CS230

Reinforcement Learning as an Approach for Language Generation

Sandy Li
sandyli@stanford.edu

Joan Bae
joanbae@stanford.edu

Adam Williams
awill13@stanford.edu

Abstract

Natural language generation is a process in which an algorithm tries to generate human-understandable text. It's a difficult task for artificial intelligence to solve and has recently become a highly explored space. In this paper, we try two approaches to language generation the (1) LSTM model and a (2) GAN network model. We achieved a result of about 45% accuracy with the LSTM model and were able to score 22/28 with the reinforcement learning approach in an environment of our own creation.

1 Introduction

Language generation, in general, is not easy and has been an ever-present problem in AI that has recently received even more attention in the space with works such as GPT-3 (1) and BERT (2). Language generation is an especially difficult task due to the complexities, nuances, and grammar rules. We propose two solutions: a long short-term memory (LSTM) model and Generative Adversarial Networks (GAN) (3)(4), combined with reinforcement learning (RL) (5) (6). We want the discriminator to work with the RL agent to give rewards that will be able to reflect the quality of the generated language. In order to make reinforcement learning powerful for our deep learning framework, we first need a good word-based generator. Then we utilize the adversarial reinforcement learning framework for creating real sentences similar to human-generated sentences.

2 Related Work

The base RL algorithm is the deep Q network (DQN) (5), more specifically the dueling double deep Q network (D3QN) (6). These papers outline a neural network-based Q learning framework. The training setup is derived from generative adversarial networks (GANs) (3)(4). From this, we draw the discriminator network that will be used to extract features using CNN layer and classify text. Another example of adversarial based language generation can be seen in (7), but the author uses the Feature-Mover's Distance as opposed to an RL framework. An RL-based approach is (8). Here, the authors used external rewards to determine the quality of the network. Another work that is more similar to ours is (9), where they use a similar adversarial RL framework, but use the REINFORCE algorithm as opposed to a deep learning approach.

CS230: Deep Learning, Spring 2021, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

3 Dataset

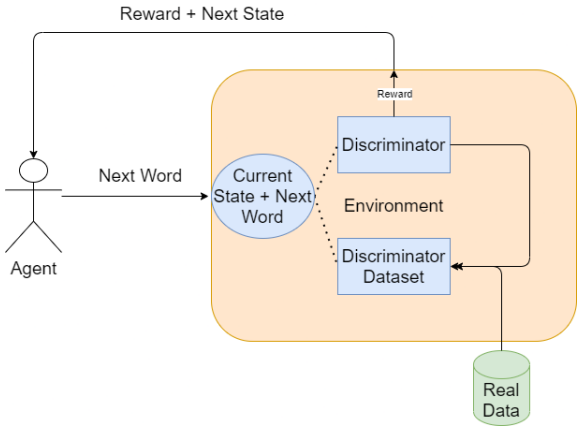
For text corpora, we use the Facebook bAbI dataset (10). There are 20 question-answering tasks in total, and each of tasks provides 10,000 training examples and 10,000 testing examples respectively. We chose this dataset because it the language used in it is relatively simple compared to other language datasets, as it only has 516 unique words. Our goal is not to provide human-like question answering so we are able to reduce volume of data we have by selecting one task among 20 tasks. We want to minimize the data for training efficiency, and because since the dataset is simple enough, it should not result in too drastic of a loss in model quality. All these English sentences will be used as inputs to the word-based language generator. For word vector representations, we selected 50-dimenstional GloVe representation as embedding model. (11)

3.1 Data Preprocessing

The bAbI dataset prepends integers to the beginning of each datapoint. This number is not useful and is distracting for our purposes so we delete the integers in our preprocessed dataset. We want to represent input sentence s as a sequence of numbers. So we first scan and collect all possible unique words (`vocabulary`) in a sentence from training/testing set. Then, we encode each word with an index value from 2 to the largest index.(`word_to_idx`, `idx_to_word`). Index value 0 is reserved for padding, and index value 1 is for indicating start of sequence token.

4 Model description and Approach

Our approach consists of two main parts - the RL agent and the environment. The action space of the RL agent includes all of the words available to it. The agent will decide which word should come next in the sequence, which is the current state. The action is then sent to the environment. The action is appended to the current state to create a more complete sentence. Inside the environment is a discriminator that will read the current state and make a decision on whether or not it was generated or it is real. If it thinks the input is real with 0.5 certainty, the reward becomes $1 * \texttt{len(s)}$. Else, the reward is 0. The current state is then stored in a separate dataset for the discriminator to train on with a label of 0, or fake. Similarly, a data point of equal size of the current state is added to the discriminator dataset, with a label of 1, or real. The state is then returned to the model along with the reward.



4.1 LSTM Text Generation

For text generation, we used the LSTM model with word embedding to create a vector for input sentence. The model is as follows: let an input sentence s be $\texttt{w}_1$, $\texttt{w}_2$, ..., $\texttt{w}_\text{m}$ where $\texttt{w} \in \texttt{vocabulary}$. Given input string s, make our prediction of the next word $\hat{y}$, where $\hat{y} \in \texttt{vocabulary}$.

For implementing text generator, we are going to iterate over the input sentences from test set, and create a list of input/expected output word vectors. For each input sentence, we create pairs of possible input substring vector with the next expected word in an output vector. Each vector size is

set to the size of longest sentence we've seen from training/testing set (`input_vector_size`). Here is an example of train dataset:

1. Input text: *Bill moved to the bedroom*
2. `input_vector_size=6`
3. `word_to_idx={Bill:2, moved:7, to:12, the:100, bedroom:9}`
4. `Output_tmp=[Bill, Moved], [Bill moved, to], [Bill moved to, the], [Bill moved to the, bedroom]`
5. `Output= [[2, 0, 0, 0, 0, 0], [7]], [[2, 7, 0, 0, 0, 0], [12]], [[2, 7, 12, 0, 0 ,0], [100]], [[2, 7, 12, 100, 0, 0], [9]]`

For training, we treat this as a standard supervised learning task. In the case of *Bill moved to the bedroom*, we will have one training example $X =$ *Bill moved to* $Y = the$ as we are trying to predict the very next word in the sequence. After we pass the input X to the LSTM, we send it through a fully connected layer and then take the softmax of the output. We can compare the output to the actual label using cross-entropy loss. Then, for prediction, we take $argmax_{\hat{y}}$ where $\hat{y} = model(X)$. This will give us the index of the word in our `idx_to_word` lookup table.

4.2 Discriminator Network Model

For building an embedding layer in the model, we initialized the embedding matrix with pretrained 50-dimenstional GloVe word vectors. The weights of embedding layer is set to this embedding matrix.

Our discriminator uses two convolutional layers combined with (leaky) relu layers to extract features. The convolutional output is flattened and sent it to a linear layer with a sigmoid activation for two-class classification. Two training sessions occur when training the discriminator - 1) Train with real language sentence 2) Train with current state(sentence) evaluated from RL model described in the following section. When training (1), our target label is Y=1. When training (2), target label is Y=0. Binary Cross Entropy is used as loss measurement, and Adam algorithm is used as optimizer. In regards to choosing hyperparameter values, we used a batch size of 128, configured learning rate to 0.0002, and set betas = (0.5, 0.999) for Adam optimization.

For training in the environment, a batch is chosen from the discriminator dataset. The batch will have a mixture of generated inputs with labels of 0 and real inputs with labels of 1. The discriminator is trained at the same rate as the RL network, at the end of each time step.

4.3 RL Network Model

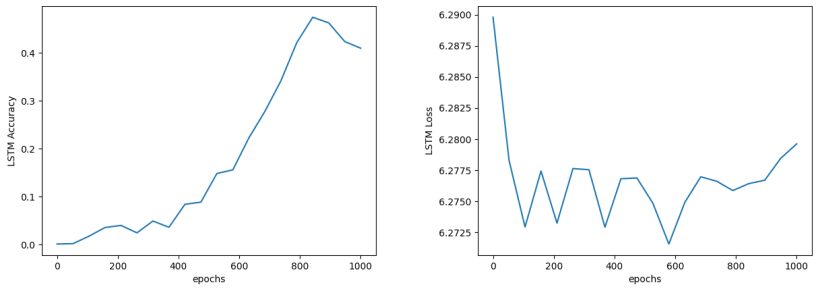
The RL network is a D3QN structure. This consists of two networks that each predict a Q value given the state. The first is the network that predicts the Q value at the current state and the second predicts the target Q value. The target for the first network is computed as $\hat{Q_{eval}}(s) = r + \gamma Q_{target}(s', a)$ where r is the reward given by the environment, γ is the discount factor which is set to 0.99, Q_{target} is the target network, s' is the next state, and a is the action taken by Q_eval at state s or the current state. $\hat{Q_{eval}}(s)$ is then compared to $Q_{eval}(s)$ using a mean squared error loss function. Every 100 steps, the target network's parameters are replaced by the main network, thus moving the targets forward.

In order to make training more efficient, the agent is also equipped with a replay buffer, which it uses to train at the same rate as the discriminator, at the end of every time step.

As an alternative experiment, we added some real data with the bAbI dataset along with the expected reward for generating the text to the replay buffer of the agent. We then pre-train the agent prior to interacting with the environment We will call these agents "Cheater" agents, as they already have information about the environment. The purpose of the cheater agent is to give the agent some kind of idea about how sentences should be structured and the appropriate actions to take given some state based on the real data. We initially believed this would cause the agent to outperform the discriminator, constantly achieving the highest score.

5 Experiments/Results/Discussion

For our baseline experiment, we trained the LSTM generator on just the bAbI dataset, trying to predict the next word. Using this method, we achieved an accuracy of around 45%.



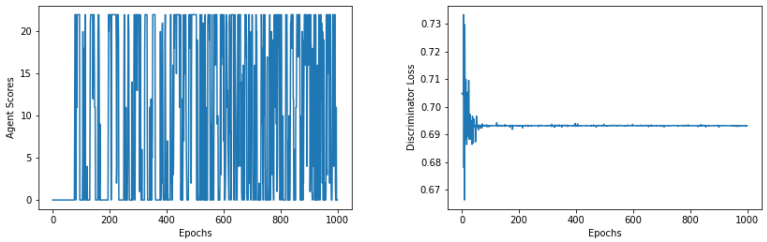
LSTM generation

We can see in the LSTM generation, the accuracy has a slow start but quickly learns but reaches a peak. Perhaps with more tuning we would have been able to achieve better results.

For our base experiment in our main approach, the agent was able to achieve a maximum score of 22 out of 28 (28 being the maximum number if the discriminator guessed the data is real each time).

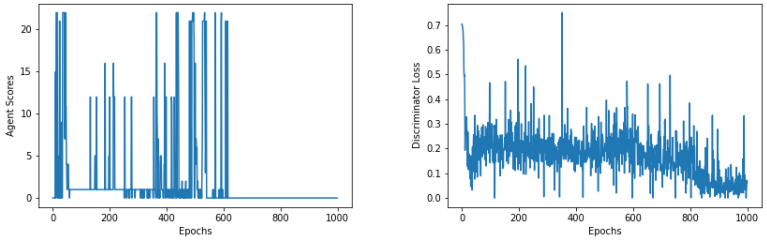
A few examples of generated sentences are

daniel back to the garden to back to bathroom kitchen kitchen daniel mary back to the kitchen yann to to garden garden bathroom the

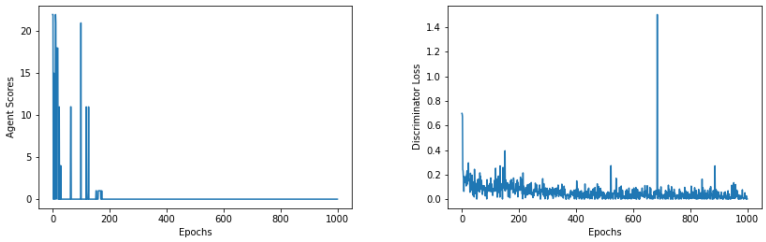


Base agent

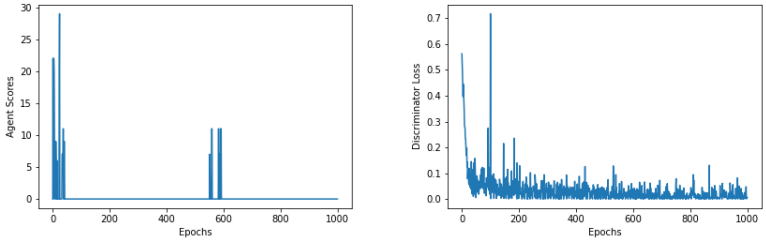
As we can see, the discriminator quickly learned to distinguish which inputs were real and fake. However, we can also see that the agent learned to adapt to the discriminator, as we can see the score is going back and forth between the maximum and minimum value. One possible assumption is that this is due to sentence length. All the sentences in the bAbI dataset are very short, which may explain why the loss of discriminator converges rapidly after 130 iteration.



Cheater agent with 10 examples



Cheater agent with 100 examples



Cheater agent with 1000 examples

We see what we consider very interesting in the cheater agents. The agents quickly maximize the score, but the discriminator seems to be able to tell if the input is fake only after a few dozen steps, bringing the agent's reward to 0 for almost every other step over training. The exception to this being the cheater with 10 episodes. the scores seem to reflect what was seen in the base agent where the agent and discriminator went back and forth until the discriminator learned enough about the text the agent generates.

6 Conclusion

Language generation continues to be a difficult problem in AI research. Language has such complex structures that it is no wonder why transformer models like GPT-3 with billions of parameters are able to achieve state of the art results. We use a basic LSTM model for our baseline agent and were able to achieve decent results on a very simple dataset. The more novel approach we address in this paper is less orthodox than the most widely accepted approaches. It involved creating our own RL environment, which makes it difficult to truly judge how effective or ineffective it is compared to more standard approaches to language generation. Although the experiment without using the cheater agent achieved the highest average results, it may be possible that the discriminator saw sentences that looked "too real", it was easy to distinguish in the cheater examples. There is a possibility that given a longer run time, the cheater agent can outperform the base agent.

7 Contributions

Sandy did input dataset pre-processing, finalized LSTM language generator, and edited reports. Joan did GAN network training, input data pre-processing for word vector representation, and most of writings. Adam did RL training, developed initial LSTM language generator, and edited reports.

References

[1] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," 2020.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019.

[3] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," 2014.

[4] J. Guo, S. Lu, H. Cai, W. Zhang, Y. Yu, and J. Wang, "Long text generation via adversarial training with leaked information," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[6] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," 2015.

[7] L. Chen, S. Dai, C. Tao, D. Shen, Z. Gan, H. Zhang, Y. Zhang, and L. Carin, "Adversarial text generation via feature-mover's distance," 2020.

[8] V. Srinivasan, S. Santhanam, and S. Shaikh, "Natural language generation using reinforcement learning with external rewards," 2019.

[9] J. Li, W. Monroe, T. Shi, S. Jean, A. Ritter, and D. Jurafsky, "Adversarial learning for neural dialogue generation," 2017.

[10] J. Weston, A. Bordes, S. Chopra, A. M. Rush, B. van Merriënboer, A. Joulin, and T. Mikolov, "Towards ai-complete question answering: A set of prerequisite toy tasks," *arXiv preprint arXiv:1502.05698*, 2015.

[11] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," 2014.