Improving Sentences with Online Policy Gradient Methods

Bhushan Suwal[a], Andrew Savage[a]

COMP 150 Reinforcement Learning, Fall 2018

[a]*Department of Computer Science, Tufts University*

Abstract

The task of sentence completion has been an active area of research, with Seq2Seq recurrent neural network models the most popular solution. The accuracy of such models depend on the the accuracy of the corpus the model is trained on, and actively correcting a huge corpus is an expensive task. Removing the biases learnt by a model after it has been trained is also expensive. To this goal, we investigate how reinforcement learning can be used to improve an LSTM model's performance by teaching the model to avoid the mistakes it learned in the corpus, as well as guiding the model to follow developer defined rules.

1. Introduction

Sentence completion is a well explored problem with many practical applications. Many of the technologies that exist in our everyday life use techniques to assist in sentence creation: messenger applications often have suggestions for the next word to use, Gmail has both automatic response suggestions and auto-complete for email writing, and Google Search has plenty of suggestions for queries when you begin typing.

Datasets to train models on are readily available, but once the models are trained the model is confined to what it learned from the corpus and retraining it to produce sentences or different styles of sentences requires expensive retraining of the model. One of the major attractions of reinforcement learning is training an agent without having explicit datasets to train on because the agent learns its parameters based on the reward function determined by the developer. We use this strength of reinforcement learning to retrain

a Long Short-Term Memory model trained on the WikiText-2 (which has 2,088,628 train tokens) to reduce typing errors.

As an RL formulation, we use the LSTM model as our learning agent. The states are the words generated so far in the sentence, and the set of actions is the next word to take. We have scores that account for the validity of the words in a sentence, the length of a sentence and the repetition of words in a sentence. We use these individual scores as reward functions on individual experiments, as well as a weighted combination of these scores on the last experiment. We observed that networks can be taught developer defined rules in very little time with optimal stopping.

2. Related Work

Sentence completion has been an area of active research. Google's Smart Compose[1] makes use of the popular Seq2Seq model[2] to learn long term dependencies for its sentence completion in Gmail. The model is made up of multi-layered LSTMs that performs well to remember values over arbitrary time intervals. LSTMs have been popular for natural language processing tasks because of their ability to learn on well on long term dependency tasks ever since their introduction by Hochreiter and Schmidhuber (1997). [3]

The use of RL to RNNs is a fairly recent domain. The use of reward functions to modify the language structure of outputs produced by an RNN was used by Li et al.[4] for avoiding repetitive loops in dialogue generation with the use of a reward function that is a combination of metrics for ease of answering, information flow and semantic coherence to encourage dialogue flow. Ranzato et al. use the BLEU score, a metric for machine translation, as a reward at test time to improve on language translation tasks. [5]

The work that inspired much of the work with the RNN model in this project was from Jaques et al., where they improve the melodies produced by an LSTM by training a DQN on a reward function that is a combination of a set of music rules and the output of the original LSTM to avoid strict rigidity caused by the rules. [6]

3. Technical Approach

We opt for a policy gradient method because of its advantages and its convenience: we already have a model that produces maximum likelihood estimates for each action in our vocabulary, and we can sample on these

estimates for exploration. It also allows us to inject a trained model as a prior for our policy parameterization. We use the REINFORCE Monte Carlo algorithm [7] to do our policy updates. The update is defined as

$$\theta_{t+1} = \theta + \alpha\gamma^t G_t \frac{\nabla_t \pi(A_t|S_t, \theta_t)}{\pi(A_t|S_t, \theta_t)}$$

where γ is the discount factor, t is the number of steps in the episode, π is the policy, α is the learning rate, G_t is the reward at time t and θ_t is the state at time t. Since this is a Monte Carlo algorithm, this suffers from high variance at the episode level. To tackle this, we update our parameters in batches of episodes. We also use a simple reinforce with baseline technique of subtracting the rewards with their mean for added stability.

We use an LSTM model as an agent, and use 200 dimension word vectors as states. This agent has the weights of an LSTM trained on over 2 million tokens of the WikiText-2 dataset over 40 epochs as a prior. The partially complete sentences are added to form the states. The set of actions was the next possible word in the sentence. An episode was defined as a series of actions until the end of an episode was reached. The weights were optimized with an Adam optimizer.

There were multiple reward functions implemented. The first one r_{short} encourages short sentences by awarding a reward at a terminal state, defined by the states {., !, ?}. The second one r_{long} encourages longer sentences with a positive reward at a non-terminal state and a zero reward at a terminal state. The third reward $r_{validity}$ function uses the PyEnchant[8] library to check if an action is a valid word, and assigns a positive reward to it and a negative reward otherwise. The fourth reward function $r_{repetition}$ aims to reduce repetition in a sentence, which we found is a common mode of failure for our models. That function awards a penalty if an action has been repeated already in the sentence so far. The final reward function is a weighted average of all of the aforementioned reward functions:

$$r_{total} = \lambda_1 r_{short} + \lambda_2 r_{long} + \lambda_3 r_{validity} + \lambda_4 r_{repetition}$$

where $\lambda_1, \lambda_2, \lambda_3$ and λ_4 are regularization parameters. Separate experiments were conducted on each of these five reward functions.

"Improving" sentences can be a very vague goal. While we do have definitions of what improvement means in the initial more defined individual experiments, like shorter sentence length when the reward is for short sentences, for the final experiment with the weighted reward function we do not

know of any metric that could define a "good" sentence. Especially in the case of machine produced sentences, the output sentence is often garbled like "Another break of matter create 1995" and it is difficult to compare it with another garbled sentence like "Another day bed is yolk sister". As such, we have relied on empirical human observation to define the progress of a model for that experiment.

4. Experiments and Results

4.1. Initial ideas and explorations

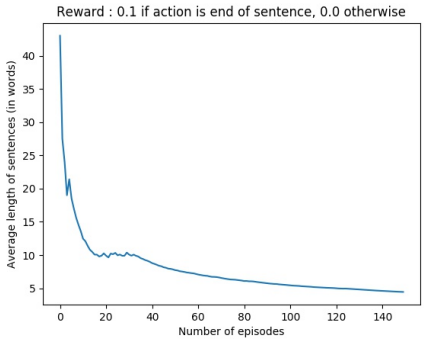
Initially we attempted to implement a character-level RNN with a partial sentence as the state and a character as an action. Initially we struggled with the exploding gradient problem characteristic to recurrent neural networks. We later discarded this approach because we realized using characters as actions would be limiting on our reward function anyway because there are few action-level rewards we could use other than determining the length of a sentence. Most other rewards would only have to be awarded at the end of the episode. We therefore switched to using words as our actions.

We also trained a policy network with two hidden layers to learn the sentences with no prior policy. The set of states was the partial sentence so far represented as the sum of its individual 200 dimension word vectors. The set of actions was the next word to produce. This experiment was to see the performance of reinforcement learning as an alternative to supervised learning, using word vectors of the next word in the training set as the reward on this occasion. The dataset used for this was the Amazon Appliances Question Answer dataset. [9]

4.2. Using rewards to motivate the length of the sentences

We initially started with simple reward functions to reward when a sentence was short and when a sentence was long (r_{short} and r_{long} in Section 3). Figures 1a. and 1b. show the plots showing the length of sentences as the training progressed.

There is a definite trend of the models following their reward functions. However, the rigidity of these rewards mean that to optimize for length the sentences produced lose all structure; for example, by the last episode of for the short sentence reward experiment, the 'sentence' being produced was the just the full stop '.'.



(a) Average sentence length decreasing with a positive reward for non-terminal states.

(b) Average sentence length increasing with a positive reward for non-terminal states.

Figure 1: The change in average length of sentences with sentence length shortening/increasing reward functions.

The short sentence model was trained for 30 iterations while the long sentence model was trained for 20 iterations. For both of them, each iteration was of 5 episodes with a learning rate of 0.01 and a discount rate of 0.01.

4.3. Removing out of dictionary words from the model

We next attempted to penalize out-of-dictionary words by giving them a negative reward. We define our dictionary by the words considered valid by the PyEnchant library. The Word-RNN model produced the following text during the first episode of training:

<eos> Kedok Sullivan County , of the Rocky Natal , which would prove by <unk> Australia , make women viewed by the <unk> in a first Gundersen on Tuesday <unk> , and the area of human paramilitary <unk> immunity whenever it became Fallen 's Animal <unk> <unk> 's long , being ruined into the afternoon , though the aversion to achieve a very " mad and an accompanying in Blood <unk> @-@ induced by <unk> 's end .

We can see that the output has some semblance of what a sentence looks like and the model has learnt some grammar, but this output is littered with words from the corpus that don't exist like '<eos>', '<unk>' and '@-@'.

We applied a validity reward function that awards a reward of 0.1 for valid words and a reward of -0.1 for invalid words.

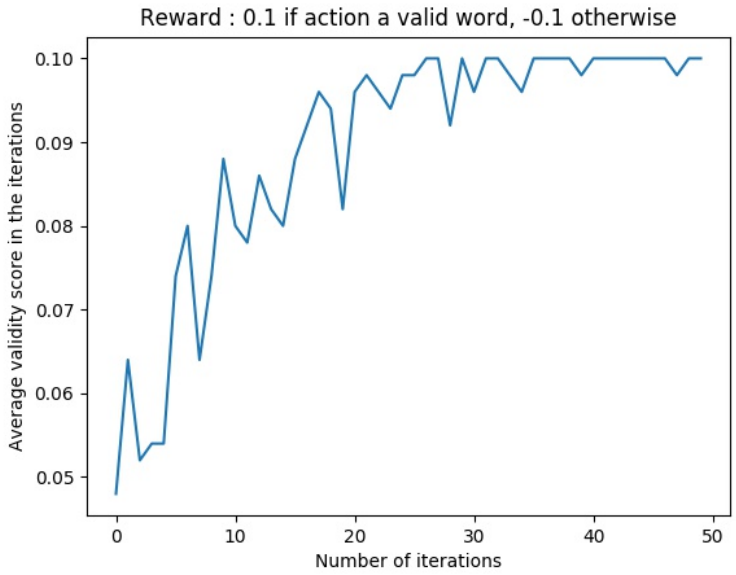


Figure 2: There is clear increase in the validity score over time.

After about iteration 30, the model produces mostly only valid sentences and has succeeded to remove out of dictionary words from it vocabulary. The text it generated at this stage was:

day . The New Jersey . The New York . The New York . The New Jersey . The first . The New Jersey . The New Jersey . The area . The New Jersey . The New Jersey . The first and that year . The New Jersey to the first in the first in the first part of the first . The New Jersey . The New Jersey . The company . The New Jersey .

While the model has learnt to produce only valid words, it has unsurprisingly lost its ability to make more interesting sentences. And while it initially may seem impressive that the model learned to almost completely remove out of dictionary vocabulary in just 30 iterations (which for this experiment was 150 episodes), it is to be noted that only the words that had a high probability of occurring in the initial random episodes were discouraged before the model resorted to repeating what it now knows are valid words.

This model was trained (at the point of optimal stopping) for 30 iterations of 5 episodes each, with a learning rate of 0.01 and a discount rate of 0.01.

4.4. Penalizing repetition in sentences, and using a weighted reward function

The last experiment showed a clear propensity of the model to 'game' the system by repeating valid words again and again. Motivated by this observation, we defined a reward function that punishes repetition in a sentence. Specifically, the reward function to avoid repetition we use is:

$$r_{repetition} = \frac{|\text{unique words in current sentence}|}{|\text{words in current sentence}|}$$

Using just this reward function made the model output longer and longer sentences in the hope of generating more unique words. Finally this motivated the use of the weighted reward function described in Section 3 and reproduced here:

$$r_{total} = \lambda_1 r_{short} + \lambda_2 r_{long} + \lambda_3 r_{validity} + \lambda_4 r_{repetition}$$

With empirical tuning, we set the $\lambda_1 = 0.1$, $\lambda_2 = 0$, $\lambda_3 = 0.4$ and $\lambda_4 = 0.5$ and trained the model for 8 iterations to obtain the following results:

In his book with the most and in the higher than his parody of the most of the small and have love by a long over pay of his most areas in this high school .

The first made up to be a British letters with about 100 840 from March results at least two women and expected .

Compared to the output generated in the very first episode without any reward observation, these sentences show a significant improvement in removing out-of-dictionary words and avoiding repetition loops while still retaining the word dependencies from the original model. We noticed that the optimal stopping point for our experiment was at 8 iterations, after which the quality of the sentences seemed to get qualitatively worse. Still, it is impressive to note that it took only 8 iterations (40 episodes) for the model to reduce poor vocabulary.

This model was trained for 8 iterations of 5 episodes each, with a learning rate of 0.01 and a discount rate of 0.01.

Table 1: Appendix of generated sentences mentioned in this paper

Sentence	Reward	Iterations
'.'	Short Sentences	30
<eos> Kedok Sullivan County , of the Rocky Natal , which would prove by <unk> Australia , make women viewed by the <unk> ...	None, Prior from LSTM model	0
day . The New Jersey . The New York . The New York . The New Jersey . The first . The New Jersey . The New Jersey . The area . The New Jersey . The New Jersey . The first and that year ...	Validity	30
In his book with the most and in the higher than his parody of the most of the small and have love by a long over pay of his most areas in this high school . *The first made up to be a British letters with about 100 840 from March results at least two women and expected .*	Weighted Reward	8

Note: An iteration was set as 5 episodes for all of these experiments.

4.5. Rewarding positive sentiments

We also aimed to reward sentences that could be categorized as positive with a positive reward with an aim to have 'happier' sentences. We chose an existing implementation from Python's nltk package called Sentiment Intensity Analyzer that provided a score $\subset [-1, 1]$ on sentences. We gave rewards only at the end of episodes, but the model failed to realize more positive sentiments clearly. This experiment resulted in failure, but we think that this would work with better tweaking of the hyperparameters.

5. Conclusion

The use of creative reward functions can be used with policy gradient methods to tune a model to obey developer-defined traits. The use of strict rules alone quickly distorts the learning of the model, as the model can learn to try to optimize for the rewards and lose its creativity. We therefore think that having a low learning rate is preferable to obtain an optimal stopping point.

Designing these reward functions may seem inconvenient and training with them is delicate to avoid the over simplification of the model. However, a model learns implicit biases from its training set and correcting these biases is not easy without expensive retraining of the entire model. Our results show that models can be re-tuned in little time compared to the amount it would take to retrain the entire model with an unbiased dataset. We offer such reward based learning as a promising path to remove biases from a model.

6. Future Work

The use of a Deep Q Network on top of an RNN similar to what Chen et al[10] have explored could be useful to explore a more robust learning of the policy. Furthermore, inherent problems in sentence completion and text generation in general like producing thought sequences that span multiple sentences have not yet been solved. The attractive qualities of RL that can make the agent learn dependencies far off in the past could be an avenue of research exploration. More specific to this project, strictly rule based rewards can make the agent get 'stiff' in its attempts to optimize for rewards. To this goal, combining the reward with the output of another model like Jacques et al.[6] explored could result in a more human-like text generation.

7. Acknowledgements

We sincerely express our gratitude to Professor Jivko Sinapov in his assistance through this project. We also are grateful for the makers of PyTorch and its wonderful open source community for being very beginner-friendly.

8. References

[1] Smart compose: Using neural networks to help write emails, https://ai.googleblog.com/2018/05/smart-compose-using-neural-networks-to.html/, 2018.

[2] Q. V. L. Ilya Sutskever, Oriol Vinyals, Sequence to sequence learning with neural networks, arXiv:1409.3215 (2014).

[3] J. S. Sepp Hochreiter, Long short term memory, Neural Computation (1997).

[4] A. R. M. G. J. G. D. J. Jiwei Li, Will Monroe, Deep reinforcement learning for dialogue generation, arXiv:1606.01541 (2016).

[5] M. A. W. Z. MarcAurelio Ranzato, Sumit Chopra, Sequence level training with recurrent neural networks, International Conference on Machine Learning (2015).

[6] R. E. T. D. E. Natasha Jaques, Shixiang Gu, Tuning recurrent neural networks for dialogue generation, arXiv:1606.01541 (2016).

[7] R. J. Williams, Simple statistical gradient-following algorithms for connectionist reinforcement learning, Machine Learning (1992).

[8] PyEnchant library, https://pypi.org/project/pyenchant/, 2018. Accessed: 2018-12-17.

[9] Amazon appliances question answer dataset, http://jmcauley.ucsd.edu/data/amazon/, 2018. Accessed: 2018-12-17.

[10] D. L. Clare Chen, Vincent Ying, Deep q-learning with recurrent neural networks, http://cs229.stanford.edu/proj2016/report/ChenYingLaird-DeepQLearningWithRecurrentNeuralNetwords-report.pdf (2016).