A Appendix

A.1 Reimplementation of Filippova and Altun (2013): additional details

In this work, we reimplement the method of Filippova and Altun (2013), who in turn implement a method partially described in Filippova and Strube (2008). There are inevitable discrepancies between our implementation and the methods described in these two prior papers.

Some discrepancies arise from differences in syntactic formalisms. To begin, prior work uses a tree transformation method which is no longer strictly compatible with UD. For instance, the tree transformation from prior work assumes PPs are headed by prepositions, which is not true in UD (Schuster and Manning, 2016). We thus reimplement the tree transformation, using the enhanced dependencies representation from CoreNLP, which provides off-the-shelf augmented modifiers and augmented conjuncts that are very similar to the augmented edge labels from prior work. We exclude a syntactic constraint based on the $\overset{sc}{\rightarrow}$ relation, which is not included in UD.

Other possible discrepancies arise from differences in part-of-speech taggers. In particular, the aforementioned tree transform from prior work adds an edge between the root node and all verbs in a sentence, as a preprocessing step. This ensures that subclauses can be removed from parse trees, and then merged together to create a compression from different clauses of a sentence. However, we found that replicating this transform literally (i.e. only adding edges from the original root to all "verbs") made it impossible for the ILP to recreate some of the gold compressions in the dataset. (We suspect that this is because our part-of-speech tagger and the original part-of-speech tagger employed in Filippova and Altun (2013) sometimes return different part-of-speech tags). Our tree transform therefore adds an edge between the root node and *all* tokens in a sentence. With this change, it is always possible for the ILP to output the gold compression.

We use Gurobi Optimization (2018) (v7) to solve the liner program. Filippova and Strube (2008) report using LPsolve.[1] We found that Gurobi sometimes segfaults nondeterminsitically during training. We implement checkpoints which save and restore the state of computation, allowing us to resume training when such errors occur. We assess convergence by examining the validation F1 score on the constrained task after each pass through the training data. The F1 score increases for each of eight passes through the training data, and then decreases slightly (drops by 10^{-3} points). We terminate training at this point.

Lastly, in Table 1 of their original paper, Filippova and Altun (2013) provide an overview of the syntactic, structural, semantic and lexical features in their model. We implement every feature explicitly described in their work, except where otherwise noted (e.g. syntactic feature not compatible with UD). However, additional features included in their model (but not explicity described in print) almost certainly affect performance.

[1] http://sourceforge.net/projects/lpsolve

A.2 Implementation of SLOR: additional details

We use the SLOR function to measure the readability of the shortened sentences produced by each compression system. Following (Lau et al., 2015), we define the function as

$$\text{SLOR} = \frac{\log P_m(\xi) - \log P_u(\xi)}{|\xi|} \qquad (1)$$

where ξ is a sequence of words, $P_u(\xi)$ is the unigram probability of this sequence of words and $P_m(\xi)$ is the probability of the sequence, assigned by a language model. $|\xi|$ is the length (in tokens) of the sentence.

We use a 3-gram language model trained on the Filippova and Altun (2013) corpus. We implement with KenLM (Heafield, 2011).

A.3 Subtree and compression brackets: additional details

Our markup input to our LSTM, $\boldsymbol{x}_j$, includes **subtree brackets** with a complex structure, used to represent the start and end of the tokens which will be pruned or inserted by an operation $o_j \in \{\text{PRUNE}, \text{INSERT}\}$. The start and end tags are each formed by concatenating two symbols: (1) a symbol o_j indicating the type of the operation proposal (i.e. prune or insert) and (2) a symbol d indicating the dependency type governing $T(v)$, such as `dobj`.

Additionally, the markup includes **compression brackets** which show the extent of the current compression (if the operation is PRUNE) or the extent of the compression which would result if the operation were to be accepted (if the operation is

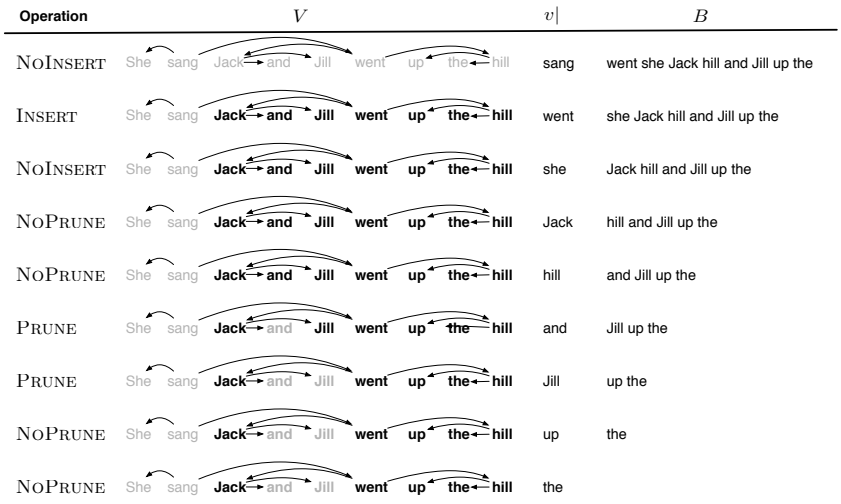

Figure 1: Nine operations of our transition-based compression return the compression: "Jack went up the hill". At each timestep, the compression has state $(V_j, [v_j|B])$. In the diagram, the tokens in V_j are shown in bold. The token v_j is shown in the third column.

INSERT). Concretely, these brackets show the extent of MAX(V_{j+1}, V_j) within s where MAX selects the largest set by cardinality and where V_{j+1} is all tokens which would be in V at step $j+1$, if the system were to execute operation o at timestep j. If $o_j =$ PRUNE then V_{j+1} will be smaller than V_j and the bracket symbols will show the extent of the current V_j. If $o_j =$ INSERT, then V_{j+1} will be larger than V_j and the compression brackets will show the extent of the compression which would result if the tokens were to be inserted.

A.4 Neural net training: additional details

We note several additional details about our neural network training procedure, including hyperparameter settings.

Batch size	135
Hidden size	907
Embeddings dim.	300
Total fully-connected layers	2
Fully-connected layers, hidden sizes	$(92, 2)$
Fully-connected layers, dropout	$(0.309, 0.408)$
Fully-connected layers, activations	Relu, Linear
Learning rate	$4.593 * 10^{-4}$
Weight decay	$2.421 * 10^{-8}$

Table 1: The hyperparameters for our model. The learning rate and weight decay parameter each clearly affect validation accuracy. The importance of other parameters is less clear.

- We train on 8 million tuples. The cardinality of the training set was bounded by available hardware resources, not by the total number of oracle tuples which can be generated with the Filippova and Altun (2013) dataset.
- We weight each training instance $(\boldsymbol{x_j}, y_j)$ using the default class weighting scheme in Scikit-learn (Pedregosa et al., 2011). The formula for assigning weights is $\frac{T_o}{2*T_{o,j}}$. T_o is the total number of training examples of accepted and rejected instances of the operation proposal o (e.g. T_o = total number of PRUNE examples + total number of NOPRUNE examples). $T_{o,j}$ is the total number of training operations labeled y_j for operations of proposal type o (e.g. $T_{o,j}$ = the total NOPRUNE operations, if $y_j = 0$ and o = PRUNE). The 2 in the numerator denotes the total number of classes. Alternative weighting methods are left for future work.
- We experimented with ELMo vectors (Peters et al., 2018), but found that we were able to achieve similar validation accuracies with much smaller embeddings. It is possible that ELMo-like vectors could be used to increase performance in the future.

References

Katja Filippova and Yasemin Altun. 2013. Overcoming the lack of parallel data in sentence compression.

In *EMNLP*.

Katja Filippova and Michael Strube. 2008. Dependency tree based sentence compression. In *Proceedings of the Fifth International Natural Language Generation Conference*.

LLC Gurobi Optimization. 2018. Gurobi optimizer reference manual.

Kenneth Heafield. 2011. KenLM: faster and smaller language model queries. In *EMNLP: Sixth Workshop on Statistical Machine Translation*.

Jey Han Lau, Alexander Clark, and Shalom Lappin. 2015. Unsupervised prediction of acceptability judgements. In *ACL*.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. 2011. Scikit-learn: Machine learning in python. *J. Mach. Learn. Res.*, 12:2825–2830.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *NAACL*.

Sebastian Schuster and Christopher D. Manning. 2016. Enhanced english universal dependencies: An improved representation for natural language understanding tasks. In *LREC*.