Homework 1

DS-GA 1011, Fall 2020

Team members:

Abed Qaddoumi (amq259), Chenjie Su (cs5998), Kunru Lu (kl3743)

Contributions:

We all solved the whole homework individually first and then compared answers and worked on the parts that were not the same, therefore the contribution was distributed most or less equally between the three team members.

PART 1

A. Context-free grammar

Here is a fragment of English grammar:

S → NP VP
NP → Det N
VP → V NP
Det → the
Det → my
N → walrus
N → manatee
N → whale
V → saw
V → visited

Question 1: Example sentence. Write a sentence that this grammar generates.

Example sentence: The walrus saw my manatee.

Question 2: Intransitive verbs.

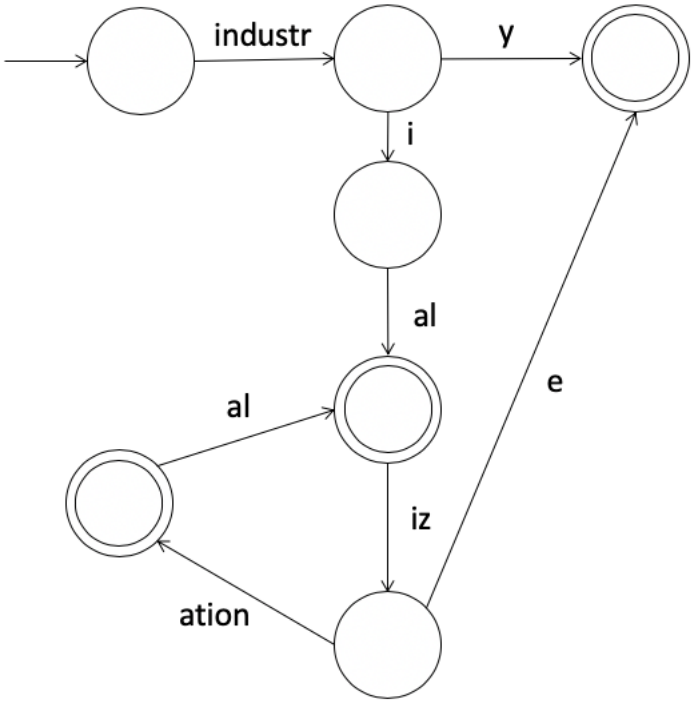
S → NP VP
NP → Det N
VP → Vi
VP → Vt NP
Det → the
Det → my
N → walrus
N → manatee
N → whale
Vt → saw
Vt → visited
Vi → smiled
Vi → chuckled

Question 3: Subject-verb agreement.

S -> NPs VPs
S -> NPp VPp
NPs -> Det Ns
NPp -> Det Np
VPs -> Vs NPs
VPs -> Vs NPp
VPp -> Vp NPs
VPp -> Vp NPp
Vs -> sees
Vs -> visits
Vp -> see
Vp -> visit
Det -> the

Det -> my
Ns -> walrus
Ns -> manatee
Ns -> whale
Np -> walruses
Np -> manatees
Np -> whales

Question 4: Morphology.



PCFGs

Specifically, we will use the classic example of syntactic ambiguity, namely prepositional phrase attachment:

1) The spy saw the professor with the telescope.

Start out by running out the next few cells, which provide functions that handle CFGs.

```
In [1]: # Given a CFG, generates a list of all parse trees generated by that grammar
def generate_all_parses(grammar):
    rule_dict = {}

    for rule in grammar:
        parts = rule.split()
        lhs = parts[0]
        rhs = " ".join(parts[2:])
        if lhs in rule_dict:
            rule_dict[lhs].append(rhs)
        else:
            rule_dict[lhs] = [rhs]


    all_parses = ["ROOT"]
    done = 0

    while not done:
        new_parses = []

        for parse in all_parses:
            words = parse.split()
            index = 0

            for word in words:
                if word in rule_dict:
                    for rule in rule_dict[word]:
                        new_sent = words
                        if len(rule.split()) > 1:
                            new_sent[index] = "( " + rule + " )"
                        else:
                            new_sent[index] = rule

                        new_parses.append(" ".join(new_sent))

                    break

                if index + 1 == len(words):
                    new_parses.append(parse)

                index += 1

        if all_parses == new_parses:
            done = 1

        all_parses = new_parses


    return all_parses
```

```
In [2]: # Helper function for removing extra spaces from a string
def despace(string):
    if "  " in string:
        return despace(string.replace("  ", " "))
    return string.strip()
```

```
In [3]: # Function that creates a dictionary of all sentences generated by a CFG and all of their possible pars
es
def make_parse_dict(grammar):
    parse_dict = {}

    for parse in generate_all_parses(grammar):
        sentence = despace(parse.replace("(", "").replace(")", ""))
        if sentence in parse_dict:
            parse_dict[sentence].append(parse)
        else:
            parse_dict[sentence] = [parse]

    return parse_dict
```

Coding Q1: Fill in the function `parse` below so that given a sentence and a grammar, it returns all possible ways of parsing that sentence given that grammar.

```
In [4]: def parse(sentence, grammar):
    a = make_parse_dict(grammar_1)
    return a[sentence]
```

Ambiguity in CFGs

Now, to warm up, we need to write a CFG which can generate both possible syntactic structures for the sentence *the spy saw the professor with the telescope.* Below is such a CFG:

```
In [5]: grammar_1 = ["ROOT -> S",
             "S -> NP VP .",
             "NP -> Det N PP",
             "NP -> Det N",
             "VP -> V NP",
             "VP -> V NP PP",
             "PP -> P Det N",
             "Det -> the",
             "N -> spy",
             "N -> professor",
             "N -> telescope",
             "N -> stopwatch",
             "P -> with",
             "V -> saw",
             "V -> timed"
          ]
```

You can use the `parse` function to parse sentences:

```
In [6]: parse("the spy saw the professor .", grammar_1)
Out[6]: ['( ( the spy ) ( saw ( the professor ) ) . )']
```

If this grammar was made correctly, and you've implemented the parse function correctly, the following line should give you both possible ways of parsing *the spy saw the professor with the telescope* .:

```
In [7]: parse("the spy saw the professor with the telescope .", grammar_1)
Out[7]: ['( ( the spy ) ( saw ( the professor ( with the telescope ) ) ) . )',
 '( ( the spy ) ( saw ( the professor ) ( with the telescope ) ) . )']
```

This example shows how a CFG can express ambiguity. In this case, there were two different ways that the CFG could generate this sentence, and that is where the ambiguity arose. However, even though both of these structures are possible, one is much more likely than the other. How can we express this?

Probabilistic CFGs

Below is a PCFG called `pcfg_even`, which is a copy of your original CFG with a probability added to each rule. Each rule is now a 2-tuple consisting of a rule and its probability, e.g. `("ROOT -> S", 1.0)`. To start, we have given equal probabilities to all rules that have the same left-hand side.

```
In [8]: pcfg_even = [("ROOT -> S", 1.0),
             ("S -> NP VP .", 1.0),
             ("NP -> Det N PP", 0.5),
             ("NP -> Det N", 0.5),
             ("VP -> V NP", 0.5),
             ("VP -> V NP PP", 0.5),
             ("PP -> P Det N", 1.0),
             ("Det -> the", 1.0),
             ("N -> spy", 0.25),
             ("N -> professor", 0.25),
             ("N -> telescope", 0.25),
             ("N -> stopwatch", 0.25),
             ("P -> with", 1.0),
             ("V -> saw", 0.5),
             ("V -> timed", 0.5)
          ]
```

Coding Q2: Write versions of the functions we've seen before so that they can handle a PCFG like above. Fill in the functions below. (Hint: they should have similar structures to the non-probabilistic versions.)

```
In [9]: def generate_all_parses_pcfg(grammar):
    rule_dict = {}

    for rule, prob in grammar:
        parts = rule.split()
        lhs = parts[0]
        rhs = " ".join(parts[2:])
        if lhs in rule_dict:
            rule_dict[lhs].append((rhs, prob))
        else:
            rule_dict[lhs] = [(rhs,prob)]

    all_parses = ["ROOT"]
    done = 0

    while not done:
        new_parses = []

        for parse in all_parses:
            index = 0
            if(parse == "ROOT"):
                words = parse.split()
                prob = 1
            else:
                words = parse[0].split()
                prob = parse[1]

            for word in words:
                if word in rule_dict:
                    for rule in rule_dict[word]:
                        new_sent = words
                        if len(rule[0].split()) > 1:
                            new_sent[index] = "( " + rule[0] + " )"
                        else:
                            new_sent[index] = rule[0]
                        new_parses.append((" ".join(new_sent), prob * rule[1]))

                    break

                if index + 1 == len(words):
                    new_parses.append(parse)

                index += 1

        if all_parses == new_parses:
            done = 1

        all_parses = new_parses

    return all_parses
```

```
In [10]: def make_parse_dict_pcfg(grammar):
    parse_dict = {}
    for parse in generate_all_parses_pcfg(grammar):
        sentence = despace(parse[0].replace("(", "").replace(")", ""))
        if sentence in parse_dict:
            parse_dict[sentence].append(parse)
        else:
            parse_dict[sentence] = [parse]
    return parse_dict
```

```
In [11]: def parse_pcfg(sentence, grammar):
    temp = make_parse_dict_pcfg(grammar)
    return temp[sentence]
```

Under this formalism, the probability of an entire parse tree then becomes the product of the probabilities of all the rules used to generate that parse tree. If you try parsing sentences now, and if you have implemented the PCFG functions correctly, the result should include a probability along with the parse:

```
In [12]: parse_pcfg("the spy saw the professor .", pcfg_even)
Out[12]: [('( ( the spy ) ( saw ( the professor ) ) . )', 0.00390625)]
```

```
In [13]: parse_pcfg("the spy saw the professor with the telescope .", pcfg_even)
Out[13]: [('( ( the spy ) ( saw ( the professor ( with the telescope ) ) ) . )',
  0.0009765625),
 ('( ( the spy ) ( saw ( the professor ) ( with the telescope ) ) . )',
  0.0009765625)]
```

Now we can manipulate the probabilities to make one parse preferred over the other. Below we've modified `pcfg_even` to make `pcfg_uneven`, which has the property that now the correct parse has a higher probability than the incorrect one.

```
In [14]: pcfg_uneven = [("ROOT -> S", 1.0),
             ("S -> NP VP .", 1.0),
             ("NP -> Det N PP", 0.1),
             ("NP -> Det N", 0.9),
             ("VP -> V NP", 0.8),
             ("VP -> V NP PP", 0.2),
             ("PP -> P Det N", 1.0),
             ("Det -> the", 1.0),
             ("N -> spy", 0.25),
             ("N -> professor", 0.25),
             ("N -> telescope", 0.25),
             ("N -> stopwatch", 0.25),
             ("P -> with", 1.0),
             ("V -> saw", 0.5),
             ("V -> timed", 0.5)
          ]
```

You can test it in the next cell:

```
In [15]: parse_pcfg("the spy saw the professor with the telescope .", pcfg_uneven)
Out[15]: [('( ( the spy ) ( saw ( the professor ( with the telescope ) ) ) . )',
  0.0005625000000000001),
 ('( ( the spy ) ( saw ( the professor ) ( with the telescope ) ) . )',
  0.0012656250000000003)]
```

Lexicalized PCFGs

Problem solved!

...but other problems remain. What happens when you run the next cell? Does this behavior seem correct? (You don't have to answer that question - it's not graded - but it will probably help you to think about it).

```
In [16]: parse_pcfg("the spy saw the professor with the stopwatch .", pcfg_uneven)
Out[16]: [('( ( the spy ) ( saw ( the professor ( with the stopwatch ) ) ) . )',
  0.0005625000000000001),
 ('( ( the spy ) ( saw ( the professor ) ( with the stopwatch ) ) . )',
  0.0012656250000000003)]
```

The problem is that we should get different judgments depending on what the object of the preposition is, and also depending on what the verb is. Think about the following four sentences:

1) the spy saw the professor with the telescope .

2) the spy saw the professor with the stopwatch .

3) the spy timed the professor with the telescope .

4) the spy timed the professor with the stopwatch .

Written Q1: For each of these four sentences, label whether the prepositional phrase (either *with the telescope* or *with the stopwatch*) should attach to the noun (*professor*) or the verb (which is either *saw* or *timed*).

The prepositional phrase should attach to the

1) verb

2) noun

3) noun

4) verb

To solve this problem, we need to *lexicalize* our PCFG, which means to make the rules dependent on specific words. In the next cell, we've gotten you started by lexicalizing the PPs to create a new grammar called `pcfg_lex_v1`. Now, there are separate nonterminals for different PPs depending on what the object of the PP is. For example, `PP_spy` means a PP whose object is `spy`.

```
In [17]: pcfg_lex_v1 = [("ROOT -> S", 1.0),
             ("S -> NP VP .", 1.0),
             ("NP -> Det N PP_spy", 0.025),
             ("NP -> Det N PP_professor", 0.025),
             ("NP -> Det N PP_telescope", 0.025),
             ("NP -> Det N PP_stopwatch", 0.025),
             ("NP -> Det N", 0.9),
             ("VP -> V NP", 0.8),
             ("VP -> V NP PP_spy", 0.05),
             ("VP -> V NP PP_professor", 0.05),
             ("VP -> V NP PP_telescope", 0.05),
             ("VP -> V NP PP_stopwatch", 0.05),
             ("PP_spy -> P Det N_spy", 1.0),
             ("PP_professor -> P Det N_professor", 1.0),
             ("PP_telescope -> P Det N_telescope", 1.0),
             ("PP_stopwatch -> P Det N_stopwatch", 1.0),
             ("Det -> the", 1.0),
             ("N -> N_spy", 0.25),
             ("N -> N_professor", 0.25),
             ("N -> N_telescope", 0.25),
             ("N -> N_stopwatch", 0.25),
             ("N_spy -> spy", 1.0),
             ("N_professor -> professor", 1.0),
             ("N_telescope -> telescope", 1.0),
             ("N_stopwatch -> stopwatch", 1.0),
             ("P -> with", 1.0),
             ("V -> saw", 0.5),
             ("V -> timed", 0.5)
            ]
```

Coding Q3: Further modify `pcfg_lex_v1` (by editing `pcfg_lex` below) so that it gives the correct judgments from your answers to Homework Q1 (that is, you should make it so that the more likely parse for each of those sentences is assigned a higher probability by `pcfg_lex` than the less likely parse). To do this, you may need to further lexicalize the PCFG (that is, you may need to add more word-specific nonterminals and more rules for the new nonterminals). You will also need to set the probabilities for the lexicalized rules so that the probabilities it gives result in the correct judgments for the four sentences.

```
In [18]: pcfg_lex = [("ROOT -> S", 1.0),
             ("S -> NP VP .", 1.0),
             ("NP -> Det N PP_spy", 0.025),
             ("NP -> Det N PP_professor", 0.025),
             ("NP -> Det N PP_telescope", 0.025),
             ("NP -> Det N PP_stopwatch", 0.025),
             ("NP -> Det N", 0.9),
             ("VP -> V NP", 0.8),
             ("VP -> V NP PP_spy", 0.05),
             ("VP -> V NP PP_professor", 0.05),
             ("VP -> V_saw NP PP_telescope", 0.0499),
             ("VP -> V_timed NP PP_telescope", 0.0001),
             ("VP -> V_saw NP PP_stopwatch", 0.0001),
             ("VP -> V_timed NP PP_stopwatch", 0.0499),
             ("PP_spy -> P Det N_spy", 1.0),
             ("PP_professor -> P Det N_professor", 1.0),
             ("PP_telescope -> P Det N_telescope", 1.0),
             ("PP_stopwatch -> P Det N_stopwatch", 1.0),
             ("Det -> the", 1.0),
             ("N -> N_spy", 0.25),
             ("N -> N_professor", 0.25),
             ("N -> N_telescope", 0.25),
             ("N -> N_stopwatch", 0.25),
             ("N_spy -> spy", 1.0),
             ("N_professor -> professor", 1.0),
             ("N_telescope -> telescope", 1.0),
             ("N_stopwatch -> stopwatch", 1.0),
             ("P -> with", 1.0),
             ("V -> V_saw", 0.5),
             ("V -> V_timed", 0.5),
             ("V_saw -> saw", 1.0),
             ("V_timed -> timed", 1.0)
           ]
```

If you lexicalized the grammar properly, the results you get below should match your intuitions about which parses are the correct ones for the different sentences.

```
In [19]: parse_pcfg("the spy saw the professor with the telescope .", pcfg_lex)
Out[19]: [('( ( the spy ) ( saw ( the professor ( with the telescope ) ) ) . )',
  0.0005625000000000001),
 ('( ( the spy ) ( saw ( the professor ) ( with the telescope ) ) . )',
  0.0025261875)]
```

```
In [20]: parse_pcfg("the spy saw the professor with the stopwatch .", pcfg_lex)
Out[20]: [('( ( the spy ) ( saw ( the professor ( with the stopwatch ) ) ) . )',
  0.0005625000000000001),
 ('( ( the spy ) ( saw ( the professor ) ( with the stopwatch ) ) . )',
  5.0625e-06)]
```

```
In [21]: parse_pcfg("the spy timed the professor with the telescope .", pcfg_lex)
Out[21]: [('( ( the spy ) ( timed ( the professor ( with the telescope ) ) ) . )',
  0.0005625000000000001),
 ('( ( the spy ) ( timed ( the professor ) ( with the telescope ) ) . )',
  5.0625e-06)]
```

```
In [22]: parse_pcfg("the spy timed the professor with the stopwatch .", pcfg_lex)
Out[22]: [('( ( the spy ) ( timed ( the professor ( with the stopwatch ) ) ) . )',
  0.0005625000000000001),
 ('( ( the spy ) ( timed ( the professor ) ( with the stopwatch ) ) . )',
  0.0025261875)]
```

Garden-path sentences

The following are examples of garden-path sentences:

1) The horse raced past the barn fell.

2) All people need to buy a house is a lot of money.

3) The complex houses married and single soldiers.

4) The old man the boats.

The name *garden-path sentences* derives from the idiom *to lead down the garden path*, which means "to mislead." In this case, these sentences mislead you because, as you hear them, you are led to believe that they should be parsed one way, but then suddenly you are forced to reconsider your assumed parse by some later words in the sentence.

Like the examples in the previous section, garden-path sentences involve ambiguity. However, there is an important difference in the types of ambiguity found in garden-path sentences vs. in *telescope*-type sentences: For the *telescope*-type sentence, the sentence has two possible parses, meaning that the sentence as a whole is ambiguous. However, for garden path sentences, there is only what is called *local ambiguity*, meaning that some subpart(s) of the sentence may be ambiguous, but once you see the whole sentence only one interpretation remains valid.

You can imagine (at least) two different general strategies for parsing a sentence. One is to start guessing the parse as you proceed from left to right, revising your parse as you encounter more words in the sentence. The other is to consider the entire sentence at once and to construct the parse based on that. The way that we are fooled by garden-path sentences argues in favor of the view that people use the incremental approach.

A crucial fact about garden-path sentences is that there is one specific point where the listener is suddenly thrown into confusion. We can appeal to probabilities to explain why this confusion arises.

Let's start out by defining a simple PCFG that can generate the horse example:

```
In [23]: pcfg_garden = [("ROOT -> S", 1.0),
             ("S -> NP VP .", 1.0),
             ("NP -> Det N", 0.95),
             ("NP -> Det N Rel", 0.05),
             ("VP -> V", 0.5),
             ("VP -> V PP", 0.5),
             ("Rel -> VBN PP", 1.0),
             ("PP -> P Det N", 1.0),
             ("Det -> the", 1.0),
             ("N -> horse", 0.5),
             ("N -> barn", 0.5),
             ("V -> raced", 0.5),
             ("V -> fell", 0.5),
             ("VBN -> raced", 0.5),
             ("VBN -> taken", 0.5),
             ("P -> past", 1.0)
            ]
```

We're going to suppose the following: While a listener is listening to the sentence, they are constantly trying to predict which word will be uttered next. To make this prediction, the listener needs to estimate a probability distribution across all possible words. For example, suppose the listener has so far only heard *the*. In this case, it is likely that the next word will be a noun, but unlikely it will be any other part of speech - in fact, given our grammar above, no other part of speech is possible. Thus, the listener's probability distribution might look like this:

- horse: 0.5
- barn: 0.5
- the: 0.0
- raced: 0.0
- fell: 0.0
- taken: 0.0
- past: 0.0

How will the listener estimate these probabilities? We will say that they consider all possible sentences that the speaker might be uttering. To find the probability of word *w* being the next word, the listener simply computes the total probability of all sentences in which *w* is the next word.

To implement this, we first need a function that generates all possible sentences that can come from this CFG. Below is a function that does this.

```
In [24]: def generate_all(grammar):

    # First, compile the grammar into a dictionary of rules
    rule_dict = {}

    for rule in grammar:
        parts = rule[0].split()
        prob = rule[1]
        lhs = parts[0]
        rhs = " ".join(parts[2:])
        if lhs in rule_dict:
            rule_dict[lhs].append((rhs, prob))
        else:
            rule_dict[lhs] = [(rhs, prob)]

    all_sents = [("ROOT", 1.0)]
    done = 0

    while not done:
        new_sents = []

        for sent in all_sents:
            words = sent[0].split()
            prob = sent[1]
            index = 0

            for word in words:
                if word in rule_dict:
                    for rule in rule_dict[word]:
                        new_sent = words
                        new_sent[index] = rule[0]
                        new_prob = prob * rule[1]
                        new_sents.append((" ".join(new_sent), new_prob))

                    break

                if index + 1 == len(words):
                    new_sents.append(sent)

                index += 1

        if all_sents == new_sents:
            done = 1

        all_sents = new_sents


    return all_sents
```

You can see the outcome of this function with the cell below.

```
In [25]: generate_all(pcfg_garden)
Out[25]: [('the horse raced .', 0.11875),
 ('the horse fell .', 0.11875),
 ('the horse raced past the horse .', 0.059375),
 ('the horse raced past the barn .', 0.059375),
 ('the horse fell past the horse .', 0.059375),
 ('the horse fell past the barn .', 0.059375),
 ('the barn raced .', 0.11875),
 ('the barn fell .', 0.11875),
 ('the barn raced past the horse .', 0.059375),
 ('the barn raced past the barn .', 0.059375),
 ('the barn fell past the horse .', 0.059375),
 ('the barn fell past the barn .', 0.059375),
 ('the horse raced past the horse raced .', 0.0015625),
 ('the horse raced past the horse fell .', 0.0015625),
 ('the horse raced past the horse raced past the horse .', 0.00078125),
 ('the horse raced past the horse raced past the barn .', 0.00078125),
 ('the horse raced past the horse fell past the horse .', 0.00078125),
 ('the horse raced past the horse fell past the barn .', 0.00078125),
 ('the horse raced past the barn raced .', 0.0015625),
 ('the horse raced past the barn fell .', 0.0015625),
 ('the horse raced past the barn raced past the horse .', 0.00078125),
 ('the horse raced past the barn raced past the barn .', 0.00078125),
 ('the horse raced past the barn fell past the horse .', 0.00078125),
 ('the horse raced past the barn fell past the barn .', 0.00078125),
 ('the horse taken past the horse raced .', 0.0015625),
 ('the horse taken past the horse fell .', 0.0015625),
 ('the horse taken past the horse raced past the horse .', 0.00078125),
 ('the horse taken past the horse raced past the barn .', 0.00078125),
 ('the horse taken past the horse fell past the horse .', 0.00078125),
 ('the horse taken past the horse fell past the barn .', 0.00078125),
 ('the horse taken past the barn raced .', 0.0015625),
 ('the horse taken past the barn fell .', 0.0015625),
 ('the horse taken past the barn raced past the horse .', 0.00078125),
 ('the horse taken past the barn raced past the barn .', 0.00078125),
 ('the horse taken past the barn fell past the horse .', 0.00078125),
 ('the horse taken past the barn fell past the barn .', 0.00078125),
 ('the barn raced past the horse raced .', 0.0015625),
 ('the barn raced past the horse fell .', 0.0015625),
 ('the barn raced past the horse raced past the horse .', 0.00078125),
 ('the barn raced past the horse raced past the barn .', 0.00078125),
 ('the barn raced past the horse fell past the horse .', 0.00078125),
 ('the barn raced past the horse fell past the barn .', 0.00078125),
 ('the barn raced past the barn raced .', 0.0015625),
 ('the barn raced past the barn fell .', 0.0015625),
 ('the barn raced past the barn raced past the horse .', 0.00078125),
 ('the barn raced past the barn raced past the barn .', 0.00078125),
 ('the barn raced past the barn fell past the horse .', 0.00078125),
 ('the barn raced past the barn fell past the barn .', 0.00078125),
 ('the barn taken past the horse raced .', 0.0015625),
 ('the barn taken past the horse fell .', 0.0015625),
 ('the barn taken past the horse raced past the horse .', 0.00078125),
 ('the barn taken past the horse raced past the barn .', 0.00078125),
 ('the barn taken past the horse fell past the horse .', 0.00078125),
 ('the barn taken past the horse fell past the barn .', 0.00078125),
 ('the barn taken past the barn raced .', 0.0015625),
 ('the barn taken past the barn fell .', 0.0015625),
 ('the barn taken past the barn raced past the horse .', 0.00078125),
 ('the barn taken past the barn raced past the barn .', 0.00078125),
 ('the barn taken past the barn fell past the horse .', 0.00078125),
 ('the barn taken past the barn fell past the barn .', 0.00078125)]
```

To measure how unexpected each word in the sentence is, we will use a metric called *surprisal*, which is closely related to probability. The surprisal of an event can be defined as follows, where *p* is the probability of that event:

$$s = -\log_2(p)$$

Below is a function that computes surprisal given a probability.

Coding Q4: Fill in the `surprisal` function below, based on the formula above.

```
In [26]: import math

def surprisal(p):
    return -math.log(p,2)
```

In a second, we will ask you to define a function that computes the surprisal of each subsequent word in the sentence. But first, we need to create a function which, given a grammar and a prefix (that is, the start of a sentence), returns a list of all sentences from that grammar starting with that prefix and the total probability of all those sentences.

Coding Q5: Complete the `possible_sentences` function in the cell below. Its output should be list (or tuple, either is fine) with 2 elements. The first element should be a list of all sentences that start with `prefix` (where each sentence is represented as either a 2-element list or a 2-element tuple with the sentence as the first element and its probability as the second element. The second element of the output should be the total probability of all sentences that you return. (You can see some examples in the cells after the function definition).

```
In [27]: def possible_sentences(prefix, grammar):
    possible_sents = generate_all(pcfg_garden)
    total_prob = 0
    answer_sents = []
    for i in possible_sents:
        temp_sent = i[0][:len(prefix)]
        if(prefix in temp_sent):
            answer_sents.append(i)
            total_prob += i[1]
    return answer_sents, total_prob
    #return possible_sents, total_prob
```

You can test this function will the cells below.

```
In [28]: possible_sentences("the horse fell", pcfg_garden)

# This should return:
#([('the horse fell .', 0.11875),
#  ('the horse fell past the horse .', 0.059375),
#  ('the horse fell past the barn .', 0.059375)],
# 0.2375)
Out[28]: ([('the horse fell .', 0.11875),
  ('the horse fell past the horse .', 0.059375),
  ('the horse fell past the barn .', 0.059375)],
 0.2375)
```

```
In [29]: possible_sentences("the horse fell past", pcfg_garden)

# This should return:
# ([('the horse fell past the horse .', 0.059375),
#  ('the horse fell past the barn .', 0.059375)],
# 0.11875)
Out[29]: ([('the horse fell past the horse .', 0.059375),
  ('the horse fell past the barn .', 0.059375)],
 0.11875)
```

```
In [30]: possible_sentences("the horse fell .", pcfg_garden)

# This should return:
# ([('the horse fell .', 0.11875)], 0.11875)
Out[30]: ([('the horse fell .', 0.11875)], 0.11875)
```

Coding Q6: Now, fill in the `incremental_surprisals` function so that it returns a list of surprisals, one for each word in the sentence.

```
In [31]: import math

def incremental_surprisals(sentence, grammar):
    words = sentence.split()
    prob_list = []
    prefix = ''
    for word in words:
        if prefix == '':
            condi_prob = possible_sentences(word, grammar)[1]/possible_sentences(prefix, grammar)[1]
            prefix = word
        else:
            condi_prob = possible_sentences(prefix + ' ' + word, grammar)[1]/possible_sentences(prefix,
 grammar)[1]
            prefix += ' ' + word
        prob_list.append(round(surprisal(condi_prob), 3))
    return prob_list
```

```
In [32]: possible_sentences("the horse raced .", pcfg_garden)
Out[32]: ([('the horse raced .', 0.11875)], 0.11875)
```

```
In [33]: possible_sentences("the horse", pcfg_garden)
Out[33]: ([('the horse raced .', 0.11875),
  ('the horse fell .', 0.11875),
  ('the horse raced past the horse .', 0.059375),
  ('the horse raced past the barn .', 0.059375),
  ('the horse fell past the horse .', 0.059375),
  ('the horse fell past the barn .', 0.059375),
  ('the horse raced past the horse raced .', 0.0015625),
  ('the horse raced past the horse fell .', 0.0015625),
  ('the horse raced past the horse raced past the horse .', 0.00078125),
  ('the horse raced past the horse raced past the barn .', 0.00078125),
  ('the horse raced past the horse fell past the horse .', 0.00078125),
  ('the horse raced past the horse fell past the barn .', 0.00078125),
  ('the horse raced past the barn raced .', 0.0015625),
  ('the horse raced past the barn fell .', 0.0015625),
  ('the horse raced past the barn raced past the horse .', 0.00078125),
  ('the horse raced past the barn raced past the barn .', 0.00078125),
  ('the horse raced past the barn fell past the horse .', 0.00078125),
  ('the horse raced past the barn fell past the barn .', 0.00078125),
  ('the horse taken past the horse raced .', 0.0015625),
  ('the horse taken past the horse fell .', 0.0015625),
  ('the horse taken past the horse raced past the horse .', 0.00078125),
  ('the horse taken past the horse raced past the barn .', 0.00078125),
  ('the horse taken past the horse fell past the horse .', 0.00078125),
  ('the horse taken past the horse fell past the barn .', 0.00078125),
  ('the horse taken past the barn raced .', 0.0015625),
  ('the horse taken past the barn fell .', 0.0015625),
  ('the horse taken past the barn raced past the horse .', 0.00078125),
  ('the horse taken past the barn raced past the barn .', 0.00078125),
  ('the horse taken past the barn fell past the horse .', 0.00078125),
  ('the horse taken past the barn fell past the barn .', 0.00078125)],
 0.5000000000000003)
```

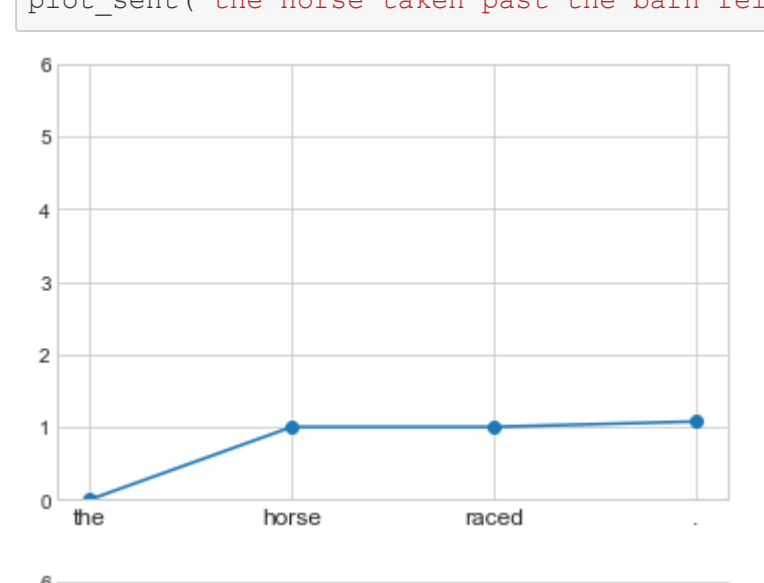
Test this function below.

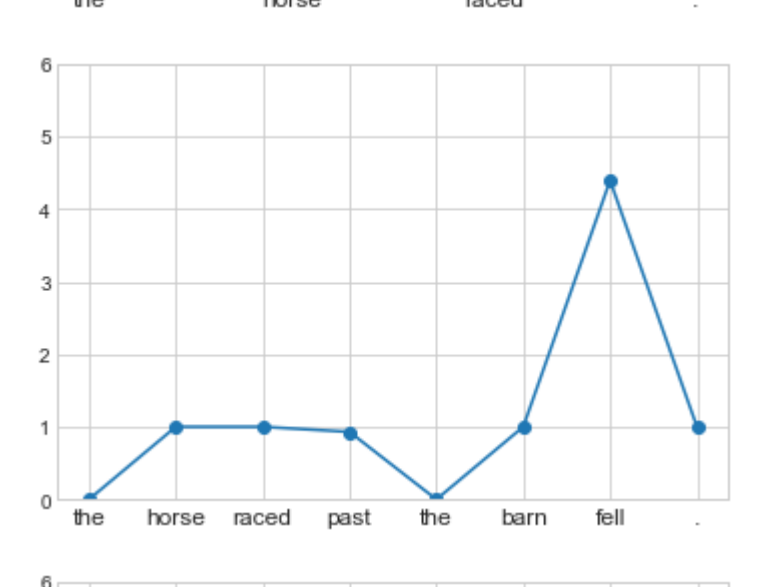
```
In [34]: incremental_surprisals("the horse raced .", pcfg_garden)
# This should return [-0.0, 1.0, 1.0, 1.074]
Out[34]: [-0.0, 1.0, 1.0, 1.074]
```

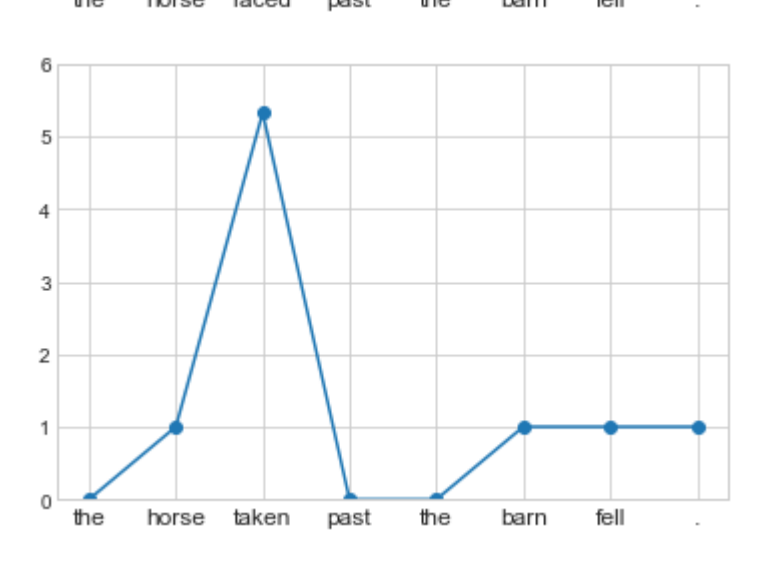
The next cell provides a function for visualizing the surprisal at each point in the sentence. Use the cell after it to try it on some sentences. You might need to install the package matplotlib.

```
In [35]: def plot_sent(sentence, rule_dict):
    sent_len = len(sentence.split())
    plt.style.use('seaborn-whitegrid')
    plt.scatter(range(sent_len), incremental_surprisals(sentence, rule_dict))
    plt.plot(range(sent_len), incremental_surprisals(sentence, rule_dict))
    plt.xticks(range(sent_len), sentence.split(), size='large')
    plt.ylim((0,6))
    plt.show()
```

```
In [36]: %matplotlib inline
import matplotlib.pyplot as plt
plot_sent("the horse raced .", pcfg_garden)
plot_sent("the horse raced past the barn fell .", pcfg_garden)
plot_sent("the horse taken past the barn fell .", pcfg_garden)
```

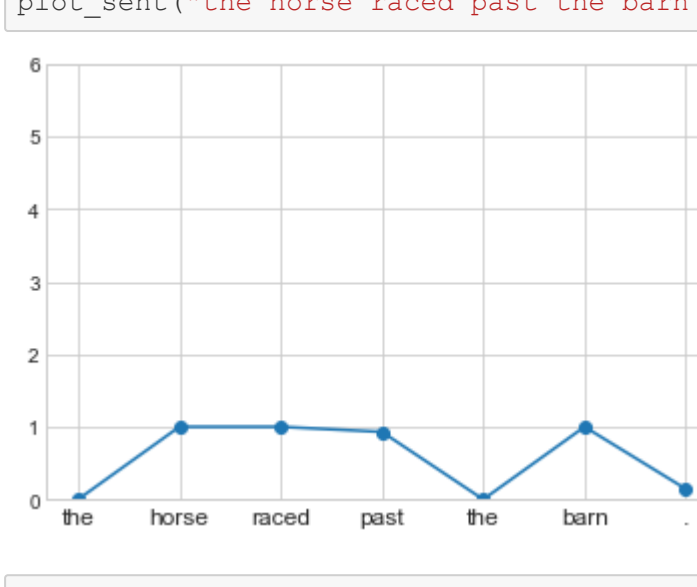




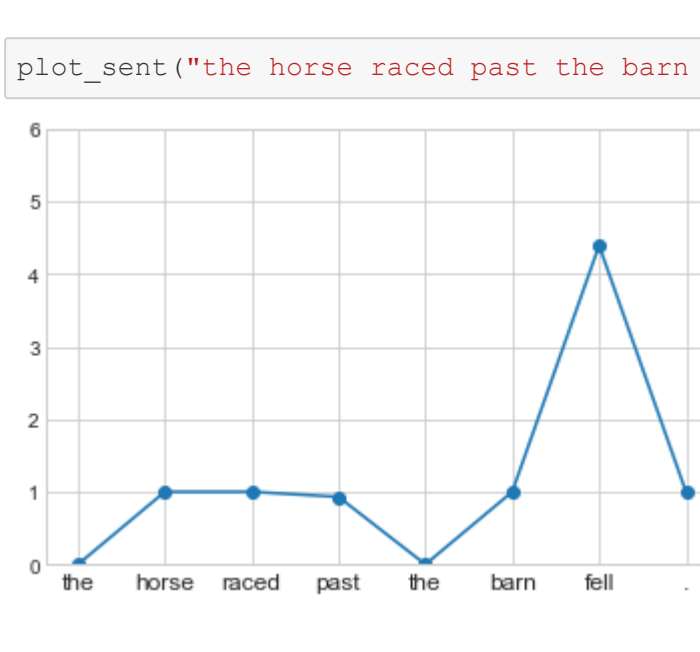


Written Q4: Using the `plot_sent` function, determine the pattern of surprisals for each word in the following two sentences: "the horse raced past the barn ." and "the horse raced past the barn fell ." (If you use `incremental_surprisals`, your answer will be a list of numerical values; if you use `plot_sent`, your answer will be the plots that the function outputs). Explain how these patterns of surprisals illustrate the garden path effect.

In [37]:
```python
plot_sent("the horse raced past the barn .", pcfg_garden)
```



In [38]:
```python
plot_sent("the horse raced past the barn fell .", pcfg_garden)
```



The plots illustrate how the surprisal of listener change after hearing each word. For the first sentence, there's no sudden change of surprisals. For the second sentence, however, we can see a sudden increase of surprisal when 'fell' is heard, where the listener is suddenly thrown into confusion. So, this indicates the garden-path effect.

Below is a PCFG that can generate the garden path sentence "The old man the boats."

In [39]:
```python
pcfg_boats_v1 = [("ROOT -> S", 1.0),
                 ("S -> NP VP .", 1.0),
                 ("NP -> Det N", 0.5),
                 ("NP -> Det Adj N", 0.5),
                 ("VP -> V NP", 1.0),
                 ("Det -> the", 1.0),
                 ("N -> old", 0.25),
                 ("N -> man", 0.25),
                 ("N -> children", 0.25),
                 ("N -> boats", 0.25),
                 ("V -> man", 0.5),
                 ("V -> operate", 0.5),
                 ("Adj -> old", 0.5),
                 ("Adj -> elderly", 0.5)
                ]
```

In [40]:
```python
possible_sentences("the old", pcfg_boats_v1)
```

Out[40]:
```
([], 0)
```

Coding Q7: Fill in `pcfg_boats` with the probabilities of the rules so that the probabilities give rise to a garden path effect, as demonstrated by the `incremental_surprisals` or `plot_sent` function. You should also make sure that no garden path effect is observed for the sentence "The children operate the boats."

In [41]:
```python
pcfg_boats = [("ROOT -> S", 1.0),
              ("S -> NP VP .", 1.0),
              ("NP -> Det N", 0.75),
              ("NP -> Det Adj N", 0.25),
              ("VP -> V NP", 0.8),
              ("VP -> V", 0.2),
              ("Det -> the", 1.0),
              ("N -> old", 0.25),
              ("N -> man", 0.25),
              ("N -> children", 0.25),
              ("N -> boats", 0.25),
              ("V -> man", 0.1),
              ("V -> operate", 0.9),
              ("Adj -> old", 0.5),
              ("Adj -> elderly", 0.5)
             ]
```

In [42]:
```python
def possible_sentences(prefix, grammar):
    possible_sents = generate_all(grammar)
    total_prob = 0
    answer_sents = []
    for i in possible_sents:
        temp_sent = i[0][:len(prefix)]
        if(prefix in temp_sent):
            answer_sents.append(i)
            total_prob += i[1]
    return answer_sents, total_prob
    #return possible_sents, total_prob
```

You can test this grammar with the following lines, which should show a garden path effect for "the old man the boats ." but not for "the children operate the boats ."

In [43]:
```python
pcfg_boats = [("ROOT -> S", 1.0),
              ("S -> NP VP .", 1.0),
              ("NP -> Det N", 0.6),
              ("NP -> Det Adj N", 0.4),
              ("VP -> V NP", 1.0),
              ("Det -> the", 1.0),
              ("N -> old", 0.02),
              ("N -> man", 0.18),
              ("N -> children", 0.40),
              ("N -> boats", 0.40),
              ("V -> man", 0.10),
              ("V -> operate", 0.90),
              ("Adj -> old", 0.60),
              ("Adj -> elderly", 0.40)
             ]
```

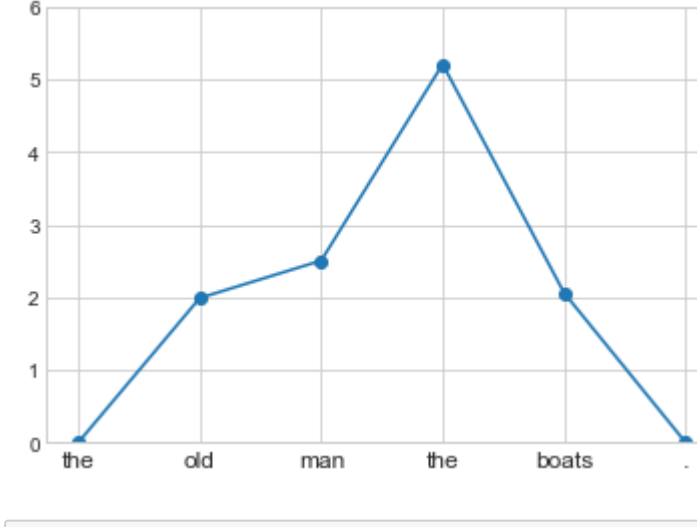
In [44]:
```python
incremental_surprisals("the old man the boats .", pcfg_boats)
```

Out[44]:
```
[-0.0, 1.989, 2.505, 5.209, 2.059, -0.0]
```

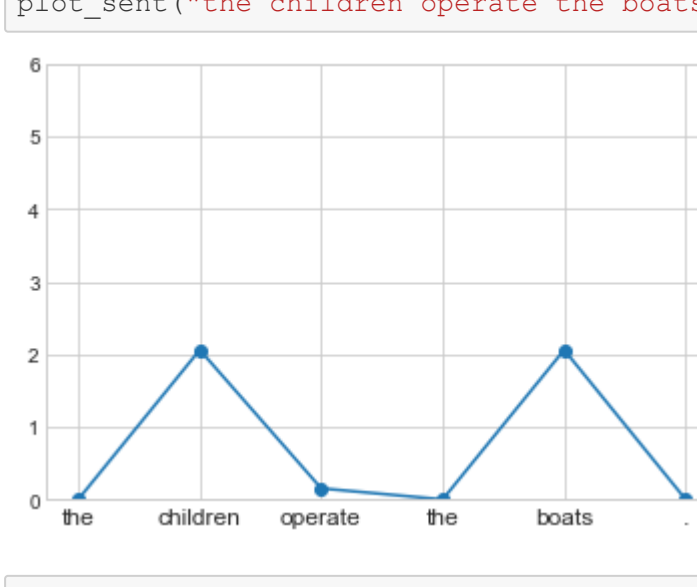
In [45]:
```python
incremental_surprisals("the children operate the boats .", pcfg_boats)
```

Out[45]:
```
[-0.0, 2.059, 0.152, -0.0, 2.059, -0.0]
```

In [46]:
```python
plot_sent("the old man the boats .", pcfg_boats)
```



In [47]:
```python
plot_sent("the children operate the boats .", pcfg_boats)
```



In [ ]:

DS-GA 1011 Homework 1 - Part 3

N-Gram Language Modeling

In [ ]:
```
import os
import json
import numpy as np
from collections import defaultdict
from tqdm import tqdm
```

Utilities

In [ ]:
```
def load_wikitext(filename='wikitext2-sentencized.json'):
    if not os.path.exists(filename):
        !wget "https://nyu.box.com/shared/static/9kb7l7ci30hb6uahhbssjlq0kctr5ii4.json" -O $filename

    datasets = json.load(open(filename, 'r'))
    for name in datasets:
        datasets[name] = [x.split() for x in datasets[name]]
    vocab = list(set([t for ts in datasets['train'] for t in ts]))
    print("Vocab size: %d" % (len(vocab)))
    return datasets, vocab

def perplexity(model, sequences):
    n_total = 0
    logp_total = 0
    for sequence in sequences:
        logp_total += model.sequence_logp(sequence)
        n_total += len(sequence) + 1
    ppl = 2 ** (- (1.0 / n_total) * logp_total)
    return ppl
```

Additive Smoothing

Fill in the ngram_prob method in the class below:

In [ ]:
```
class NGramAdditive(object):
    def __init__(self, n, delta, vsize):
        self.n = n
        self.delta = delta
        self.count = defaultdict(lambda: defaultdict(float))
        self.total = defaultdict(float)
        self.vsize = vsize

    def estimate(self, sequences):
        for sequence in sequences:
            padded_sequence = ['<bos>']*(self.n-1) + sequence + ['<eos>']
            for i in range(len(padded_sequence) - self.n+1):
                ngram = tuple(padded_sequence[i:i+self.n])
                prefix, word = ngram[:-1], ngram[-1]
                self.count[prefix][word] += 1
                self.total[prefix] += 1

    def sequence_logp(self, sequence):
        padded_sequence = ['<bos>']*(self.n-1) + sequence + ['<eos>']
        total_logp = 0
        for i in range(len(padded_sequence) - self.n+1):
            ngram = tuple(padded_sequence[i:i+self.n])
            total_logp += np.log2(self.ngram_prob(ngram))
        return total_logp

    def ngram_prob(self, ngram):
        prefix, word = ngram[:-1], ngram[-1]
        prob = (self.count[prefix][word] + self.delta) / (self.total[prefix] + self.delta * self.vsize)
        return prob
```

In [ ]:
```
datasets, vocab = load_wikitext()

delta = 0.0005
for n in [2, 3, 4]:
    lm = NGramAdditive(n=n, delta=delta, vsize=len(vocab)+1)  # +1 is for <eos>
    lm.estimate(datasets['train'])

    print("Baseline (Additive smoothing, n=%d, delta=%.4f)) Train Perplexity: %.3f" % (n, delta, perple
xity(lm, datasets['train'])))
    print("Baseline (Additive smoothing, n=%d, delta=%.4f)) Valid Perplexity: %.3f" % (n, delta, perple
xity(lm, datasets['valid'])))
```

```
--2020-10-07 00:00:10--  https://nyu.box.com/shared/static/9kb7l7ci30hb6uahhbssjlq0kctr5ii4.json
Resolving nyu.box.com (nyu.box.com)... 107.152.24.197
Connecting to nyu.box.com (nyu.box.com)|107.152.24.197|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: /public/static/9kb7l7ci30hb6uahhbssjlq0kctr5ii4.json [following]
--2020-10-07 00:00:10--  https://nyu.box.com/public/static/9kb7l7ci30hb6uahhbssjlq0kctr5ii4.json
Reusing existing connection to nyu.box.com:443.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://nyu.app.box.com/public/static/9kb7l7ci30hb6uahhbssjlq0kctr5ii4.json [following]
--2020-10-07 00:00:10--  https://nyu.app.box.com/public/static/9kb7l7ci30hb6uahhbssjlq0kctr5ii4.json
Resolving nyu.app.box.com (nyu.app.box.com)... 107.152.24.201
Connecting to nyu.app.box.com (nyu.app.box.com)|107.152.24.201|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://public.boxcloud.com/d/1/b1!vS-um405cYBNyzA6zMvQy66NXjGXx5NUP8cPzIpX--MxR6H12MMCM-wh
KQ7uOHysjikXJICjCLvCf4GClz_7y7OA9U5Ia_wnnRFh35bMGspblVggbJA6XPg74UO6ON-0aMTmgk4xns17SVeWzjBVzQvao9qys
Itlr41ZShCxzaIrrXGs1mwxAsDHwkZn9hS_dIqspztYnOn40r1J_swC-sbYl0Yfg4h2rcpA1HonkG--79LMV_0UrQgmVfz_cw_54q
O8LGV5JirkOc5qsKvPilVbXVSKJE6DzfkKTLu0vNoLoCasX7y53uMbYZd0KLjJwvEcoYlfjfkmZY9NX1U2wySpYsWOJgjeTnkx3vd
k91Egd7fj2X64yGgM9GA3AzajnUmcP69QZRV823rMG1D-FK1OdYryCM0MxcMcNBtdlDMhgY1k3as5WER3agncNDl4mooK_zPyAE50
fbUz8gmhCKOALds-VPr2Jw3O5kXAZ0xbH4Emj0M-M8Vh7vsbw7Oy-W-x775T3UYypfOW2i3oapQ-0-ZOlkb_VbaKoQwlHtyrv3_TR
2FFjYe6gt4XX68Iu8VOLrUPnAI6qtcnBwoy3ZP0j6N_USBiW4DNxz-VZIw4LStoM94Zc5GNn_Wqkr352m6M2NwNNJtCofkqnrTLAr
dx86cYmMLj1o55IZam_TkQazU78vRGq6tc3_Gv8yTkq7gdav23hfc6djYaLf9VzdAtzg5WMyOvDLZQOHOFhUCB8xj3BILuUOQBOD2
IwE5YWm2VA5EqFx1ABM2SpKOIZyM0InlfzUrby9njbI8__DUK8R-0dhA0B6x3wPVgJkDtnvzlazQKugYjHJH-MeaaUpaHVUceSDzz
ccF77NzdNsW2s0Y9QpiIoH2yz_ojd46CzCNEDjYB9nJbxuhJo-zcyLSmPMM-KljQbNg1doBbXspp_ZCnalh8C4T_PHUdmvOylHj07
cWlQzpTgbPPne1Tr6209UTEHri2Q4z1bRZOM35AH8kC34CcSTH5oVvirnq9QowD7u962CRIzC3FRqXxC5bCdfVaiPg1O8bbDxWGn9
UuEIdxio8fiurd2865xUIvqB9LkQcIEcKbHEg6OOdgTYottsJ-sTjiRcpK1Ef5UlePrqm4rrYQj2GvP4_HEG4ijw7p4oi4Hh0n4Y2
f3oSFQ1woeZ7vX9SQH26hezEgUqhyTSaxuRfj9ZZfS4XyVuopJUKqIpQztuj2tlaw72UD1fenWJe0GWKg2b_2tyJ3qfqYyz-f817A
jhe5q82ucvxu-kozijaNvJoeeXYX811rEDBzXLq1gDV8OHlrWsSaDp_bnbUTJxf-MVpxVga5u_ZQR3BLuVSWH6eStQhBrbVJkK4cy
iS13Ff2lbM9GuSfEQSGXiiQLyPRVg1Fea2aRln0QasmfjsJtDJn83AG6pA./download [following]
--2020-10-07 00:00:11--  https://public.boxcloud.com/d/1/b1!vS-um405cYBNyzA6zMvQy66NXjGXx5NUP8cPzIpX-
-MxR6H12MMCM-whKQ7uOHysjikXJICjCLvCf4GClz_7y7OA9U5Ia_wnnRFh35bMGspblVggbJA6XPg74UO6ON-0aMTmgk4xns17SV
eWzjBVzQvao9qysItlr41ZShCxzaIrrXGs1mwxAsDHwkZn9hS_dIqspztYnOn40r1J_swC-sbYl0Yfg4h2rcpA1HonkG--79LMV_0
UrQgmVfz_cw_54qO8LGV5JirkOc5qsKvPilVbXVSKJE6DzfkKTLu0vNoLoCasX7y53uMbYZd0KLjJwvEcoYlfjfkmZY9NX1U2wySp
YsWOJgjeTnkx3vdk91Egd7fj2X64yGgM9GA3AzajnUmcP69QZRV823rMG1D-FK1OdYryCM0MxcMcNBtdlDMhgY1k3as5WER3agncN
Dl4mooK_zPyAE50fbUz8gmhCKOALds-VPr2Jw3O5kXAZ0xbH4Emj0M-M8Vh7vsbw7Oy-W-x775T3UYypfOW2i3oapQ-0-ZOlkb_Vb
aKoQwlHtyrv3_TR2FFjYe6gt4XX68Iu8VOLrUPnAI6qtcnBwoy3ZP0j6N_USBiW4DNxz-VZIw4LStoM94Zc5GNn_Wqkr352m6M2Nw
NNJtCofkqnrTLArdx86cYmMLj1o55IZam_TkQazU78vRGq6tc3_Gv8yTkq7gdav23hfc6djYaLf9VzdAtzg5WMyOvDLZQOHOFhUCB
8xj3BILuUOQBOD2IwE5YWm2VA5EqFx1ABM2SpKOIZyM0InlfzUrby9njbI8__DUK8R-0dhA0B6x3wPVgJkDtnvzlazQKugYjHJH-M
eaaUpaHVUceSDzzccF77NzdNsW2s0Y9QpiIoH2yz_ojd46CzCNEDjYB9nJbxuhJo-zcyLSmPMM-KljQbNg1doBbXspp_ZCnalh8C4
T_PHUdmvOylHj07cWlQzpTgbPPne1Tr6209UTEHri2Q4z1bRZOM35AH8kC34CcSTH5oVvirnq9QowD7u962CRIzC3FRqXxC5bCdfV
aiPg1O8bbDxWGn9UuEIdxio8fiurd2865xUIvqB9LkQcIEcKbHEg6OOdgTYottsJ-sTjiRcpK1Ef5UlePrqm4rrYQj2GvP4_HEG4i
jw7p4oi4Hh0n4Y2f3oSFQ1woeZ7vX9SQH26hezEgUqhyTSaxuRfj9ZZfS4XyVuopJUKqIpQztuj2tlaw72UD1fenWJe0GWKg2b_2t
yJ3qfqYyz-f817Ajhe5q82ucvxu-kozijaNvJoeeXYX811rEDBzXLq1gDV8OHlrWsSaDp_bnbUTJxf-MVpxVga5u_ZQR3BLuVSWH6
eStQhBrbVJkK4cyiS13Ff2lbM9GuSfEQSGXiiQLyPRVg1Fea2aRln0QasmfjsJtDJn83AG6pA./download
Resolving public.boxcloud.com (public.boxcloud.com)... 107.152.24.200
Connecting to public.boxcloud.com (public.boxcloud.com)|107.152.24.200|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 12714601 (12M) [application/octet-stream]
Saving to: ‘wikitext2-sentencized.json’

wikitext2-sentenciz 100%[===================>]  12.12M  31.8MB/s    in 0.4s

2020-10-07 00:00:11 (31.8 MB/s) - ‘wikitext2-sentencized.json’ saved [12714601/12714601]

Vocab size: 33175
Baseline (Additive smoothing, n=2, delta=0.0005)) Train Perplexity: 90.228
Baseline (Additive smoothing, n=2, delta=0.0005)) Valid Perplexity: 525.825
Baseline (Additive smoothing, n=3, delta=0.0005)) Train Perplexity: 26.768
Baseline (Additive smoothing, n=3, delta=0.0005)) Valid Perplexity: 2577.128
Baseline (Additive smoothing, n=4, delta=0.0005)) Train Perplexity: 19.947
Baseline (Additive smoothing, n=4, delta=0.0005)) Valid Perplexity: 9570.901
```

In [ ]:
```
datasets, vocab = load_wikitext()

delta = 0.005
for n in [2, 3, 4]:
    lm = NGramAdditive(n=n, delta=delta, vsize=len(vocab)+1)  # +1 is for <eos>
    lm.estimate(datasets['train'])

    print("Baseline (Additive smoothing, n=%d, delta=%.4f)) Train Perplexity: %.3f" % (n, delta, perple
xity(lm, datasets['train'])))
    print("Baseline (Additive smoothing, n=%d, delta=%.4f)) Valid Perplexity: %.3f" % (n, delta, perple
xity(lm, datasets['valid'])))
```

```
Vocab size: 33175
Baseline (Additive smoothing, n=2, delta=0.0050)) Train Perplexity: 138.787
Baseline (Additive smoothing, n=2, delta=0.0050)) Valid Perplexity: 475.942
Baseline (Additive smoothing, n=3, delta=0.0050)) Train Perplexity: 116.398
Baseline (Additive smoothing, n=3, delta=0.0050)) Valid Perplexity: 2875.057
Baseline (Additive smoothing, n=4, delta=0.0050)) Train Perplexity: 139.759
Baseline (Additive smoothing, n=4, delta=0.0050)) Valid Perplexity: 10697.654
```

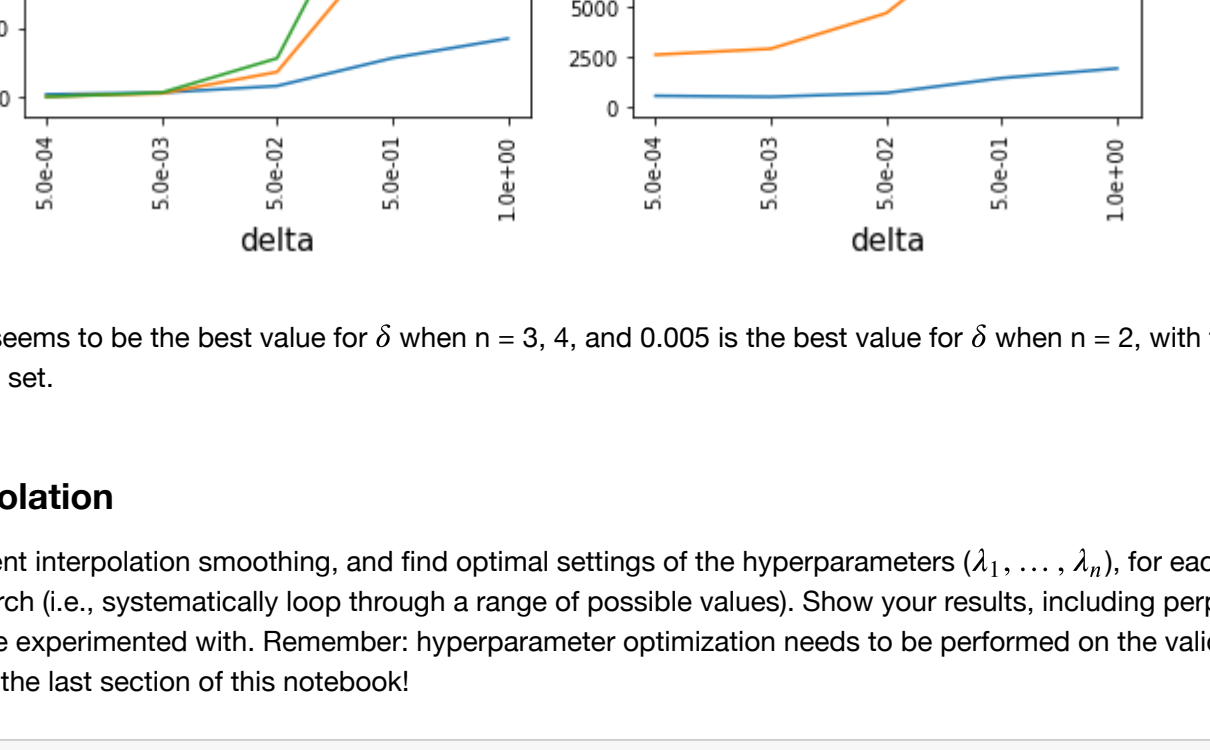
Is 0.005 a good value for δ? Compare the model's perplexity across several orders of magnitude of δ (e.g., 0.0005, 0.005, 0.05, 0.5, and 1).

In [ ]:
```
from tqdm import tqdm
from itertools import product

ns = [2, 3, 4]
deltas = np.array([0.0005, 0.005, 0.05, 0.5, 1])

lms = {}
combos = list(product(ns, deltas))
for n, d in tqdm(combos, total=len(combos)):
    lm = NGramAdditive(n=n, delta=d, vsize=len(vocab)+1)
    lm.estimate(datasets['train'])
    lms[n, d] = lm

T = np.zeros((len(ns), len(deltas)))
V = np.zeros((len(ns), len(deltas)))

for i, n in tqdm(enumerate(ns), total=len(ns)):
    lm = lms[n, 0.0005]
    for j, delta in enumerate(deltas):
        lm.delta = delta

        T[i, j] = perplexity(lm, datasets['train'])
        V[i, j] = perplexity(lm, datasets['valid'])
```

```
100%|██████████| 15/15 [00:59<00:00,  3.97s/it]
100%|██████████| 3/3 [01:59<00:00, 39.96s/it]
```

In [ ]:
```
import matplotlib.pyplot as plt

fig, axs = plt.subplots(1, 2, figsize=(10, 4))
for i in range(len(ns)):
    axs[0].plot(T[i])
    axs[1].plot(V[i])

axs[0].set_title("Train Perplexity", fontsize=15)
axs[1].set_title("Val Perplexity", fontsize=15)

axs[0].set_xticks(np.arange(len(deltas)))
axs[1].set_xticks(np.arange(len(deltas)))

axs[0].set_xticklabels(['{:.1e}'.format(x) for x in deltas], rotation=90)
axs[1].set_xticklabels(['{:.1e}'.format(x) for x in deltas], rotation=90)

axs[0].set_xlabel('delta', fontsize=15)
axs[1].set_xlabel('delta', fontsize=15)

axs[0].set_ylabel('ppl', fontsize=15)
axs[1].set_ylabel('ppl', fontsize=15)

plt.legend(ns, fontsize=15)
```

Out[ ]: `<matplotlib.legend.Legend at 0x7f232052a0f0>`



0.0005 seems to be the best value for δ when n = 3, 4, and 0.005 is the best value for δ when n = 2, with the smallest preplexity results for the valid set.

Interpolation

Implement interpolation smoothing, and find optimal settings of the hyperparameters $(\lambda_1, \ldots, \lambda_n)$, for each of $n \in \{2, 3, 4\}$. You can use grid search (i.e., systematically loop through a range of possible values). Show your results, including perplexity for all of the lambda values you have experimented with. Remember: hyperparameter optimization needs to be performed on the validation set -- do not touch the test set until the last section of this notebook!

In [ ]:
```
class NGramInterpolation(object):
    def __init__(self, n=2, lmd_0=0, lmd_1=0, lmd_2=0, lmd_3=0, vsize=0):
        self.n = n
        self.lmd_0 = lmd_0
        self.lmd_1 = lmd_1
        self.lmd_2 = lmd_2
        self.lmd_3 = lmd_3
        self.vsize = vsize
        self.all_count = []
        self.all_pre_total = []
        self.all_word_total = []

    def set_detla(self, new_delta):
        lmd_0 = new_delta[0]
        lmd_1 = new_delta[1]
        lmd_2 = new_delta[2]
        lmd_3 = new_delta[3]

    def estimate(self, sequences):
        for num in range(2,self.n+1):
            count = defaultdict(lambda: defaultdict(float))
            pre_total = defaultdict(float)
            word_total = defaultdict(float)
            for sequence in sequences:
                padded_sequence = ['<bos>']*(num-1) + sequence + ['<eos>']
                for i in range(len(padded_sequence) - num+1):
                    ngram = tuple(padded_sequence[i:i+num])
                    prefix, word = ngram[:-1], ngram[-1]
                    count[prefix][word] += 1
                    pre_total[prefix] += 1
                    word_total[word] += 1
            self.all_count.append(count)
            self.all_pre_total.append(pre_total)
            self.all_word_total.append(word_total)

    def sequence_logp(self, sequence):
        padded_sequence = ['<bos>']*(self.n-1) + sequence + ['<eos>']
        total_logp = 0
        for i in range(len(padded_sequence) - self.n+1):
            ngram = tuple(padded_sequence[i:i+self.n])
            total_logp += np.log2(self.ngram_prob(ngram))
        return total_logp

    def ngram_prob(self, ngram):
        prefix, word = ngram[:-1], ngram[-1]
        uni_prob = self.all_word_total[0][word] / self.vsize
        prob = self.lmd_0 * uni_prob + (1 - self.lmd_0) * 1 / self.vsize

        if self.n >= 2:
            if self.all_pre_total[0][prefix] == 0:
                bi_prob = 1 / self.vsize
            else:
                bi_prob = self.all_count[0][prefix][word] / self.all_pre_total[0][prefix]
            prob = self.lmd_1 * bi_prob + (1 - self.lmd_1) * prob

            if self.n >= 3:
                if self.all_pre_total[1][prefix] == 0:
                    tri_prob = 1/self.vsize
                else:
                    tri_prob = self.all_count[1][prefix][word] / self.all_pre_total[1][prefix]
                prob = self.lmd_2 * tri_prob + (1 - self.lmd_2) * prob

                if self.n == 4:
                    if self.all_pre_total[2][prefix] == 0:
                        four_prob = 1/self.vsize
                    else:
                        four_prob = self.all_count[2][prefix][word] / self.all_pre_total[2][prefix]
                    prob = self.lmd_3 * four_prob + (1 - self.lmd_3) * prob

        return prob
```

In [ ]:
```
import itertools
d = [(a, 10-a) for a in range(1,10)]
d_2 = []
for i in d:
    d_2.append((i[0]/10, i[1]/10, 0, 0))

d = [(a, b-a, 10-b) for a, b in itertools.combinations(range(1, 10), 2)]
d_3 = []
for i in d:
    d_3.append((i[0]/10, i[1]/10, i[2]/10, 0))

d = [(a, b-a, c-b, 10-c) for a, b, c in itertools.combinations(range(1, 10), 3)]
d_4 = []
for i in d:
    d_4.append((i[0]/10, i[1]/10, i[2]/10, i[3]/10))
```

In [ ]:
```
datasets, vocab = load_wikitext()

for n in [2]:
    for i in tqdm(d_2):
        lm = NGramInterpolation(n=n, lmd_0=i[0], lmd_1=i[1], lmd_2=i[2], lmd_3=i[3], vsize=len(vocab)+1
)  # +1 is for <eos>
        lm.estimate(datasets['train'])
        print("Baseline (Interpolation smoothing, n=%d, delta_1=%.4f, delta_2=%.4f, delta_3=%.4f, delta
_4=%.4f)) Train Perplexity: %.3f" % (n, i[0], i[1], i[2], i[3], perplexity(lm, datasets['train'])))
        print("Baseline (Interpolation smoothing, n=%d, delta_1=%.4f, delta_2=%.4f, delta_3=%.4f, delta
_4=%.4f)) Valid Perplexity: %.3f" % (n, i[0], i[1], i[2], i[3], perplexity(lm, datasets['valid'])))
        print("")
```

```
  0%|          | 0/9 [00:00<?, ?it/s]
Vocab size: 33175
Baseline (Interpolation smoothing, n=2, delta_1=0.1000, delta_2=0.9000, delta_3=0.0000, delta_4=0.000
0)) Train Perplexity: 73.006
 11%|█         | 1/9 [00:09<01:17,  9.65s/it]
Baseline (Interpolation smoothing, n=2, delta_1=0.1000, delta_2=0.9000, delta_3=0.0000, delta_4=0.000
0)) Valid Perplexity: 194.200

Baseline (Interpolation smoothing, n=2, delta_1=0.2000, delta_2=0.8000, delta_3=0.0000, delta_4=0.000
0)) Train Perplexity: 60.399
 22%|██        | 2/9 [00:19<01:07,  9.66s/it]
Baseline (Interpolation smoothing, n=2, delta_1=0.2000, delta_2=0.8000, delta_3=0.0000, delta_4=0.000
0)) Valid Perplexity: 119.467

Baseline (Interpolation smoothing, n=2, delta_1=0.3000, delta_2=0.7000, delta_3=0.0000, delta_4=0.000
0)) Train Perplexity: 48.675
 33%|███       | 3/9 [00:29<00:58,  9.68s/it]
Baseline (Interpolation smoothing, n=2, delta_1=0.3000, delta_2=0.7000, delta_3=0.0000, delta_4=0.000
0)) Valid Perplexity: 80.776

Baseline (Interpolation smoothing, n=2, delta_1=0.4000, delta_2=0.6000, delta_3=0.0000, delta_4=0.000
0)) Train Perplexity: 39.445
 44%|████      | 4/9 [00:38<00:48,  9.68s/it]
Baseline (Interpolation smoothing, n=2, delta_1=0.4000, delta_2=0.6000, delta_3=0.0000, delta_4=0.000
0)) Valid Perplexity: 57.870

Baseline (Interpolation smoothing, n=2, delta_1=0.5000, delta_2=0.5000, delta_3=0.0000, delta_4=0.000
0)) Train Perplexity: 32.451
 56%|█████     | 5/9 [00:48<00:38,  9.63s/it]
Baseline (Interpolation smoothing, n=2, delta_1=0.5000, delta_2=0.5000, delta_3=0.0000, delta_4=0.000
0)) Valid Perplexity: 43.289

Baseline (Interpolation smoothing, n=2, delta_1=0.6000, delta_2=0.4000, delta_3=0.0000, delta_4=0.000
0)) Train Perplexity: 27.180
 67%|██████    | 6/9 [00:58<00:28,  9.67s/it]
Baseline (Interpolation smoothing, n=2, delta_1=0.6000, delta_2=0.4000, delta_3=0.0000, delta_4=0.000
0)) Valid Perplexity: 33.509

Baseline (Interpolation smoothing, n=2, delta_1=0.7000, delta_2=0.3000, delta_3=0.0000, delta_4=0.000
0)) Train Perplexity: 23.209
 78%|███████   | 7/9 [01:07<00:19,  9.64s/it]
Baseline (Interpolation smoothing, n=2, delta_1=0.7000, delta_2=0.3000, delta_3=0.0000, delta_4=0.000
0)) Valid Perplexity: 26.681

Baseline (Interpolation smoothing, n=2, delta_1=0.8000, delta_2=0.2000, delta_3=0.0000, delta_4=0.000
0)) Train Perplexity: 20.254
 89%|████████  | 8/9 [01:17<00:09,  9.68s/it]
Baseline (Interpolation smoothing, n=2, delta_1=0.8000, delta_2=0.2000, delta_3=0.0000, delta_4=0.000
0)) Valid Perplexity: 21.770

Baseline (Interpolation smoothing, n=2, delta_1=0.9000, delta_2=0.1000, delta_3=0.0000, delta_4=0.000
0)) Train Perplexity: 18.216
100%|██████████| 9/9 [01:26<00:00,  9.64s/it]
Baseline (Interpolation smoothing, n=2, delta_1=0.9000, delta_2=0.1000, delta_3=0.0000, delta_4=0.000
0)) Valid Perplexity: 18.183
```

In [ ]:
```
for n in [3]:
    for i in tqdm(d_3):
        lm = NGramInterpolation(n=n, lmd_0=i[0], lmd_1=i[1], lmd_2=i[2], lmd_3=i[3], vsize=len(vocab)+1
)  # +1 is for <eos>
        lm.estimate(datasets['train'])
        print("Baseline (Interpolation smoothing, n=%d, delta_1=%.4f, delta_2=%.4f, delta_3=%.4f, delta
_4=%.4f)) Train Perplexity: %.3f" % (n, i[0], i[1], i[2], i[3], perplexity(lm, datasets['train'])))
        print("Baseline (Interpolation smoothing, n=%d, delta_1=%.4f, delta_2=%.4f, delta_3=%.4f, delta
_4=%.4f)) Valid Perplexity: %.3f" % (n, i[0], i[1], i[2], i[3], perplexity(lm, datasets['valid'])))
        print("")
```

```
  0%|          | 0/36 [00:00<?, ?it/s]
Baseline (Interpolation smoothing, n=3, delta_1=0.1000, delta_2=0.1000, delta_3=0.8000, delta_4=0.000
0)) Train Perplexity: 9.249
  3%|▎         | 1/36 [00:14<08:41, 14.90s/it]
Baseline (Interpolation smoothing, n=3, delta_1=0.1000, delta_2=0.1000, delta_3=0.8000, delta_4=0.000
0)) Valid Perplexity: 230.696

Baseline (Interpolation smoothing, n=3, delta_1=0.1000, delta_2=0.2000, delta_3=0.7000, delta_4=0.000
0)) Train Perplexity: 10.196
  6%|▌         | 2/36 [00:32<08:55, 15.74s/it]
Baseline (Interpolation smoothing, n=3, delta_1=0.1000, delta_2=0.2000, delta_3=0.7000, delta_4=0.000
0)) Valid Perplexity: 200.681

Baseline (Interpolation smoothing, n=3, delta_1=0.1000, delta_2=0.3000, delta_3=0.6000, delta_4=0.000
0)) Train Perplexity: 11.499
  8%|▊         | 3/36 [00:47<08:32, 15.52s/it]
Baseline (Interpolation smoothing, n=3, delta_1=0.1000, delta_2=0.3000, delta_3=0.6000, delta_4=0.000
0)) Valid Perplexity: 189.698

Baseline (Interpolation smoothing, n=3, delta_1=0.1000, delta_2=0.4000, delta_3=0.5000, delta_4=0.000
0)) Train Perplexity: 13.346
 11%|█         | 4/36 [01:02<08:12, 15.38s/it]
Baseline (Interpolation smoothing, n=3, delta_1=0.1000, delta_2=0.4000, delta_3=0.5000, delta_4=0.000
0)) Valid Perplexity: 190.059

Baseline (Interpolation smoothing, n=3, delta_1=0.1000, delta_2=0.5000, delta_3=0.4000, delta_4=0.000
0)) Train Perplexity: 16.114
 14%|█▍        | 5/36 [01:17<07:53, 15.26s/it]
Baseline (Interpolation smoothing, n=3, delta_1=0.1000, delta_2=0.5000, delta_3=0.4000, delta_4=0.000
0)) Valid Perplexity: 200.649

Baseline (Interpolation smoothing, n=3, delta_1=0.1000, delta_2=0.6000, delta_3=0.3000, delta_4=0.000
0)) Train Perplexity: 20.656
 17%|█▋        | 6/36 [01:35<07:57, 15.90s/it]
Baseline (Interpolation smoothing, n=3, delta_1=0.1000, delta_2=0.6000, delta_3=0.3000, delta_4=0.000
0)) Valid Perplexity: 224.528

Baseline (Interpolation smoothing, n=3, delta_1=0.1000, delta_2=0.7000, delta_3=0.2000, delta_4=0.000
0)) Train Perplexity: 29.392
 19%|█▉        | 7/36 [01:49<07:30, 15.55s/it]
Baseline (Interpolation smoothing, n=3, delta_1=0.1000, delta_2=0.7000, delta_3=0.2000, delta_4=0.000
0)) Valid Perplexity: 272.197

Baseline (Interpolation smoothing, n=3, delta_1=0.1000, delta_2=0.8000, delta_3=0.1000, delta_4=0.000
0)) Train Perplexity: 53.069
 22%|██▏       | 8/36 [02:04<07:10, 15.39s/it]
Baseline (Interpolation smoothing, n=3, delta_1=0.1000, delta_2=0.8000, delta_3=0.1000, delta_4=0.000
0)) Valid Perplexity: 381.367

Baseline (Interpolation smoothing, n=3, delta_1=0.2000, delta_2=0.1000, delta_3=0.7000, delta_4=0.000
0)) Train Perplexity: 9.224
 25%|██▌       | 9/36 [02:19<06:52, 15.27s/it]
Baseline (Interpolation smoothing, n=3, delta_1=0.2000, delta_2=0.1000, delta_3=0.7000, delta_4=0.000
0)) Valid Perplexity: 121.949

Baseline (Interpolation smoothing, n=3, delta_1=0.2000, delta_2=0.2000, delta_3=0.6000, delta_4=0.000
0)) Train Perplexity: 10.167
 28%|██▊       | 10/36 [02:37<06:54, 15.96s/it]
Baseline (Interpolation smoothing, n=3, delta_1=0.2000, delta_2=0.2000, delta_3=0.6000, delta_4=0.000
0)) Valid Perplexity: 112.651

Baseline (Interpolation smoothing, n=3, delta_1=0.2000, delta_2=0.3000, delta_3=0.5000, delta_4=0.000
0)) Train Perplexity: 11.533
 31%|███       | 11/36 [02:52<06:31, 15.66s/it]
Baseline (Interpolation smoothing, n=3, delta_1=0.2000, delta_2=0.3000, delta_3=0.5000, delta_4=0.000
0)) Valid Perplexity: 110.239

Baseline (Interpolation smoothing, n=3, delta_1=0.2000, delta_2=0.4000, delta_3=0.4000, delta_4=0.000
0)) Train Perplexity: 13.583
 33%|███▎      | 12/36 [03:07<06:10, 15.42s/it]
Baseline (Interpolation smoothing, n=3, delta_1=0.2000, delta_2=0.4000, delta_3=0.4000, delta_4=0.000
0)) Valid Perplexity: 113.301

Baseline (Interpolation smoothing, n=3, delta_1=0.2000, delta_2=0.5000, delta_3=0.3000, delta_4=0.000
0)) Train Perplexity: 16.892
 36%|███▌      | 13/36 [03:22<05:51, 15.28s/it]
Baseline (Interpolation smoothing, n=3, delta_1=0.2000, delta_2=0.5000, delta_3=0.3000, delta_4=0.000
0)) Valid Perplexity: 122.504

Baseline (Interpolation smoothing, n=3, delta_1=0.2000, delta_2=0.6000, delta_3=0.2000, delta_4=0.000
0)) Train Perplexity: 23.006
 39%|███▉      | 14/36 [03:39<05:49, 15.91s/it]
Baseline (Interpolation smoothing, n=3, delta_1=0.2000, delta_2=0.6000, delta_3=0.2000, delta_4=0.000
0)) Valid Perplexity: 141.139

Baseline (Interpolation smoothing, n=3, delta_1=0.2000, delta_2=0.7000, delta_3=0.1000, delta_4=0.000
0)) Train Perplexity: 38.105
 42%|████▏     | 15/36 [03:54<05:26, 15.56s/it]
Baseline (Interpolation smoothing, n=3, delta_1=0.2000, delta_2=0.7000, delta_3=0.1000, delta_4=0.000
0)) Valid Perplexity: 179.871

Baseline (Interpolation smoothing, n=3, delta_1=0.3000, delta_2=0.1000, delta_3=0.6000, delta_4=0.000
0)) Train Perplexity: 9.088
 44%|████▍     | 16/36 [04:08<05:06, 15.32s/it]
Baseline (Interpolation smoothing, n=3, delta_1=0.3000, delta_2=0.1000, delta_3=0.6000, delta_4=0.000
0)) Valid Perplexity: 78.315

Baseline (Interpolation smoothing, n=3, delta_1=0.3000, delta_2=0.2000, delta_3=0.5000, delta_4=0.000
0)) Train Perplexity: 10.107
 47%|████▋     | 17/36 [04:23<04:48, 15.17s/it]
Baseline (Interpolation smoothing, n=3, delta_1=0.3000, delta_2=0.2000, delta_3=0.5000, delta_4=0.000
0)) Valid Perplexity: 75.126

Baseline (Interpolation smoothing, n=3, delta_1=0.3000, delta_2=0.3000, delta_3=0.4000, delta_4=0.000
0)) Train Perplexity: 11.647
 50%|█████     | 18/36 [04:41<04:46, 15.89s/it]
Baseline (Interpolation smoothing, n=3, delta_1=0.3000, delta_2=0.3000, delta_3=0.4000, delta_4=0.000
0)) Valid Perplexity: 75.470

Baseline (Interpolation smoothing, n=3, delta_1=0.3000, delta_2=0.4000, delta_3=0.3000, delta_4=0.000
0)) Train Perplexity: 14.102
 53%|█████▎    | 19/36 [04:56<04:24, 15.55s/it]
Baseline (Interpolation smoothing, n=3, delta_1=0.3000, delta_2=0.4000, delta_3=0.3000, delta_4=0.000
0)) Valid Perplexity: 79.293

Baseline (Interpolation smoothing, n=3, delta_1=0.3000, delta_2=0.5000, delta_3=0.2000, delta_4=0.000
0)) Train Perplexity: 18.491
 56%|█████▌    | 20/36 [05:10<04:05, 15.33s/it]
Baseline (Interpolation smoothing, n=3, delta_1=0.3000, delta_2=0.5000, delta_3=0.2000, delta_4=0.000
0)) Valid Perplexity: 87.753

Baseline (Interpolation smoothing, n=3, delta_1=0.3000, delta_2=0.6000, delta_3=0.1000, delta_4=0.000
0)) Train Perplexity: 28.606
 58%|█████▊    | 21/36 [05:25<03:47, 15.14s/it]
Baseline (Interpolation smoothing, n=3, delta_1=0.3000, delta_2=0.6000, delta_3=0.1000, delta_4=0.000
0)) Valid Perplexity: 104.689

Baseline (Interpolation smoothing, n=3, delta_1=0.4000, delta_2=0.1000, delta_3=0.5000, delta_4=0.000
0)) Train Perplexity: 8.946
 61%|██████    | 22/36 [05:40<03:29, 14.94s/it]
Baseline (Interpolation smoothing, n=3, delta_1=0.4000, delta_2=0.1000, delta_3=0.5000, delta_4=0.000
0)) Valid Perplexity: 55.287

Baseline (Interpolation smoothing, n=3, delta_1=0.4000, delta_2=0.2000, delta_3=0.4000, delta_4=0.000
0)) Train Perplexity: 10.114
 64%|██████▍   | 23/36 [05:57<03:23, 15.62s/it]
Baseline (Interpolation smoothing, n=3, delta_1=0.4000, delta_2=0.2000, delta_3=0.4000, delta_4=0.000
0)) Valid Perplexity: 54.499

Baseline (Interpolation smoothing, n=3, delta_1=0.4000, delta_2=0.3000, delta_3=0.3000, delta_4=0.000
0)) Train Perplexity: 11.966
 67%|██████▋   | 24/36 [06:11<03:03, 15.32s/it]
Baseline (Interpolation smoothing, n=3, delta_1=0.4000, delta_2=0.3000, delta_3=0.3000, delta_4=0.000
0)) Valid Perplexity: 55.948

Baseline (Interpolation smoothing, n=3, delta_1=0.4000, delta_2=0.4000, delta_3=0.2000, delta_4=0.000
0)) Train Perplexity: 15.203
 69%|██████▉   | 25/36 [06:26<02:45, 15.08s/it]
Baseline (Interpolation smoothing, n=3, delta_1=0.4000, delta_2=0.4000, delta_3=0.2000, delta_4=0.000
0)) Valid Perplexity: 60.031

Baseline (Interpolation smoothing, n=3, delta_1=0.4000, delta_2=0.5000, delta_3=0.1000, delta_4=0.000
0)) Train Perplexity: 22.307
 72%|███████▏  | 26/36 [06:41<02:29, 14.96s/it]
Baseline (Interpolation smoothing, n=3, delta_1=0.4000, delta_2=0.5000, delta_3=0.1000, delta_4=0.000
0)) Valid Perplexity: 68.352

Baseline (Interpolation smoothing, n=3, delta_1=0.5000, delta_2=0.1000, delta_3=0.4000, delta_4=0.000
0)) Train Perplexity: 8.875
 75%|███████▌  | 27/36 [06:58<02:20, 15.58s/it]
Baseline (Interpolation smoothing, n=3, delta_1=0.5000, delta_2=0.1000, delta_3=0.4000, delta_4=0.000
0)) Valid Perplexity: 41.400

Baseline (Interpolation smoothing, n=3, delta_1=0.5000, delta_2=0.2000, delta_3=0.3000, delta_4=0.000
0)) Train Perplexity: 10.295
 78%|███████▊  | 28/36 [07:12<02:02, 15.29s/it]
Baseline (Interpolation smoothing, n=3, delta_1=0.5000, delta_2=0.2000, delta_3=0.3000, delta_4=0.000
0)) Valid Perplexity: 41.709

Baseline (Interpolation smoothing, n=3, delta_1=0.5000, delta_2=0.3000, delta_3=0.2000, delta_4=0.000
0)) Train Perplexity: 12.740
 81%|████████  | 29/36 [07:27<01:46, 15.18s/it]
Baseline (Interpolation smoothing, n=3, delta_1=0.5000, delta_2=0.3000, delta_3=0.2000, delta_4=0.000
0)) Valid Perplexity: 43.686

Baseline (Interpolation smoothing, n=3, delta_1=0.5000, delta_2=0.4000, delta_3=0.1000, delta_4=0.000
0)) Train Perplexity: 17.929
 83%|████████▎ | 30/36 [07:42<01:29, 15.00s/it]
Baseline (Interpolation smoothing, n=3, delta_1=0.5000, delta_2=0.4000, delta_3=0.1000, delta_4=0.000
0)) Valid Perplexity: 48.066

Baseline (Interpolation smoothing, n=3, delta_1=0.6000, delta_2=0.1000, delta_3=0.3000, delta_4=0.000
0)) Train Perplexity: 8.962
 86%|████████▌ | 31/36 [08:00<01:19, 15.82s/it]
Baseline (Interpolation smoothing, n=3, delta_1=0.6000, delta_2=0.1000, delta_3=0.3000, delta_4=0.000
0)) Valid Perplexity: 32.333

Baseline (Interpolation smoothing, n=3, delta_1=0.6000, delta_2=0.2000, delta_3=0.2000, delta_4=0.000
0)) Train Perplexity: 10.848
 89%|████████▉ | 32/36 [08:14<01:01, 15.45s/it]
Baseline (Interpolation smoothing, n=3, delta_1=0.6000, delta_2=0.2000, delta_3=0.2000, delta_4=0.000
0)) Valid Perplexity: 33.221

Baseline (Interpolation smoothing, n=3, delta_1=0.6000, delta_2=0.3000, delta_3=0.1000, delta_4=0.000
0)) Train Perplexity: 14.760
 92%|█████████▏| 33/36 [08:29<00:45, 15.24s/it]
Baseline (Interpolation smoothing, n=3, delta_1=0.6000, delta_2=0.3000, delta_3=0.1000, delta_4=0.000
0)) Valid Perplexity: 35.611

Baseline (Interpolation smoothing, n=3, delta_1=0.7000, delta_2=0.1000, delta_3=0.2000, delta_4=0.000
0)) Train Perplexity: 9.363
 94%|█████████▍| 34/36 [08:44<00:30, 15.09s/it]
Baseline (Interpolation smoothing, n=3, delta_1=0.7000, delta_2=0.1000, delta_3=0.2000, delta_4=0.000
0)) Valid Perplexity: 26.114

Baseline (Interpolation smoothing, n=3, delta_1=0.7000, delta_2=0.2000, delta_3=0.1000, delta_4=0.000
0)) Train Perplexity: 12.387
 97%|█████████▋| 35/36 [09:01<00:15, 15.79s/it]
Baseline (Interpolation smoothing, n=3, delta_1=0.7000, delta_2=0.2000, delta_3=0.1000, delta_4=0.000
0)) Valid Perplexity: 27.426

Baseline (Interpolation smoothing, n=3, delta_1=0.8000, delta_2=0.1000, delta_3=0.1000, delta_4=0.000
0)) Train Perplexity: 10.562
100%|██████████| 36/36 [09:15<00:00, 15.44s/it]
Baseline (Interpolation smoothing, n=3, delta_1=0.8000, delta_2=0.1000, delta_3=0.1000, delta_4=0.000
0)) Valid Perplexity: 21.764
```

In [ ]:

```python
for n in [4]:
    for i in tqdm(d_4):
        lm = NGramInterpolation(n=n, lmd_0=i[0], lmd_1=i[1], lmd_2=i[2], lmd_3=i[3], vsize=len(vocab)+1)  # +1 is for <eos>
        lm.estimate(datasets['train'])
        print("Baseline (Interpolation smoothing, n=%d, delta_1=%.4f, delta_2=%.4f, delta_3=%.4f, delta_4=%.4f)) Train Perplexity: %.3f" % (n, i[0], i[1], i[2], i[3], perplexity(lm, datasets['train'])))
        print("Baseline (Interpolation smoothing, n=%d, delta_1=%.4f, delta_2=%.4f, delta_3=%.4f, delta_4=%.4f)) Valid Perplexity: %.3f" % (n, i[0], i[1], i[2], i[3], perplexity(lm, datasets['valid'])))
        print("")
```

```
  0%|          | 0/84 [00:00<?, ?it/s]
Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.1000, delta_3=0.1000, delta_4=0.7000)) Train Perplexity: 3.262
  1%|          | 1/84 [00:24<34:26, 24.90s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.1000, delta_3=0.1000, delta_4=0.7000)) Valid Perplexity: 309.702

Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.1000, delta_3=0.2000, delta_4=0.6000)) Train Perplexity: 3.734
  2%|▏         | 2/84 [00:45<32:22, 23.69s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.1000, delta_3=0.2000, delta_4=0.6000)) Valid Perplexity: 277.921

Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.1000, delta_3=0.3000, delta_4=0.5000)) Train Perplexity: 4.392
  4%|▍         | 3/84 [01:09<32:06, 23.78s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.1000, delta_3=0.3000, delta_4=0.5000)) Valid Perplexity: 265.451

Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.1000, delta_3=0.4000, delta_4=0.4000)) Train Perplexity: 5.369
  5%|▌         | 4/84 [01:30<30:28, 22.86s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.1000, delta_3=0.4000, delta_4=0.4000)) Valid Perplexity: 266.726

Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.1000, delta_3=0.5000, delta_4=0.3000)) Train Perplexity: 6.965
  6%|▌         | 5/84 [01:54<30:33, 23.21s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.1000, delta_3=0.5000, delta_4=0.3000)) Valid Perplexity: 281.693

Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.1000, delta_3=0.6000, delta_4=0.2000)) Train Perplexity: 10.033
  7%|▋         | 6/84 [02:15<29:13, 22.48s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.1000, delta_3=0.6000, delta_4=0.2000)) Valid Perplexity: 315.623

Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.1000, delta_3=0.7000, delta_4=0.1000)) Train Perplexity: 18.416
  8%|▊         | 7/84 [02:39<29:25, 22.92s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.1000, delta_3=0.7000, delta_4=0.1000)) Valid Perplexity: 386.538

Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.2000, delta_3=0.1000, delta_4=0.6000)) Train Perplexity: 3.734
 10%|█         | 8/84 [03:03<29:26, 23.24s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.2000, delta_3=0.1000, delta_4=0.6000)) Valid Perplexity: 277.921

Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.2000, delta_3=0.2000, delta_4=0.5000)) Train Perplexity: 4.387
 11%|█         | 9/84 [03:24<28:12, 22.57s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.2000, delta_3=0.2000, delta_4=0.5000)) Valid Perplexity: 262.383

Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.2000, delta_3=0.3000, delta_4=0.4000)) Train Perplexity: 5.351
 12%|█▏        | 10/84 [03:48<28:19, 22.97s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.2000, delta_3=0.3000, delta_4=0.4000)) Valid Perplexity: 259.615

Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.2000, delta_3=0.4000, delta_4=0.3000)) Train Perplexity: 6.916
 13%|█▎        | 11/84 [04:08<27:04, 22.25s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.2000, delta_3=0.4000, delta_4=0.3000)) Valid Perplexity: 268.446

Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.2000, delta_3=0.5000, delta_4=0.2000)) Train Perplexity: 9.892
 14%|█▍        | 12/84 [04:32<27:16, 22.73s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.2000, delta_3=0.5000, delta_4=0.2000)) Valid Perplexity: 291.674

Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.2000, delta_3=0.6000, delta_4=0.1000)) Train Perplexity: 17.851
 15%|█▌        | 13/84 [04:56<27:18, 23.08s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.2000, delta_3=0.6000, delta_4=0.1000)) Valid Perplexity: 340.020

Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.3000, delta_3=0.1000, delta_4=0.5000)) Train Perplexity: 4.392
 17%|█▋        | 14/84 [05:17<26:08, 22.41s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.3000, delta_3=0.1000, delta_4=0.5000)) Valid Perplexity: 265.451

Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.3000, delta_3=0.2000, delta_4=0.4000)) Train Perplexity: 5.351
 18%|█▊        | 15/84 [05:41<26:17, 22.86s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.3000, delta_3=0.2000, delta_4=0.4000)) Valid Perplexity: 259.615

Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.3000, delta_3=0.3000, delta_4=0.3000)) Train Perplexity: 6.900
 19%|█▉        | 16/84 [06:05<26:23, 23.29s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.3000, delta_3=0.3000, delta_4=0.3000)) Valid Perplexity: 264.327

Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.3000, delta_3=0.4000, delta_4=0.2000)) Train Perplexity: 9.824
 20%|██        | 17/84 [06:26<25:10, 22.55s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.3000, delta_3=0.4000, delta_4=0.2000)) Valid Perplexity: 281.117

Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.3000, delta_3=0.5000, delta_4=0.1000)) Train Perplexity: 17.539
 21%|██▏       | 18/84 [06:50<25:23, 23.08s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.3000, delta_3=0.5000, delta_4=0.1000)) Valid Perplexity: 317.450

Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.4000, delta_3=0.1000, delta_4=0.4000)) Train Perplexity: 5.369
 23%|██▎       | 19/84 [07:11<24:10, 22.32s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.4000, delta_3=0.1000, delta_4=0.4000)) Valid Perplexity: 266.726

Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.4000, delta_3=0.2000, delta_4=0.3000)) Train Perplexity: 6.916
 24%|██▍       | 20/84 [07:35<24:26, 22.91s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.4000, delta_3=0.2000, delta_4=0.3000)) Valid Perplexity: 268.446

Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.4000, delta_3=0.3000, delta_4=0.2000)) Train Perplexity: 9.824
 25%|██▌       | 21/84 [07:56<23:25, 22.31s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.4000, delta_3=0.3000, delta_4=0.2000)) Valid Perplexity: 281.117

Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.4000, delta_3=0.4000, delta_4=0.1000)) Train Perplexity: 17.439
 26%|██▌       | 22/84 [08:20<23:38, 22.88s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.4000, delta_3=0.4000, delta_4=0.1000)) Valid Perplexity: 310.623

Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.5000, delta_3=0.1000, delta_4=0.3000)) Train Perplexity: 6.965
 27%|██▋       | 23/84 [08:44<23:38, 23.25s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.5000, delta_3=0.1000, delta_4=0.3000)) Valid Perplexity: 281.693

Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.5000, delta_3=0.2000, delta_4=0.2000)) Train Perplexity: 9.892
 29%|██▊       | 24/84 [09:05<22:25, 22.43s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.5000, delta_3=0.2000, delta_4=0.2000)) Valid Perplexity: 291.674

Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.5000, delta_3=0.3000, delta_4=0.1000)) Train Perplexity: 17.539
 30%|██▉       | 25/84 [09:29<22:31, 22.91s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.5000, delta_3=0.3000, delta_4=0.1000)) Valid Perplexity: 317.450

Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.6000, delta_3=0.1000, delta_4=0.2000)) Train Perplexity: 10.033
 31%|███       | 26/84 [09:49<21:29, 22.24s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.6000, delta_3=0.1000, delta_4=0.2000)) Valid Perplexity: 315.623

Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.6000, delta_3=0.2000, delta_4=0.1000)) Train Perplexity: 17.851
 32%|███▏      | 27/84 [10:13<21:30, 22.65s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.6000, delta_3=0.2000, delta_4=0.1000)) Valid Perplexity: 340.020

Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.7000, delta_3=0.1000, delta_4=0.1000)) Train Perplexity: 18.416
 33%|███▎      | 28/84 [10:37<21:34, 23.11s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.1000, delta_2=0.7000, delta_3=0.1000, delta_4=0.1000)) Valid Perplexity: 386.538

Baseline (Interpolation smoothing, n=4, delta_1=0.2000, delta_2=0.1000, delta_3=0.1000, delta_4=0.6000)) Train Perplexity: 3.486
 35%|███▍      | 29/84 [10:58<20:28, 22.34s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.2000, delta_2=0.1000, delta_3=0.1000, delta_4=0.6000)) Valid Perplexity: 150.024

Baseline (Interpolation smoothing, n=4, delta_1=0.2000, delta_2=0.1000, delta_3=0.2000, delta_4=0.5000)) Train Perplexity: 4.036
 36%|███▌      | 30/84 [11:22<20:34, 22.87s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.2000, delta_2=0.1000, delta_3=0.2000, delta_4=0.5000)) Valid Perplexity: 140.635

Baseline (Interpolation smoothing, n=4, delta_1=0.2000, delta_2=0.1000, delta_3=0.3000, delta_4=0.4000)) Train Perplexity: 4.848
 37%|███▋      | 31/84 [11:42<19:34, 22.17s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.2000, delta_2=0.1000, delta_3=0.3000, delta_4=0.4000)) Valid Perplexity: 138.398

Baseline (Interpolation smoothing, n=4, delta_1=0.2000, delta_2=0.1000, delta_3=0.4000, delta_4=0.3000)) Train Perplexity: 6.150
 38%|███▊      | 32/84 [12:06<19:41, 22.73s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.2000, delta_2=0.1000, delta_3=0.4000, delta_4=0.3000)) Valid Perplexity: 142.459

Baseline (Interpolation smoothing, n=4, delta_1=0.2000, delta_2=0.1000, delta_3=0.5000, delta_4=0.2000)) Train Perplexity: 8.573
 39%|███▉      | 33/84 [12:30<19:37, 23.08s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.2000, delta_2=0.1000, delta_3=0.5000, delta_4=0.2000)) Valid Perplexity: 154.056

Baseline (Interpolation smoothing, n=4, delta_1=0.2000, delta_2=0.1000, delta_3=0.6000, delta_4=0.1000)) Train Perplexity: 14.752
 40%|████      | 34/84 [12:51<18:39, 22.39s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.2000, delta_2=0.1000, delta_3=0.6000, delta_4=0.1000)) Valid Perplexity: 178.114

Baseline (Interpolation smoothing, n=4, delta_1=0.2000, delta_2=0.2000, delta_3=0.1000, delta_4=0.5000)) Train Perplexity: 4.036
 42%|████▏     | 35/84 [13:16<18:48, 23.02s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.2000, delta_2=0.2000, delta_3=0.1000, delta_4=0.5000)) Valid Perplexity: 140.635

Baseline (Interpolation smoothing, n=4, delta_1=0.2000, delta_2=0.2000, delta_3=0.2000, delta_4=0.4000)) Train Perplexity: 4.836
 43%|████▎     | 36/84 [13:36<17:53, 22.37s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.2000, delta_2=0.2000, delta_3=0.2000, delta_4=0.4000)) Valid Perplexity: 136.651

Baseline (Interpolation smoothing, n=4, delta_1=0.2000, delta_2=0.2000, delta_3=0.3000, delta_4=0.3000)) Train Perplexity: 6.109
 44%|████▍     | 37/84 [14:00<17:54, 22.86s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.2000, delta_2=0.2000, delta_3=0.3000, delta_4=0.3000)) Valid Perplexity: 138.304

Baseline (Interpolation smoothing, n=4, delta_1=0.2000, delta_2=0.2000, delta_3=0.4000, delta_4=0.2000)) Train Perplexity: 8.452
 45%|████▌     | 38/84 [14:21<17:00, 22.18s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.2000, delta_2=0.2000, delta_3=0.4000, delta_4=0.2000)) Valid Perplexity: 146.098

Baseline (Interpolation smoothing, n=4, delta_1=0.2000, delta_2=0.2000, delta_3=0.5000, delta_4=0.1000)) Train Perplexity: 14.315
 46%|████▋     | 39/84 [14:45<17:01, 22.69s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.2000, delta_2=0.2000, delta_3=0.5000, delta_4=0.1000)) Valid Perplexity: 163.127

Baseline (Interpolation smoothing, n=4, delta_1=0.2000, delta_2=0.3000, delta_3=0.1000, delta_4=0.4000)) Train Perplexity: 4.848
 48%|████▊     | 40/84 [15:09<16:55, 23.08s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.2000, delta_2=0.3000, delta_3=0.1000, delta_4=0.4000)) Valid Perplexity: 138.398

Baseline (Interpolation smoothing, n=4, delta_1=0.2000, delta_2=0.3000, delta_3=0.2000, delta_4=0.3000)) Train Perplexity: 6.109
 49%|████▉     | 41/84 [15:30<16:05, 22.45s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.2000, delta_2=0.3000, delta_3=0.2000, delta_4=0.3000)) Valid Perplexity: 138.304

Baseline (Interpolation smoothing, n=4, delta_1=0.2000, delta_2=0.3000, delta_3=0.3000, delta_4=0.2000)) Train Perplexity: 8.413
 50%|█████     | 42/84 [15:54<16:08, 23.06s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.2000, delta_2=0.3000, delta_3=0.3000, delta_4=0.2000)) Valid Perplexity: 143.634

Baseline (Interpolation smoothing, n=4, delta_1=0.2000, delta_2=0.3000, delta_3=0.4000, delta_4=0.1000)) Train Perplexity: 14.111
 51%|█████     | 43/84 [16:15<15:19, 22.42s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.2000, delta_2=0.3000, delta_3=0.4000, delta_4=0.1000)) Valid Perplexity: 156.577

Baseline (Interpolation smoothing, n=4, delta_1=0.2000, delta_2=0.4000, delta_3=0.1000, delta_4=0.3000)) Train Perplexity: 6.150
 52%|█████▏    | 44/84 [16:40<15:19, 22.98s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.2000, delta_2=0.4000, delta_3=0.1000, delta_4=0.3000)) Valid Perplexity: 142.459

Baseline (Interpolation smoothing, n=4, delta_1=0.2000, delta_2=0.4000, delta_3=0.2000, delta_4=0.2000)) Train Perplexity: 8.452
 54%|█████▎    | 45/84 [17:04<15:11, 23.36s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.2000, delta_2=0.4000, delta_3=0.2000, delta_4=0.2000)) Valid Perplexity: 146.098

Baseline (Interpolation smoothing, n=4, delta_1=0.2000, delta_2=0.4000, delta_3=0.3000, delta_4=0.1000)) Train Perplexity: 14.111
 55%|█████▍    | 46/84 [17:25<14:21, 22.66s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.2000, delta_2=0.4000, delta_3=0.3000, delta_4=0.1000)) Valid Perplexity: 156.577

Baseline (Interpolation smoothing, n=4, delta_1=0.2000, delta_2=0.5000, delta_3=0.1000, delta_4=0.2000)) Train Perplexity: 8.573
 56%|█████▌    | 47/84 [17:49<14:17, 23.18s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.2000, delta_2=0.5000, delta_3=0.1000, delta_4=0.2000)) Valid Perplexity: 154.056

Baseline (Interpolation smoothing, n=4, delta_1=0.2000, delta_2=0.5000, delta_3=0.2000, delta_4=0.1000)) Train Perplexity: 14.315
 57%|█████▋    | 48/84 [18:10<13:32, 22.58s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.2000, delta_2=0.5000, delta_3=0.2000, delta_4=0.1000)) Valid Perplexity: 163.127

Baseline (Interpolation smoothing, n=4, delta_1=0.2000, delta_2=0.6000, delta_3=0.1000, delta_4=0.1000)) Train Perplexity: 14.752
 58%|█████▊    | 49/84 [18:35<13:27, 23.09s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.2000, delta_2=0.6000, delta_3=0.1000, delta_4=0.1000)) Valid Perplexity: 178.114

Baseline (Interpolation smoothing, n=4, delta_1=0.3000, delta_2=0.1000, delta_3=0.1000, delta_4=0.5000)) Train Perplexity: 3.723
 60%|█████▉    | 50/84 [18:56<12:43, 22.47s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.3000, delta_2=0.1000, delta_3=0.1000, delta_4=0.5000)) Valid Perplexity: 91.747

Baseline (Interpolation smoothing, n=4, delta_1=0.3000, delta_2=0.1000, delta_3=0.2000, delta_4=0.4000)) Train Perplexity: 4.398
 61%|██████    | 51/84 [19:20<12:38, 22.98s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.3000, delta_2=0.1000, delta_3=0.2000, delta_4=0.4000)) Valid Perplexity: 88.732

Baseline (Interpolation smoothing, n=4, delta_1=0.3000, delta_2=0.1000, delta_3=0.3000, delta_4=0.3000)) Train Perplexity: 5.468
 62%|██████▏   | 52/84 [19:44<12:27, 23.35s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.3000, delta_2=0.1000, delta_3=0.3000, delta_4=0.3000)) Valid Perplexity: 89.414

Baseline (Interpolation smoothing, n=4, delta_1=0.3000, delta_2=0.1000, delta_3=0.4000, delta_4=0.2000)) Train Perplexity: 7.407
 63%|██████▎   | 53/84 [20:05<11:41, 22.64s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.3000, delta_2=0.1000, delta_3=0.4000, delta_4=0.2000)) Valid Perplexity: 93.999

Baseline (Interpolation smoothing, n=4, delta_1=0.3000, delta_2=0.1000, delta_3=0.5000, delta_4=0.1000)) Train Perplexity: 12.105
 64%|██████▍   | 54/84 [20:29<11:34, 23.14s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.3000, delta_2=0.1000, delta_3=0.5000, delta_4=0.1000)) Valid Perplexity: 104.211

Baseline (Interpolation smoothing, n=4, delta_1=0.3000, delta_2=0.2000, delta_3=0.1000, delta_4=0.4000)) Train Perplexity: 4.398
 65%|██████▌   | 55/84 [20:50<10:51, 22.45s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.3000, delta_2=0.2000, delta_3=0.1000, delta_4=0.4000)) Valid Perplexity: 88.732

Baseline (Interpolation smoothing, n=4, delta_1=0.3000, delta_2=0.2000, delta_3=0.2000, delta_4=0.3000)) Train Perplexity: 5.447
 67%|██████▋   | 56/84 [21:15<10:45, 23.06s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.3000, delta_2=0.2000, delta_3=0.2000, delta_4=0.3000)) Valid Perplexity: 88.222

Baseline (Interpolation smoothing, n=4, delta_1=0.3000, delta_2=0.2000, delta_3=0.3000, delta_4=0.2000)) Train Perplexity: 7.331
 68%|██████▊   | 57/84 [21:40<10:37, 23.61s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.3000, delta_2=0.2000, delta_3=0.3000, delta_4=0.2000)) Valid Perplexity: 91.099

Baseline (Interpolation smoothing, n=4, delta_1=0.3000, delta_2=0.2000, delta_3=0.4000, delta_4=0.1000)) Train Perplexity: 11.832
 69%|██████▉   | 58/84 [22:01<09:52, 22.78s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.3000, delta_2=0.2000, delta_3=0.4000, delta_4=0.1000)) Valid Perplexity: 98.485

Baseline (Interpolation smoothing, n=4, delta_1=0.3000, delta_2=0.3000, delta_3=0.1000, delta_4=0.3000)) Train Perplexity: 5.468
 70%|███████   | 59/84 [22:25<09:41, 23.24s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.3000, delta_2=0.3000, delta_3=0.1000, delta_4=0.3000)) Valid Perplexity: 89.414

Baseline (Interpolation smoothing, n=4, delta_1=0.3000, delta_2=0.3000, delta_3=0.2000, delta_4=0.2000)) Train Perplexity: 7.331
 71%|███████▏  | 60/84 [22:50<09:30, 23.75s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.3000, delta_2=0.3000, delta_3=0.2000, delta_4=0.2000)) Valid Perplexity: 91.099

Baseline (Interpolation smoothing, n=4, delta_1=0.3000, delta_2=0.3000, delta_3=0.3000, delta_4=0.1000)) Train Perplexity: 11.745
 73%|███████▎  | 61/84 [23:11<08:48, 22.98s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.3000, delta_2=0.3000, delta_3=0.3000, delta_4=0.1000)) Valid Perplexity: 96.718

Baseline (Interpolation smoothing, n=4, delta_1=0.3000, delta_2=0.4000, delta_3=0.1000, delta_4=0.2000)) Train Perplexity: 7.407
 74%|███████▍  | 62/84 [23:35<08:35, 23.44s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.3000, delta_2=0.4000, delta_3=0.1000, delta_4=0.2000)) Valid Perplexity: 93.999

Baseline (Interpolation smoothing, n=4, delta_1=0.3000, delta_2=0.4000, delta_3=0.2000, delta_4=0.1000)) Train Perplexity: 11.832
 75%|███████▌  | 63/84 [23:57<07:58, 22.78s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.3000, delta_2=0.4000, delta_3=0.2000, delta_4=0.1000)) Valid Perplexity: 98.485

Baseline (Interpolation smoothing, n=4, delta_1=0.3000, delta_2=0.5000, delta_3=0.1000, delta_4=0.1000)) Train Perplexity: 12.105
 76%|███████▌  | 64/84 [24:21<07:46, 23.32s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.3000, delta_2=0.5000, delta_3=0.1000, delta_4=0.1000)) Valid Perplexity: 104.211

Baseline (Interpolation smoothing, n=4, delta_1=0.4000, delta_2=0.1000, delta_3=0.1000, delta_4=0.4000)) Train Perplexity: 4.009
 77%|███████▋  | 65/84 [24:43<07:12, 22.76s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.4000, delta_2=0.1000, delta_3=0.1000, delta_4=0.4000)) Valid Perplexity: 62.722

Baseline (Interpolation smoothing, n=4, delta_1=0.4000, delta_2=0.1000, delta_3=0.2000, delta_4=0.3000)) Train Perplexity: 4.895
 79%|███████▊  | 66/84 [25:07<07:00, 23.36s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.4000, delta_2=0.1000, delta_3=0.2000, delta_4=0.3000)) Valid Perplexity: 62.124

Baseline (Interpolation smoothing, n=4, delta_1=0.4000, delta_2=0.1000, delta_3=0.3000, delta_4=0.2000)) Train Perplexity: 6.471
 80%|███████▉  | 67/84 [25:33<06:45, 23.87s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.4000, delta_2=0.1000, delta_3=0.3000, delta_4=0.2000)) Valid Perplexity: 63.884

Baseline (Interpolation smoothing, n=4, delta_1=0.4000, delta_2=0.1000, delta_3=0.4000, delta_4=0.1000)) Train Perplexity: 10.149
 81%|████████  | 68/84 [25:55<06:12, 23.29s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.4000, delta_2=0.1000, delta_3=0.4000, delta_4=0.1000)) Valid Perplexity: 68.675

Baseline (Interpolation smoothing, n=4, delta_1=0.4000, delta_2=0.2000, delta_3=0.1000, delta_4=0.3000)) Train Perplexity: 4.895
 82%|████████▏ | 69/84 [26:21<06:01, 24.11s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.4000, delta_2=0.2000, delta_3=0.1000, delta_4=0.3000)) Valid Perplexity: 62.124

Baseline (Interpolation smoothing, n=4, delta_1=0.4000, delta_2=0.2000, delta_3=0.2000, delta_4=0.2000)) Train Perplexity: 6.437
 83%|████████▎ | 70/84 [26:43<05:28, 23.49s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.4000, delta_2=0.2000, delta_3=0.2000, delta_4=0.2000)) Valid Perplexity: 62.998

Baseline (Interpolation smoothing, n=4, delta_1=0.4000, delta_2=0.2000, delta_3=0.3000, delta_4=0.1000)) Train Perplexity: 10.002
 85%|████████▍ | 71/84 [27:07<05:09, 23.84s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.4000, delta_2=0.2000, delta_3=0.3000, delta_4=0.1000)) Valid Perplexity: 66.461

Baseline (Interpolation smoothing, n=4, delta_1=0.4000, delta_2=0.3000, delta_3=0.1000, delta_4=0.2000)) Train Perplexity: 6.471
 86%|████████▌ | 72/84 [27:32<04:50, 24.20s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.4000, delta_2=0.3000, delta_3=0.1000, delta_4=0.2000)) Valid Perplexity: 63.884

Baseline (Interpolation smoothing, n=4, delta_1=0.4000, delta_2=0.3000, delta_3=0.2000, delta_4=0.1000)) Train Perplexity: 10.002
 87%|████████▋ | 73/84 [27:54<04:18, 23.50s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.4000, delta_2=0.3000, delta_3=0.2000, delta_4=0.1000)) Valid Perplexity: 66.461

Baseline (Interpolation smoothing, n=4, delta_1=0.4000, delta_2=0.4000, delta_3=0.1000, delta_4=0.1000)) Train Perplexity: 10.149
 88%|████████▊ | 74/84 [28:19<04:00, 24.05s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.4000, delta_2=0.4000, delta_3=0.1000, delta_4=0.1000)) Valid Perplexity: 68.675

Baseline (Interpolation smoothing, n=4, delta_1=0.5000, delta_2=0.1000, delta_3=0.1000, delta_4=0.3000)) Train Perplexity: 4.412
 89%|████████▉ | 75/84 [28:45<03:40, 24.45s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.5000, delta_2=0.1000, delta_3=0.1000, delta_4=0.3000)) Valid Perplexity: 45.920

Baseline (Interpolation smoothing, n=4, delta_1=0.5000, delta_2=0.1000, delta_3=0.2000, delta_4=0.2000)) Train Perplexity: 5.711
 90%|█████████ | 76/84 [29:07<03:09, 23.67s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.5000, delta_2=0.1000, delta_3=0.2000, delta_4=0.2000)) Valid Perplexity: 46.403

Baseline (Interpolation smoothing, n=4, delta_1=0.5000, delta_2=0.1000, delta_3=0.3000, delta_4=0.1000)) Train Perplexity: 8.665
 92%|█████████▏| 77/84 [29:32<02:49, 24.27s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.5000, delta_2=0.1000, delta_3=0.3000, delta_4=0.1000)) Valid Perplexity: 48.732

Baseline (Interpolation smoothing, n=4, delta_1=0.5000, delta_2=0.2000, delta_3=0.1000, delta_4=0.2000)) Train Perplexity: 5.711
 93%|█████████▎| 78/84 [29:54<02:21, 23.60s/it]
Baseline (Interpolation smoothing, n=4, delta_1=0.5000, delta_2=0.2000, delta_3=0.1000, delta_4=0.2000)) Valid Perplexity: 46.403

Baseline (Interpolation smoothing, n=4, delta_1=0.5000, delta_2=0.2000, delta_3=0.2000, delta_4=0.1000)) Train Perplexity: 8.605
 94%|█████████▍| 79/84 [30:20<02:01, 24.26s/it]
```

```
Baseline (Interpolation smoothing, n=4, delta_1=0.5000, delta_2=0.2000, delta_3=0.2000, delta_4=0.100
0)) Valid Perplexity: 48.032

Baseline (Interpolation smoothing, n=4, delta_1=0.5000, delta_2=0.3000, delta_3=0.1000, delta_4=0.100
0)) Train Perplexity: 8.665

 95%|█████████| 80/84 [30:43<01:34, 23.70s/it]

Baseline (Interpolation smoothing, n=4, delta_1=0.5000, delta_2=0.3000, delta_3=0.1000, delta_4=0.100
0)) Valid Perplexity: 48.732

Baseline (Interpolation smoothing, n=4, delta_1=0.6000, delta_2=0.1000, delta_3=0.1000, delta_4=0.200
0)) Train Perplexity: 5.087

 96%|█████████| 81/84 [31:08<01:12, 24.19s/it]

Baseline (Interpolation smoothing, n=4, delta_1=0.6000, delta_2=0.1000, delta_3=0.1000, delta_4=0.200
0)) Valid Perplexity: 35.291

Baseline (Interpolation smoothing, n=4, delta_1=0.6000, delta_2=0.1000, delta_3=0.2000, delta_4=0.100
0)) Train Perplexity: 7.508

 98%|█████████| 82/84 [31:34<00:49, 24.67s/it]

Baseline (Interpolation smoothing, n=4, delta_1=0.6000, delta_2=0.1000, delta_3=0.2000, delta_4=0.100
0)) Valid Perplexity: 36.393

Baseline (Interpolation smoothing, n=4, delta_1=0.6000, delta_2=0.2000, delta_3=0.1000, delta_4=0.100
0)) Train Perplexity: 7.508

 99%|█████████| 83/84 [31:56<00:24, 24.09s/it]

Baseline (Interpolation smoothing, n=4, delta_1=0.6000, delta_2=0.2000, delta_3=0.1000, delta_4=0.100
0)) Valid Perplexity: 36.393

Baseline (Interpolation smoothing, n=4, delta_1=0.7000, delta_2=0.1000, delta_3=0.1000, delta_4=0.100
0)) Train Perplexity: 6.587

100%|█████████| 84/84 [32:22<00:00, 23.12s/it]

Baseline (Interpolation smoothing, n=4, delta_1=0.7000, delta_2=0.1000, delta_3=0.1000, delta_4=0.100
0)) Valid Perplexity: 28.220
```

Now compare the performance of the two smoothing methods on the test set. Use the best hyperparameters you found for each.

When n = 2,

1) Additive Smoothing

In [ ]:
```
delta = 0.005

lm = NGramAdditive(n=2, delta=delta, vsize=len(vocab)+1)  # +1 is for <eos>
lm.estimate(datasets['train'])

print("Baseline (Additive smoothing, n=%d, delta=%.4f)) Test Perplexity: %.3f" % (n, delta, perplexity(
lm, datasets['test'])))
```

```
Baseline (Additive smoothing, n=4, delta=0.0050)) Test Perplexity: 447.054
```

2) Interpolation Smoothing

In [ ]:
```
delta = (0.9000, 0.1000, 0, 0)

lm = NGramInterpolation(n=2, lmd_0=delta[0], lmd_1=delta[1], lmd_2=delta[2], lmd_3=delta[3], vsize=len(
vocab)+1)  # +1 is for <eos>
lm.estimate(datasets['train'])

print("Baseline (Interpolation smoothing, n=%d, delta_1=%.4f, delta_2=%.4f, delta_3=%.4f, delta_4=%.4f
)) Valid Perplexity: %.3f" % (n, delta[0], delta[1], delta[2], delta[3], perplexity(lm, datasets['test'
])))
```

```
Baseline (Interpolation smoothing, n=4, delta_1=0.9000, delta_2=0.1000, delta_3=0.0000, delta_4=0.000
0)) Valid Perplexity: 17.478
```

When n = 3,

1) Additive Smoothing

In [ ]:
```
delta = 0.0005

lm = NGramAdditive(n=3, delta=delta, vsize=len(vocab)+1)  # +1 is for <eos>
lm.estimate(datasets['train'])

print("Baseline (Additive smoothing, n=%d, delta=%.4f)) Test Perplexity: %.3f" % (n, delta, perplexity(
lm, datasets['test'])))
```

```
Baseline (Additive smoothing, n=4, delta=0.0005)) Test Perplexity: 2417.329
```

2) Interpolation Smoothing

In [ ]:
```
delta = (0.8000, 0.1000, 0.1000, 0)

lm = NGramInterpolation(n=3, lmd_0=delta[0], lmd_1=delta[1], lmd_2=delta[2], lmd_3=delta[3], vsize=len(
vocab)+1)  # +1 is for <eos>
lm.estimate(datasets['train'])

print("Baseline (Interpolation smoothing, n=%d, delta_1=%.4f, delta_2=%.4f, delta_3=%.4f, delta_4=%.4f
)) Valid Perplexity: %.3f" % (n, delta[0], delta[1], delta[2], delta[3], perplexity(lm, datasets['test'
])))
```

```
Baseline (Interpolation smoothing, n=4, delta_1=0.8000, delta_2=0.1000, delta_3=0.1000, delta_4=0.000
0)) Valid Perplexity: 20.954
```

When n = 4,

1) Additive Smoothing

In [ ]:
```
delta = 0.0005

lm = NGramAdditive(n=4, delta=delta, vsize=len(vocab)+1)  # +1 is for <eos>
lm.estimate(datasets['train'])

print("Baseline (Additive smoothing, n=%d, delta=%.4f)) Test Perplexity: %.3f" % (n, delta, perplexity(
lm, datasets['test'])))
```

```
Baseline (Additive smoothing, n=4, delta=0.0005)) Test Perplexity: 9209.615
```

2) Interpolation Smoothing

In [ ]:
```
delta = (0.7000, 0.1000, 0.1000, .1000)

lm = NGramInterpolation(n=3, lmd_0=delta[0], lmd_1=delta[1], lmd_2=delta[2], lmd_3=delta[3], vsize=len(
vocab)+1)  # +1 is for <eos>
lm.estimate(datasets['train'])

print("Baseline (Interpolation smoothing, n=%d, delta_1=%.4f, delta_2=%.4f, delta_3=%.4f, delta_4=%.4f
)) Valid Perplexity: %.3f" % (n, delta[0], delta[1], delta[2], delta[3], perplexity(lm, datasets['test'
])))
```

```
Baseline (Interpolation smoothing, n=4, delta_1=0.7000, delta_2=0.1000, delta_3=0.1000, delta_4=0.100
0)) Valid Perplexity: 23.700
```