Improving Quality and Efficiency in Plan-based Neural Data-to-Text Generation

Amit Moryossef[†] **Ido Dagan**[†] **Yoav Goldberg**[†‡]
amitmoryossef@gmail.com, {dagan,yogo}@cs.biu.ac.il

[†]Bar Ilan University, Ramat Gan, Israel
[‡]Allen Institute for Artificial Intelligence

Abstract

We follow the step-by-step approach to neural data-to-text generation we proposed in Moryossef et al. (2019), in which the generation process is divided into a text-planning stage followed by a plan-realization stage. We suggest four extensions to that framework: (1) we introduce a trainable neural planning component that can generate effective plans several orders of magnitude faster than the original planner; (2) we incorporate typing hints that improve the model's ability to deal with unseen relations and entities; (3) we introduce a verification-by-reranking stage that substantially improves the faithfulness of the resulting texts; (4) we incorporate a simple but effective referring expression generation module. These extensions result in a generation process that is faster, more fluent, and more accurate.

1 Introduction

In the data-to-text generation task (D2T), the input is data encoding facts (e.g., a table, a set of tuples, or a small knowledge graph), and the output is a natural language text representing those facts.[1] In neural D2T, the common approaches train a neural end-to-end encoder-decoder system that encodes the input data and decodes an output text. In recent work (Moryossef et al., 2019) we proposed to adopt ideas from "traditional" language generation approaches (i.e. Reiter and Dale (2000); Walker et al. (2007); Gatt and Krahmer (2017)) that separate the generation into a *planning* stage that determines the order and structure of the expressed facts, and a *realization* stage that maps the plan to natural language text. We show that by breaking the task this way, one can achieve the same fluency of neural generation systems while being able to better control the form of the generated text and to improve its correctness by reducing missing facts and "hallucinations", common in neural systems.

In this work we adopt the step-by-step framework of Moryossef et al. (2019) and propose four independent extensions that improve aspects of our original system: we suggest a new plan generation mechanism, based on a trainable-yet-verifiable neural decoder, that is orders of magnitude faster than the original one (§3); we use knowledge of the plan structure to add typing information to plan elements. This improves the system's performance on unseen relations and entities (§4); the separation of planning from realizations allows the incorporation of a simple output verification heuristic that drastically improves the correctness of the output (§5); and finally we incorporate a post-processing referring expression generation (REG) component, as proposed but not implemented in our previous work, to improve the naturalness of the resulting output (§6).

2 Step-by-step Generation

We provide a brief overview of the step-by-step system. See Moryossef et al. (2019) for further details. The system works in two stages. The first stage (planning) maps the input facts (encoded as a directed, labeled graph, where nodes represent entities and edges represent relations) to text plans, while the second stage (realization) maps the text plans to natural language text.

The text plans are a sequence of sentence plans—each of which is a tree— representing the ordering of facts and entities within the sentence. In other words, the plans determine the separation of facts into sentences, the ordering of sentences, and the ordering of facts and entities within each sentence. This stage is *completely verifiable*:

[1] In this paper, we focus on a setup where the desired output represents *all* and *only* the facts expressed in the dataset. Other variants also involve content selection, allowing the process to select which subset of the facts to express.

Proceedings of The 12th International Conference on Natural Language Generation, pages 377–382,
Tokyo, Japan, 28 Oct - 1 Nov, 2019.

the text plans are guaranteed to faithfully encode all and only the facts from the input. The realization stage then translates the plans into natural language sentences, using a neural sequence-to-sequence system, resulting in fluent output.

3 Fast and Verifiable Planner

The data-to-plan component in Moryossef et al. (2019) exhaustively generates all possible plans, scores them using a heuristic, and chooses the highest scoring one for realization. While this is feasible with the small input graphs in the WebNLG challenge (Colin et al., 2016), it is also very computationally intensive, growing exponentially with the input size. We propose an alternative planner which works in linear time in the size of the graph and remains verifiable: generated plans are guaranteed to represent the input faithfully.

The original planner works by first enumerating over all possible splits into sentences (sub-graphs), and for each sub-graph enumerating over all possible undirected, unordered, Depth First Search (DFS) traversals, where each traversal corresponds to a sentence plan. Our planner combines these into a single process. It works by performing a series of what we call *random truncated DFS traversals*. In a DFS traversal, a node is visited, then its children are visited recursively in order. Once all children are visited, the node "pops" back to the parent. In a *random truncated traversal*, the choice of which children to visit next, as well as whether to go to the next children or to "pop", is non-deterministic (in practice, our planner decides by using a neural-network controller). Popping at a node before visiting all its children truncates the DFS: further descendants of that node will not be visited in this traversal. It behaves as a DFS on a graph where edges to these descendants do not exist. Popping the starting node terminates the traversal.

Our planner works by choosing a node with a non-zero degree and performing a truncated DFS traversal from that node. Then, all edges visited in the traversal are removed from the input graph, and the process repeats (performing another truncated DFS) until no more edges remain. Each truncated DFS traversal corresponds to a sentence plan, following the DFS-to-plan procedure of Moryossef et al. (2019): the linearized plan is generated incrementally at each step of the traversal. This process is linear in the number of edges in the graph.

At training time, we use the plan-to-DFS mapping to perform the correct sequence of traversals, and train a neural classifier to act as a controller, choosing which action to perform at each step. At test time, we use the controller to guide the truncated DFS process. This mechanism is inspired by transition based parsing (Nivre and McDonald, 2008). The action set at each stage is dynamic. During traversal, it includes the available children at each stage and POP. Before traversals, it includes a *choose-i* action for each available node n_i. We assign a score to each action, normalize with softmax, and train to choose the desired one using cross-entropy loss. At test time, we either greedily choose the best action, or we can sample plans by sampling actions according to their assigned probabilities.

Feature Representation and action scoring. Each graph node n_i corresponds to an entity x_{n_i}, and has an associated embedding vector $\mathbf{x_{n_i}}$. Each relation r_i is associated with an embedding vector $\mathbf{r_i}$. Each labeled input graph edge $e_k = (n_i, r_\ell, n_j)$ is represented as a projected concatenated vector $\mathbf{e_k} = \mathbf{E}(\mathbf{x_{n_i}}; \mathbf{r}_\ell; \mathbf{x_{n_j}})$, where $\mathbf{E}$ is a projection matrix. Finally, each node n_i is then represented as a vector $\mathbf{n_i} = \mathbf{V}[\mathbf{x_{n_i}}; \sum_{e_j \in \pi(i)} \mathbf{e_j}; \sum_{e_j \in \pi^{-1}(i)} \mathbf{e_j}]$, where $\pi(i)$ and $\pi^{-1}(i)$ are the incoming and outgoing edges from node n_i. The traverse-to-child-via-edge-e_j action is represented as $\mathbf{e_j}$, choose-node-i is represented as $\mathbf{n_i}$ and pop-to-node-i is represented as $\mathbf{n_i} + \mathbf{p}$ where $\mathbf{p}$ is a learned vector. The score for an action a at time t is calculated as a dot-product between the action representation and the LSTM state over the symbols generated in the plan so far. Thus, each decision takes into account the immediate surrounding of the node in the graph, and the plan structure generated so far.

Speed On a 7 edges graph, the planner of Moryossef et al. (2019) takes an average of 250 seconds to generate a plan, while our planner takes 0.0025 seconds, 5 orders of magnitude faster.

4 Incorporating typing information for unseen entities and relations

In Moryossef et al. (2019), the sentence plan trees were linearized into strings that were then fed to a neural machine translation decoder (OpenNMT) (Klein et al., 2017) with a copy mecha-

nism. This linearization process is lossy, in the sense that the linearized strings do not explicitly distinguish between symbols that represent *entities* (e.g., BARACK_OBAMA) and symbols that represent *relations* (e.g., works-for). While this information can be deduced from the position of the symbol within the structure, there is a benefit in making it more explicit. In particular, the decoder needs to act differently when decoding relations and entities: entities are copied, while relations need to be verbalized. By making the typing information explicit to the decoder, we make it easier for it to generalize this behavior distinction and apply it also for *unseen* entities and relations. We thus expect the typing information to be especially useful for the unseen part of the evaluation set.

We incorporate typing information by concatenating to the embedding vector of each input symbol one of three embedding vectors, **S**, **E** or **R**, where **S** is concatenated to structural elements (opening and closing brackets), **E** to entity symbols and **R** to relation symbols.

5 Output verification

While the plan generation stage is guaranteed to be faithful to the input, the translation process from plans to text is based on a neural seq2seq model and may suffer from known issues with such models: hallucinating facts that do not exist in the input, repeating facts, or dropping facts. While the clear mapping between plans and text helps to reduce these issues greatly, the system in Moryossef et al. (2019) still has 2% errors of these kinds.

Existing approaches: soft encouragement via neural modules. Recent work in neural text generation and summarization attempt to address these issues by trying to map the textual outputs back to structured predicates, and comparing these predicates to the input data. Kiddon et al. (2016) uses a neural checklist model to avoid the repetition of facts and improve coverage. Agarwal et al. (2018) generate k-best output candidates with beam search, and then try to map each candidate output back to the input structure using a reverse seq2seq model trained on the same data. They then select the highest scoring output candidate that best translates back to the input. Mohiuddin and Joty (2019) reconstructs the input in training time, by jointly learning a back-translation model and enforcing the back-translation to reconstruct the input. Both of these approaches are "soft" in the sense that they crucially rely on the internal dynamics or on the output of a neural network module that may or may not be correct.

Our proposal: explicit verification. The separation between planning and realization provided by the step-by-step framework allows incorporating a robust and straightforward *verification step*, that does not rely on brittle information extraction procedures or trust neural network models.

The plan-to-text generation handles each sentence individually and translates entities as copy operations. We thus have complete knowledge of the generated entities and their locations. We can then assess the correctness of an output sentence by comparing[2] its sequence of entities to the entity sequence in the corresponding sentence plan, which is guaranteed to be complete.

We then decode k-best outputs and rerank them based on their correctness scores, tie-breaking using model scores. We found empirically that, with a beam of size 5 we find at least one candidate with an exact match to the plan's entity sequence in 99.82% of the cases for seen entities and relations compared to 98.48% at 1-best, and 72.3% for cases of unseen entities and relations compared to 58.06% at 1-best. In the remaining cases, we set the system to continue searching by trying other plans, by going down the list of plans (when using the exhaustive planner of Moryossef et al. (2019)) or by sampling a new plan (when using the linear time planner suggested in this paper).

6 Referring Expressions

The step-by-step system generates entities by first generating an indexed entity symbols, and then lexicalizing each symbol to the string associated with this entity in the input structure (i.e., all occurrences of the entity *11TH MISSISSIPPI INFANTRY MONUMENT* will be lexicalized with the full name rather than "*it*" or "*the monument*"). This results in correct but somewhat unnatural structures. In contrast, end-to-end neural generation systems are trained on text that includes referring expressions, and generate them naturally as part of the decoding process, resulting in natural looking text. However, the generated referring expressions are sometimes incorrect. Moryossef et al. (2019) suggests the possibility of handling

[2] We use Levenshtein-distance (Levenshtein, 1966).

this with a post-processing referring-expression generation step (REG). Here, we propose a concrete REG module and demonstrate its effectiveness. One option is to use a supervised REG module (Ferreira et al., 2018), that is trained to lexicalize in-context mentions. Such an approach is suboptimal for our setup as it is restricted to the entities and contexts it seen in training, and is prone to error on unseen entities and contexts.

Our REG solution lexicalizes the first mention of each entity as its associated string and attempts to generate referring expressions to subsequent mentions. The generated referring expressions can take the form "PRON", "X" or "THE X" where PRON is a pronoun[3], and X is a word appearing in the entity's string (allowing, e.g., *John*, or *the monument*). We also allow referring to its entity with its entire associated string. We restrict the set of allowed pronouns for each entity according to its type (male, female, plural-animate, unknown-animate, inanimate).[4] We then take, for each entity mention individually, the referring expression that receives the best language model score in context, using a strong unsupervised neural LM (BERT (Devlin et al., 2018)). The system is guaranteed to be correct in the sense that it will not generate wrong pronouns. It also has failure modes: it is possible for the system to generate ambiguous referring expressions (e.g., *John is Bob's father. He works as a nurse.*), and may lexicalize *Boston University* as *Boston*. We find that the second kind of mistake is rare as it is handled well by the language model. It can also be controlled by manually restricting the set of possible referring expression to each entity. Similarly, it is easy to extend the system to support other lexicalizations of entities by extending the sets of allowed lexicalizations (for example, supporting abbreviations, initials or nicknames) either as user-supplied inputs or using heuristics.

7 Evaluation and Results

We evaluate each of the introduced components separately. Tables listing their interactions are available in the appendix. The appendix also lists some qualitative outputs. The main trends that we observe are:

- The new planner causes a small drop in BLEU, but is orders of magnitude faster (§7.1).
- Typing information causes a negligible drop in BLEU overall, but improves results substantially for the *unseen* portion of the dataset (§7.2).
- The verification step is effective at improving the faithfulness of the output, practically eliminating omitted and overgenerated facts, reducing the number of wrong facts, and increasing the number of correctly expressed facts. This is based on both manual and automatic evaluations. (§7.3).
- The referring expression module is effective, with an intrinsic correctness of 92.2%. It substantially improves BLEU scores. (§7.4).

Setup We evaluate on the WebNLG dataset (Colin et al., 2016), comparing to the step-by-step systems described in Moryossef et al. (2019), which are state of the art. Due to randomness inherent in neural training, our reported automatic evaluation measures are based on an average of 5 training runs of each system (neural planner and neural realizer), each run with a different random seed.

7.1 Neural Planner vs Exhaustive Planner

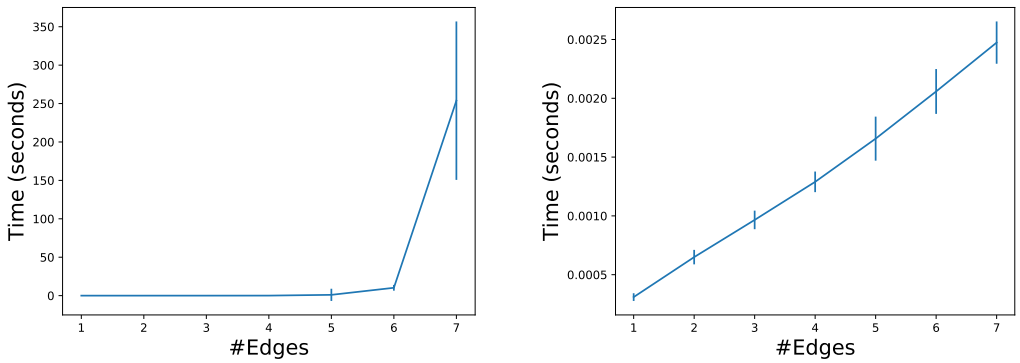
We compare the exhaustive planner from Moryossef et al. (2019) to our neural planner, by replacing the planner component in the Moryossef et al. (2019) system. Moving to the neural planner exhibits a small drop in BLEU (46.882 dropped to 46.506). However, figure 1 indicates 5 orders of magnitude (100,000x) speedup for graphs with 7 edges, and a linear growth in time for number of edges compared to exponential time for the exhaustive planner.

7.2 Effect of Type Information

We repeat the coverage experiment in (Moryossef et al., 2019), counting the number of output texts that contain all the entities in the input graph, and, of these text, counting the ones in which the entities appear in the exact same order as the plan. Incorporating typing information reduced the number of texts not containing all entities by 18% for the *seen* part of the test set, and 16% for the unseen part. Moreover, for the text containing all

[3] One of *he, his, him, himself, she, her, hers, herself, they, them, theirs, it, its, itself.*

[4] We extract the types from DBPedia pages for the entities. In case we cannot deduce a type, we do not allow any pronoun.



(a) Exhaustive Planner (b) Neural Planner

Figure 1: Average (+std) planning time (seconds) for different graph sizes, for exhaustive vs neural planner.

	Moryosef et al StrongNeural	Moryosef et al BestPlan	Exhaustive +Verify	Neural +Verify
Expressed	360	417	**426**	405
Omitted	41	6	**0**	2
Wrong	39	17	**14**	30
Over-generation	29	3	**0**	4
Wrong REG	-	-	**0**	3

Table 1: Manual correctness analysis comparing our systems with the ones from Moryossef et al. (2019).

entities, the number of texts that did not follow the plan's entity order is reduced by 46% for the *seen* part of the test set, and by 35% for the *unseen* part. We also observe a small drop in BLEU scores, which we attribute to some relations being verbalized more freely (though correctly).

7.3 Effect of Output Verification

The addition of output verification resulted in negligible changes in BLEU, reinforcing that automatic metrics are not sensitive enough to output accuracy. We thus performed manual analysis, following the procedure in Moryossef et al. (2019). We manually inspect 148 samples from the seen part of the test set, containing 440 relations, counting expressed, omitted, wrong and over-generated (hallucinated) facts.[5] We compare to the **StrongNeural** and **BestPlan** systems from Moryossef et al. (2019). Results in Table 1 indicate that the effectiveness of the verification process in ensuring correct output, reducing the already small number of ommited and overgenerated facts to 0 (with the exhaustive planner) and keeping it small (with the fast neural planner).

[5]A wrong fact is one in which a fact exists between the two entities, but the text implies a different fact from the graph, while over-generated is either repeating facts or inventing new facts.

7.4 Referring Expression Module

Intrinsic evaluation of the REG module. We manually reviewed 1,177 pairs of entities and referring expressions generated by the system. We find that 92.2% of the generated referring expressions refer to the correct entity.

From the generated expressions, 325 (27.6%) were pronouns, 192 (16.3%) are repeating a one-token entity as is, and 505 (42.9%) are generating correct shortening of a long entity. In 63 (5.6%) of the cases the system did not find a good substitute and kept the entire entity intact. Finally, 92 (7.82%) are wrong referrals. Overall, 73.3% of the non-first mentions of entities were replaced with suitable shorter and more fluent expressions.

Effect on BLEU scores. As can be seen in Table 2, using the REG module increases BLEU scores for both the exhaustive and the neural planner.

	-	REG
Exhaustive Planner	46.882	47.338
Neural Planner	46.506	47.124

Table 2: Effect of the REG component on BLEU score

8 Conclusions

We adopt the planning-based neural generation framework of Moryossef et al. (2019) and extend it to be orders of magnitude faster and produce more correct and more fluent text. We conclude that these extensions not only improve the system of Moryossef et al. (2019) but also highlight the flexibility and advantages of the step-by-step framework for text generation.

Acknowledgements

This work was supported in part by the German Research Foundation through the German-Israeli Project Cooperation (DIP, grant DA 1600/1-1) and by a grant from Reverso and Theo Hoffenberg.

References

Shubham Agarwal, Marc Dymetman, and Eric Gaussier. 2018. Char2char generation with reranking for the e2e nlg challenge. *arXiv preprint arXiv:1811.05826*.

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

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*.

Thiago Castro Ferreira, Diego Moussallem, Ákos Kádár, Sander Wubben, and Emiel Krahmer. 2018. Neuralreg: An end-to-end approach to referring expression generation. *arXiv preprint arXiv:1805.08093*.

Albert Gatt and Emiel Krahmer. 2017. Survey of the state of the art in natural language generation: Core tasks, applications and evaluation. *CoRR*, abs/1703.09902.

Chloé Kiddon, Luke Zettlemoyer, and Yejin Choi. 2016. Globally coherent text generation with neural checklist models. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 329–339.

Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M Rush. 2017. Opennmt: Open-source toolkit for neural machine translation. *arXiv preprint arXiv:1701.02810*.

Vladimir I Levenshtein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710.

Tasnim Mohiuddin and Shafiq Joty. 2019. Revisiting adversarial autoencoder for unsupervised word translation with cycle consistency and improved training. *arXiv preprint arXiv:1904.04116*.

Amit Moryossef, Yoav Goldberg, and Ido Dagan. 2019. Step-by-step: Separating planning from realization in neural data-to-text generation. *arXiv preprint arXiv:1904.03396*.

Joakim Nivre and Ryan McDonald. 2008. Integrating graph-based and transition-based dependency parsers.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. 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.

Ehud Reiter and Robert Dale. 2000. *Building natural language generation systems*. Cambridge university press.

Marilyn A Walker, Amanda Stent, François Mairesse, and Rashmi Prasad. 2007. Individual and domain adaptation in sentence planning for dialogue. *Journal of Artificial Intelligence Research*, 30:413–456.