Analyzing performance of RL algorithms on bigram language model MDP

Tanvi Sahay
University of Massachusetts Amherst

December 2017

1 Introduction

Statistical language models a.k.a language models are probabilistic models that assign probability to the occurrence of a sentence using information of cooccurrence of words extracted from a corpus of baseline text. In this project, I explore how a simple 2-gram or bigram model can be modeled as an MDP and how different RL algorithms perform in learning a policy that can produce fixed length coherent and meaningful sentences. One of the prime challenges of defining an MDP is to design a reward function that is in accordance with the goal you are trying to achieve. In this project, I explore 4 different reward functions and compare them in order to determine which ones best reflect the desired objective, by computing a coherence score for sentences produced by the agent and comparing it with sample sentences extracted from the training corpus. I analyze and compare the performance of three RL algorithms: SARSA, Q-Learning and Q(λ), on this task on the basis of their average return of rewards.

In the following sections, I describe how the bigram language model was translated to an MDP, what reward functions were used in learning the policy and how the three algorithms performed on this task. Section 2 describes the bigram language in more detail, section 3 describes how the model was translated to an MDP, what reward function were used and how the policy was learned, section 4 describes the dataset used for extracting reward values and section 5 provides details of the experiments conducted and results obtained.

2 Bigram Language Model

A bigram language model or a 2-gram language model predicts the next word based on a single previous word. Mathematically, it computes the probability of a word as follows:

$$P(w_n|H) = P(w_n|w_{n-1}) = \frac{count(w_{n-1}w_n)}{count(w_{n-1})} \tag{1}$$

where w_n is the n^{th} word to be predicted, w_{n-1} is the word seen at $(n-1)^{th}$ time step, H is the history till time step n, $count(w_{n-1}w_n)$ is the number of times the word pair $w_{n-1}w_n$ was seen in a corpus of text and $count(w_{n-1})$ is the number of times word w_{n-1} was seen in a corpus of text. Based on this model, probability is assigned to a sentence by calculating the joint probability of all words in the sentence, as shown in equation 2.

$$P(s) = P(w_1w_2w_3w_4....w_n) = P(w_1|START)\prod_{k=1}^{n-1} P(w_{k+1}|w_k) \tag{2}$$

Here $START$ denotes the beginning of a sentence. As seen here, in a bigram model, dependencies only go up to 2 terms. Since probabilities lie between 0 and 1, multiplying enough bigram probabilities can result in underflow. Thus, traditionally, probability of a sentence is calculated in log domain, as shown in equation 3.

$$P(s) = exp(log^{P(w_1|START)} + \sum_{k=1}^{n-1} log^{P(w_{k+1}|w_k)}) \tag{3}$$

3 Problem Description

3.1 Environment

In order to completely define the bigram language model as an MDP, we need to define its states, actions, transition function, reward function, initial state distribution and discounting factor. This MDP $M_{lm} = (S, A, P, R, d_0, \gamma)$ can be defined as follows:

- States: States of the MDP were all possible words present in the vocabulary of the base text corpus. To avoid excess computation, the vocabulary was limited to 500 words. The terminal state was token '.'.
- Actions: The set of actions contained all words that could be predicted as the next word, which is equal to the vocabulary of the base text.
- Transition Function: The transition function used was deterministic in nature i.e. the next state governed by action taken was reached with a probability of 1.
- Rewards: Four separate reward functions were defined and a separate policy was learned to maximize each of these reward functions.
  - **reward 1**: The first reward function calculated the bigram probability of co-occurrence of the two states (s_t, s_{t+1}) and translated the value to an integer lying in the range of 1-10. Mathematically,

    $$R(s, a, s') = ceiling(\frac{count(ss')}{count(s)} * 10) \tag{4}$$

    The intuition behind choosing this score as a reward function was that in maximizing the reward obtained from this function, the agent would learn a policy that generates sentences whose bigram subsequences would have a high probability of co-occurrence.
  - **reward 2**: Part-of-speech tags (pos tags) are lexical categories such as noun, verb, adjective etc. that govern the structure of any sentence. Therefore, the second reward function calculated the probability of co-occurrence of the part of speech tags of the pair of previous and next states (part of speech tags of $s_t and s_{t+1}$) and translated it to an integer lying in the range 1-10. Mathematically,

    $$R(s, a, s') = ceiling(\frac{count(POS(s)POS(s'))}{count(POS(s))} * 10) \tag{5}$$

    This reward function was chosen so that with the intuition that in trying to maximize this reward, the agent would learn a policy that would generate bigrams that were grammatically correct.
  - **reward 3 and reward 4**: The above two rewards either look at the word co-occurrence or the part of speech tag co-occurrence separately. To account for both the part of speech tag sequence and the actual word sequence, the next two rewards were a combination of rewards 1 and 2. Reward function 3 calculated the product of the rewards 1 and 2 while reward function 4 calculated their average.
- $d_0(s)$: The initial state distribution was uniform and at the beginning of each episode, a state was chosen uniformly at random from the state space.
- gamma(γ) : In all the experiments, value of γ was fixed to 1.

The MDP was defined as a finite horizon MDP with maximum time steps = 10.

3.2 Agent

The agent implemented an ϵ-greedy policy which was learned using state-action value function based update rules. For each update rule experimented with, q(s,a) was defined as: $q_w(s, a) = w^T \phi(s, a)$ where $\phi(s, a)$ was a one-hot vector, with each position in the vector representing a state-action pair. The update rules used were:

1. SARSA : The sarsa update is an on-policy algorithm that uses the TD error given in equation 6, for linear function approximation $q_w(s,a) = w^T\phi(s,a)$, to update the q values for each state-action pair as given in equation 7.

$$\delta_t = R_t + \gamma * w^T\phi(s',a') - w^T\phi(s,a) \tag{6}$$

$$w = w + \alpha\delta_t \frac{\partial q_w(s,a)}{\partial w} = \alpha\delta_t\phi(s,a) \tag{7}$$

Here, R_t is the reward obtained at t, (s',a') is the state action pair at time $t+1$ and (s,a) is the state-action pair at time t. At the end of each time step, current states and actions are set to the next states and actions i.e.

$$(s,a) \leftarrow (s',a') \tag{8}$$

2. Q Learning : Q Learning is an off policy algorithm that uses the TD error given in equation 9, for linear function approximation, to update the q values for each state-action pair as given in equation 10.

$$\delta_t = R_t + \gamma * max_{a'} w^T\phi(s',a') - w^T\phi(s,a) \tag{9}$$

$$w = w + \alpha\delta_t \frac{\partial q_w(s,a)}{\partial w} = \alpha\delta_t\phi(s,a) \tag{10}$$

At the end of each time step, current state is set to next state i.e.

$$s \leftarrow s' \tag{11}$$

3. Q(λ) : Q(λ) uses the TD error update along with e-traces given in equations 12 and 13 respectively, for linear function approximation, to update q values for each state-action pair as given in equation 14. This update is the backward view of Q(λ) that allows us to update $q(s,a)$ at each time step instead of waiting for the episode to end.

$$\delta_t = R_t + \gamma * max_{a'} w^T\phi(s',a') - w^T\phi(s,a) \tag{12}$$

$$e = \gamma\lambda e + \phi(s,a) \tag{13}$$

$$w = w + \alpha\delta_t e \tag{14}$$

3.3 Reward Function Evaluation and Hyperparameter Optimization

To evaluate the learned policies, 1000 sample sentences were generated for each reward function and for each sentence, its probability of occurrence was calculated(3). To provide a single score to learned policy, an average of these probabilities over all 1000 sentences was calculated. These scores were compared with average sentence probability of 1000 sentences drawn from the training text corpus, to estimate how the quality of sentences generated from each learned policy compared with actual sentences. For each algorithm, hyperparameters were tuned by searching over a fixed set of permutations of values and best hyperparameters were chosen to be those that maximized the average sentence probability score. Table 1 shows the range of hyperparameters searched over, for each algorithm. The same ranges were used for all reward functions and were chosen through manual observation.

Table 1: Set of hyperparameters searched

	SARSA	Q Learning	Q(λ)
ϵ	0.05, 0.001, 0.0001	0.05, 0.001, 0.0001	0.001, 0.0001
α	0.09, 0.1, 0.5, 0.9	0.05, 0.09, 0.1, 0.5, 0.9	0.09, 0.1, 0.5, 0.9
λ	NA	NA	0.9, 0.5, 0.1

4 Dataset

To compute the word co-occurrence and pos tag co-occurrence probabilities, three books of the English novelist Jane Austen were downloaded from the Project Gutenberg website. Text from all the books was cleaned to remove all punctuations other than '.' and vocabulary was limited to 500 words. Words were sorted according to their frequency and the top 499 words were chosen. In addition to that, all words that did not occur in the top 499 were assigned a special token 'unk'. After cleaning, the text corpus containing 500 unique words and a total of 525965 tokens.

5 Experimentation and Results

A total of 12 experiments were carried out in the form of unique update rule - reward function pairs and for each of these experiments, hyperparameters were optimized as explained in section 3.3. For each experimental pair, the best hyperparameters found have been provided in table 2. Corresponding to these best parameters, the average return of rewards are shown in figure 1 for SARSA, figure 2 for Q Learning and figure 3 for Q(λ).

Table 2: Best Hyperparameters for each reward function

	sarsa		qlearning		qlambda		
	ϵ	α	ϵ	α	ϵ	α	λ
reward1	0.001	0.9	0.001	0.5	0.001	0.9	0.9
reward2	0.001	0.9	0.001	0.05	0.0001	0.09	0.5
reward3	0.05	0.09	0.05	0.9	0.0001	0.1	0.5
reward4	0.05	0.9	0.0001	0.1	0.001	0.9	0.1

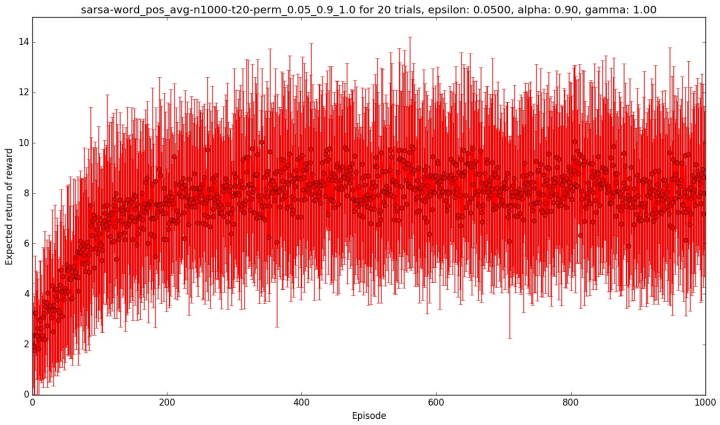

Figure 1: Average expected return of rewards for 20 trials using SARSA update

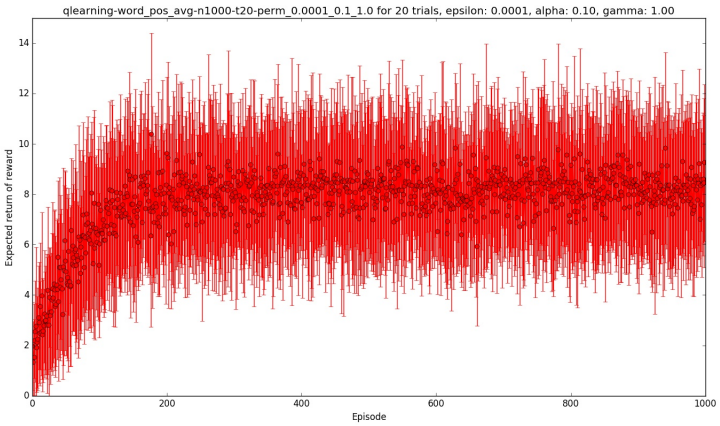

Figure 2: Average expected return of rewards for 20 trials using Q Learning update

Table 3 shows how each experimental performed in terms of the average sentence log probability score. Last column of the table is as estimate of probability of actual sentences, averaged over 1000 length-10 sentences extracted from the training text corpus. As can be seen from these results, reward function 4 provides the closest resemblance in terms of sentence probabilities to actual sentences present in the test corpus. This seems reasonable since the reward function 4 takes into account both the word co-occurrence and the pos tag co-occurrence, which allows the agent to learn policies that optimize both these values. On the other hand, reward functions 2 and 3 perform poorly because in reward function 2, only pos tags are taken into account and in reward 3, values are biased towards pos tags or word pairs that have a very high probability of occurrence. A bigram whose word sequence is rarely seen but whose pos tag

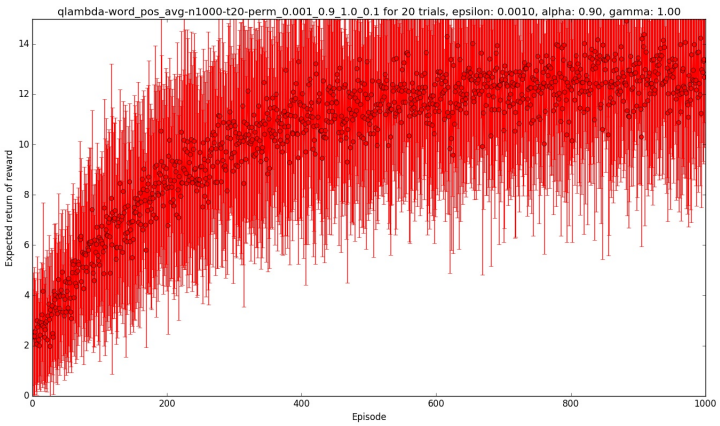

Figure 3: Average expected return of rewards for 20 trials using Q(λ) update

Table 3: Average sentence probabilities for each model (*e-04)

	SARSA	Q Learning	Q(λ)	baseline
reward 1	19.5275	16.0263	30.9983	311.9264
reward 2	9.7899	1.9959	5.9214	311.9264
reward 3	8.5425	7.5111	48.3262	311.9264
reward 4	24.0697	12.5941	59.1307	311.9264

sequence occurs very frequently would still get a high reward when following reward function 3, while reward function 4 will compensate for it by averaging the two rewards.

6 Conclusion

In this project, I analyzed the performance of different RL algorithms on the task of generating sentences by translating a bigram language model into an MDP. I explored the challenging task of designing rewards for an MDP by experimenting with 4 different reward functions and comparing them on the basis of how well the policy learned using each of these rewards reflected the original goal of generating coherent sentences. As can be seen from the results above, reward functions 1 and 4 in general performed better than reward function 2 with reward function 3 performing better in some cases and worse in others. This is because reward function 1 has information of the actual occurrence of words, which is more informative than co-occurrence counts of part of speech tags only. On the other hand, pos tag co-occurrence adds the information of grammatical correctness to the rewards. Thus, reward function 4 performs consistently well with all update rules because it allows discrediting a word-pair if the pos-tag sequence for that pair has not been seen frequently enough, a feature missing in reward 3. Explaining via a simple example: if a word-pair reward corresponds to 1 and the pos-tag pair reward for that word pair is 10, reward function 3 would still return a reward of 10, while function 4 would return a reward of 5, thus reducing the reward given to a word pair whose pos tag sequence occurs frequently in the underlying text but whose actual word sequence is rarely seen. This is advantageous as it encourages generation of words that are both seen frequently and have a more probable pos tag sequence. In general, with proper hyperparameter tuning and appropriately designed rewards, action value function based RL algorithms can work well on the bigram language model MDP.