HW3: Language Modeling

In this homework, you will be implementing a bigram language model on a dataset containing news headlines. Specifically, you will be tasked to complete the `count_bigrams`, `get_probability`, `sample_word`, and `calculate_perplexity` functions.

Setup

Here we will be importing the necessary modules, as well as doing some basic preprocessing of the text. You do not need to modify any code here.

[1]

```
import math
import random
import re
import string

# Reading in the training and development datasets
with open("headlines.train", 'r') as f:
    headlines_train = f.readlines()
with open("headlines.dev", 'r') as f:
    headlines_dev = f.readlines()

# Removing excess punctuation and newline
regex = re.compile('[%s]' % re.escape(string.punctuation))
headlines_train = [regex.sub('', h.split("\n")[0]) for h in headlines_tra
headlines_dev = [regex.sub('', h.split("\n")[0]) for h in headlines_dev]

# Defining UNK, START and STOP tokens
UNK_TOKEN = "<UNK>"
START_TOKEN = "<START>"
STOP_TOKEN = "<STOP>"
```

Before we begin, let's look at what some of the headlines look like. Run the following code block as many times as you want to get a sense of what kind of headlines we hope to generate.

[2]

```
for headline in random.sample(headlines_train, 5):
    print(headline)
    print()
```

```
vcat to hear police station push

ex mayor laments loss of council experience

pinpointing the source of sea level rise

new shire administrator seeks lightning ridge

ashes highlights day three
```

1.a) Counting Bigrams

Below we have `count_unigrams` implemented for you. Implement `count_bigrams`, which takes in a headline as text, and updates the given bigram dictionary with the bigram counts for that text.

[3]

```python
def count_unigrams(text, unigram_dict):
    """
    :param text: A headline, consisting of a string of words
    :param unigram_dict: A dictionary containing unigrams as keys and the
    """
    tokens = [START_TOKEN] + text.split(" ") + [STOP_TOKEN]
    for i in range(len(tokens)):
        unigram = tokens[i]
        if unigram not in unigram_dict:
            unigram_dict[unigram] = 1
        else:
            unigram_dict[unigram] += 1

def count_bigrams(text, bigram_dict):
    """
    :param text: A headline, consisting of a string of words
    :param bigram_dict: A dictionary containing bigrams as keys and their
    """
    tokens = [START_TOKEN] + text.split() + [STOP_TOKEN]

    for i in range(len(tokens) - 1):
        first_word = tokens[i]
        second_word = tokens[i + 1]
        bigram = first_word + " " + second_word
        bigram_dict[bigram] = bigram_dict.get(bigram, 0) + 1
    return bigram_dict
```

1.b) Computing Probability

Implement `get_probability`, which calculates the probability of seeing a word given the previous word, along with Laplace smoothing parameterized by `alpha`.

```
[4]  def get_probability(target, context, unigram_dict, bigram_dict, alpha, vo
         """
         :param target: The word whose probability of seeing is being computed
         :param context: The word directly preceeding the target word
         :param unigram_dict: A dictionary containing unigrams as keys and the
         :param bigram_dict: A dictionary containing bigrams as keys and their
         :param alpha: The amount of additive smoothing being applied
         :param vocab_size: The size of the training vocabulary
         :return: The probability of seeing the target word given the context
         """
         # YOUR CODE HERE
         counts_context = unigram_dict.get(context,0)
         bigram = context + " " + target
         counts_context_target = bigram_dict.get(bigram, 0)
         prob = (counts_context_target + alpha) / (counts_context + vocab_size
         return prob
```

1.c) Word Sampling

Once we can calculate the desired probabilities, we can now use that to sample words for generation. Implement `sample_word`, which samples a new word in accordance with its probability of following the previous word.

```
[5]  def sample_word(words, probs):
         """
         :param words: The list of words to sample from
         :param probs: The probabilities of seeing each word; probs[i] is the
         :return: A word whose sampling likelihood is the probability of being
         """
         # YOUR CODE HERE
         r = random.random()
         CDF = 0
         x = 0

         for i in range(len(probs)):
             CDF = CDF + probs[i]
             x = i
             if CDF >= r:
                 break
         return words[x]
```

Generating New Headlines

Now that we've all the key parts of our language model completed, let's see how well we can generate new headlines! Run the following code block to complete the algorithm, and the subsequent code block as many times as you want to see what kind of new headlines we are able to generate.

```
[11]  alpha = .000001 # Change this to see different levels of smoothing affect
      min_freq = 10 # The minimum word frequency to be present in the vocabular

      # The following are used to keep track of and remove infrequent words
      low_freq = set()
      all_words = {}

      def generate_headline(unigram_dict, bigram_dict, alpha):
          sent = [START_TOKEN]
          while not sent[-1] == STOP_TOKEN:
              words = list(vocab)
              probs = [get_probability(word, sent[-1], unigram_dict, bigram_dic
              next_word = sample_word(words, probs)
              sent.append(next_word)
          print(' '.join(sent[1:-1]))

      def replace_text_train(text):
          return " ".join([UNK_TOKEN if t in low_freq else t for t in text.spli

      def replace_text_dev(text):
          return " ".join([UNK_TOKEN if t not in vocab else t for t in text.spl

      # Finding all words with low frequency
      for h in headlines_train:
          count_unigrams(h, all_words)
      for word, count in all_words.items():
          if count <= min_freq:
              low_freq.add(word)
      # Replacing low frequency words from training dataset with UNK
      headlines_train_clean = [replace_text_train(h) for h in headlines_train]

      # Initialize unigram and bigram dictionaries
      unigrams = {}
      bigrams = {}

      # Generating unigram and bigram dictionaries
      for h in headlines_train_clean:
          count_unigrams(h, unigrams)
          count_bigrams(h, bigrams)

      # Creating the training vocabulary
      vocab = set([item for sublist in map(lambda x: x.split(" "), headlines_tr
    vocab.add(START_TOKEN)
    vocab.add(STOP_TOKEN)

    # Replacing unseen vocabulary from development dataset with UNK
    headlines_dev_clean = [replace_text_dev(h) for h in headlines_dev]
```

[18]

```
for _ in range(5):
    generate_headline(unigrams, bigrams, alpha)
    print()
```

```
sri lanka says he was killed in <UNK> to <UNK> as police out as first time

pocket <UNK> rate

powerline funding cut for tiger airways plane plan

operation begins voting good progress says

<UNK>
```

2.a) Calculating Perplexity

Once we're able to generate new headlines, we can see how well our algorithm works when encountering text from an unseen development set. Implement `calculate_perplexity` below and run the code block below that to see what the perplexity of our algorithm is.

[19]

```
def calculate_perplexity(text, unigram_dict, bigram_dict, alpha, vocab_si
    """
    :param text: A headline, consisting of a string of words
    """
    # YOUR CODE HERE
    count_unigrams(text, unigram_dict)
    count_bigrams(text, bigram_dict)
    words = text.split()
    sum_probs = 0
    for i in range(len(words)-1):
        sum_probs += math.log(get_probability(words[i+1], words[i], unigr
    return math.exp(-sum_probs/len(words))
```

[20]

```
perplexities = []
for h in headlines_dev_clean:
    perp = calculate_perplexity(h, unigrams, bigrams, alpha, len(vocab))
    perplexities.append(perp)
```

```
print("Average perplexity on dev set: %.02f" % (sum(perplexities) / len(p
```

```
Average perplexity on dev set: 54.89
```

2.b) Parameter Experimentation

Play around with modifying the smoothing parameter `alpha`. How does changing the values of that parameter affect the generated headlines, as well as the perplexity on the development set? Write two such observations in the cell below. Find a value for `alpha` that gives the lowest perplexities; you should be able to get a perplexity below 1000. What value of `alpha` gets the optimal perplexity value?

Lowering alpha usually brought the average perplexity down. I raised alpha a few magnitudes for the first time and headlines were sometimes a paragraph long and incoherent. Seems like the lowest I can get is 54.88 with 10^{-6}

3) RNN Language Model

List the three headlines you generated from running the rnn_language_model.ipynb notebook.

mp mp foreign investment boost for states coast

livestock plane creek restoration

plane to make excites with nets it aussie chopper