602–610.
- Julia Hockenmaier and Mark Steedman. 2007. Ccg-bank: a corpus of ccg derivations and dependency structures extracted from the penn treebank. *Computational Linguistics*, 33(3):355–396.
- Aravind K Joshi and Yves Schabes. 1997. Tree-adjoining grammars. In *Handbook of formal languages*, pages 69–123. Springer.
- Jungo Kasai, Robert Frank, R Thomas McCoy, Owen Rambow, and Alexis Nasr. 2017. Tag parsing with neural networks and vector representations of supertags. In *Conference on Empirical Methods in Natural Language Processing*, pages 1712–1722.
- Jungo Kasai, Robert Frank, Pauli Xu, William Merrill, and Owen Rambow. 2018. [End-to-end graph-based TAG parsing with neural networks](#). In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, pages 1181–1194.
- Diederik P. Kingma and Jimmy Ba. 2015. [Adam: A method for stochastic optimization](#). In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

- Mike Lewis, Kenton Lee, and Luke Zettlemoyer. 2016. Lstm ccg parsing. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 221–231.
- Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. 2019. Multi-task deep neural networks for natural language understanding. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4487–4496, Florence, Italy. Association for Computational Linguistics.
- Minh-Thang Luong, Quoc V Le, Ilya Sutskever, Oriol Vinyals, and Lukasz Kaiser. 2015. Multi-task sequence to sequence learning. *arXiv preprint arXiv:1511.06114*.
- Xuezhe Ma and Eduard Hovy. 2016. End-to-end sequence labeling via bi-directional lstm-cnns-crf. *arXiv preprint arXiv:1603.01354*.
- 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.
- Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch. In *NIPS-W*.
- Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*.
- Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.
- Cicero D Santos and Bianca Zadrozny. 2014. Learning character-level representations for part-of-speech tagging. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1818–1826.
- Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.
- Ashish Vaswani, Yonatan Bisk, Kenji Sagae, and Ryan Musa. 2016. Supertagging with lstms. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 232–237.
- David Vilares, Mostafa Abdou, and Anders Søgaard. 2019. Better, faster, stronger sequence tagging constituent parsers. 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 3372–3383, Minneapolis, Minnesota. Association for Computational Linguistics.
- Wenduan Xu. 2016. Lstm shift-reduce ccg parsing. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1754–1764.
- Wenduan Xu, Michael Auli, and Stephen Clark. 2015. Ccg supertagging with a recurrent neural network. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, volume 2, pages 250–255.