Assignment 3
CSE 517: Natural Language Processing - University of Washington
Winter 2022

Solution 1.2

The purpose of this problem is to develop a language model for unigrams, bigrams, and trigrams. The dataset is divided into three files: a training set, a validation set, and a testing set. I began coding by importing the three text files into a Jupyter notebook and preparing the training data for preprocessing.

Data Preprocessing: I simply started by adding a pad in the beginning and end with special tokens, START = "<s> " and STOP = "</s>". For n >= 2, n-1 SOS tokens are added, otherwise only one is added. Where n is the order of the n-gram model.

Next step was to create the UNK tokens. Tokens which appeared less than three times in the training data were assigned as UNK = "<UNK>" token. After that the training data was striped by removing all whitespace characters (newlines and spaces) from the end of each line.

```
def add_sentence_tokens(sentences):
    return ['{}{} {}'.format(START, s, STOP) for s in sentences]

def replace_tringletons(tokens):
    vocab = nltk.FreqDist(tokens)
    return [token if vocab[token] >= 3 else UNK for token in tokens]

def preprocess(sentences):
    sentences = add_sentence_tokens(sentences)
    tokens = ' '.join(sentences).split(' ')
    tokens = replace_tringletons(tokens)
    return tokens
```

Language model: The n-gram language model for a given corpus was constructed by pre-processing the corpus and then computing the probabilities for each n-gram. To prevent a situation in which test data assigns a probability of zero to unseen words, I choose to use Lapace smoothing. By simply adding one to all the unigram, bigram, and trigram counts before normalizing the probabilities, the Laplace smoothing helps overcome the sparse data problem. Now, all the counts that may be zero will have a count of one. For instance, if the unsmoothed maximum likelihood estimates of the word w_i unigram probability is its count c_i normalized by the total number of word tokens N:

$$P(w_i) = \frac{c_i}{N}$$

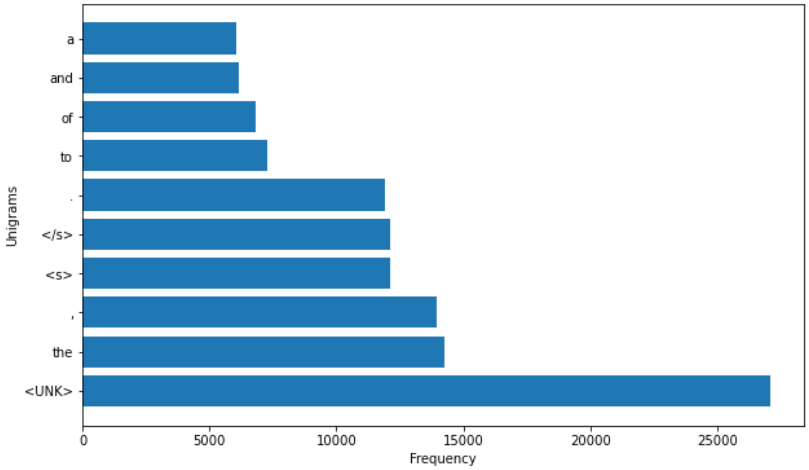
The probability after Laplace smoothing is shown below. Here, V denotes the words in the vocabulary.

$$P_{Laplace}(w_i) = \frac{c_i + 1}{N + V}$$

```
def Laplace_smoothing(n_gram, n_count):
    lm_gram = n_gram[:-1]
    lm_count = m_vocab[m_gram]
    return (n_count + laplace) / (lm_count + laplace * vocab_size)
```

The plot below illustrates the frequency distribution of the top ten unigrams. We see that the UNK token, along with special tokens such as START and STOP, has the highest frequency in the

corpus. Since we did not remove the stop words, we see more tokens such as 'the', 'to', 'of', 'and', and 'a'.



Next step in the process was to determine the perplexity, which deterministic transformation of log-likelihood into an information theoretic quantity. I simply used the formula below formula to calculate the perplexity: $Perplexity(w) = 2^{\frac{-l(w)}{N}}$, where N is the total number of tokens. Below is the snippet of the formula that I used for the perplexity function in my code.

```
return math.exp((-1/N) * sum(map(math.log, probabilities)))
```

The final step in the process was to calculate the perplexity for three unigram, bigram, and trigram language models using training, validation, and test data sets. The following code snippet show the perplexity for a unigram language model using the training data set, and table 1 summarizes the results for all three language models using the train, validation, and test data sets.

```
lm = LanguageModel(train, 1)
print("Vocabulary size: {}".format(len(lm.vocab)))
perplexity = lm.perplexity(train)
print("Model perplexity: {:.3f}".format(perplexity))
```

```
Vocabulary size: 26603
Model perplexity: 888.297
```

Table 1. Language Model Perplexity

		Perplexity			Perplexity with Laplace Smoothing
Data Set	**Vocabulary size**	**Unigram**	**Bigram**	**Trigram**	**Unigram**
Training	26603	223.961	1.0146	1.00	207.606
Validation	9607	223.174	1.014	1.000	206.88
Test	9599	625.497	8.29	1.412	649.786

Solution 1.3

In this part of the assignment, we are asked to implement the linear interpolation smoothing between unigram, bigram, and trigram models. Below is the formula that I used for an interpolated trigram model:

$$P\ interpolation(w_n|w_{n-1}, w_{n-2}) = \lambda_3 P_3(w_n|w_{n-1}, w_{n-2}) + \lambda_2 P_2(w_n|w_{n-1}) + \lambda_1 P_1(w_n)$$

Here, each token, $w_n, n = 1,2 \ldots N$ and $\lambda_3 + \lambda_2 + \lambda_1 = 1$
I first started by taking the unsmoothed empirical probability for the unigram, bigram, and trigram from the previous n-gram model that I created. The perplexity formula is given as:

```
cross_entropy = (-1)*sum(logpx)/len(logpx)
perplexity = 2**cross_entropy
```

Perplexity scores using different lambda values are shown in Table 2.

Table 2. Perplexity score for different lambda set

Lambda Set	Training data set	Validation data set	Test data set
[0.5, 0.3, 0.2]	255.464	254.563	232.746
[0.8, 0.1, 0.1]	202.119	201.428	251.783
[0.4, 0.2, 0.4]	288.013	286.964	269.576
[0.1, 0.4, 0.5]	588.515	586.088	439.359
[0.1, 0.3, 0.6]	590.136	587.692	468.231

Perplexity score when half training data is used in Table 3.

Table 3. Perplexity score when half training set used

	Perplexity			Perplexity with Laplace Smoothing
Data Set	Unigram	Bigram	Trigram	Unigram
Training	175.701	1.013	1.000	157.392
Validation	175.021	1.013	1.000	156.794
Test	476.813	5.702	1.264	500.776

Perplexity score for different hyperparameter when half training data is used (Interpolation Smoothing) in Table 3.

Table 4. Perplexity score for different hyperparameter (Interpolation Smoothing)

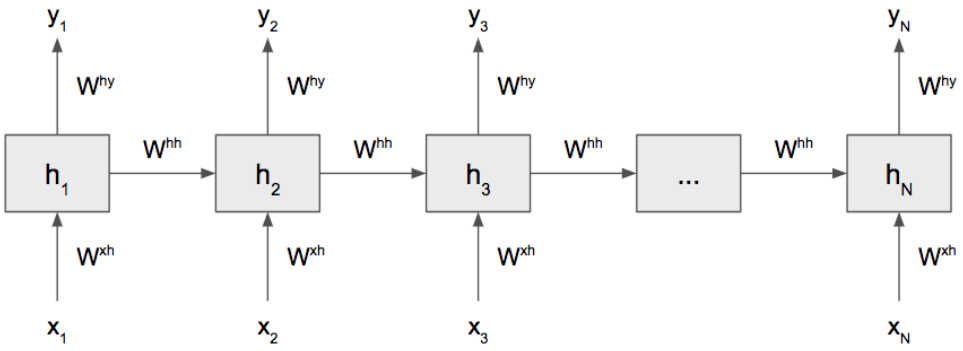
Lambda Set	Training data set	Validation data set	Test data set
[0.5, 0.3, 0.2]	193.501	192.640	233.367
[0.8, 0.1, 0.1]	153.961	153.302	237.957
[0.4, 0.2, 0.4]	217.535	216.548	273.797
[0.1, 0.4, 0.5]	436.979	434.786	479.983
[0.1, 0.3, 0.6]	438.158	435.957	509.923

Hence, we found that above the perplexity score improved when we the size of the training data is reduced.

Solution 2.

In this problem, we used wikitext2 as our dataset, which is already split into train, validation, and test sets, and the corpus provided does not need to be processed because it is already tokenized at the word level. The only change I made in the pre-process was to replace newlines with STOP </s> tokens. When I was doing minor pre-processing, I observed that the training text had several titles, such as gaming, plot, development, and so on. As a result, I chose to omit them; this may cause poor perplexity, but the wikitext2 appeared clearer with the same number of tokens as before this step.

For this problem I used the Recurrent Neural Network Model. This type of model does not rely on the n-gram assumption of restricted context; in fact, it may contain arbitrarily distant information, while still being able to be used in statistical and computational models. The RNN picture below contains boxes for each input in the sequence. We feed an input $x_t$at time t into a specific cell to get a hidden state h, which we then use to make an output y_t. This is how we proceed till we reach the finish of the sequence. The most important feature of an RNN is recurrence (shown in an arrow), which links one hidden state block to another. This recurrence demonstrates a dependence on all information prior to time t.



The objective here is to determine the probability $P(w|u)$ where $w \in v$ denotes a word and u denotes the context established by the prior words. Rather than calculating word probabilities solely based on (smoothed) relative frequencies, we can approach language modeling as a machine learning task and optimize the log conditional probability of a corpus through parameter estimation.

The dataset that we have used have the 2075677 number of tokens in training, 216347 in validation and 244102 in the test data. The vocabulary size came out to be 33278. Code snippet is provided below for the evaluation criteria.

```
def rnnmodel_evaluation(datapath):
    model.eval()
    loss = 0.
    num_of_tokens = len(corpus.dictionary)
    hidden = model.init_hidden(10)
    with torch.no_grad():
        for k in range(0, datapath.size(0) - 1, BPTT):
            data, result = get_batch(datapath, k)
            output, hidden = model(data, hidden)
            output = output.view(-1, num_of_tokens)
            total_loss += len(data) * CRITERION(output, result).item()
            hidden = repackage_hidden(hidden)
    return total_loss / len(datapath)
```

Here, the parameter bptt (backpropagation through time is 3 in our case) specifies the maximum time step backward to which the gradient should backpropagate, which is equal to the batch's sequence length.

Table 5. Hyperparameters used					
Embedding size	**Features**	**Epochs**	**Dropout**	**Perplexity**	**Loss**
650	33278	40	0.5	75.96	4.486

Solution 3.

I fully concur with the authors' assertion that previous language models ignored the underlying meaning of natural language, preferring to learn some reflection of meaning in the linguistic form, which is extremely beneficial for a variety of tasks. This is evident from both the octopus's observation of A and B's conversation and the paper's argument about how humans interact and rely on shared attention and intersubjectivity. We discovered from babies that the physical environment, as well as interaction with others, is critical for meaningful language learning rather than passive language learning. I particularly admire the author's big-picture perspective, which emphasizes the importance of a top-down approach that aims to provide a thorough understanding of the task with clear end goals, rather than focusing exclusively on a bottom-up approach that implies partial success on a particular challenge. Additionally, the paper discusses practices that can be considered when developing meaningful language models, such as cultivating humility toward language, comprehending when creating new tasks, carefully evaluating the task to determine if it can perform well across multiple tasks, and finally, not only considering the data that we already have but also paying equal attention to the data that is missing and its impact on the model overall.

With this, I believe we have arrived at a point where science has made significant progress in the field of natural language understanding by assisting humanity in completing language tasks (e.g., chatbot). Now that we are aware of the language model's lack of meaning, I believe that scientists will find a way to incorporate semantics into the language model, resulting in a language model that may not be perfect but significantly improved.

References:

[1] Speech and Language Processing. Daniel Jurafsky & James H. Martin. Copyright © 2021. All rights reserved. Draft of December 29, 2021.
[2] Murphy, Kevin P. Machine learning: a probabilistic perspective. MIT press, 2012.
[3] Natural Language Processing by Jacob Eisenstein
[4] McKinney, Wes. "Pandas, python data analysis library." URL http://pandas. pydata. org (2015).
[5] Link: https://medium.com/the-artificial-impostor/notes-neural-language-model-with-pytorch-a8369ba80a5c
[6] Link: https://www.analyticsvidhya.com/blog/2020/08/build-a-natural-language-generation-nlg-system-using-pytorch/
[9] Natural Language Processing with Deep Learning CS224N by Christopher Manning
[9] Ed discussion board for A3 Assignment and discord group.