How Helpful is Inverse Reinforcement Learning for Table-to-Text Generation?

Sayan Ghosh[†*] **Zheng Qi**[‡*] **Snigdha Chaturvedi**[†] **Shashank Srivastava**[†]
† UNC Chapel Hill
‡ University of Pennsylvania
{sayghosh, snigdha, ssrivastava}@cs.unc.edu
issacqzh@seas.upenn.edu

Abstract

Existing approaches for the Table-to-Text task suffer from issues such as missing information, hallucination and repetition. Many approaches to this problem use Reinforcement Learning (RL), which maximizes a single manually defined reward, such as BLEU. In this work, we instead pose the Table-to-Text task as Inverse Reinforcement Learning (IRL) problem. We explore using multiple interpretable unsupervised reward components that are combined linearly to form a composite reward function. The composite reward function and the description generator are learned jointly. We find that IRL outperforms strong RL baselines marginally. We further study the generalization of learned IRL rewards in scenarios involving domain adaptation. Our experiments reveal significant challenges in using IRL for this task.

1 Introduction

Table-to-Text generation focuses on explaining tabular data in natural language. This is increasingly relevant due to the vast amounts of tabular data created in domains including e-commerce, healthcare and industry (for example, infoboxes in Wikipedia, tabular product descriptions in online shopping sites, etc.). Table-to-Text can make data easily accessible to non-experts and can automate certain pipelines like auto-generation of product descriptions. Traditional methods approached the general problem of converting structured data to text using slot-filling techniques (Kukich, 1983; Reiter and Dale, 2000; McKeown, 1992; Cawsey et al., 1997; Konstas and Lapata, 2013; Flanigan et al., 2016). While recent advances in data-to-text generation using neural networks (Sutskever et al., 2011; Mei et al., 2015; Gardent et al., 2017; Wiseman et al., 2017; Song et al., 2018; Zhao et al., 2020) have led to improved fluency, current systems still suffer from issues such as lack of coverage (where the generated text misses information present in the source), repetition (where the generated text repeats information) and hallucination (where the generated text asserts information not present in the source)(Lee et al., 2019). A significant reason for these issues is that models often lack explicit inductive biases to avoid these problems. Most extant approaches utilize Reinforcement Learning-based (RL) training, using a single reward (such as BLEU or task-specific rewards) that optimizes for a specific aspect. For example, Liu et al. (2019) and Nishino et al. (2020) use domain-specific rewards to improve the accuracy of medical report generation.

However, defining a single reward that addresses all of the above-described issues is difficult. To use multiple reward components with RL, one has to manually find an optimal set of weights of each component either through a trial-and-error approach or expensive grid search which gets infeasible as the number of such reward components increases. Inverse Reinforcement Learning (Abbeel and Ng, 2004; Ratliff et al., 2006; Ziebart et al., 2008) can be a natural approach for this task since it can learn an underlying composite reward function from labeled examples incorporating multiple rewards. Motivated by existing applications of IRL in other domains and tasks (Finn et al., 2016; Fu et al., 2017; Shi et al., 2018), we explore its utility for Table-to-Text generation. We diverge from previous work on IRL in designing a set of intuitive and interpretable reward components that are linearly combined to get the reward function. Figure 1 illustrates the overall idea of this work. We learn a "Description Generator" (also referred as policy later) to generate descriptions given a table. The IRL framework includes "Reward Approximator" that leverages the "expert" or the ground-truth de-

[*] Authors contributed equally.

Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Short Papers), pages 71–79
August 1–6, 2021.

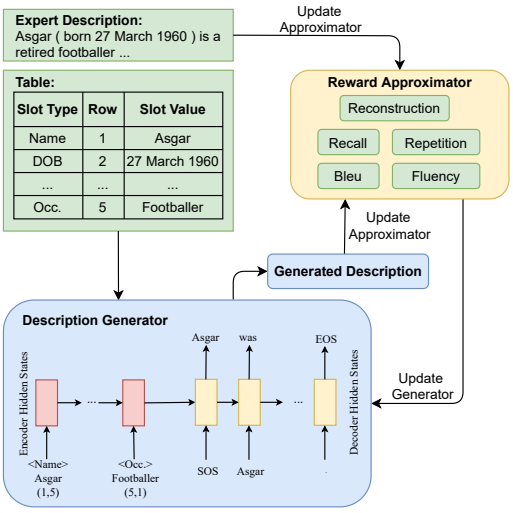

Figure 1: We frame the Table-to-Text task under Inverse Reinforcement Learning framework using multiple reward components

scriptions corresponding to tables to jointly learn the underlying composite reward function combining multiple reward components such as "Recall", "Fluency", etc. This composite reward function quantifies the quality of the generated descriptions. We see IRL performs at par with RL baselines. For investigating when IRL helps and when it does not, we conduct experiments to evaluate generalization capabilities of IRL in limited data setting and identify challenges involved in IRL training. Our contributions are:

- We formulate a set of interpretable reward components and learn the composite linear reward function in a data-driven manner for Table-to-Text generation[1].
- We study the utility of IRL for Table-to-Text generation.

2 Method

The training data for this task consists of pairs of tables and corresponding natural language descriptions, as shown in Figure 1. A table T is a sequence of tuples of slot types (e.g. "Name") and slot values (e.g. "Asgar") and let D denote the expert description. We formulate the "Table-to-Text" task as generating D from source table T. In the rest of this section, we first explain how to formulate Table-to-Text under the IRL framework, followed by the formulation of the reward components and a brief description of the text generation network that is at the core of our method.

2.1 Table-to-Text as IRL

We pose Table-to-Text under the IRL framework where we aim to jointly learn a policy for generating description from the table and the underlying composite reward function. At the core of our approach, we have a neural description generator that we adapt from Wang et al. (2018). The description generator is first trained using maximum likelihood estimation (MLE) followed by fine-tuning it using Maximum Entropy Inverse Reinforcement Learning (MaxEnt IRL) (Ziebart et al., 2008). Under the MaxEnt IRL framework, we iteratively perform two steps: (1) approximate the underlying composite reward function by leveraging the expert descriptions and the current policy for description generation; (2) Using the updated reward function, we update the current policy for description generation using RL. In this work, we model the composite reward $R_\phi(D)$ as a linear combination of multiple reward components.

$$R_\phi(D) = \sum_{t=1}^{\tau} \phi^\intercal C^t \tag{1}$$

where ϕ is a weight vector, C^t is the vector of reward component values at step t in a generated description and τ denotes total steps.

Following the MaxEnt IRL paradigm, we assume the expert descriptions come from a log-linear distribution ($p_\phi(D)$) on reward values. The objective of the reward approximator ($J_r(\phi)$) is to maximize the likelihood of the expert descriptions. The partition function for this distribution ($p_\phi(D)$) is approximated by using importance sampling from the learned description generation policy. For sake of brevity, we skip the mathematical derivation here. Please refer to Appendix A.1 for detailed derivation. We draw N expert descriptions and M descriptions from the learned policy. The gradient of the objective ($J_r(\phi)$) w.r.t. reward function parameters ϕ is then the difference between the expected expert reward and expected reward obtained by the policy (Ziebart et al., 2008):

$$\nabla_\phi J_r(\phi) = \frac{1}{N}\sum_{i=1}^{N} \nabla_\phi R_\phi(D_i) - \frac{1}{\sum_j \beta_j}\sum_{j=1}^{M} \beta_j \nabla_\phi R_\phi(D'_j) \tag{2}$$

where D_i and D'_j are drawn from the training data and the learned policy respectively and β's are importance sampling weights.

[1]Code and dataset splits for the paper are provided in https://github.com/issacqzh/IRL_Table2Text

The linear functional form of the reward simplifies individual weight updates as a simple difference of the expected expert and the expected roll out reward component from policy. Weight update for component c is:

$$\nabla_\phi J_r(\phi)_c = \frac{1}{N}\sum_{i=1}^{N} c_i - \frac{1}{\sum_j \beta_j}\sum_{j=1}^{M} \beta_j c'_j \quad (3)$$

where c_i is total value of reward component over all steps for i^{th} expert description and c'_j is total value of reward component over all steps for j^{th} generated description. To stabilize training when learning the policy for description generation we mix in weighted MLE loss with the policy gradient loss before backpropagation. Please refer to supplementary material (Appendix A.5) for model training details.

2.2 Reward Components

We aim to find a reward function that can combine multiple characteristics present in a good description such as faithfulness to the table and fluency. To encourage faithfulness, we use *recall* and *reconstruction* as reward components, while to characterize grammatical correctness and fluency we use *repetition* and *perplexity*. We also consider *BLEU* score as a reward component. BLEU is a supervised reward component as it requires ground-truth descriptions for its computation. However, all other reward components are unsupervised.

- **Recall:** Fraction of slot values in the table mentioned in the description.
- **Reconstruction:** We use QA models to extract answers from the description against a few "extractor" slot types (for example, "What is the name of the person in the description?" is used as a question for the slot type "Name_ID"). Details about other extractor slot types are provided in Appendix A.3. Reconstruction score is the average of lexical overlap scores between predicted and true slot values, corresponding to the extractor slot types present in the table.
- **Repetition:** Fraction of unique trigrams in the description.
- **Perplexity:** This is the normalized perplexity of the description calculated using GPT-2 model (Radford et al., 2019).
- **BLEU:** This is the BLEU score (Papineni et al., 2002) of the description.

Additional details on implementation of reward components are in Appendix A.3.

3 Experiments and Results

In this section we describe our experiments and their results in detail.

3.1 Data and Metrics

Wang et al. (2018) proposed a dataset of tables and their corresponding descriptions related to people and animals from Wikipedia. However, the original released dataset is noisy (many descriptions have low precision/recall, most examples have very few distinct slot types, etc.). For our experiments, we filtered this dataset to get a smaller high-quality dataset of 4623 examples using the following criteria : (1) Recall (defined in §2.2) of 1.0 (2) High precision (fraction of entities in the description mentioned in the table) greater than 0.7 (3) number of distinct slot types greater than 6. We split the entire dataset as 80%, 10% and 10% for training, validation and testing respectively. Details of the dataset are provided in Appendix A.2. To aid reproducibility we make the data splits used by us publicly available[2].

For evaluation, we report BLEU (Papineni et al., 2002) and ROUGE (Lin, 2004) along with their harmonic mean (called F1 hereon). Additionally, we report the mean reward value for Recall and Perplexity as proxies for faithfulness and fluency of generated descriptions respectively.

MODEL	B	R	F1	REC.	PPL
MLE	23.78	42.11	30.40	0.82	-73.38
RL with					
B	28.03	43.75	34.17	0.87	-42.17
Rec., B	28.02	43.77	34.16	0.88	-39.43
Rec., B, PPL	28.22	43.12	34.11	0.89	-43.91
All	28.21	43.23	34.14	0.89	-40.77
IRL with					
Rec., B	27.96	43.52	34.04	0.88	-40.27
Rec., B, PPL	28.25	**43.81**	34.35	0.89	-40.19
All	**28.42**	43.19	34.28	0.89	-40.11
IRL (using multipliers) with					
Rec., B	28.41	43.53	**34.38**	0.89	**-38.53**
Rec., B, PPL	28.16	43.35	34.14	**0.90**	-40.86

Table 1: Test set performance for various models measured using BLEU (B), ROUGE(R), F1, Recall(Rec.) and Perplexity (PPL). Using IRL instead of RL gives marginal improvement in performance

3.2 Automatic Evaluation

Table 1 shows the performance of models trained using maximum likelihood estimation (MLE), RL

[2]Data splits are provided in https://github.com/issacqzh/IRL_Table2Text

and IRL. For RL and IRL we report results with various sets of reward components. When using multiple reward components with RL we consider the total reward as the uniformly weighted sum of each component. We note that while IRL variants achieve higher performance than RL methods for all metrics, the gain in performance is marginal.

In Table 1 we choose the best model for each setting based on the performance on the validation split. For the best IRL (All) model, we find the learned weights for repetition, recall, BLEU, reconstruction and perplexity are 0.02, 0.12, 0.65, 0.05 and 0.15 respectively. However, we noticed that the weights of the IRL reward components failed to converge in our training runs. This is a consequence of the fact that reward components such as BLEU achieve their maximum value for the ground-truth description, and the value quickly drops as descriptions diverge from the ground truth description. Thus the gap between the expert value and the value achieved by the model for BLEU is always large, hindering the convergence of weights in IRL (Eqn 3). This results in a peaked distribution of weights where the model tends to favor the BLEU reward component excessively. We attempt scaling down the expert BLEU reward values by using multiplier. We dynamically update the multiplier using an adaptive binary search method (refer to Appendix A.4 for details) to induce convergence in weights. We observe that the multiplier acts as a "regularizer" in learning a more balanced weight for the reward components considered. For example, when we train IRL with BLEU, recall and perplexity without using multiplier, the learned weights of the components are 0.72, 0.15 and 0.13 respectively. On using multipliers for IRL training, the learned weights for BLEU, recall and perplexity are 0.45, 0.31 and 0.24 respectively. The model variants using multipliers get the best F1 score as seen in the second last row of Table 1.

We also find that having more reward components does not help IRL improve significantly. We note that IRL using all reward components gets the best BLEU but suffers a marginal drop in ROUGE.

3.3 Domain adaptation

To evaluate if rewards learned using IRL generalize better to unseen data distributions, we evaluate it for scenarios involving domain adaptation. For this, we divide the dataset into disjoint subsets of categories involving people in sports, academia, art, etc. (category details in Appendix A.2). Each category has different table schemas. We train RL and IRL models on one category and test them on a different category. Since training on a single category limits the amount of labelled data, we consider training with unsupervised rewards that do not rely on the ground truth. Table 2 shows the F1 results when using IRL and RL with recall, perplexity and reconstruction. For each training category, we show results of the test category with the highest absolute value of relative change in F1. We notice mixed results. For instance, when training on the "Sports" domain, IRL's performance is much worse than RL. This may be because slot types with high frequency in the "Sports" category are significantly different from all other categories. Thus, IRL may be susceptible to learning a reward function that overfits the domain and actually generalizes worse than a fixed reward function. However, in several cases IRL leads to big improvements in performance (e.g. when training on Politics, Law, and Military) indicating the promise of this method in limited data settings.

TRAIN CAT.	TEST CAT.	RL	IRL
Politics	Sports	18.59	**21.04**
Law	Academia	28.54	**31.17**
Military	Politics	30.07	**32.01**
Art	Academia	**32.78**	31.37
Academia	Sports	**21.25**	20.67
Sports	Academia	**24.43**	22.25

Table 2: F1 scores on using IRL and RL for domain adaptation. IRL leads to higher F1 scores in a few settings indicating its usefulness for domain adaptation. However, IRL performs worse than RL when trained on domains which have significantly different slot types with high frequency (e.g. "Sports").

4 Discussion

We highlight some challenges with IRL training that potentially hinder IRL to get significantly better than RL baselines. Further, we discuss qualitative differences between RL and IRL models.

4.1 Challenges in IRL training

Importance of reward components: During training, for most reward components, their values for expert and generated descriptions are close. However, the values of BLEU for generated descriptions are quite smaller than the BLEU value for expert descriptions. This shadows the contribution of other reward components irrespective of

the weights assigned to them. Since BLEU optimizes for n-gram overlap with the expert text, it is undesirable to drop this component as it leads to text degeneration. As described in Section 3.2, we use adaptive multipliers to alleviate its dominance. However, its effect is limited and the method does not correspond to optimizing a fixed objective.

Unstable training: To stabilize training, we mix the weighted MLE loss (cross-entropy loss) and the policy gradient objective. However, these losses can differ largely in scale. Having a larger weight to MLE loss diminishes the contribution of reward components, while larger weight to policy gradient leads to degeneration.

These observations indicate the need for future research on training paradigms and better-designed reward components to address these challenges.

4.2 Qualitative analysis

Using only BLEU as a reward leads to generated descriptions that fit a general template resembling descriptions from the most common category ("Sports"). Including other reward components helps the model avoid this behavior. We still observe hallucination from both IRL and RL fine-tuned models. However, hallucinated information generated from IRL fine-tuned models often matches the overall theme (for example, it generates incorrect football league names but gets the name of the club mentioned in the table correct). Appendix A.7 shows an example of description generated by IRL (All) model.

5 Conclusion

We present an approach using IRL for Table-to-Text generation using a set of interpretable reward components. While the approach outperforms RL, improvements are marginal, and we identify several challenges. In particular, using metrics like BLEU as reward components is problematic, since they affect weight convergence for IRL. Based on our study, the application of IRL for Table-to-Text generation would broadly benefit from designing better-calibrated reward components and improvements in training paradigms. We hope our exploration encourages the community to engage in interesting directions of future work.

References

Pieter Abbeel and Andrew Y Ng. 2004. Apprenticeship learning via inverse reinforcement learning. In Proceedings of the twenty-first international conference on Machine learning, page 1.

Alison J Cawsey, Bonnie L Webber, and Ray B Jones. 1997. Natural language generation in health care.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. 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 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Chelsea Finn, Paul Christiano, Pieter Abbeel, and Sergey Levine. 2016. A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models. arXiv preprint arXiv:1611.03852.

Jeffrey Flanigan, Chris Dyer, Noah A Smith, and Jaime G Carbonell. 2016. Generation from abstract meaning representation using tree transducers. In Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 731–739.

Justin Fu, Katie Luo, and Sergey Levine. 2017. Learning robust rewards with adversarial inverse reinforcement learning. arXiv preprint arXiv:1710.11248.

Claire Gardent, Anastasia Shimorina, Shashi Narayan, and Laura Perez-Beltrachini. 2017. Creating training corpora for nlg micro-planning. In 55th annual meeting of the Association for Computational Linguistics (ACL).

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Ioannis Konstas and Mirella Lapata. 2013. A global model for concept-to-text generation. Journal of Artificial Intelligence Research, 48:305–346.

Karen Kukich. 1983. Design of a knowledge-based report generator. In 21st Annual Meeting of the Association for Computational Linguistics, pages 145–150.

Katherine Lee, Orhan Firat, Ashish Agarwal, Clara Fannjiang, and David Sussillo. 2019. Hallucinations in neural machine translation.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In Text summarization branches out, pages 74–81.

Guanxiong Liu, Tzu-Ming Harry Hsu, Matthew McDermott, Willie Boag, Wei-Hung Weng, Peter Szolovits, and Marzyeh Ghassemi. 2019. Clinically accurate chest x-ray report generation. In Machine Learning for Healthcare Conference, pages 249–269. PMLR.

Kathleen McKeown. 1992. Text generation. Cambridge University Press.

Hongyuan Mei, Mohit Bansal, and Matthew R Walter. 2015. What to talk about and how? selective generation using lstms with coarse-to-fine alignment. arXiv preprint arXiv:1509.00838.

Toru Nishino, Ryota Ozaki, Yohei Momoki, Tomoki Taniguchi, Ryuji Kano, Norihisa Nakano, Yuki Tagawa, Motoki Taniguchi, Tomoko Ohkuma, and Keigo Nakamura. 2020. Reinforcement learning with imbalanced dataset for data-to-text medical report generation. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings, pages 2223–2236.

Travis E Oliphant. 2006. A guide to NumPy, volume 1. Trelgol Publishing USA.

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 of the Association for Computational Linguistics, pages 311–318.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In Advances in Neural Information Processing Systems 32, pages 8024–8035. Curran Associates, Inc.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. OpenAI blog, 1(8):9.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for SQuAD. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), pages 784–789, Melbourne, Australia. Association for Computational Linguistics.

Nathan D Ratliff, J Andrew Bagnell, and Martin A Zinkevich. 2006. Maximum margin planning. In Proceedings of the 23rd international conference on Machine learning, pages 729–736.

Ehud Reiter and Robert Dale. 2000. Building Natural Language Generation Systems. Studies in Natural Language Processing. Cambridge University Press.

Zhan Shi, Xinchi Chen, Xipeng Qiu, and Xuanjing Huang. 2018. Toward diverse text generation with inverse reinforcement learning. arXiv preprint arXiv:1804.11258.

Linfeng Song, Yue Zhang, Zhiguo Wang, and Daniel Gildea. 2018. A graph-to-sequence model for amr-to-text generation. arXiv preprint arXiv:1805.02473.

Ilya Sutskever, James Martens, and Geoffrey E Hinton. 2011. Generating text with recurrent neural networks. In ICML.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake Vand erPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods, 17:261–272.

Qingyun Wang, Xiaoman Pan, Lifu Huang, Boliang Zhang, Zhiying Jiang, Heng Ji, and Kevin Knight. 2018. Describing a knowledge base. In Proceedings of the 11th International Conference on Natural Language Generation, pages 10–21. Association for Computational Linguistics.

Sam Wiseman, Stuart M Shieber, and Alexander M Rush. 2017. Challenges in data-to-document generation. arXiv preprint arXiv:1707.08052.

Chao Zhao, Marilyn Walker, and Snigdha Chaturvedi. 2020. Bridging the structural gap between encoding and decoding for data-to-text generation. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 2481–2491, Online. Association for Computational Linguistics.

Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. 2008. Maximum entropy inverse reinforcement learning. In Aaai, volume 8, pages 1433–1438. Chicago, IL, USA.

A Appendices

A.1 Derivation of gradient for MaxEnt IRL

We show the detailed mathematical steps to approximate the gradient calculation under the MaxEnt IRL framework for Table-to-Text generation.

We assume that expert descriptions are drawn from a distribution $p_\phi(D)$.

$$p_\phi(D) = \frac{1}{Z} exp(R_\phi(D)) \text{ and } Z = \int_D exp(R_\phi(D)) \quad (4)$$

where the reward function, $R_\phi(D)$ has parameters ϕ, and Z is the partition function. The total reward

of a description is sum of rewards at each step. Let $q_\theta(D)$ be the policy for description generation. We maximise the log-likelihood of the samples in the training set (Equation 5).

$$J_r(\phi) = \frac{1}{N}\sum_{n=1}^{N} log(p_\phi(D_n)) = \frac{1}{N}\sum_{n=1}^{N} R_\phi(D_n) - logZ \tag{5}$$

The gradient w.r.t. reward parameters is given by

$$\begin{aligned}\nabla_\phi J_r(\phi) &= \frac{1}{N}\sum_{n} \nabla_\phi R_\phi(D_n) \\ &\quad - \frac{1}{Z}\int_D exp(R_\phi(D))\nabla_\phi R_\phi(D) dD \\ &= \mathbb{E}_{D\sim p_{data}} \nabla_\phi R_\phi(D) - \mathbb{E}_{D\sim p_{\phi(D)}} \nabla_\phi R_\phi(D)\end{aligned} \tag{6}$$

The partition function requires enumerating all possible descriptions which makes this intractable. This is tackled by approximating the partition function by sampling descriptions from the policy using importance sampling. The importance weight β_i for a generated description D_i is given by

$$\beta_i \propto \frac{exp(R_\phi(D_i))}{q_\theta(D_i)} \tag{7}$$

The gradient is now approximated as:

$$\nabla_\phi J_r(\phi) = \frac{1}{N}\sum_{i=1}^{N} \nabla_\phi R_\phi(D_i) - \frac{1}{\sum_j \beta_j}\sum_{j=1}^{M} \beta_j \nabla_\phi R_\phi(D'_j) \tag{8}$$

where D_i and D'_j are drawn from training data and $q_\theta(D)$ respectively.

A.2 Dataset statistics

We split the entire dataset as 80%, 10% and 10% for training, validation and testing respectively. Table 3 shows the statistics for our dataset.

Table 4 shows the various disjoint category splits of our data.

A.3 Detailed description of some reward components

- **Reconstruction:** We use Question Answering models to extract answers from the description corresponding to few slot types. For example, to extract the name from the description we ask a question "What is the name of the person?". The questions corresponding to each slot type is pre-determined. We extract values for four most common slot types occurring in the dataset – "name", "place of birth", "place of death" and "country". We will refer to these slots as "extraction slot types". The questions for these extractor slot types are "What is the name of the person in the description?", "What is the place of birth of the person in the description?", "What is the place of death of the person in the description?" and "Which country does the person in the description belong to?" respectively. All extraction slot types are not present in every table of the dataset (example, "place of death" is not present for a living sportsperson). Following SQUAD-like (Rajpurkar et al., 2018) formalisation, for each slot-type we train a BERT-based (Devlin et al., 2019) model to get the answer from the description given the question. We calculate overlap score of predicted answer with the correct answer (slot value from table). The final reconstruction score is the arithmetic mean of answer overlap scores corresponding to the extractor slot types present in the table.
- **Perplexity:** This is the negative perplexity of the description. We further normalize it by using

$$\frac{\text{Perplexity} - \text{Perplexity}_{low}}{\text{Perplexity}_{high} - \text{Perplexity}_{low}} \tag{9}$$

where Perplexity_{high} and Perplexity_{low} are the maximum and minimum perplexity of expert texts and texts generated by pretrained MLE model respectively.

A.4 Learning Multiplier for BLEU

Let us assume that after the i^{th} iteration of IRL, we have the multiplier value as m_i. Let b be the average BLEU score obtained by the model. For $(i+1)^{th}$ iteration we update the multiplier value as

$$m_{i+1} = \frac{m_i + b}{2} \tag{10}$$

In case the change in weight is less than 0.00001, we instead increase multiplier value by 0.1. The maximum of multiplier value is 1. We start with initial multiplier value (m_0) as 1.

A.5 Training details

Model parameters We follow the same training scheme and model parameters from Wang et al. (2018). Our model roughly has around 7.8M parameters. We perform MLE for 20 epochs. For RL finetuning we perform 100 epochs. For the IRL

TYPE	SIZE	REC.	PREC.	# SENT./TAB.	# SLOTS/SENT.	# SLOTS/TAB.	# W/SENT.	# W/TAB.
Train	3700	1.0	0.82	4.61	1.86	8.58	14.85	68.52
Val	461	1.0	0.82	4.67	1.86	8.70	15.10	70.54
Test	462	1.0	0.82	4.60	1.86	8.57	14.54	66.87
Total	4623	1.0	0.82	4.62	1.86	8.59	14.85	68.56

Table 3: Dataset statistics

Slot Type	Row	Slot Value
Name ID	1	William Duval (ice hockey)
Country of citizenship	2	Canada
Date of birth	3	August 3, 1877
Date of death	4	June 7, 1905
Sport	5	Ice hockey
Position played on team / Specialty	6	Defenceman
Place of birth	7	Ottawa

Reference:
William Duval (ice hockey) (August 3 1877 – June 7 1905) was a Canadian professional Ice hockey Defenceman who played for the Ottawa Hockey Club and the Pittsburgh Victorias in the late 1890s and early 1900s . born in Ottawa Canada Duval played intermediate hockey for the Ottawa Aberdeens and Ottawa Atlantic Railway teams before joining the Ottawa Hockey Club in the 1899 – 1900 season . he played two further seasons for Ottawa and was named captain prior to the 1902 season . duval died due to alcoholism on June 7 1905 . duval had previously worked for the Canada Atlantic Railway in Ottawa .

IRL All:
William Duval (ice hockey) (August 3 1877 - June 7 1905) was a Canada professional Ice hockey Defenceman who played eleven seasons in the National Hockey League of six . he was born in Ottawa Ontario Canada .

Figure 2: Example of generated description using IRL (All) model

CATEGORY	# SAMPLES
Academia	2152
Art	4736
Politics	2974
Sports	17434
Law	586
Military	4170
Unknown	14096
All	46148

Table 4: Data statistics for categories

model, we perform two weight updates followed by five RL epochs and this is repeated 20 times. For training we use Adam optimizer (Kingma and Ba, 2014). We choose the hyperparameters and best epoch for each model by obtaining results on the validation set using beam search with beam size of 3.

Hyper-parameter tuning We adapt the model and optimizer hyper-parameters from Wang et al. (2018). For choosing the weights for cross-entropy loss and policy gradient loss we tried combinations in the set 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9 keeping sum of weights as 1. For IRL reward component weight updates we sample 500 descriptions from ground-truth and from the descriptions generated from the policy. We chose the size as 500 based on validation set performance. Based on the performance on the validation set we chose 0.9 as policy gradient loss weight and 0.1 for cross-entropy loss. This also helps to bring both the loss terms in same scale.

Software and hardware specifications All the models are coded using Pytorch 1.4.0[3] (Paszke et al., 2019) and related libraries like numpy (Oliphant, 2006), scipy (Virtanen et al., 2020) etc. We run all experiments on GeForce RTX 2080 GPU of size 12 GB. The system has 256 GB RAM and 40 CPU cores.

Time for training and inference It takes around 16 seconds for one epoch of MLE training while it takes close to 150 seconds for an epoch when using RL fine-tuning with all the reward components. The reward component weight approximation stage of IRL is very fast and takes less than a second generally.

A.6 Validation set results

Table 5 shows the results on validation set for the models in Table 1 of main paper.

A.7 Qualitative example

Table 2 shows an example of the output generated by the IRL (All) model along with the reference

[3] https://pytorch.org/

MODEL	B	R	F1	REC.	PPL
MLE	23.64	41.17	30.03	0.83	-74.75
RL with					
B	26.61	**42.12**	32.61	0.86	-39.89
Rec., B	26.88	42.05	32.79	0.87	**-34.06**
Rec., B, PPL	27.10	41.72	32.86	0.88	-43.47
All	26.87	41.87	32.73	0.88	-39.14
IRL with					
Rec., B	26.85	41.79	32.69	0.87	-37.40
Rec., B, PPL	27.09	42.10	**32.97**	0.88	-40.13
All	**27.23**	41.70	32.94	0.88	-39.87
IRL (using multipliers) with					
Rec., B	27.02	42.00	32.88	0.86	-34.83
Rec., B, PPL	27.02	41.67	32.78	**0.89**	-40.87

Table 5: Performance on the validation set for various models measured using BLEU (B), ROUGE(R), F1, Recall(Rec.) and Perplexity(PPL)

description.