WebNLG Final Project Report

Rajeev Bhatt Ambati, Agnese Chiatti, Tejas Mahale, Thanh Tran

April 22, 2018

1 Introduction

This report provides an outline of the experimental pipelines we have tested based on the insights gathered from the interim trials. We refrain from describing the overarching task, which was part of the interim report, and focus on marginal customizations and improvement introduced during this second stage.

From the performance of the Neural Machine Translation model, we have observed that there are two main issues that have to be addressed in order to improve its performance: 1) handling unknown tokens and the misplacing of nouns, 2) identifying effective data preparation strategies. The first problem is due to the difficulty of neural networks in a simple operation of copying the input to the output. Pointer Generator Networks as hypothesized in [10] are proven to do the copy operation efficiently and therefore can be incorporated to address our problem. Secondly, most of the submissions for the WebNLG challenge have used classic Neural Machine Translation models and a few other variants like the attention model. Surprisingly, they have very high BLEU scores compared to our NMT models. This can be simply attributed to the preprocessing step before training the model. Thus, we have attempted to further refine our data representation before feeding it to the latter models, also inspired by another observation: rule-based language generation models such as UPF-FORGE were scoring higher on the unseen classes than on the classes employed for training [2], hinting to the need for finding more abstract and generalized representations of the lexical instances at hand.

In brief, we tested for the impact of delexicalization, aggregation and integration of grammar-based structures (namely constituency trees based on POS tags) on the final performance, where performance is here evaluated with respect to BLEU scores. Further, we controlled for the effects of adopting alternative language generation models for this task (i.e., seq2seq NMT with attention model and Pointer Generator Networks), in combination with the pre-processing pipelines that were found effective.

This seminal set of experiments showed that adopting a pre-processing routine that combines improved delexicalisation with aggregation increased the BLEU score by 6 units on seen classes and by 25 units on a combination of seen and unseen classes (i.e., test set), outperforming the previous baseline. Further, we found the application of an alternative seq2seq NMT architecture that includes attention models to be beneficial in terms of overall performance on both sets. Eventually, we discovered that embedding grammar-based structures as part of our input feature set, although not leading to top-performing results, can relatively mitigate the issue of learning longer, compound sentences.

2 Data Preparation

Pre-processing subtasks are essential to ensure a good mapping between the RDF Triples (subject, predicate, object) and a target coherent English structure. These typically include expression generation, delexicalisation, aggregations, surface realisation, and sentence segmentation [2]. In the baseline preprocessing, we decided to clean up the RDF triples by replacing underscores with spaces, removing quotation marks, and splitting the punctuations from text in both modified triples and lex. Next, we performed delexicalisation by replacing the corresponding subject with its entity type from DBPedia and through

replacing all the objects with the capitalized predicate,if found. No aggregation is performed within this baseline approach. For example, the original:

```
RDF Triples: <3Arena | location | "North Wall Quay">
     Lex:The 3Arena is located at the North Wall Quay.
```

becomes

```
RDF Triples: WRITTENWORK author AUTHOR
     WRITTENWORK mediaType MEDIATYPE
     WRITTENWORK numberOfPages NUMBEROFPAGES

     Lex: AUTHOR is the author of WRITTENWORK which can
     be found in hardcover and has NUMBEROFPAGES pages .
```

2.1 Improved Delexicalisation

While the delexicalisation process above performs well on seen data, the performance on unseen class is significantly lower due to overfitting caused by the following reasons: (i) using predicates as a replacement for the objects, (ii) relying on the default DBPedia entity type without generalizing them. To mitigate the latter, we employed the following pipeline:

1. Apply Regular Expressions and Query DBPedia to collect properties for the objects and subjects. Unknown entities are labeled UNK - There are around 600 unknowns after scraping and querying
2. Combining different DBPedia entity types and properties types into a set of generalize properties. E.g Astraunaut and Author are combined into Person. The entire set include
3. Encode each triple as:

   ENTITYi <ENTITYi PROPERTY> PREDICATE ENTITYj <ENTITYj PROPERTY>

4. Automatically encode the lex English sentence with ENTITYith using dynamic programming

For example, the following triple and lex:

```
TRIPLES:
        Addiction_(journal) | publisher | Wiley-Blackwell
        Addiction_(journal) | ISSN_number | "1360-0443"
        Addiction_(journal) | LCCN_number | 93645978
        Addiction_(journal) | abbreviation | "Addiction"

LEX: The Addiction Journal is published by Wiley - Blackwell and is abbreviated
to Addiction. The ISSN number is 1360 - 0443 and the LCCN number is 93645978 .
```

Becomes

```
TRIPLES: ENTITY1 UNIVERSITY publisher ENTITY2
ORGANIZATION ENTITY1 UNIVERSITY issn number ENTITY3 UNK
ENTITY1 UNIVERSITY lccn number ENTITY4 NUMBER ENTITY1
UNIVERSITY abbreviation ENTITY5 UNK
LEX: The ENTITY5 Journal is published by ENTITY2 and is abbreviated to ENTITY1
number is ENTITY3 and the LCCN number is ENTITY4 .
```

2.2 Aggregation

Next, we apply appropriate re-lexicalisation by not repeating the property type of an ENTITY if that ENTITY appeared before in the same sentence. This will furthur reduce the triples' sentence length and

make the context vector generated by the NMT's encoder more focus without losing any information regarding the relationship between the subject and object.

For Example, given the above TRIPLES, the property UNIVERSITY associated with ENTITY1 show up only once now:

```
TRIPLES: ENTITY1 UNIVERSITY publisher ENTITY2
ORGANIZATION ENTITY1 issn number ENTITY3 UNK ENTITY1
lccn number ENTITY4 NUMBER ENTITY1 abbreviation ENTITY5 UNK
```

2.3 Constituency Parsed Trees

The results obtained by [8] through a pipeline-based system where quite surprising. In the UPF-FORGE case, after manually generating templates summarizing the different grammar structured represented throughout the WebNLG data set, triples were reduced to a Predicate-Argument tree-like structure and information on the related POS tags was also attached. This strategy led to top performance, among the competing systems, on the unseen classes, but significantly lower performance on the development set consisting of only seen classes.
Our tests on enhancing data representation to embed grammar-based properties stemmed from two main intuitions: (i) relying on a manual process for template definition could be rather inefficient and not sufficiently robust to handle different corpora, (ii) combining grammar-based attributes with delexicalisation (that led to the best results for UMEL on seen classes) has the potential to provide a synergy to scale and generalize more effectively.

However, one potential drawback that we expect is the higher computational cost of training on longer data units (i.e., obtained by concatenating delexicalised triples with parsed trees) and the risk of underfitting (as mentioned in Section 2.1 as well). In other words, the more we generalize to higher-level structures, the more we are able to manage any new incoming class, at the expenses of performance on the training sample.

Based on these premises, we set up a different pre-processing routine than UPF-FORGE [8]. First, we consider the lex set provided for training, and use the Stanford Core NLP library [6] to create one constituency tree for each input sentence. In constituency parsing, tokens are represented through their related Part-of-Speech tags and linked in a tree structure that differentiates terminal nodes from non-terminal nodes. The underlying grammar provides a graphic depiction of the sentence based on its constituency relations, i.e., correspondences between different clauses are identified based on atomic subject-predicate relations.

For instance, this sample delexicalised lex obtained from the previous pre-processing steps:

```
The ENTITY5 Journal is published by ENTITY2 and is abbreviated to ENTITY1
number is ENTITY3 and the LCCN number is ENTITY4.
```

would be parsed to the following tree:

```
(ROOT
        (S
        (S        (NP (DT The) (NNP ENTITY5)
        (NNP Journal))
        (VP          (VP (VBZ is)
        (VP (VBN published)              (PP (IN by)
        (NP (NN ENTITY2)))))
       (CC and)          (VP (VBZ is)            (VP (VBN abbreviated)
       (PP (TO to)
       (NP
       (NP (NN ENTITY1)
       (NN number))
       (SBAR       (S
```

```
        (VP (VBZ is)
        (NP (NN ENTITY3)))))))))))

         (CC and)
         (S
         (NP (DT the) (NNP LCCN) (NN number))
         (VP (VBZ is)
                  (NP (NN ENTITY4))))
(. .)))
```

Further, to derive the same auxiliary data for development and test sets: (i) we considered each triple set in the development set, (ii) we compared it against 50 randomly selected triples available for training, (iii) similarity between strings is computed after representing triple through their related POS tags and transforming those POS tags to their Wordnet counterpart [3], specifically following a procedure that mimes the algorithm proposed in [7]. Ultimately, (iv) the constituency tree linked to the most similar training example is reused and paste over to the development set. All the aforementioned steps are applied to the testing set as well.
Eventually, the produced trees are concatenated line-by-line with the original files containing sequences of delexicalised and aggregated triples (produced as described in Sections 2.1 and 2.2), to comply to the format expected by the NMT seq2seq model.

The choice of 50 as random sample size is based on the fact that, intuitively, this pairwise comparison is of order $O(n^2)$, hence requiring to take optimization needs into careful consideration. Plus, we noticed that even with as few examples as 50, similarity were on average ranging between 0.7 and 1, presumably due to the lack of syntactic diversity in input triples (i.e., entities are recognized as nouns anyways). This evidence raises another problem: different sequences of triples could be found similar even though their actual tree would be fairly different. We chose to leverage computational cost and annotation cost of manual template generation with desired accuracy for the purpose of this analysis.

3 Models

3.1 Sequence-to-Sequence model with Attention

Neural Machine Translation has improved quite significantly from its predecessors like the statistical machine translation models, this can be largely owed to encoder-decoder type architectures proposed in [11]. Although they are quite effective, a potential issue with this encoder-decoder type architecture is that the neural network should be able to compress all of the information in the input sequence into a latent representation. The decoder later decompresses the latent representation into the output but most of the information is either lost or is not coarse, especially for long sentences. In order to address this issue, [1] proposed a model that tries to align and translate jointly. Each time a the model generates a word, it calculates a probability distribution over the positions in source sentence where the most relevant information is concentrated. The model then predicts a target word based on the context vectors associated with these source positions and all the previous generated target words. It can be described by the following mathematical equations:

In the attention model architecture, the conditional probability is defined as follows:

$$p\left(y_i \mid y_1, ..., y_{i-1}, x\right) = g\left(y_{i-1}, s_i.c_i\right)$$

Where s_i is an RNN hidden state for time ii, computed by

$$s_i = f\left(s_{i-1}, y_{i-1}, c_i\right)$$

The context vector c_i depends on a sequence of annotations $(h_1, ..., h_{T_x})$ to which an encoder maps the input sentence. Each annotation h_i contains information about the whole input sequence with a strong focus on the parts surrounding the i-th word of the input sequence.

The context vector c_i is, then, computed as a weighted sum of these annotations h_i:

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

The weight α_{ij} of each annotation h_j is given by

$$\alpha_{ij} = \frac{exp\left(e_{ij}\right)}{\sum_{k=1}^{T_x} exp\left(e_{ik}\right)}$$

where

$$e_{ij} = a\left(s_{i-1}, h_j\right)$$

is an alignment model which scores how well the input around the position j and the output at position i match. The score is based on the RNN hidden state s_{i-1} and the j-th annotation h_j of the input sequence. There are many ways of calculating this alignment score. One is to use a feed-forward network for on top s_{i-1}, h_j as used in [1] and the other by [5] is as follows:

$$a\left(s_{i-1}, h_j\right) = \begin{cases} h_j^T s_{i-1} & dot \\ h_j^T W_a s_{i-1} & general \\ v_a^T W_a \left[h_j; s_{i-1}\right] & concat \end{cases}$$

There are also other ways of scoring such as taking a simple dot product between s_{i-1}, h_j and multi-head attention which is outside the scope of this report. We have used Bahdanau attention as implemented in the tensorflow NMT repository. One other reason attention may help is that our input concatenation has repetitions of the same word, so the attention distribution could be high only at different words and help the decoder. Though the attention strategy may help the decoder, neural networks in general cannot do a simple operation of copying the input to the output. In the next Section, we will describe Pointer Generator Networks whose architecture is developed specifically for this purpose.

3.2 Pointer Generator Network

A hybrid pointer-generator network is proposed in [10] which can copy words from source text via pointing. This functionality aids accurate reproduction of information, while retaining the ability to produce novel words through the generator. As discussed earlier, simple encoder-decoder architecture exhibit undesirable behavior such as inaccurately reproducing factual details (misplacing nouns in our case), an inability to deal with out-of-vocabulary (OOV) words, and repeating themselves. The attention distribution α_i and the context vector c_i are calculated as in section-3.1. In addition, the generation probability $p_{gen} \in [0, 1]$ for timestep i is calculated from the context vector c_i, the decoder state s_{i-1} and the decoder input x_i

$$p_{gen} = \sigma(w_c^T c_i + w_s^T s_{i-1} + w_x^T x_t + b_{ptr})$$

where w_c, w_c, w_x and scalar b_{ptr} are learnable parameters and σ is the sigmoid function. Next, p_{gen} is used as a soft switch to choose between generating a word from vocabulary by sampling from P_{vocab} , or copying a word from the input sequence by sampling from the attention distribution α_i. For each document let the extended vocabulary denote the union of the vocabulary and all the words in the input sequence. The following is the probability distribution over the extended vocabulary:

$$P(w) = p_{gen} P_{vocab}(w) + (1 - p_{gen}) \sum_{i:w_i=w} \alpha_i$$

Note that if w is an OOV word, then $P_{vocab}(w)$ is zero; similarly if w doesn't appear in the input sequence, then $\sum_{i:w_i=w} \alpha_i$ is zero. During training, the loss for timestep t is the negative log-likelihood of the target word w_t^* for that timestep:

$$loss_t = -\log P\left(w_t^*\right)$$

and the overall loss for the whole sequence is:

$$loss = \frac{1}{T}\sum_{t=0}^{T} loss_t$$

4 Experimental Setting

The remainder of this Section provides details on the architectural settings and hyperparameters choices adopted for each tested pipeline.
All the run experiments were compared with a baseline approach. We selected the best results from our interim reports as baseline. Specifically, the baseline not only applies linearization to triples associated with the same lex, but also delexicalizes through the default pre-processing script provided as part of the WebNLG challenge resources [2]. Then, NMT seq2seq off-the-shelf solution illustrated in [4] is applied to the same data set.

4.1 NMT seq2seq

For this pipeline, we used the model described in [4] on two different data sets:(i) triple sets post delexicalisation and aggregation, (ii) delexicalised sequences integrated with constituency parsed trees.

In both cases, the architecture used is the same as in the baseline top-performing case, for control reasons. The latter consists of 2 layers over 320 LSTM units. Dropout rate is set to 0.30 and 0.35 respectively, as that rate seemed to mirror the class distribution in a satisfactory way, based on our interim observations. For the first corpus, we run for 7000 iterations but the number of epochs is increased to 20000 instead, for the case that integrates constituency trees, as we hypothesized that it would take more iteration to learn a larger feature set.

4.2 NMT seq2seq with attention

Secondly, after retaining the data collection that led to the best performance in 5.1, we applied a variation of sequence to sequence model that integrates attention mechanisms. The parameters used in this workflow are the same introduced for delexicalised and aggregated input with base NMT seq2seq: dropout was 0.3, for 2 layers and 320 units, over 7000 iterations.

4.3 Pointer generator network

For implementing the code, we took help from the publicly available repository of the paper [10] and modified it to our purpose. We have use a learning rate of 0.15 and tensorflow Adam Optimizer for out training. The loss started at 9.82 and decreased till 4.59 after which we have tried to decrease the learning rate to 0.001 but it didn't help the optimizer to move away from local optimum. Observing that the model has a lot of repetition in words, We have also tried the coverage mechanism but that also didn't work for our cause.

5 Evaluation

Results were evaluated with respect to the BLEU score, also included as part of the WebNLG challenge. BLEU (bilingual evaluation understudy) is a state-of-the art approach to evaluate machine-translated test, after translation from a source language to a target language. Ground-truth is formed by manually-transcribed lex sets provided for this task.
The BLEU metric essentially compares the n-gram representing the generated sentence with the gold

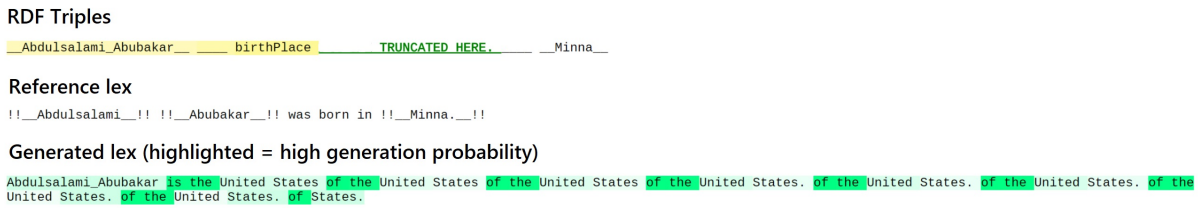

Figure 1: The visualization of generated output. The green highlighting shows the generation probability. The OOV word "Abdulsalani_Abubakar" have low generation probability as it is copied from input and the words "of" and "the" have high generation probabilities.

standard n-gram, as $P = \frac{m}{w_t}$, where m is the number of generated terms matching the reference and w_t is the total number of tokens constituting the generated sentence.
Moreover, the modified precision metric introduced in [9], averages the count of n-gram matches across the total number of n-gram counts forming the full corpus.

Note that one set of modified triples can be mapped to multiple lexes; thus there are multiple methods to evaluate the performance of the model. In the case of the WebNLG Paper[2], the BLEU score is computed from multiple references by splitting the model's translation and the lex files. In our case, the BLEU score was computed by comparing the translation and the lex files directly, which is used for most of our analysis. However, we still compute the BLEU score according to the WebNLG paper's method for references and comparisions [2]. Further, we differentiated two sub-cases, besides evaluating on the full sequence set. Specifically, BLEU scores were also computed separately for longer and shorter sentences, where the threshold to discriminate between the two is set to 100 characters.

6 Experimental Results and Discussion

BLEU scores obtained for each experimental pipeline are reported in Table 1. As introduced in Section 5, besides evaluating on all input sequences, we also discriminated between phrases that contain less than 100 characters and phrases that are longer than 100 characters. Table 2 shows overall results obtained when using the evaluation script provided by the challenge organizers, for easier comparison with the challenge ranking.
The data preparation routines described in Sections 2.1 and 2.2 are indicated here as *"Delex+Aggr"*, whereas the integration of auxiliary grammar-based features (as detailed in Section 2.3) is marked as *"Delex+CTrees"*. The seq2seq NMT architecture derived from [4] (*"AttNMTseq2seq"*) is compared to the Pointer Generator based system (*"PointGen"*) when applied to the same pre-processed set.

Table 1: BLEU scores obtained in each tested pipeline, compared with the baseline scores taken as reference from interim trails.

Approach	**Development Set**			**Test Set**		
	Short phrases	Long phrases	Full set	Short phrases	Long phrases	Full set
Baseline	32.51	30.80	31.91	4.91	2.05	2.82
NMTseq2seq (Delex+Aggr)	40.79	36.20	37.21	30.38	26.48	27.49
NMTseq2seq (Delex+CTrees)	26.49	31.44	31.39	21.42	25.65	24.90
AttNMTseq2seq (Delex+Aggr)	44.56	39.31	40.82	33.42	28.33	30.07

Overall, the results did confirm that applying more refined data preparation strategies can lead to improved performance, as all the experimental flows improved performance when compared to the baseline

Table 2: BLEU scores obtained in each tested pipeline, with official evaluation script. *Interim NMT (nonDelex)* refers to results obtained on triples sentences from prior attempts that were not delexicalized nor aggregated, i.e., not to be confused with Baseline approach in Table 1

Approach	**Development Set**	**Test Set**
NMTseq2seq (Delex+Aggr)	47.11	32.16
NMTseq2seq (Delex+CTrees)	40.00	29.46
AttNMTseq2seq (Delex+Aggr)	53.22	35.85
PointGen (Delex+Aggr)	19.2	8.6
Interim NMT (nonDelex)	19.1	1.3

Table 3: Hyper-parameter tuning for NMT model with 4 layers.

dropout	BLEU
0.2	2.2
0.4	3.4
0.5	2.6

score taken as reference from the interim attempts. This evidence holds on both development and the unseen test data, hence indicating a more robust generalization to unseen classes.
Applying Delexicalization and Aggregation in sequence, as well as implying more complex models that embed attention mechanisms, led to further marginal improvement in performance, at both development and test stage. Interestingly, all the tested models performed better on shorter phrases than on longer feature sequences, except for the method that integrated constituency parsed trees. Our speculation is that, although this methods provides only a proxy for grammatical structures (i.e., subject-predicate structures are derived, for development and test, based on a simple heuristic and on a small random sample), relying on auxiliary information on correspondences between different clauses relatively increased the system's capacity to learn more complex and longer sentences. Figure 1 shows the visualization of the output of pointer generator network on test set. The increase in performance compared to the interim results can be attributed to the capacity of pointer generator networks to copy from input to output. Tables 3, 4, 5 show the hyper-parameter tuning for both plain NMT model and also with attention.

Based on the explanations in Section 5, our evaluation procedure tends to underestimate scores compared to the official challenge evaluation script. However, scores based on the WebNLG challenge official script are summarized in Table 2. Thus, our top performing system would have placed first on unseen classes, beating UPF-FORGE with 35.85 BLEU score on test.

7 Conclusion

In this report we showed how integrating a more refined pre-processing pipeline (i.e., delexicalisation that makes use of external data sources Web-scraped from DBPedia and following aggregation) helped increasing performance, especially on unseen class entities. Further, we found the combination of said pre-processing with alternative models for language generation to also be beneficial. One of our hypothesis was that grammar-based templates would have aid quality of generated text, but we discovered that aggregating the delexicalised input led to relatively better results than concatenated with constituency trees. One interesting finding, however, was that including tree-data led to better performance on longer sequences than on shorter sequences, behaving differently from all the other tested models.

Table 4: Hyper-parameter tuning for NMT model with a dropout of 0.2.

layers	BLEU
2	1.5
4	2.2
6	2.5

Table 5: Hyper-parameter tuning for NMT-attention model with 4 layers.

dropout	BLEU
0.2	2.7
0.4	3.2
0.5	3.2

This motivates running additional future experiment, both on Attention-embedded seq2seq and Pointer Generator Networks, on the data collection that includes auxiliary trees, to test whether the advantages introduced to handle compound phrases are complementary to the higher quality ensured by those architectures. Further, the rationale used to select the best candidate tree on development and test (i.e., derived from the training set) could be further refined and improved (e.g., averaging across top-k candidates instead of just picking the maximized similarity, or including external data to parse trees for unseen classes).
Along those same lines, we showed how Pointer Generator Network can support input copy to output, however the employed parameters and underlying performance could be further test [1], or combined with attention based models, to further assess its effectiveness on this task. Ultimately, as previously mentioned in interim report, one potential direction of this work is testing the use of recursive neural networks. Recursive neural networks were previously shown to perform well for parsing and therefore we can infer that they have the capacity to model tree structured data. Since our WebNLG task is an inverse parsing problem, in a way it is also a tree structured data, thus suggesting the testing of Recursive Neural Networks as promising next step.

References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014.

[2] Emilie Colin, Claire Gardent, Yassine Mrabet, Shashi Narayan, and Laura Perez-Beltrachini. The webnlg challenge: Generating text from dbpedia data. In *Proceedings of the 9th International Natural Language Generation conference*, pages 163–167, 2016.

[3] Adam Kilgarriff. Wordnet: An electronic lexical database, 2000.

[4] Minh-Thang Luong, Eugene Brevdo, and Rui Zhao. Neural machine translation (seq2seq) tutorial. *https://github.com/tensorflow/nmt*, 2017.

[5] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. *CoRR*, abs/1508.04025, 2015.

[6] Christopher Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky. The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, pages 55–60, 2014.

[1] The authors in the pointer generator network trained it for 230k iterations for 2-3 days in order to achieve best performance. We trained for only a few iterations, thus training longer and searching for optimal hyper-parameters may help.

[7] Rada Mihalcea, Courtney Corley, Carlo Strapparava, et al. Corpus-based and knowledge-based measures of text semantic similarity. In *AAAI*, volume 6, pages 775–780, 2006.

[8] Simon Mille and Stamatia Dasiopoulou. Forge at webnlg 2017. 2017.

[9] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics, 2002.

[10] Abigail See, Peter J. Liu, and Christopher D. Manning. Get to the point: Summarization with pointer-generator networks. *CoRR*, abs/1704.04368, 2017.

[11] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014.