Language Modeling through Inverse Reinforcement Learning

Ronak Mehta, Cailin Winston, Peter Michael
Department of Computer Science, Department of Statistics
University of Washington, Seattle
ronakdm, cailinw, petermic

December 30, 2020

1 Introduction

Recent work has highlighted the nontrivial connection between inverse reinforcement learning (IRL) and generative modelling (GM), two subfields of machine learning that were previously thought to be disparate Finn et al. [2018], Shi et al. [2018], Ho and Ermon [2016]. In the GM problem, the agent is given samples $x^{(1)}, ..., x^{(n)} \sim p$ of a random variable and is tasked with generating new samples from the distribution. For high-dimensional, structured random variables such as images, text, and audio, the current state-of-the-art performance is held by variants of Generative Adversarial Networks (GANs) Goodfellow et al. [2014]. However, GANs are notoriously difficult to optimize stably Salimans et al. [2016]. Sequential data, such as text, pose additional challenges for generation, as samples are dependent, warranting a new GM framework for this kind of data.

In the IRL problem, on the other hand, an agent is given multiple trajectories (sequences) of state-action pairs, $\{(s_1^{(i)}, a_1^{(i)}), ..., (s_T^{(i)}, a_T^{(i)})\}_{i=1}^n$, from which to infer the Markov Decision Process (MDP) that these trajectories attempt to solve. After this, a planning algorithm can solve for the optimal policy from the estimated MDP. Such an approach aids generalization by imposing additional structure on the types of policies to be learned; however, a large amount of "expert" trajectories (which may be expensive and noisy) may be needed to learn an effective policy Arora and Doshi [2019].

Given the promises and challenges of both of these areas of study, text generation presents a fruitful opportunity to leverage the connection between them. Specifically, text generation can be considered an MDP, in which states are represented by words (or an p-gram of words), actions can represent the next word, policies are stochastic, and an unknown reward function encourages realistic sounding sentences. The structure of the MDP can promote generalizable policies for sampling high-dimensional, sparse sequences of states, while the troves of publicly available text data acts as trajectories for realistic language. Finally, the learned reward of IRL can be used to train any of the successful reinforcement learning (RL) algorithms on a new text dataset of interest.

2 Background and Related Work

Common approaches to generative modelling include variational autoencoders (VAEs) and GANs [Kingma and Welling, 2014, Goodfellow et al., 2014, Finn et al., 2018]. For sentence generation,

these can be augmented with autoregressive models, possibly given some starter words. As mentioned before, training generative models can be challenging; GANs suffer from mode collapse, while VAEs can suffer from posterior collapse. Many of these approaches are based on neural network architectures that have been designed such that the number of parameters does not scale with the dimension of the input Goodfellow et al. [2016]. The recently popularized transformer architecture also achieves this property, by employing a self-attention mechanism that has achieve state-of-the-art performance in language tasks, and is trainable in a highly parallelized manner Vaswani et al. [2017].

Finn et al. [2016] presents a deep approach to inverse reinforcement learning, which is a variant of maximum entropy IRL, while Arora and Doshi [2019] outlines many classical approaches to the problem. Interestingly, Ho and Ermon [2016] attempts IRL by exploting a connection to GANs. While a theoretical connection between generative modeling and maximum entropy IRL has been established Finn et al. [2018], an extensive empirical understanding is in its early stages.

Shi et al. [2018] presents, to our knowledge, the only application of IRL to the language modelling problem. They present a maximum entropy IRL approach to text generation, which involves alternately training a reward approximator and a generator. While they implement the generator with an LSTM neural network, we aim to explore modeling the generator with a transformer-based network. Transformers provide many inherent advantages over LSTMs because of direct access to elements earlier in the sequence, allowing for more effective modeling of long-term dependencies, as well as the ability to be parallelized efficiently. This is a huge step-up from LSTMs, which can have issues modeling relationships in long sequences due to the vanishing gradients.

3 Approach

3.1 Problem Formulation

Here we formally describe the problem statement. We are given a vocabulary $\mathcal{V}$ of (string) words, including a `EOS` word that indicates "end-of-sentence". The words are represented as one-hot encoded vectors $w \in \mathcal{W} = \{0,1\}^{|\mathcal{V}|}$. A sentence of length T is represented by the sequence of words $x = (x_1, x_2, ..., x_T) \in \mathcal{X} = \cup_{T=1}^{\infty} \mathcal{W}^T$. Natural sentences are drawn from a fixed distribution $x \sim p$, from which we only have samples $x^{(1)}, ..., x^{(n)}$. Our goal, from a GM perspective, is to build a generator to sample new sentences x' from the same distribution p.

To apply IRL, we formulate text generation as an Markov Decision Process (MDP). The state space $\mathcal{S}$ is the hidden states of a pre-trained language model. The action space $\mathcal{A} = \mathcal{W}$, representing the next word in the sequence. The policy $\pi_\theta : \mathcal{S} \to \Delta^{|\mathcal{A}|-1}$ is stochastic, in that it maps the context to a distribution over $\mathcal{A}$, and samples the action from that distribution. The transition function is deterministic and can be disregarded. The reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ takes a context and a word, and assesses the realism or naturalness of the action word following the state context. The reward for a trajectory τ is the sum of the rewards of all the (s_t, a_t): $\mathcal{R}_\phi(\tau) = \sum_{t=1}^{T} r_\phi(s_t, a_t)$. While this reward function is unknown, we have a set of expert demonstrations (natural sentences) in a text corpus. Thus, the IRL goal is to learn this reward function using a neural function approximator for both the reward function and the optimal policy.

The goal then is both to learn a reward function that gives high reward to sentences that are similar to expert demonstrations and to learn a policy to generate sentences with high expected reward. We model both the reward function approximator and generation policy with neural networks.

3.2 Reward Approximator

The reward approximator learns to give high reward to realistic-sounding sentences and low reward to others. The objective then is to maximize the log-likelihood of samples from the training set of expert demonstrations, using Maximum Likelihood Estimation, and is as follows:

$$\begin{aligned}\mathcal{J}_r(\phi) &= \mathbb{E}_{\tau\sim p_{data}}[\mathcal{R}_\phi(\tau)] - \mathbb{E}_{\tau\sim q_\theta(\tau)}[\mathcal{R}_\phi(\tau)] \\ &= \frac{1}{N}\sum_{n=1}^{N}\mathcal{R}_\phi(\tau_n) - \log Z\end{aligned}$$

We use Maximum Likelihood Estimation to compute the expected reward of real texts and importance sampling to sample trajectories from the current policy (generator). The estimated gradient update is as follows, where $w_j = \frac{exp(\mathcal{R}_\phi(\tau_j))}{q_\theta(\tau_j)}$:

$$\nabla_\phi \mathcal{J}_r(\phi) = \frac{1}{N}\sum_{i=1}^{N}\nabla_\phi \mathcal{R}_\phi(\tau_i) - \frac{1}{\sum_{j=1}^{M} w_j}\sum_{j=1}^{M} w_j \nabla_\phi \mathcal{R}_\phi(\tau_j)$$

We use a multilayer perceptron to model the reward $r(s_t, a_t)$. It contains a single hidden layer, taking in the last hidden state of the generator concatenated to an embedding of the action taken at that state, and outputting the reward. The reward model learns this action embedding as well.

3.3 Text Generator

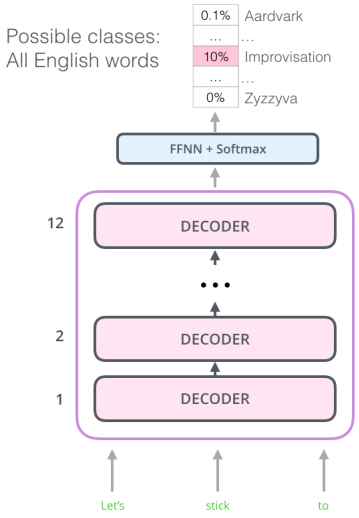
The generator learns to generate sequences that have high reward. The objective is entropy regularized policy gradient, as we want to maximize the expected reward of trajectories generated plus a regularization term.

$$\mathcal{J}_g(\theta) = \mathbb{E}_{\tau\sim q_\theta}[\mathcal{R}_\phi(\tau)] + \mathcal{H}(q_\theta(\tau))$$

The estimated gradient update is as follows, as per the policy gradient theorem:

$$\nabla_\phi \mathcal{J}_g(\theta) = \sum_t \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \ \log_\theta(a_t|s_t)\cdot(R_\phi(\tau_{t:T} - \log \pi_\theta(a_t|s_t) - 1)\right],$$

where $R_\phi(\tau_{t:T}$ is the reward to go. However, by itself this gradient estimate has prohibitively high variance, so applied Rao-Blackwellization and replaced $R_\phi(\tau_{t:T}$ with $r(s_t, a_t) + V(s_{t+1})$ (current reward plus value-to-go), which can be estimated with Monte Carlo rollouts. We also added gradient clipping to alleviate some of the variance issues.



We used the GPT-2 Radford et al. [2019] architecture for this task. GPT-2 is a Transformer architecture Vaswani et al. [2017] for language modeling. It contains only Transformer decoder blocks and is autoregressive. A figure displaying it is above. This model is trained on a huge corpus of text, over 40 gigabytes. In order to compare with the previous work, we wanted to change the vocabulary to that of the COCO captions dataset Chen et al. [2015]. We also don't want to retrain the whole network due to constrained compute resources. After all, the COCO dataset is in the same language as the dataset the network was trained on (English), so it is unnecessary to retrain the whole thing. What we did is two-fold:

(a) Change the output (softmax) layer to classify into the COCO vocabulary (change the number of output units to the new vocab size).

(b) Tokenize the data with the **GPT2 tokenizer** for input into the network, which allows us to utilize the model's learned embeddings and weights (this prevents retraining the whole network).

We performed pre-training to get the softmax layer up and running using the traditional cross-entropy loss. We then switched the the reinforcement learning objective to complete training.

3.4 Optimization Algorithm

The generator and rewarder objectives are optimized alternately. The rewarder model is trained on expert demonstrations (sequences drawn from our text corpus) and learner demonstrations (sequences generated by the generator), while the generator is trained on trajectories drawn from

the current policy. This is shown in Algorithm 1.

Algorithm 1: IRL for Text Generation

while *not converged* **do**
  for n_r *iterations* **do**
    Draw $\tau_1, \tau_2, ..., \tau_N \sim p_{data}$
    Draw $\tau'_1, \tau'_2, ..., \tau'_M \sim q_\theta$
    Update $\phi \leftarrow \phi + \alpha \nabla_\phi J_r(\phi)$
  end
  for n_g *epochs* **do**
    Draw $\tau_1, \tau_2, ..., \tau_N \sim q_\theta$
    Update $\theta \leftarrow \theta + \beta \nabla_\theta J_g(\theta)$
  end
end

4 Training

We used `PyTorch` for neural network implementations, the `HuggingFace` library for a pre-trained GPT2 model and tokenizer, and the `nltk` package for common language preprocessing and evaluation functions. We utilized the free Google Colaboratory GPU resources for training.

5 Results

We evaluated our model on the Image Caption COCO dataset Chen et al. [2015]. We used a dataset size of 80,000 sequences, and the vocabulary size is 4939. We used a sequence length of 32. We found that selected $n_r > n_g$ produced better results, as described in Shi et al. [2018]. The following list shows the hyperparameters that we selected.

(a) Number of IRL iterations: 100

(b) Number of Generator pretrain iterations: 120

(c) Embedding dimension: 768

(d) Action embedding dimension: 32

(e) Size of only hidden layer for MLP: 128

(f) Generator batch size: 32

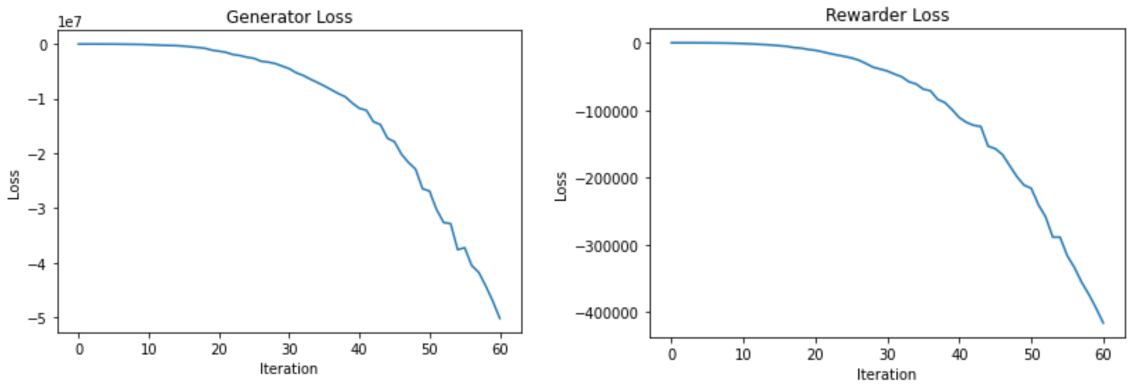
(g) Rollout number: 4

(h) Generator learning rate: 5e-5

(i) Rewarder learning rate: 0.001

(j) Gradient clipping max norm: 1.0

(k) Number of reward iterations per generator iteration ($n_r : n_g$): 5



Unfortunately, we hit many roadblocks. First, we were constrained my limited compute resources, namely, Google Colab, where we frequently reached usage limits. Second, we implemented the architecture in PyTorch from scratch, which took a considerable amount of time. Third, as shown in our loss plots, we encountered issues with exploding gradients, which made our cost function assign high cost to every trajectory, making it hard for the generator to learn.

In order to combat these, we are implementing weight regularization, adding a baseline for policy gradient, and fine-tuning the last two layers of the GPT2 network.

Here are some (silly) example sentences (trajectories) from our model:

(a) shrimp decor rams painted quaint island

(b) Three males revealing the handheld policeman letter

(c) weathered people fighting berry peels

6 Conclusion & Future Work

Merging two seemingly disparate areas, text generation and inverse reinforcement learning, has shown promising results. It is able to alleviate many of the disadvantages of traditional text generation methods, such as reward sparsity and mode collapse. The use of the reward function provides a denser training signal and alleviates mode collapse by making the objective the KL divergence without the estimate of the data distribution, which was the root of the issue. Transformer networks have dominated the sequence modeling world for the past three years, and its use in this framework brings substantial potential. We hope that our work has provided a stepping stone for future works to build upon. In particular, in the future, we hope to pursue works such as:

(a) Use the learned reward to train a generator on other datasets (transfer learning).

(b) Use larger models such as GPT2-Large.

(c) Try out different hidden state representations, such as from the middle of the Transformer instead of the end.

(d) Use different network architectures for the reward function.

(e) Apply this framework to other types of tasks, such as image generation or inferential text generation.

7 Acknowledgements

We'd like to thank Professor Byron Boots, Sandesh Adhikary, and Jake Sacks for their insightful discussions.

References

Saurabh Arora and Prashant Doshi. A survey of inverse reinforcement learning: Challenges, methods and progress, 2019.

Xinlei Chen, Hao Fang, Tsung-Yi Lin, Ramakrishna Vedantam, Saurabh Gupta, Piotr Dollar, and Lawrence Zitnick. Microsoft coco captions: Data col-lection and evaluation server. In *arXiv*, 2015.

Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization, 2016.

Chelsea Finn, Paul Christiano, Pieter Abbeel, and Sergey Levine. A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models. In *arXiv*, 2018.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 2672–2680, Cambridge, MA, USA, 2014. MIT Press.

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29, pages 4565–4573. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper/2016/file/cc7e2b878868cbae992d1fb743995d8f-Paper.pdf.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2014.

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

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, and Xi Chen. Improved techniques for training gans. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29, pages 2234–2242. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper/2016/file/8a3363abe792db2d8761d6403605aeb7-Paper.pdf.

Zhan Shi, Xinchi Chen, Xipeng Qiu, and Xuanjing Huang. Toward diverse text generation with inverse reinforcement learning. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, IJCAI'18, page 4361–4367. AAAI Press, 2018. ISBN 9780999241127.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.