Appendix for Query-focused Sentence Compression in Linear Time (Handler and O'Connor, EMNLP 2019)

A Appendix

A.1 Algorithm

We formally present the VERTEX ADDITION compression algorithm, using notation defined in §**??**. ℓ linearizes a vertex set, based on left-to-right position in S. $|P|$ indicates the number of tokens in the priority queue.

Algorithm 1: VERTEX ADDITION

input: $s = (V, E)$, $Q \subseteq V$, $b \in \mathbb{R}^+$
$C \leftarrow Q$; $P \leftarrow V \setminus Q$;
while $\ell(C) < b$ *and* $|P| > 0$ **do**
  $v \leftarrow \text{pop}(P)$;
  if $p(y = 1) > .5$ *and* $\ell(C \cup \{v\}) \leq b$ **then**
    $C \leftarrow C \cup \{v\}$
  end
end
return $\ell(C)$

A.2 Neural network tuning and optimization

We learn network parameters for VERTEX ADDITION$_{NN}$ by minimizing cross-entropy loss against oracle decisions y_i. We optimize with ADAGRAD (Duchi et al., 2011). We learn input embeddings after initializing randomly. The hyperparameters of our network and training procedure are: the learning rate, the dimensionality of input embeddings, the weight decay parameter, the batch size, and the hidden state size of the LSTM. We tune via random search (Bergstra and Bengio, 2012), selecting parameters which achieve highest accuracy in predicting oracle decisions for the validation set. We train for 15 epochs, and we use parameters from the best-performing epoch (by validation accuracy) at test time.

Learning rate	0.025
Embedding dim.	315
Weight decay	1.88×10^{-9}
Hidden dim.	158
Batch size	135

Table 1: Hyperparameters for VERTEX ADDITION$_{NN}$

A.3 Reimplementation of Filippova and Altun (2013)

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.

1. Where the original authors train on only 100,000 sentences, we learn weights with the full training set to compare fairly with VERTEX ADDITION (each model trains on the full training set).

2. We use Gurobi Optimization (2018) (v8) to solve the liner program. Filippova and Strube (2008) report using LPsolve.[1]

3. We implement with the common Universal Dependencies (UD, v1) framework (Nivre et al., 2016). Prior work (Filippova and Strube, 2008) implements with older dependency formalisms (Briscoe et al., 2006; de Marneffe et al., 2006).

4. 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 described in the table. We do not implement features which are not described in the paper.

5. Filippova and Altun (2013) augment edge labels in the dependency parse of S as a preprocessing step. We reimplement this step using off-the-shelf augmented modifiers and augmented conjuncts available with the enhanced dependencies representation in CoreNLP (Schuster and Manning, 2016).

6. Filippova and Altun (2013) preprocess dependency parses by adding an edge between the root node and all verbs in a sentence.[2] We found that replicating this transform literally (i.e. only adding edges from the original root to all tokens tagged as verbs) made it impossible for the ILP to recreate some gold compressions. (We suspect that this is due to differences in output from part-of-speech taggers). We thus add an edge between the root node and *all* tokens in a sentence during preprocessing, allowing the ILP to always return the gold compression.

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

[2] This step ensures that subclauses can be removed from parse trees, and then merged together to create a compression from different clauses of a sentence.

We assess convergence of the ILP by examining validation F1 score on the traditional sentence compression task. We terminate training after six epochs, when F1 score stabilizes (i.e. changes by fewer than 10^{-3} points).

A.4 Implementation of SLOR

We use the SLOR function to measure the readability of the shortened sentences produced by each compression system. SLOR normalizes the probability of a token sequence assigned from a language model by adjusting for both the probability of the individual unigrams in the sentence and for the sentence length.[3]

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 training set of the Filippova and Altun (2013) corpus. We implement with KenLM (Heafield, 2011). Because compression often results in shortenings where the first token is not capitalized (e.g. a compression which begins with the third token in S) we ignore case when calculating language model probabilities.

A.5 Latency evaluation

To measure latency, for each technique, we sample 100,000 sentences with replacement from the test set. We observe the mean time to compress each sentence using Python's built-in *timeit* module. In order to minimize effects from unanticipated confounds in measuring latency, we repeat this experiment three separate times (with a one hour delay between experiments). Thus in total we collect 300,000 observations for each compression technique. We observe that runtimes are log normal, and thus report each latency as the geometric mean of 300,000 observations. We use an Intel Xeon processor with a clock rate of 2.80GHz.

[3]Longer sentences are always less probable than shorter sentences; rarer words make a sequence less probable.

A.6 Compression ratios

When comparing sentence compression systems, it is important to ensure that all approaches use the same rate of compression (Napoles et al., 2011). Following Filippova et al. (2015), we define the compression ratio as the character length of the compression divided by the character length of the sentence. We present test set compression ratios for all methods in Table 2. Because ratios are similar, our comparison is appropriate.

RANDOM	0.405
ILP	0.408
ABLATED	0.387
VERTEX ADDITION$_{LR}$	0.403
VERTEX ADDITION$_{NN}$	0.405
C_g Train	0.384
C_g Test	0.413

Table 2: Mean test time compression ratios for all techniques. We also show mean ratios for gold compressions C_g across the train and test sets.

.

References

James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305.

Ted Briscoe, John Carroll, and Rebecca Watson. 2006. The second release of the RASP system. In *Proceedings of the COLING/ACL 2006 Interactive Presentation Sessions*.

John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159.

Katja Filippova, Enrique Alfonseca, Carlos A Colmenares, Lukasz Kaiser, and Oriol Vinyals. 2015. Sentence compression by deletion with LSTMs. In *EMNLP*.

Katja Filippova and Yasemin Altun. 2013. Overcoming the lack of parallel data in sentence compression. In *EMNLP*. https://github.com/google-research-datasets/sentence-compression.

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 (v8).

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*.

Marie-Catherine de Marneffe, Bill MacCartney, and Christopher D. Manning. 2006. Generating typed dependency parses from phrase structure parses. In *LREC*.

Courtney Napoles, Benjamin Van Durme, and Chris Callison-Burch. 2011. Evaluating sentence compression: Pitfalls and suggested remedies. In *Proceedings of the Workshop on Monolingual Text-To-Text Generation*.

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajic, Christopher D. Manning, Ryan T. McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, Reut Tsarfaty, and Daniel Zeman. 2016. Universal Dependencies v1: A multilingual treebank collection. In *LREC*.

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