FIT5217 Assignment 2

Student Name: Yi Jie Ng

Student ID: 31158145

Common functions Implementation (Part A)

1. Import library

In [1]:
```
!pip3 install torch numpy matplotlib
```

```
Requirement already satisfied: torch in c:\users\ngyij\anaconda3\lib\site-packages (2.2.2)
Requirement already satisfied: numpy in c:\users\ngyij\anaconda3\lib\site-packages (1.24.3)
Requirement already satisfied: matplotlib in c:\users\ngyij\anaconda3\lib\site-packages (3.7.2)
Requirement already satisfied: filelock in c:\users\ngyij\anaconda3\lib\site-packages (from torch) (3.9.0)
Requirement already satisfied: typing-extensions>=4.8.0 in c:\users\ngyij\anaconda3\lib\site-packages (from torch) (4.9.0)
Requirement already satisfied: sympy in c:\users\ngyij\anaconda3\lib\site-packages (from torch) (1.11.1)
Requirement already satisfied: networkx in c:\users\ngyij\anaconda3\lib\site-packages (from torch) (3.1)
Requirement already satisfied: jinja2 in c:\users\ngyij\anaconda3\lib\site-packages (from torch) (3.1.2)
Requirement already satisfied: fsspec in c:\users\ngyij\anaconda3\lib\site-packages (from torch) (2023.4.0)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\ngyij\anaconda3\lib\site-packages (from matplotlib) (1.0.5)
Requirement already satisfied: cycler>=0.10 in c:\users\ngyij\anaconda3\lib\site-packages (from matplotlib) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\ngyij\anaconda3\lib\site-packages (from matplotlib) (4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\ngyij\anaconda3\lib\site-packages (from matplotlib) (1.4.4)
Requirement already satisfied: packaging>=20.0 in c:\users\ngyij\anaconda3\lib\site-packages (from matplotlib) (23.1)
Requirement already satisfied: pillow>=6.2.0 in c:\users\ngyij\anaconda3\lib\site-packages (from matplotlib) (9.4.0)
Requirement already satisfied: pyparsing<3.1,>=2.3.1 in c:\users\ngyij\anaconda3\lib\site-packages (from matplotlib) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\ngyij\anaconda3\lib\site-packages (from matplotlib) (2.8.2)
Requirement already satisfied: six>=1.5 in c:\users\ngyij\anaconda3\lib\site-packages (from python-dateutil>=2.7->matplotlib)
(1.16.0)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\ngyij\anaconda3\lib\site-packages (from jinja2->torch) (2.1.1)
Requirement already satisfied: mpmath>=0.19 in c:\users\ngyij\anaconda3\lib\site-packages (from sympy->torch) (1.3.0)
```

In [2]:
```
## Requirements
from __future__ import unicode_literals, print_function, division
from io import open
import unicodedata
import string
import re
import random

import torch
import torch.nn as nn
from torch import optim
import torch.nn.functional as F

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

2. Word Dictionary

In [3]:
```
SOS_token = 0 # start token
EOS_token = 1 # end token

# class which map the word to unique index and vice verca
class Lang:
    def __init__(self, name):
        self.name = name
        self.word2index = {}
        self.word2count = {}
        self.index2word = {0: "SOS", 1: "EOS"}
        self.n_words = 2  # Count SOS and EOS
        self.min_sentence = 150
        self.max_sentence = 0

    def addSentence(self, sentence):
        if len(sentence.split(' ')) > self.max_sentence:
            self.max_sentence = len(sentence.split(' ')) # record the max length of word in dic
        if len(sentence.split(' ')) < self.min_sentence:
            self.min_sentence = len(sentence.split(' ')) # record the min length of word in dic

        for word in sentence.split(' '):
            self.addWord(word)

    def addWord(self, word):
        if word not in self.word2index:
            self.word2index[word] = self.n_words
            self.word2count[word] = 1
            self.index2word[self.n_words] = word
            self.n_words += 1
        else:
            self.word2count[word] += 1
```

3. Text Normalisation

a. Common normalisation techniques

In [4]:
```
# Turn a Unicode string to plain ASCII, thanks to
# https://stackoverflow.com/a/518232/2809427
def unicodeToAscii(s):
    return ''.join(
        c for c in unicodedata.normalize('NFD', s)
        if unicodedata.category(c) != 'Mn'
    )

def removeWord(s):
    # remove redundant words
    remove_words = ["c", "tb", "ts", "lg", "x", "lb", "sl", "tsp", "and", "h", "softened", "optional", "ea", "pk",
                    "cn", "ounce", "up", "ups", "ct", "sm", "md", "oz", "fl",'from', 'hours', 'medium', 'blender',
                    'until', 'serving', 'food', 'boil', 'seeds', 'achieve', 'let', 'fold', 'dissolves', 'syrup',
                    'soften', 'saucepan', 'with', 'strawberrie', 'along', 'least', 'cooled', 'processor',
                    'temperature', 'it', 'any', 'container', 'combine', 'remove', 'mixture', 'room', 'consistency',
                    'serve', 'minutes', 'well', 'sit', 'a', 'puree', 'or', 'chilled', 'cool', 'bowl',
                    'to', 'mixed', 'gently', 'for', 'into', 'heat', 'simmer', 'in', 'if', 'large', 'at',
                    'smooth', 'the', 'allow', 'bring', 'stir', 'blend', 'strain', 'desired', 'freeze',
                    'stirring', 'before', 'pour']

    words = re.split(r'(\s+)', s)
    words_removed = [word for word in words if (word not in remove_words)]
    # Join the filtered words back into a string
    s = ''.join(words_removed)
    return s


def addCommonWords(my_lang1):
    # most common ingredients words
    words_to_add = ["vanilla", "chocolate", "strawberry", "blueberry", "raspberry", "blackberry", "banana", "orange",
                    "lemon", "lime", "apple", "pear", "peach", "plum", "apricot", "kiwi", "melon", "watermelon", "grape",
                    "pineapple", "coconut", "mango", "pomegranate", "fig", "cherry", "peanut", "ice", "almond", "cashew", "walnut",
                    "hazelnut", "pecan", "pistachio", "macadamia", "sesame", "sunflower seed", "pumpkin seed", "flaxseed",
                    "chia seed", "quinoa", "cream", "amaranth", "buckwheat", "oats", "barley", "vanilla ice cream", "wheat", "rice",
                    "teff", "millet", "sorghum", "farro", "bulgur", "kamut", "nutmeg", "cinnamon", "ginger", "cloves", "allspice",
                    "cardamom", "vanilla ice", "almond extract", "lemon juice", "orange juice", "lime extract",
                    "coconut extract", "maple syrup", "honey", "molasses", "agave nectar", "corn syrup", "rice syrup",
                    "barley malt", "sugar", "brown sugar", "powdered sugar", "molasses", "cornstarch", "tapioca"]

    for word in words_to_add:
        my_lang1.addWord(word)
    return my_lang1

# Lowercase, trim, and remove non-letter and number characters
def normalizeString(s):
    s = unicodeToAscii(s.lower().strip())
```

```
  s = re.sub(r"([,.!?])", r" \1", s)
  s = re.sub(r"[^a-zA-Z\s]+", r" ", s)
    s = re.sub(r"[0-9]+", r"NUM", s) # replace number with unified NUM token
  s = removeWord(s)
  return s

Lowercase, trim, and remove non-letter and number characters
ef normalizeRecipe(s):
  s = unicodeToAscii(s.lower().strip())
  s = re.sub(r"([,.!?])", r" \1", s)
  s = re.sub(r"[^a-zA-Z\s]+", r" ", s)
  s = re.sub(r'\s+', ' ', s).strip()
  return s
```

b. Normalisation techniques for ingredients

In [5]:
```
def remove_extra_spaces(s):
    # Replace multiple spaces with a single space
    s = re.sub(r'\s+', ' ', s).strip()
    return s

# given an ingredients from pair, convert the ingredients into unique ingredient
def seperated_ingredient(pair):
    unseperated_ingredient = pair

    ingredients_list = unseperated_ingredient.split('\t')
    for i in range(len(ingredients_list)):
        ingredients_list[i] = remove_extra_spaces(ingredients_list[i])

    cleaned_ingredients = set()
    for ingredient in ingredients_list:
        cleaned_ingredient = ingredient.strip()
        if cleaned_ingredient != '':
            cleaned_ingredients.add(cleaned_ingredient)

    return (' '.join(list(cleaned_ingredients)))
```

4. Load dictionaries and pairs

In [6]:
```
import pandas as pd
def readIngredientsRecipe(preprocessing=False):
    print("Reading lines...")
    # read the file
    train_dataset = pd.read_csv('Cooking_Dataset/train.csv')
    dev_dataset = pd.read_csv('Cooking_Dataset/dev.csv')
    test_dataset = pd.read_csv('Cooking_Dataset/test.csv')

    # drop the empty cell
    train_dataset.dropna(inplace=True)
    dev_dataset.dropna(inplace=True)
    test_dataset.dropna(inplace=True)

    # convert int/float cell value to string
    train_dataset = train_dataset.astype(str)
    dev_dataset = dev_dataset.astype(str)
    test_dataset = test_dataset.astype(str)

    # length of dataset
    train_length = len(train_dataset['Ingredients'].values)
    dev_length = len(dev_dataset['Ingredients'].values)
    test_length = len(test_dataset['Ingredients'].values)

    if not preprocessing:# if without text preprocessing
        train_pairs = [[(train_dataset['Ingredients'].values[i]), (train_dataset['Recipe'].values[i])]
                       for i in range(train_length)]
        dev_pairs = [[(dev_dataset['Ingredients'].values[i]), (dev_dataset['Recipe'].values[i])]
                       for i in range(dev_length)]
        test_pairs = [[(test_dataset['Ingredients'].values[i]), (test_dataset['Recipe'].values[i])]
                       for i in range(test_length)]

    else: # if with text preprocessing
        train_pairs = [[seperated_ingredient(normalizeString(train_dataset['Ingredients'].values[i])),
                       normalizeRecipe(train_dataset['Recipe'].values[i])] for i in range(train_length)]
        dev_pairs = [[seperated_ingredient(normalizeString(dev_dataset['Ingredients'].values[i])),
                       normalizeRecipe(dev_dataset['Recipe'].values[i])] for i in range(dev_length)]
        test_pairs = [[seperated_ingredient(normalizeString(test_dataset['Ingredients'].values[i])),
                       normalizeRecipe(test_dataset['Recipe'].values[i])] for i in range(test_length)]
    # make Lang instances
    input_lang = Lang("Ingredients")
    output_lang = Lang("Recipe")
```

```
    return input_lang, output_lang, train_pairs, dev_pairs, test_pairs
```

5. Filter and trim the dataset with MAX_LENGTH

In [7]:
```
MAX_LENGTH = 150

def filterPair(p):
    return len(p[0].split(' ')) < MAX_LENGTH and \
        len(p[1].split(' ')) < MAX_LENGTH


def filterPairs(pairs):
    filtered_pairs = []
    filtered_indices = [] # for testing set to trace the unfiltered index. Indices used for updating the generated csv file

    for index, pair in enumerate(pairs):
        if filterPair(pair):
            filtered_pairs.append(pair)
            filtered_indices.append(index)

    return filtered_pairs, filtered_indices
```

6. Load Data

In [8]:
```
def prepareData(preprocessing=False):
    input_lang, output_lang, train_pairs, dev_pairs, test_pairs =readIngredientsRecipe(preprocessing)

    # training set
    print("Read %s train sentence pairs" % len(train_pairs))
    print("Counting train words...")
    train_pairs, _ = filterPairs(train_pairs) # trim data
    print("Trimmed to %s train sentence pairs" % len(train_pairs))
    for train_pair in train_pairs:
        input_lang.addSentence(train_pair[0])
        output_lang.addSentence(train_pair[1])
    print("Counted words:")
    print(input_lang.name, input_lang.n_words)
    print(output_lang.name, output_lang.n_words)

    # validating set
    print("Read %s dev sentence pairs" % len(dev_pairs))
    print("Counting dev words...")
    dev_pairs, _ = filterPairs(dev_pairs) # trim data
    print("Trimmed to %s dev sentence pairs" % len(dev_pairs))
    for dev_pair in dev_pairs:
        input_lang.addSentence(dev_pair[0])
        output_lang.addSentence(dev_pair[1])

    # testing set
    print("Read %s test sentence pairs" % len(test_pairs))
    print("Counting test words...")
    test_pairs, filtered_indices = filterPairs(test_pairs) # trim data
    print("Trimmed to %s test sentence pairs" % len(test_pairs))
    for test_pair in test_pairs:
        input_lang.addSentence(test_pair[0])
        output_lang.addSentence(test_pair[1])

    input_lang = addCommonWords(input_lang) # most common ingredient words added

    return input_lang, output_lang, train_pairs, dev_pairs, test_pairs, filtered_indices
```

7. Pair to Tensor functions

a. Normal pair to tensor functions

In [9]:
```
# match each index of word of given sentence from lang
def indexesFromSentence(lang, sentence):
    return [lang.word2index[word] for word in sentence.split(' ')]

# convert index to tensors
def tensorFromSentence(lang, sentence):
    indexes = indexesFromSentence(lang, sentence)
    indexes.append(EOS_token)
    return torch.tensor(indexes, dtype=torch.long, device=device).view(-1, 1)


def tensorsFromPair(pair):
    input_tensor = tensorFromSentence(input_lang, pair[0])
    target_tensor = tensorFromSentence(output_lang, pair[1])
    return (input_tensor, target_tensor)
```

b. pair to tensor functions which return original word of pair to keep track the randomized original word (Extension 2 purpose)

In [10]:
```
def tensorsFromPairWithWords(pair):
    input_tensor = tensorFromSentence(input_lang, pair[0])
    target_tensor = tensorFromSentence(output_lang, pair[1])
    return pair[0], (input_tensor, target_tensor)
```

8. Time Tracing functions

In [11]:
```
import time
import math


def asMinutes(s):
    m = math.floor(s / 60)
    s -= m * 60
    return '%dm %ds' % (m, s)


def timeSince(since, percent):
    now = time.time()
    s = now - since
    es = s / (percent)
    rs = es - s
    return '%s (- %s)' % (asMinutes(s), asMinutes(rs))
```

9. Function for train and valid loss implementation

In [12]:
```
import matplotlib.pyplot as plt
plt.switch_backend('agg')
import matplotlib.ticker as ticker
import numpy as np
%matplotlib inline

def showPlot(n_iters, train_points, valid_points, title):
    x_range = [i for i in range(0, n_iters-1, 100)]
    plt.figure()
    fig, ax = plt.subplots()
    # this locator puts ticks at regular intervals
    loc = ticker.MultipleLocator(base=0.2)
    ax.yaxis.set_major_locator(loc)
    plt.xlabel('iteration')
    plt.ylabel('loss')
    plt.plot(x_range, train_points, label='train loss')
    plt.plot(x_range, valid_points, label='dev loss')
    plt.legend()
    plt.title(title)
```

10. Encoder RNN

In [13]:
```
class EncoderRNN(nn.Module):
    def __init__(self, input_size, hidden_size):
        super(EncoderRNN, self).__init__()
        self.hidden_size = hidden_size

        self.embedding = nn.Embedding(input_size, hidden_size)
        self.lstm = nn.LSTM(hidden_size, hidden_size)

    def forward(self, input, hidden):
        embedded = self.embedding(input).view(1, 1, -1)
        output = embedded
        output, (hidden, cell_state) = self.lstm(output, hidden)
        return output, (hidden, cell_state)

    def initHidden(self):
        return (torch.zeros(1, 1, self.hidden_size, device=device),
                torch.zeros(1, 1, self.hidden_size, device=device))
```

11. Decoder RNN without attention

In [14]:
```
class DecoderRNN(nn.Module):
    def __init__(self, hidden_size, output_size):
        super(DecoderRNN, self).__init__()
        self.hidden_size = hidden_size

        self.embedding = nn.Embedding(output_size, hidden_size)
        self.lstm = nn.LSTM(hidden_size, hidden_size)
        self.out = nn.Linear(hidden_size, output_size)
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, input, hidden):
        output = self.embedding(input).view(1, 1, -1)
        output = F.relu(output)
        output, (hidden, cell_state) = self.lstm(output, hidden)
        output = self.softmax(self.out(output[0]))
        return output, (hidden, cell_state)

    def initHidden(self):
        return (torch.zeros(1, 1, self.hidden_size, device=device),
                torch.zeros(1, 1, self.hidden_size, device=device))
```

12. Train and validate function for RNN without attention

In [15]:
```
teacher_forcing_ratio = 1.0

def train(input_tensor, target_tensor, encoder, decoder, encoder_optimizer, decoder_optimizer, criterion, max_length=MAX_LENGTH):
    encoder_hidden = encoder.initHidden()

    encoder_optimizer.zero_grad()
    decoder_optimizer.zero_grad()

    input_length = input_tensor.size(0)
    target_length = target_tensor.size(0)

    encoder_outputs = torch.zeros(max_length, encoder.hidden_size, device=device)

    loss = 0

    for ei in range(input_length):
        encoder_output, encoder_hidden = encoder(
            input_tensor[ei], encoder_hidden)
        encoder_outputs[ei] = encoder_output[0, 0]

    decoder_input = torch.tensor([[SOS_token]], device=device)

    decoder_hidden = encoder_hidden

    use_teacher_forcing = True if random.random() < teacher_forcing_ratio else False

    if use_teacher_forcing:
        # Teacher forcing: Feed the target as the next input
        for di in range(target_length):
            decoder_output, decoder_hidden = decoder(
                decoder_input, decoder_hidden)
            loss += criterion(decoder_output, target_tensor[di])
            decoder_input = target_tensor[di]  # Teacher forcing

    else:
        # Without teacher forcing: use its own predictions as the next input
        for di in range(target_length):
            decoder_output, decoder_hidden = decoder(
                decoder_input, decoder_hidden)
            topv, topi = decoder_output.topk(1)
            decoder_input = topi.squeeze().detach()  # detach from history as input

            loss += criterion(decoder_output, target_tensor[di])
            if decoder_input.item() == EOS_token:
                break
```

```
    loss.backward()

    encoder_optimizer.step()
    decoder_optimizer.step()

    return loss.item() / target_length

# validate function which does not backward and record the gradient of loss. validation aims to evaluate the general loss
def validate(input_tensor, target_tensor, encoder, decoder, criterion, max_length=MAX_LENGTH):
    encoder_hidden = encoder.initHidden()

    input_length = input_tensor.size(0)
    target_length = target_tensor.size(0)

    encoder_outputs = torch.zeros(max_length, encoder.hidden_size, device=device)

    loss = 0

    for ei in range(input_length):
        encoder_output, encoder_hidden = encoder(
            input_tensor[ei], encoder_hidden)
        encoder_outputs[ei] = encoder_output[0, 0]

    decoder_input = torch.tensor([[SOS_token]], device=device)

    decoder_hidden = encoder_hidden

    # no teacher forcing in validating set
    for di in range(target_length):
        decoder_output, decoder_hidden = decoder(
            decoder_input, decoder_hidden)
        topv, topi = decoder_output.topk(1)
        decoder_input = topi.squeeze().detach()  # detach from history as input

        loss += criterion(decoder_output, target_tensor[di])
        if decoder_input.item() == EOS_token:
            break

    return loss.item() / target_length


def trainIters(encoder, decoder, n_iters, print_every=1000, plot_every=100, learning_rate=0.001):
    start = time.time()
    # training
    plot_losses = []
    print_loss_total = 0  # Reset every print_every
```

```
plot_loss_total = 0  # Reset every plot_every

# validating
plot_valid_losses = []
print_valid_loss_total = 0  # Reset every print_every
plot_valid_loss_total = 0  # Reset every plot_every

encoder_optimizer = optim.Adam(encoder.parameters(), lr=learning_rate) # Adam optimizer
decoder_optimizer = optim.Adam(decoder.parameters(), lr=learning_rate) # Adam optimizer

# convert pairs to tensors
training_pairs = [tensorsFromPair(random.choice(pairs))
                  for i in range(n_iters)]
validating_pairs = [tensorsFromPair(random.choice(dev_pairs))
                  for i in range(n_iters)]    # validating set size

criterion = nn.NLLLoss()

for iter in range(1, n_iters + 1):
    # training
    training_pair = training_pairs[iter - 1]
    input_tensor = training_pair[0]
    target_tensor = training_pair[1]
    # validating
    validating_pair = validating_pairs[iter - 1]
    valid_input_tensor = validating_pair[0]
    valid_target_tensor = validating_pair[1]

    # training
    loss = train(input_tensor, target_tensor, encoder,
                 decoder, encoder_optimizer, decoder_optimizer, criterion)
    print_loss_total += loss
    plot_loss_total += loss
    # validating
    valid_loss = validate(valid_input_tensor, valid_target_tensor, encoder, decoder, criterion)
    print_valid_loss_total += valid_loss
    plot_valid_loss_total += valid_loss

    if iter % print_every == 0:
        print_loss_avg = print_loss_total / print_every
        print_loss_total = 0

        print_valid_loss_avg = print_valid_loss_total / print_every
        print_valid_loss_total = 0
        print('%s (%d %d%%) Train Loss: %.4f | Validation Loss: %.4f' % (timeSince(start, iter / n_iters),
                                     iter, iter / n_iters * 100, print_loss_avg, print_valid_loss_avg))
```

```
        if iter % plot_every == 0:
            plot_loss_avg = plot_loss_total / plot_every
            plot_valid_loss_avg = plot_valid_loss_total / plot_every
            plot_losses.append(plot_loss_avg)
            plot_valid_losses.append(plot_valid_loss_avg)
            plot_loss_total = 0
            plot_valid_loss_total = 0

    return plot_losses, plot_valid_losses
```

13. Bleu, Meteor and Avg given and extra items Evaluation Functions

In [16]:

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.translate.bleu_score import sentence_bleu, SmoothingFunction
from nltk.translate.meteor_score import meteor_score


def bleu_meteor(ingredient_sample, my_prediction, my_target):
    # Tokenize recipes
    my_prediction_tokens = word_tokenize(my_prediction)
    my_target_tokens = word_tokenize(my_target)

    # Calculate BLEU-4 score
    # bleu 4 means 4-grams hence weights=(0.25, 0.25, 0.25, 0.25)
    bleu_4_score = sentence_bleu([my_target_tokens], my_prediction_tokens, weights=(0.25, 0.25, 0.25, 0.25))

    # Calculate METEOR score
    my_meteor_score = meteor_score([my_target_tokens], my_prediction_tokens)

    # calculate the given and extra items
    ingredient_word = identifyIngredients(ingredient_sample) # see the given ingredients
    # check how many items of gold recipe match with real ingredient
    gold_given_items, _ = identifyIngredientsExtra(my_target, ingredient_word)
    # check how many items of generated ingredient matched with gold recipe
    given_items, extra_items = identifyIngredientsExtra(my_prediction, gold_given_items)

    similar_items = []
    for item in given_items:
        if item in gold_given_items:
            similar_items.append(given_items)

    # control the boundary
    if len(similar_items) == 0 or len(given_items) == 0:
        given_avg = 0
    else:
        given_avg = len(given_items) / len(similar_items)

    if given_avg < 0:
        given_avg = 0
    if given_avg > 1:
        given_avg = 1

    extra_avg = len(extra_items)

    return given_avg, extra_avg, bleu_4_score, my_meteor_score
```

```
def identifyIngredients(my_sentences):
    # trace the ingredients
    ingredient_word = set()
    sentences_splitted = my_sentences.split()
    i = 0
    while i < len(sentences_splitted):
        current_word_guess = sentences_splitted[i]
        if current_word_guess in input_lang.word2index: # if the word is an ingredient
            j = i + 1
            while j < len(sentences_splitted): # check following words are also part of ingredients or not
                if current_word_guess + " " + sentences_splitted[j] not in input_lang.word2index:
                    break
                else:
                    current_word_guess = current_word_guess + " " + sentences_splitted[j]
                    j += 1
            i = j
            ingredient_word.add(current_word_guess)
        else:
            i += 1

    ingredient_word = list(ingredient_word)
    return ingredient_word


def identifyIngredientsExtra(recipe_sample, ingredient_word):
    extra_items = set()
    given_items = set()
    sentences_splitted = recipe_sample.split()
    i = 0
    while i < len(sentences_splitted):
        current_word_guess = sentences_splitted[i]
        if current_word_guess in input_lang.word2index: # if the word is an ingredient
            j = i + 1
            while j < len(sentences_splitted): # check following words are also part of ingredients or not
                if current_word_guess + " " + sentences_splitted[j] not in input_lang.word2index:
                    break
                else:
                    current_word_guess = current_word_guess + " " + sentences_splitted[j]
                    j += 1
            i = j
            if not(current_word_guess in ingredient_word): # if it is not a given ingredients means it is an extra word
                extra_items.add(current_word_guess)
            else:
                given_items.add(current_word_guess)

        else:
```

```
            i += 1

    extra_items = list(extra_items)
    given_items = list(given_items)
    return given_items, extra_items
```

14. Testing Evaluation for RNN without attention

In [17]:

```
def test(input_tensor, target_tensor, encoder, decoder, criterion, max_length=MAX_LENGTH):
    given_items = 0
    extra_items = 0
    decoded_words = []

    encoder_hidden = encoder.initHidden()

    input_length = input_tensor.size(0)
    target_length = target_tensor.size(0)

    encoder_outputs = torch.zeros(max_length, encoder.hidden_size, device=device)

    loss = 0

    for ei in range(input_length):
        encoder_output, encoder_hidden = encoder(
            input_tensor[ei], encoder_hidden)
        encoder_outputs[ei] = encoder_output[0, 0]

    decoder_input = torch.tensor([[SOS_token]], device=device)

    decoder_hidden = encoder_hidden

    # no teacher forcing in testing set
    for di in range(target_length):
        decoder_output, decoder_hidden = decoder(
            decoder_input, decoder_hidden)
        topv, topi = decoder_output.topk(1)
        decoder_input = topi.squeeze().detach()  # detach from history as input

        loss += criterion(decoder_output, target_tensor[di])

        topv_word, topi_word = decoder_output.data.topk(1)
        if topi_word.item() == EOS_token:
            decoded_words.append('<EOS>')
            break
        else:
            decoded_words.append(output_lang.index2word[topi_word.item()])


        if decoder_input.item() == EOS_token:
            break

    return loss.item() / target_length, decoded_words
```

```
def testIters(encoder, decoder):
    # testing loss
    loss_total = 0
    given_total = 0
    extra_total = 0
    criterion = nn.NLLLoss()
    bleu_total = 0
    meteor_total = 0
    prediction_words_total = []

    for iter in range(1, len(test_pairs) + 1): # evaluate all the test data
        # testing
        testing_pair = tensorsFromPair(test_pairs[iter - 1])
        input_tensor = testing_pair[0]
        target_tensor = testing_pair[1]
        input_word = test_pairs[iter - 1][0]

        loss, prediction_words = test(input_tensor, target_tensor, encoder, decoder, criterion)
        prediction_words = ' '.join(prediction_words)
        prediction_words_total.append(prediction_words)

        loss_total += loss

        given, extra, current_bleu, current_meteor = bleu_meteor(input_word, prediction_words, test_pairs[iter - 1][1])
        given_total += given
        extra_total += extra
        bleu_total += current_bleu
        meteor_total += current_meteor


    loss_avg = loss_total / len(test_pairs)
    given_avg = given_total / len(test_pairs)
    extra_avg = extra_total / len(test_pairs)
    bleu_avg = bleu_total / len(test_pairs)
    meteor_avg = meteor_total / len(test_pairs)

    return loss_avg, given_avg, extra_avg, bleu_avg, meteor_avg, prediction_words_total


def update_csv_recipe(prediction_words_total, column_header, my_test_indices):
    # Read the CSV file into a pandas DataFrame
    df = pd.read_csv("generated_31158145.csv")

    # Update the specified column with the prediction for the specified indices
    for i in range(len(my_test_indices)):
        index = my_test_indices[i]
```

```
        df.at[index, column_header] = prediction_words_total[i]

    # Set cell value as None for indices not in the list
    for j in range(len(df)):
        if j not in my_test_indices:
            df.at[j, column_header] = "TRIMMED DATA"

    # Write the updated DataFrame back to the CSV file
    df.to_csv("generated_31158145.csv", index=False)
```

15. Evaluation for model using decoder without attention

In [18]:
```
# sentence that required to predict
evaluation_sentence = "2 c sugar, 1/4 c lemon juice, 1 c water, 1/3 c orange juice, 8 c strawberries"

def evaluate(encoder, decoder, sentence, max_length=MAX_LENGTH):
    with torch.no_grad():
        input_tensor = tensorFromSentence(input_lang, sentence)
        input_length = input_tensor.size()[0]
        encoder_hidden = encoder.initHidden()

        encoder_outputs = torch.zeros(max_length, encoder.hidden_size, device=device)

        for ei in range(input_length):
            encoder_output, encoder_hidden = encoder(input_tensor[ei],
                                                     encoder_hidden)
            encoder_outputs[ei] += encoder_output[0, 0]

        decoder_input = torch.tensor([[SOS_token]], device=device)  # SOS

        decoder_hidden = encoder_hidden

        decoded_words = []
        decoder_attentions = torch.zeros(max_length, max_length)

        for di in range(max_length):
            decoder_output, decoder_hidden = decoder(
                decoder_input, decoder_hidden)
            topv, topi = decoder_output.data.topk(1) # return the word with highest probability
            if topi.item() == EOS_token:
                decoded_words.append('<EOS>')
                break
            else:
                decoded_words.append(output_lang.index2word[topi.item()])

            decoder_input = topi.squeeze().detach()

        return decoded_words


def evaluateRandomly(encoder, decoder, n=5):
    for i in range(n):
        pair = random.choice(test_pairs)
        print('>', pair[0])
        print('=', pair[1])
        output_words= evaluate(encoder, decoder, pair[0])
        output_sentence = ' '.join(output_words)
        print('<', output_sentence)
```

```
        print('')


def evaluateSpecific(encoder, decoder, words, preprocess=False):
    # function specifically check and return the predicted word based on the input sentence given
    if preprocess:
        words = seperated_ingredient(normalizeString(words))
    print('>', words)
    output_words = evaluate(encoder, decoder, words)
    output_sentence = ' '.join(output_words)
    print('<', output_sentence)
    print('')
```

Implementation of Baseline 1: Sequence-to-Sequence model without attention

1. Prepare the dictionaries, train dev and test pairs

In [19]:
```
input_lang, output_lang, pairs, dev_pairs, test_pairs, my_test_indices = prepareData(False) # preprocessing = false
print("")
print("training pair example: ", random.choice(pairs))
print("")
print("dev pair example: ", random.choice(dev_pairs))
print("")
print("testing pair example: ", random.choice(test_pairs))
print("-----------------------------------------------------------------------")
print("max ingredients sentence length: ", input_lang.max_sentence)
print("min ingredients sentence length: ", input_lang.min_sentence)
print("max recipe sentence length: ", output_lang.max_sentence)
print("min recipe sentence length: ", output_lang.min_sentence)
```

```
Reading lines...
Read 100925 train sentence pairs
Counting train words...
Trimmed to 79434 train sentence pairs
Counted words:
Ingredients 107361
Recipe 35037
Read 793 dev sentence pairs
Counting dev words...
Trimmed to 641 dev sentence pairs
Read 773 test sentence pairs
Counting test words...
Trimmed to 620 test sentence pairs

training pair example:  ['3 lb chicken pieces\t1/4 c  vegetable oil\t3/4 c  lemon juice\t1 tb lemon juice\tflour for dredging\t
1    lemon; sliced thin\t1/2 ts -salt\t3 tb brown sugar pepper                          1/2 c  chicken stock', 'combine chicke
n and lemon juice . marinate at least 6 hours or overnight , turning occaisionally . remove chicken , discard marinade , reserv
ing 1 tbsp . combine flour , salt and pepper ; heat oil in large skillet over medium-high heat , fry in several batches . trans
fer to casserole large enough to hold chicken in single layer . combine peel and brown sugar in bowl , sprinkle over chicken .
combine stock and 1 tbsp reserved marinate , pour around chicken . bake at 350f 35-40 minutes . serves 4 -lrb- generous portion
s -rrb- or 6 . good served hot , even better served cold . great for picnics . ']

dev pair example:  ['1/2    banana\t1 tb apple juice concentrate\tgrape nuts', 'insert a stick in banana . dip in juice and rol
l in cereal . wrap in plastic wrap and freeze . thaw a few minutes and eat ! ']

testing pair example:  ['8    bacon slices,cut 1 1/2"\t1/2 c  chopped onion\t1/2 c  chopped green bell pepper\t53 oz can pork a
nd beans\t1/4 c  molasses\t1/4 ts tabasco sauce', 'heat oven to 375 degrees.fry bacon until crisp ; set aside.reserve 2 tablesp
oons drippings in pan.saute onion and green pepper in drippings until tender.combine beans , molasses and red pepper sauce in a
21/2 quart casserole.bake 40 to 45 minutes.top with bacon . ']
---------------------------------------------------------------
max ingredients sentence length:  149
min ingredients sentence length:  1
max recipe sentence length:  149
min recipe sentence length:  1
```

2.Train and validate for Baseline 1 model

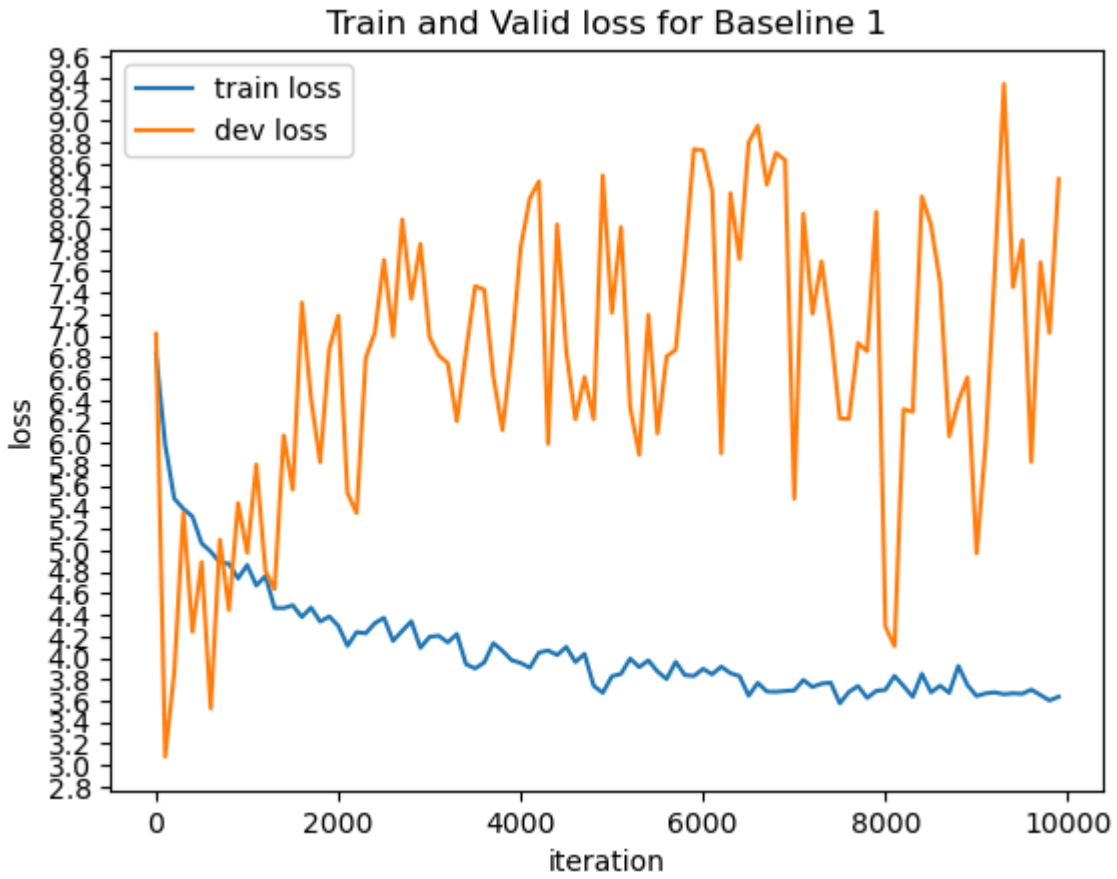
In [20]:
```
hidden_size = 256
n_iters = 10000
baseline1_encoder = EncoderRNN(input_lang.n_words, hidden_size).to(device) # normal encoder
baseline1_decoder = DecoderRNN(hidden_size, output_lang.n_words).to(device) # normal decoder

baseline1_plot_losses, baseline1_plot_valid_losses = trainIters(baseline1_encoder, baseline1_decoder, n_iters, print_every=1000)
```

```
3m 40s (- 33m 7s) (1000 10%) Train Loss: 5.3557 | Validation Loss: 4.6961
7m 11s (- 28m 45s) (2000 20%) Train Loss: 4.5282 | Validation Loss: 5.8275
10m 41s (- 24m 56s) (3000 30%) Train Loss: 4.2407 | Validation Loss: 6.9866
14m 8s (- 21m 12s) (4000 40%) Train Loss: 4.0740 | Validation Loss: 6.8086
17m 38s (- 17m 38s) (5000 50%) Train Loss: 3.9510 | Validation Loss: 7.2968
21m 6s (- 14m 4s) (6000 60%) Train Loss: 3.8865 | Validation Loss: 7.0868
24m 47s (- 10m 37s) (7000 70%) Train Loss: 3.7819 | Validation Loss: 8.2512
28m 19s (- 7m 4s) (8000 80%) Train Loss: 3.7058 | Validation Loss: 6.9958
31m 32s (- 3m 30s) (9000 90%) Train Loss: 3.7517 | Validation Loss: 6.3888
34m 47s (- 0m 0s) (10000 100%) Train Loss: 3.6580 | Validation Loss: 7.2261
```

3. Visualisation for train and valid loss

In [21]:
```
showPlot(n_iters, baseline1_plot_losses, baseline1_plot_valid_losses, "Train and Valid loss for Baseline 1")
```

```
<Figure size 640x480 with 0 Axes>
```



4. Evaluate the metrics using Test Data, save the prediction to csv file

In [22]:
```
ine1_test_loss, baseline1_given_items, baseline1_extra_items, baseline1_bleu, baseline1_meteor, baseline1_prediction_words = test

("baseline 1 - Test loss: {}, Avg % given: {}, Avg extra: {}, BLEU: {}, METEOR: {}".format(
aseline1_test_loss, baseline1_given_items, baseline1_extra_items, baseline1_bleu, baseline1_meteor))

e_csv_recipe(baseline1_prediction_words, "Generated Recipe - Baseline 1", my_test_indices)
```

```
C:\Users\ngyij\anaconda3\Lib\site-packages\nltk\translate\bleu_score.py:552: UserWarning:
The hypothesis contains 0 counts of 4-gram overlaps.
Therefore the BLEU score evaluates to 0, independently of
how many N-gram overlaps of lower order it contains.
Consider using lower n-gram order or use SmoothingFunction()
  warnings.warn(_msg)
C:\Users\ngyij\anaconda3\Lib\site-packages\nltk\translate\bleu_score.py:552: UserWarning:
The hypothesis contains 0 counts of 3-gram overlaps.
Therefore the BLEU score evaluates to 0, independently of
how many N-gram overlaps of lower order it contains.
Consider using lower n-gram order or use SmoothingFunction()
  warnings.warn(_msg)
C:\Users\ngyij\anaconda3\Lib\site-packages\nltk\translate\bleu_score.py:552: UserWarning:
The hypothesis contains 0 counts of 2-gram overlaps.
Therefore the BLEU score evaluates to 0, independently of
how many N-gram overlaps of lower order it contains.
Consider using lower n-gram order or use SmoothingFunction()
  warnings.warn(_msg)

baseline 1 - Test loss: 9.042197013169032, Avg % given: 0.3032258064516129, Avg extra: 22.18064516129032, BLEU: 0.0070037242513
835305, METEOR: 0.11199092633604503
```

5. Check 5 predictions randomly

In [23]:
```
evaluateRandomly(baseline1_encoder, baseline1_decoder)
```

> 5 sl brown bread; toasted   150 g raw smoked ham; sliced such as parma ham   250 g emmental cheese; sliced 400 g mushrooms in season   1   shallot; chopped   40 g butter   20 g flour   1 dl dry white wine   1 dl brown stock
1 dl cream   marjoram   salt   pepper; freshly ground parsley; chopped   5   unpeeled pears, poached in syrup
25 g   = 1 oz. 1   dl = 3.5 fl. oz.   2.5 dl = 1 cup
= cover the slices of toasted bread with rhe ham , reserving 1 ham slices/servings . sweat the mushrooms in the butter with the shallot . mix in the flour , seasoning -lrb- except marjoram -rrb- , white wine and brown stock and simmer for 5 minutes . add the cream and marjoram , quickly bring to a boil , the remove from the heat . adjust the seasoning . place 3/4 of the mushrooms on top of the ham and bread slices . cover each with the reserved slice of ham and a slide of cheese . brown under the grill . garnish with the remaining mushrooms and chopped parsley . serve on a plate with thin slices of hot pear .
< in a large skillet , heat oil in a large skillet over medium-high heat . add onion and saute until tender . add remaining ingredients and stir until well blended . add remaining ingredients and mix well . pour into greased 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9

> 2 lb round steak (elk or deer)   4   medium-sized carrots   salt   flour   pepper shortening   1/2 lb pork sausage
= pound thinly cut steak with saucer edge or meat hammer . cut into 4 inch sqares . sprinkle with salt and pepper and spread with sausage meat . peel carrots and quarter lengthwise . place several strips on each piece of meat . roll and tie with string or fasten with toothpicks or skewers . flour lightly , brown in hot shortening . partly cover with water , cover pan , and cook in moderate oven -lrb- 359 degrees f -rrb- until tender 1-1/2 to 2 hours .
< in a large skillet , heat oil in a large skillet over medium-high heat . add onion and cook until tender . add remaining ingredients and stir until well blended . add remaining ingredients and mix well . pour into greased 9 '' pie plate . bake at 350 degrees for 30 minutes . <EOS>

> 5 sl brown bread; toasted   150 g raw smoked ham; sliced such as parma ham   250 g emmental cheese; sliced 400 g mushrooms in season   1   shallot; chopped   40 g butter   20 g flour   1 dl dry white wine   1 dl brown stock
1 dl cream   marjoram   salt   pepper; freshly ground parsley; chopped   5   unpeeled pears, poached in syrup
25 g   = 1 oz. 1   dl = 3.5 fl. oz.   2.5 dl = 1 cup
= cover the slices of toasted bread with rhe ham , reserving 1 ham slices/servings . sweat the mushrooms in the butter with the shallot . mix in the flour , seasoning -lrb- except marjoram -rrb- , white wine and brown stock and simmer for 5 minutes . add the cream and marjoram , quickly bring to a boil , the remove from the heat . adjust the seasoning . place 3/4 of the mushrooms on top of the ham and bread slices . cover each with the reserved slice of ham and a slide of cheese . brown under the grill . garnish with the remaining mushrooms and chopped parsley . serve on a plate with thin slices of hot pear .
< in a large skillet , heat oil in a large skillet over medium-high heat . add onion and saute until tender . add remaining ingredients and stir until well blended . add remaining ingredients and mix well . pour into greased 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9

> 1 pt milk   3 oz imported chocolate (milk   chocolate or semi-sweet)
= pour 1/4 cup of the milk into a small , thick-bottomed pot . break the chocolate into small bits and add to the milk in the pot . over very low heat , let the chocolate melt , then pour in the rest of the milk , bit by bit , whisking well with each addition . when the chocolate is melted and the milk faintly steaming , remove from the heat , beat vigorously with a whisk until frothy , and serve . 11/2 oz . chocolate per cup of milk and sweeten to taste .
< combine all ingredients in a small bowl . pour into a greased 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 13 x 9 x 13 x 9 x 13 x 9 x 13 x 9 x 13 x 9 x 13 x 9 x 13 x 9 x 13 x 9 x 13 x 9 x 13 x 9 x

```
> cooking light 9-95    tomatoes, undrained     carolyn shaw 8-95       1 cn (14.5 oz) fat free beef    4 c  water      broth
2 c  chopped onion      3 cl garlic, chopped    1 1/2 c  quartered small red    1 c  sliced zucchini    potatoes        1 c  to
rn spinach      1 c  dried great northern or navy       1/2 c  uncooked alphabet or other       beans   sm pasta        1/2 c
sliced carrots  1 tb dried rosemary, crushed    1/2 c  quartered mushrooms      1 ts salt       1/2 c  uncooked pearl barley
1 ts rubbed sage        1/2 lb lean boneless round steak,       1/2 ts pepper   cut in half inch pieces 1/4 ts ground nutmeg
1 cn (14.5 oz) pasta style chunky       1/2 c  grated parmesan cheese
= combine first 11 ingredients -lrb- water through garlic -rrb- in slow cooker . cover and cook on high 6 hours . add next 8 in
gredients -lrb- zucchini through nutmeg -rrb- ; cover and cook on high an additional 30 minutes or until beans are tender . lad
le into soup bowls and sprinkle with cheese .
< in a large skillet , heat oil in a large skillet over medium-high heat . add onion and saute until tender . add remaining ing
redients and stir until well blended . add remaining ingredients and mix well . pour into greased 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9
x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9
x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9 x 9
```

6. Check the evaluation sentence prediction

In [24]:
```
baseline1_output_sentence = evaluateSpecific(baseline1_encoder, baseline1_decoder, evaluation_sentence, False)
```

```
> 2 c sugar, 1/4 c lemon juice, 1 c water, 1/3 c orange juice, 8 c strawberries
< in a large skillet , heat oil in a large skillet over medium-high heat . add onion and cook until tender . add remaining ingr
edients and stir until well blended . add remaining ingredients and mix well . pour into greased 9 '' pie plate . bake at 350 d
egrees for 30 minutes .  <EOS>
```

Common functions Implementation (Part B)

1. Decoder RNN with attention

In [25]:
```
class AttnDecoderRNN(nn.Module):
    def __init__(self, hidden_size, output_size, dropout_p=0.1, max_length=MAX_LENGTH):
        super(AttnDecoderRNN, self).__init__()
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.dropout_p = dropout_p
        self.max_length = max_length

        self.embedding = nn.Embedding(self.output_size, self.hidden_size)
        self.dropout = nn.Dropout(self.dropout_p)
        self.lstm = nn.LSTM(self.hidden_size, self.hidden_size)
        self.out = nn.Linear(self.hidden_size*2, self.output_size)

    def forward(self, input, hidden, encoder_outputs):
        embedded = self.embedding(input).view(1, 1, -1)
        embedded = self.dropout(embedded)

        _, (hidden, cell_state) = self.lstm(embedded, hidden)

        attn_weights = F.softmax(torch.bmm(hidden, encoder_outputs.T.unsqueeze(0)),dim=-1)
        attn_output = torch.bmm(attn_weights, encoder_outputs.unsqueeze(0))

        concat_output = torch.cat((attn_output[0], hidden[0]), 1)

        output = F.log_softmax(self.out(concat_output), dim=1)

        return output, (hidden, cell_state), attn_weights

    def initHidden(self):
        return (torch.zeros(1, 1, self.hidden_size, device=device),
                torch.zeros(1, 1, self.hidden_size, device=device))
```

2. Train and validate functions for decoder using attention

In [26]:
```
teacher_forcing_ratio = 1.0


def train_attn(input_tensor, target_tensor, encoder, decoder, encoder_optimizer, decoder_optimizer,
               criterion, max_length=MAX_LENGTH):
    encoder_hidden = encoder.initHidden()

    encoder_optimizer.zero_grad()
    decoder_optimizer.zero_grad()

    input_length = input_tensor.size(0)
    target_length = target_tensor.size(0)

    encoder_outputs = torch.zeros(max_length, encoder.hidden_size, device=device)

    loss = 0

    for ei in range(input_length):
        encoder_output, encoder_hidden = encoder(
            input_tensor[ei], encoder_hidden)
        encoder_outputs[ei] = encoder_output[0, 0]

    decoder_input = torch.tensor([[SOS_token]], device=device)

    decoder_hidden = encoder_hidden

    use_teacher_forcing = True if random.random() < teacher_forcing_ratio else False

    if use_teacher_forcing:
        # Teacher forcing: Feed the target as the next input
        for di in range(target_length):
            decoder_output, decoder_hidden, decoder_attention = decoder(
                decoder_input, decoder_hidden, encoder_outputs)
            loss += criterion(decoder_output, target_tensor[di])
            decoder_input = target_tensor[di]  # Teacher forcing

    else:
        # Without teacher forcing: use its own predictions as the next input
        for di in range(target_length):
            decoder_output, decoder_hidden, decoder_attention = decoder(
                decoder_input, decoder_hidden, encoder_outputs)
            topv, topi = decoder_output.topk(1)
            decoder_input = topi.squeeze().detach()  # detach from history as input

            loss += criterion(decoder_output, target_tensor[di])
            if decoder_input.item() == EOS_token:
```

```
                break

    loss.backward()

    encoder_optimizer.step()
    decoder_optimizer.step()

    return loss.item() / target_length


def validate_attn(input_tensor, target_tensor, encoder, decoder, criterion, max_length=MAX_LENGTH):
    encoder_hidden = encoder.initHidden()

    input_length = input_tensor.size(0)
    target_length = target_tensor.size(0)

    encoder_outputs = torch.zeros(max_length, encoder.hidden_size, device=device)

    loss = 0

    for ei in range(input_length):
        encoder_output, encoder_hidden = encoder(
            input_tensor[ei], encoder_hidden)
        encoder_outputs[ei] = encoder_output[0, 0]

    decoder_input = torch.tensor([[SOS_token]], device=device)

    decoder_hidden = encoder_hidden

    for di in range(target_length):
        decoder_output, decoder_hidden, decoder_attention = decoder(
            decoder_input, decoder_hidden, encoder_outputs)
        topv, topi = decoder_output.topk(1)
        decoder_input = topi.squeeze().detach()  # detach from history as input

        loss += criterion(decoder_output, target_tensor[di])
        if decoder_input.item() == EOS_token:
            break

    return loss.item() / target_length


def trainIters_attn(encoder, decoder, n_iters, print_every=1000, plot_every=100, learning_rate=0.001):
    start = time.time()
    plot_losses = []
    print_loss_total = 0  # Reset every print_every
```

```
plot_loss_total = 0  # Reset every plot_every

# validating
plot_valid_losses = []
print_valid_loss_total = 0  # Reset every print_every
plot_valid_loss_total = 0  # Reset every plot_every

encoder_optimizer = optim.Adam(encoder.parameters(), lr=learning_rate)
decoder_optimizer = optim.Adam(decoder.parameters(), lr=learning_rate)

# convert pairs to tensors
training_pairs = [tensorsFromPair(random.choice(pairs))
                  for i in range(n_iters)]
validating_pairs = [tensorsFromPair(random.choice(dev_pairs))
                  for i in range(n_iters)]    # validating set size
criterion = nn.NLLLoss()

for iter in range(1, n_iters + 1):
    # training
    training_pair = training_pairs[iter - 1]
    input_tensor = training_pair[0]
    target_tensor = training_pair[1]
    # validating
    validating_pair = validating_pairs[iter - 1]
    valid_input_tensor = validating_pair[0]
    valid_target_tensor = validating_pair[1]

    # training
    loss = train_attn(input_tensor, target_tensor, encoder,
                 decoder, encoder_optimizer, decoder_optimizer, criterion)
    print_loss_total += loss
    plot_loss_total += loss
    # validating
    valid_loss = validate_attn(valid_input_tensor, valid_target_tensor, encoder, decoder, criterion)
    print_valid_loss_total += valid_loss
    plot_valid_loss_total += valid_loss

    if iter % print_every == 0:
        print_loss_avg = print_loss_total / print_every
        print_loss_total = 0

        print_valid_loss_avg = print_valid_loss_total / print_every
        print_valid_loss_total = 0
        print('%s (%d %d%%) Train Loss: %.4f | Validation Loss: %.4f' % (timeSince(start, iter / n_iters),
                                     iter, iter / n_iters * 100, print_loss_avg, print_valid_loss_avg))
```

```
        if iter % plot_every == 0:
            plot_loss_avg = plot_loss_total / plot_every
            plot_valid_loss_avg = plot_valid_loss_total / plot_every
            plot_losses.append(plot_loss_avg)
            plot_valid_losses.append(plot_valid_loss_avg)
            plot_loss_total = 0
            plot_valid_loss_total = 0

    return plot_losses, plot_valid_losses
```

3. Testing evaluation for decoder with attention

In [27]:

```
def test_attn(input_tensor, target_tensor, encoder, decoder, criterion, max_length=MAX_LENGTH):
    given_items = 0
    extra_items = 0
    decoded_words = []

    encoder_hidden = encoder.initHidden()

    input_length = input_tensor.size(0)
    target_length = target_tensor.size(0)

    encoder_outputs = torch.zeros(max_length, encoder.hidden_size, device=device)

    loss = 0

    for ei in range(input_length):
        encoder_output, encoder_hidden = encoder(
            input_tensor[ei], encoder_hidden)
        encoder_outputs[ei] = encoder_output[0, 0]

    decoder_input = torch.tensor([[SOS_token]], device=device)

    decoder_hidden = encoder_hidden

    for di in range(target_length): # no teacher forcing
        decoder_output, decoder_hidden, decoder_attention = decoder(
            decoder_input, decoder_hidden, encoder_outputs)
        topv, topi = decoder_output.topk(1)
        decoder_input = topi.squeeze().detach()  # detach from history as input

        loss += criterion(decoder_output, target_tensor[di])

        topv_word, topi_word = decoder_output.data.topk(1)
        if topi_word.item() == EOS_token:
            decoded_words.append('<EOS>')
            break
        else:
            decoded_words.append(output_lang.index2word[topi_word.item()])

        if decoder_input.item() == EOS_token:
            break

    return loss.item() / target_length, decoded_words


def testIters_attn(encoder, decoder):
    loss_total = 0
```

```
    given_total = 0
    extra_total = 0
    criterion = nn.NLLLoss()
    bleu_total = 0
    meteor_total = 0
    prediction_words_total = []

    for iter in range(1, len(test_pairs) + 1): # evaluate all the test
        # testing
        testing_pair = tensorsFromPair(test_pairs[iter - 1])
        input_tensor = testing_pair[0]
        target_tensor = testing_pair[1]
        input_word = test_pairs[iter - 1][0]

        loss, prediction_words = test_attn(input_tensor, target_tensor, encoder, decoder, criterion)
        prediction_words = ' '.join(prediction_words)
        prediction_words_total.append(prediction_words)

        loss_total += loss
        given, extra, current_bleu, current_meteor = bleu_meteor(input_word, prediction_words, test_pairs[iter - 1][1])
        given_total += given
        extra_total += extra
        bleu_total += current_bleu
        meteor_total += current_meteor


    loss_avg = loss_total / len(test_pairs)
    given_avg = given_total / len(test_pairs)
    extra_avg = extra_total / len(test_pairs)
    bleu_avg = bleu_total / len(test_pairs)
    meteor_avg = meteor_total / len(test_pairs)

    return loss_avg, given_avg, extra_avg, bleu_avg, meteor_avg, prediction_words_total
```

4. Evaluation for decoder with attention

In [28]:
```
def evaluate_attn(encoder, decoder, sentence, max_length=MAX_LENGTH):
    with torch.no_grad():
        input_tensor = tensorFromSentence(input_lang, sentence)
        input_length = input_tensor.size()[0]
        encoder_hidden = encoder.initHidden()

        encoder_outputs = torch.zeros(max_length, encoder.hidden_size, device=device)

        for ei in range(input_length):
            encoder_output, encoder_hidden = encoder(input_tensor[ei],
                                                     encoder_hidden)
            encoder_outputs[ei] += encoder_output[0, 0]

        decoder_input = torch.tensor([[SOS_token]], device=device)  # SOS

        decoder_hidden = encoder_hidden

        decoded_words = []
        decoder_attentions = torch.zeros(max_length, max_length)

        for di in range(max_length):
            decoder_output, decoder_hidden, decoder_attention = decoder(
                decoder_input, decoder_hidden, encoder_outputs)
            decoder_attentions[di] = decoder_attention.data
            topv, topi = decoder_output.data.topk(1)
            if topi.item() == EOS_token:
                decoded_words.append('<EOS>')
                break
            else:
                decoded_words.append(output_lang.index2word[topi.item()])

            decoder_input = topi.squeeze().detach()

        return decoded_words, decoder_attentions[:di + 1]

#  randomly return 5 prediction from test pairs
def evaluateRandomly_attn(encoder, decoder, n=5):
    for i in range(n):
        pair = random.choice(test_pairs)
        print('>', pair[0])
        print('=', pair[1])
        output_words, attention= evaluate_attn(encoder, decoder, pair[0])
        output_sentence = ' '.join(output_words)
        print('<', output_sentence)
        print('')
```

```
# predict and return the predicted outcome based on the sentence given
def evaluateSpecific_attn(encoder, decoder, words, preprocess=False):
    if preprocess:
        words = seperated_ingredient(normalizeString(words))
    print('>', words)
    output_words, attention= evaluate_attn(encoder, decoder, words)
    output_sentence = ' '.join(output_words)
    print('<', output_sentence)
    print('')
```

Implementation of Baseline 2: Sequence-to-Sequence model with attention

1. Prepare data

In [29]:
```
input_lang, output_lang, pairs, dev_pairs, test_pairs, my_test_indices  = prepareData(False)
print("-----------------------------------------------------------------------")
print("max ingredients sentence length: ", input_lang.max_sentence)
print("min ingredients sentence length: ", input_lang.min_sentence)
print("max recipe sentence length: ", output_lang.max_sentence)
print("min recipe sentence length: ", output_lang.min_sentence)
```

```
Reading lines...
Read 100925 train sentence pairs
Counting train words...
Trimmed to 79434 train sentence pairs
Counted words:
Ingredients 107361
Recipe 35037
Read 793 dev sentence pairs
Counting dev words...
Trimmed to 641 dev sentence pairs
Read 773 test sentence pairs
Counting test words...
Trimmed to 620 test sentence pairs
-----------------------------------------------------------------------
max ingredients sentence length:  149
min ingredients sentence length:  1
max recipe sentence length:  149
min recipe sentence length:  1
```

2. Train and fit

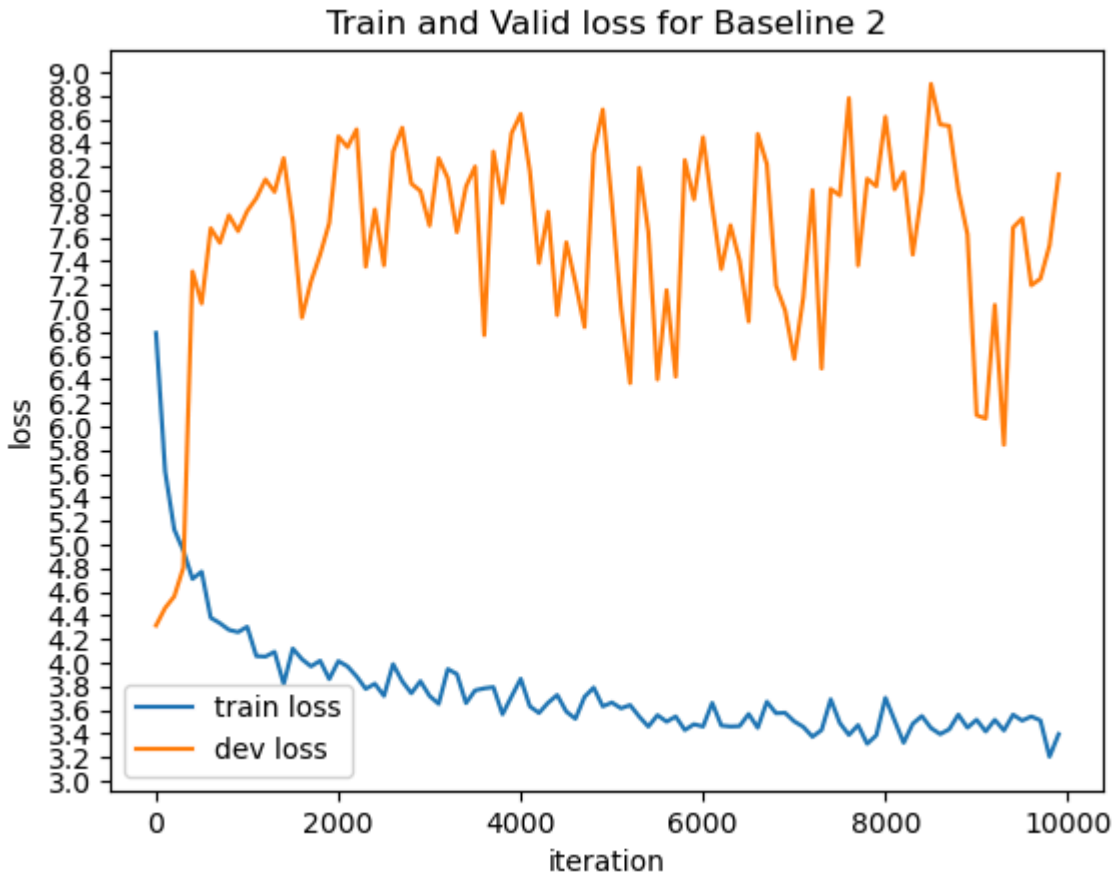
In [30]:
```
hidden_size = 256
n_iters = 10000
baseline2_encoder = EncoderRNN(input_lang.n_words, hidden_size).to(device)
baseline2_decoder = AttnDecoderRNN(hidden_size, output_lang.n_words, dropout_p=0.1).to(device)

baseline2_plot_losses, baseline2_plot_valid_losses = trainIters_attn(baseline2_encoder,
                                                                     baseline2_decoder, n_iters, print_every=1000)
```

```
4m 25s (- 39m 53s) (1000 10%) Train Loss: 4.9209 | Validation Loss: 6.3163
8m 53s (- 35m 34s) (2000 20%) Train Loss: 4.0313 | Validation Loss: 7.7153
13m 23s (- 31m 14s) (3000 30%) Train Loss: 3.8594 | Validation Loss: 8.0782
17m 50s (- 26m 45s) (4000 40%) Train Loss: 3.7493 | Validation Loss: 7.9411
22m 14s (- 22m 14s) (5000 50%) Train Loss: 3.6691 | Validation Loss: 7.7552
26m 37s (- 17m 44s) (6000 60%) Train Loss: 3.5423 | Validation Loss: 7.3242
31m 2s (- 13m 18s) (7000 70%) Train Loss: 3.5335 | Validation Loss: 7.6504
35m 34s (- 8m 53s) (8000 80%) Train Loss: 3.4495 | Validation Loss: 7.6384
40m 23s (- 4m 29s) (9000 90%) Train Loss: 3.4850 | Validation Loss: 8.1826
44m 47s (- 0m 0s) (10000 100%) Train Loss: 3.4597 | Validation Loss: 7.0583
```

3. Visualisation of train and valid loss for Baseline 2

In [31]:
```
showPlot(n_iters, baseline2_plot_losses, baseline2_plot_valid_losses, "Train and Valid loss for Baseline 2")
```

```
<Figure size 640x480 with 0 Axes>
```



4. Evaluate the metrics using Test Data, save the prediction to csv file

In [32]:
```
baseline2_test_loss, baseline2_given_items, baseline2_extra_items, baseline2_bleu, baseline2_meteor, baseline2_prediction_words
    baseline2_encoder, baseline2_decoder)

print("baseline 2 - Test loss: {}, Avg % given: {}, Avg extra: {}, BLEU: {}, METEOR: {}".format(
    baseline2_test_loss, baseline2_given_items, baseline2_extra_items, baseline2_bleu, baseline2_meteor))

update_csv_recipe(baseline2_prediction_words, "Generated Recipe - Baseline 2", my_test_indices)
```

```
C:\Users\ngyij\anaconda3\Lib\site-packages\nltk\translate\bleu_score.py:552: UserWarning:
The hypothesis contains 0 counts of 4-gram overlaps.
Therefore the BLEU score evaluates to 0, independently of
how many N-gram overlaps of lower order it contains.
Consider using lower n-gram order or use SmoothingFunction()
  warnings.warn(_msg)
C:\Users\ngyij\anaconda3\Lib\site-packages\nltk\translate\bleu_score.py:552: UserWarning:
The hypothesis contains 0 counts of 3-gram overlaps.
Therefore the BLEU score evaluates to 0, independently of
how many N-gram overlaps of lower order it contains.
Consider using lower n-gram order or use SmoothingFunction()
  warnings.warn(_msg)
C:\Users\ngyij\anaconda3\Lib\site-packages\nltk\translate\bleu_score.py:552: UserWarning:
The hypothesis contains 0 counts of 2-gram overlaps.
Therefore the BLEU score evaluates to 0, independently of
how many N-gram overlaps of lower order it contains.
Consider using lower n-gram order or use SmoothingFunction()
  warnings.warn(_msg)

baseline 2 - Test loss: 6.3945219474209125, Avg % given: 0.5516129032258065, Avg extra: 23.12258064516129, BLEU: 0.009824200682
94867, METEOR: 0.14715451500418
```

5. Randomly print 5 outcomes of the prediction from test set

In [33]:
```
evaluateRandomly_attn(baseline2_encoder, baseline2_decoder)
```

> 2 c flour 1 ts baking soda 1/2 ts salt 1/2 ts baking powder 1 c margarine 1 c white sugar 1 c br
own sugar 2 eggs 1 ts vanilla 2 c oats 6 oz semi-sweet chocolate chips 1 c nuts
= sift together the flour , baking soda , salt , and baking powder . cream the margarine and the sugars together . add the eggs and beat . add the flour mixture and mix well . add the vanilla , oats , chocolate chips , and nuts . grease a 13x9x2 pan , and press mixture in evenly . bake in a preheated oven 15 minutes at 350 f. judean scott fort worth , tx in the gustine , tx p-tc c
ookbook
< preheat oven to 350 degrees f. grease muffin tins . in large bowl , combine flour , baking soda , salt , and cinnamon . mix w
ell . add flour , baking powder , salt , and cinnamon . mix well . add remaining ingredients . mix well . add remaining ingredi
ents . bake at 350 degrees for 45 minutes . <EOS>

> 1 smoked turkey wing 1 md yellow onion, diced 2 garlic cloves, minced 2 lb fresh green beans 12 t
o 16 small red creamer potatoes 1 qt water salt, if needed pepper, to taste
= chef regina charboneau makes these for her 2-year-old son , jean-luc . place a heavy sauce pot over medium heat and let prehe
at for about 3 minutes ; add turkey wing , stir and turn until it begins to release some of its oil . -rrb- add diced onion and stir to brown for about 3 minutes , adding garlic at end . add green beans , potatoes and water -lrb- enough to just cover the vegetables -rrb- then cook over medium heat until potatoes are tender , which may take 35 to 45 minutes , depending on size of potatoes . season with pepper and salt to taste -lrb- the turkey wing usually adds enough salt -rrb- . serves 6 . san francisco chronicle , 6/29/92 .
< in a large bowl , combine all ingredients except lettuce . mix well . <EOS>

> 4 chicken breast halves, skinned, boned and patted dry 3/4 c buttermilk 3 tb parmesan cheese 1/2 c dry brea
d crumbs 1/2 ts rosemary 1/2 ts thyme 1/4 ts garlic powder 1/4 ts onion powder 1/4 ts black pepper
= cover a baking sheet with foil and lightly coat with nonstick cooking spray . in a shallow disk combine all ingredients excep
t chicken and buttermilk . in a separate dish , dip chicken in buttermilk then roll in dry mixture and place on baking sheet . bake at 400 ! for 35 to 40 minutes until golden .
< in a large skillet , heat oil over medium heat . add chicken broth , onion , garlic , and garlic powder . cook over medium he
at until chicken is no longer pink . add chicken broth , celery , onion , garlic , and garlic . cook over medium heat until chi
cken is no longer pink . add chicken broth and cook over medium heat until chicken is no longer pink . remove from heat and add chicken broth . stir in broth , and rice . <EOS>

> 1 c bread crumbs 1 c grated parmesan cheese 2 tb fresh parsley -- chopped 2 garlic cloves -- chopped salt --
to taste black pepper -- to taste 3/4 c margarine -- melted chicken
= chickens into serving-sized pieces . place this mixture in a shallow bowl . place 1/2 cup of melted margarine in a shallow di
sh . preheat oven to 350 degrees . have ready a shallow baking pan . dip chicken pieces first in the margarine , then into the bread crumb mixture . place in baking pan in a single layer . drizzle rest of melted margarine evenly over chicken . bake for 4
5 minutes , turning only once .
< in a large skillet , heat the oil over medium-high heat . add the onion , garlic , garlic , and pepper . cook over medium hea
t until the bread is golden brown , about 3 minutes . add the garlic , and cook over medium heat until the liquid is absorbed . add the remaining ingredients except the bread crumbs . stir in the yogurt and the remaining ingredients . cover and refrigerat
e for at least one hour . <EOS>

> 1 c semisweet chocolate chips 1 pk (3 oz) cream cheese 1/2 c condensed milk 2 tb cashews, ground 1/2 c
chopped walnuts 1/4 ts almond extract 1 1/2 c flour 1/2 ts baking powder 1/4 ts salt 3/4 c sugar 1/2 c butter,
softened 1 ea egg 1/4 ts almond extract
= make the filling first . in a saucepan , combine chocolate chips , cream cheese and milk . melt over low heat , stirring cons
tantly . remove from heat . stir in nuts and almond extract ; blend well . set aside . in a large bowl , combine all remaining

```
ingredients . mix well until crumbly . press half the crust mixture into an oiled 7 x 11-inch pan . spread filling over crust .
sprinkle rest of crust mixture over filling . bake at 350 degrees for 20-25 minutes .
< mix the dry ingredients together . add the dry ingredients and mix well . add the flour and mix well . add the flour and mix
well . add the milk and mix well . add the flour and mix well . add the egg and mix well . add the egg and mix well . mix well
. add the milk and mix well . add the flour and mix well . add the flour and mix well . add the milk and mix well . add the mil
k and mix well . add the flour and mix well . add the flour and mix well . add the flour and mix well . add the flour and mix w
ell . add the milk and mix well . add the egg and mix well . add the milk and mix well . add the flour and mix well .
```

6. Evaluate and save the prediction for evaluation sentence

In [34]:
```
baseline2_output_sentence = evaluateSpecific_attn(baseline2_encoder, baseline2_decoder, evaluation_sentence, False)
```

```
> 2 c sugar, 1/4 c lemon juice, 1 c water, 1/3 c orange juice, 8 c strawberries
< mix all ingredients together . in a large bowl , combine all ingredients . mix well . pour into a greased 9 '' x 9 '' x 13 ''
baking pan . bake at 350 deg . for 20 minutes or until golden brown .  <EOS>
```

Implementation of Extension 1: Sequence-to-Sequence model with attention and text-preprocessing

1. Prepare the preprocess data

In [35]:
```
input_lang, output_lang, pairs, dev_pairs, test_pairs, my_test_indices  = prepareData(True) # preprocess = True
print("")
print("training pair example: ", random.choice(pairs))
print("")
print("dev pair example: ", random.choice(dev_pairs))
print("")
print("testing pair example: ", random.choice(test_pairs))
print("-----------------------------------------------------------------------")
print("max ingredients sentence length: ", input_lang.max_sentence)
print("min ingredients sentence length: ", input_lang.min_sentence)
print("max recipe sentence length: ", output_lang.max_sentence)
print("min recipe sentence length: ", output_lang.min_sentence)
```

```
Reading lines...
Read 100925 train sentence pairs
Counting train words...
Trimmed to 87770 train sentence pairs
Counted words:
Ingredients 14479
Recipe 27381
Read 793 dev sentence pairs
Counting dev words...
Trimmed to 695 dev sentence pairs
Read 773 test sentence pairs
Counting test words...
Trimmed to 685 test sentence pairs

training pair example:  ['spicy tomato juice pepper chili powder cubes salt chuck steak trimmed cut water clove garlic minced o
nion chopped cayenne pepper beef broth worcestershire sauce masa', 'brown meat and onions in a skillet drain add garlic tomato
juice broth worcestershire chili powder salt pepper and cayenne pepper simmer over low heat for hours adding more tomato juice
if needed combine masa and water whisk until smooth add to chili and cook for minutes or until thickened garnish with shredded
cheese sour cream and sliced jalapeno peppers if desired']

dev pair example:  ['apricot jam dijon mustard extra firm tofu thinly sliced garlic cloves crushed onion chopped scallions chop
ped tamari pn dill orange juice vegetable oil', 'pan fry onion and garlic in oil gradually add tamari mustard orange juice and
apricot puree stir until you have a smooth sauce in separate pan fry tofu in oil and add to sauce cook mixture over low heat un
til sauce reduces by half place tofu in serving dish and garnish with chopped shallots and dill serve with stir fried vegetable
s rice or a salad']

testing pair example:  ['shark steaks juice of three limes chopped tequila juice of three lemons finely chopped red pepper oliv
e oil pepper finely chopped cilantro jicama orange salad recipe fresh jalapenos finely finely chopped yellow follows chopped gr
een onions fresh cilantro sprigs salt pepper coco lopez', 'in a mixing bowl whisk all the ingredients together except for the s
hark and olive oil season with salt and pepper place the fish in a glass casserole dish pour the marinate over the shark and re
frigerate for hours remove the fish from the marinate heat the olive oil in a saut pan sear the shark for two minutes on each s
ide remove from the pan and slice mound the salad in the center of the plate arrange the shark around the salad garnish with fr
esh cilantro and essence']
---------------------------------------------------------------
max ingredients sentence length:  144
min ingredients sentence length:  1
max recipe sentence length:  149
min recipe sentence length:  1
```

2. Train the extension 1 model

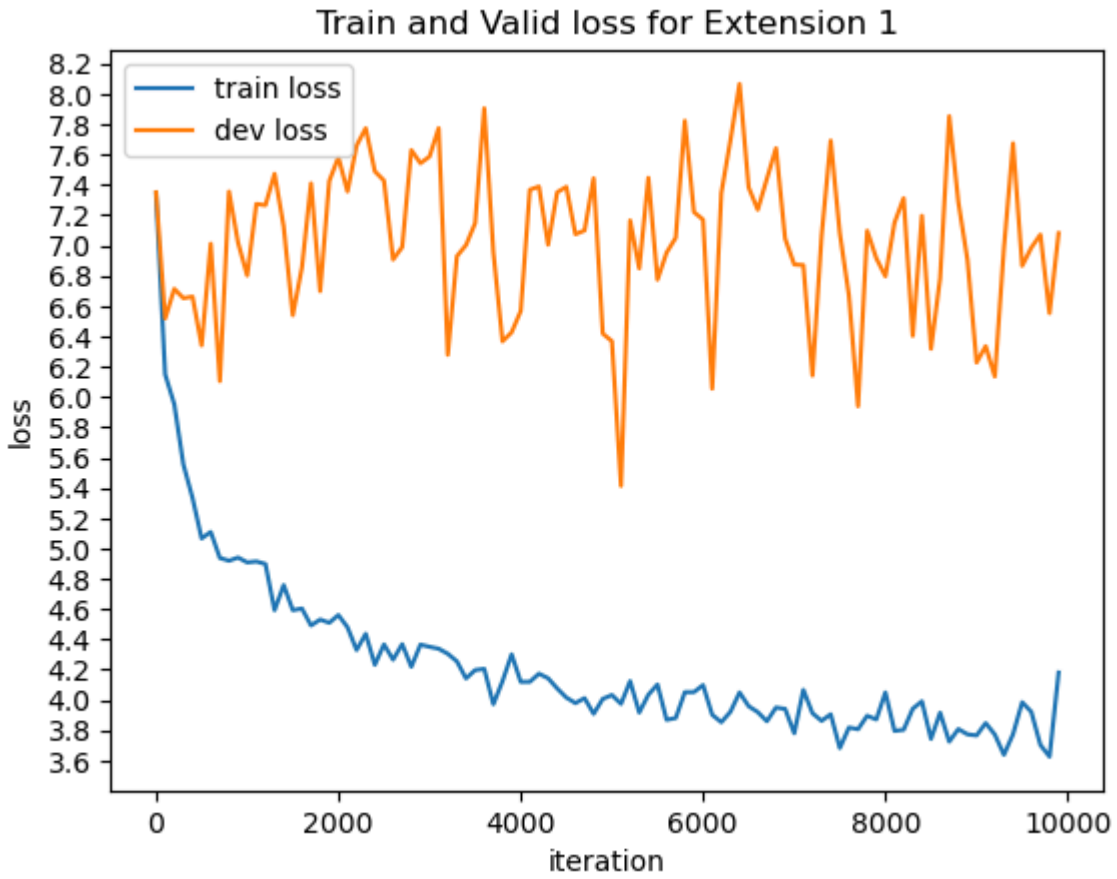
In [36]:
```
hidden_size = 256
n_iters = 10000
extension1_encoder = EncoderRNN(input_lang.n_words, hidden_size).to(device)
extension1_decoder = AttnDecoderRNN(hidden_size, output_lang.n_words, dropout_p=0.1).to(device)

extension1_plot_losses, extension1_plot_valid_losses = trainIters_attn(extension1_encoder,
                                                                       extension1_decoder, n_iters, print_every=1000)
```

```
3m 59s (- 35m 53s) (1000 10%) Train Loss: 5.5302 | Validation Loss: 6.7723
7m 40s (- 30m 41s) (2000 20%) Train Loss: 4.6798 | Validation Loss: 7.0863
11m 28s (- 26m 46s) (3000 30%) Train Loss: 4.3625 | Validation Loss: 7.4355
15m 11s (- 22m 46s) (4000 40%) Train Loss: 4.2191 | Validation Loss: 7.0371
18m 57s (- 18m 57s) (5000 50%) Train Loss: 4.0556 | Validation Loss: 7.1098
22m 32s (- 15m 1s) (6000 60%) Train Loss: 4.0039 | Validation Loss: 6.9060
26m 34s (- 11m 23s) (7000 70%) Train Loss: 3.9455 | Validation Loss: 7.3073
30m 19s (- 7m 34s) (8000 80%) Train Loss: 3.8604 | Validation Loss: 6.8343
33m 59s (- 3m 46s) (9000 90%) Train Loss: 3.8549 | Validation Loss: 7.0002
36m 48s (- 0m 0s) (10000 100%) Train Loss: 3.8216 | Validation Loss: 6.7908
```

3. Visualisation for train and valid loss

In [37]:
```
showPlot(n_iters, extension1_plot_losses, extension1_plot_valid_losses, "Train and Valid loss for Extension 1")
```

```
<Figure size 640x480 with 0 Axes>
```



4. Evaluate the metrics using test set, update the prediction to csv file

In [38]:
```
extension1_test_loss, extension1_given_items, extension1_extra_items, extension1_bleu, extension1_meteor, extension1_prediction_
    extension1_encoder, extension1_decoder)
print("extension 1 - Test loss: {}, Avg % given: {}, Avg extra: {}, BLEU: {}, METEOR: {}".format(
    extension1_test_loss, extension1_given_items, extension1_extra_items, extension1_bleu, extension1_meteor))

update_csv_recipe(extension1_prediction_words, "Generated Recipe - Extended 1", my_test_indices)
```

```
C:\Users\ngyij\anaconda3\Lib\site-packages\nltk\translate\bleu_score.py:552: UserWarning:
The hypothesis contains 0 counts of 4-gram overlaps.
Therefore the BLEU score evaluates to 0, independently of
how many N-gram overlaps of lower order it contains.
Consider using lower n-gram order or use SmoothingFunction()
  warnings.warn(_msg)
C:\Users\ngyij\anaconda3\Lib\site-packages\nltk\translate\bleu_score.py:552: UserWarning:
The hypothesis contains 0 counts of 3-gram overlaps.
Therefore the BLEU score evaluates to 0, independently of
how many N-gram overlaps of lower order it contains.
Consider using lower n-gram order or use SmoothingFunction()
  warnings.warn(_msg)
C:\Users\ngyij\anaconda3\Lib\site-packages\nltk\translate\bleu_score.py:552: UserWarning:
The hypothesis contains 0 counts of 2-gram overlaps.
Therefore the BLEU score evaluates to 0, independently of
how many N-gram overlaps of lower order it contains.
Consider using lower n-gram order or use SmoothingFunction()
  warnings.warn(_msg)

extension 1 - Test loss: 7.35031530172775, Avg % given: 0.7065693430656934, Avg extra: 9.64963503649635, BLEU: 0.01576338995230
5612, METEOR: 0.16522021979101054
```

5. Randomly evaluate 5 example

In [39]:
```
evaluateRandomly_attn(extension1_encoder, extension1_decoder)
```

> l more water leaves epazote onions kg black beans dried
= let the dried black beans stand one night with water enough for to cover all next day cook with the same water add more if ne eded adding epazote leaves onion and salt when the beans are cooking you only can add boiling water if need to add more
< in a large saucepan bring water to a boil over high heat reduce heat to low and simmer for minutes or until tender drain and rinse with cold water drain and rinse with cold water drain rinse with cold water drain and rinse with cold water to stop cooki ng spray in a large saucepan bring water to a boil over high heat reduce heat to low and simmer for minutes or until tender dra in off fat stir in water and water bring to a boil reduce heat and simmer for minutes or until tender drain off fat stir in wat er and cook until soft add beans and cook stirring occasionally until thickened and bubbly stirring constantly add remaining in gredients except water and bring to a boil reduce heat and simmer for minutes or until thickened serve immediately <EOS>

> garlic cloves sliced green bell pepper diced dried oregano can tomato sauce removed dry red wine op sweet italian sausage arr ot thinly sliced zucchini sliced ounces purchased grated chopped onion chopped tomatoes about chorizo sausage casings parmesan cheese freshly dried basil fresh cheese tortelli beef stock canned broth
= saute italian sausage in heavy dutch oven over medium high heat until cooked through crumbling with back of spoon about minut es using slotted spoon transfer sausage to large bowl pour off all but tablespoon drippings from dutch oven add onion and garli c to dutch oven and saute until translucent about minutes return sausage to dutch oven add stock tomatoes tomato sauce zucchini carrot bell pepper wine basil and oregan bring to simmer before continuing rrb add tortellini to soup and cook until tender abo ut minutes season soup to taste with salt and pepper ladle soup into bowls sprinkle with parmesan and serve servings
< heat oven to f in large saucepan combine remaining ingredients except onion and garlic in a large saucepan bring to a boil re duce heat to low and simmer for minutes or until tender drain and rinse with cold water drain and rinse with cold water to stop cooking spray with cooking spray add onion and saute until sausage is softened add tomatoes and tomatoes and cook until sausage is tender about minutes add tomatoes and tomato sauce and simmer uncovered for minutes or until sausage is tender and sausage i s no longer pink stirring occasionally to prevent sticking add tomatoes and cook stirring occasionally until mixture thickens s lightly thickened and bubbly stirring occasionally about minutes or until sausage is tender and sausage is tender serve with to mato sauce or tomato sauce <EOS>

> cold strong french roast coffee grated orange peel granulated sugar tawny port ds cinnamon
= conbibe ingredients and mix in a blender cup at high speed pour into chilled wine glasses milligrams cholesterol milligrams s odium
< combine first ingredients in a large saucepan bring to a boil reduce heat and simmer for minutes or until thickened stirring occasionally remove from heat and stir in the orange juice and orange extract and continue to cook until the mixture thickens s lightly and the mixture thickens and the mixture thickens and the mixture is thickened and bubbly the mixture stirring constant ly remove from heat and let stand for minutes add the cinnamon and cook stirring constantly until the mixture thickens slightly thickened and bubbly stirring constantly add the cream and orange juice and bring to a boil stirring constantly reduce the heat and simmer for minutes or until the mixture thickens and thickens add the cinnamon and nutmeg and stir until the mixture thicke ns and the mixture thickens stir in the orange juice and orange extract and orange juice and stir until the mixture thickens an d thickened stirring constantly

> onion small grated ground beef chili powder water green chilies tomato sauce monterey jack cheese taco shells
= coarsely break taco shells in quart bowl mix ground beef and onion cover bowl with waxed paper cook at high lrb rrb minutes s tirring and breaking up meat into small pieces every minutes tilt bowl skim and discard excess fat stir in tomato sauce water a nd chili powder cover and cook at high minutes or until meat mixture thickens stirring after minutes stir in undrained green ch ilies and shredded cheese in shallow quart casserole place of the coarsely broken tacos top with meat mixture cover with casser ole lid or large plate cook at medium high lrb rrb to minutes until meat mixture is hot let stand still covered minutes sprinkl e with remaining tacos
< combine all ingredients and mix well chill until firm <EOS>

```
> corn butter margarine quick oats uncooked salt sugar nestle toll house milk chocolate morsels all purpose flour milk vanilla
extract
= preheat oven to f melt butter in medium saucepan over low heat remove from heat stir in oats sugar flour corn syrup milk vani
lla extract and salt mix well drop by measuring teaspoonfuls about apart onto foil lined cookie sheets spread thin with rubber
spatula bake minutes cool on cookie sheets peel foil away from cookies stir until smooth spread chocolate on flat side of half
the cookies top with remaining cookies
< in a large saucepan combine flour baking soda and salt and stir until blended stir in flour and salt and stir until blended s
tir in milk and salt and stir until blended stir in flour and milk stir until smooth stir in chocolate and vanilla pour over ba
tter bake at f for minutes or until toothpick inserted in center comes out clean cool minutes before removing rim of pan cool o
n wire rack minutes before removing rim of pan <EOS>
```

Evaluate and save the prediction for evaluation sentence

In [40]:
```
extension1_output_sentence = evaluateSpecific_attn(extension1_encoder, extension1_decoder, evaluation_sentence, True)
```

```
> sugar lemon juice water orange juice strawberries
< in a large saucepan combine water orange juice orange juice and water bring to a boil stirring constantly until thickened and
bubbly stirring constantly add lemon juice and lemon juice to taste and cook until thickened stirring constantly remove from he
at and stir in lemon juice and lemon juice stir in lemon juice and lemon juice and lemon juice and bring to a boil stirring con
stantly until the mixture thickens and thickens stir in the orange juice and lemon juice and lemon juice and cook stirring cons
tantly until the mixture thickens stirring constantly remove from heat and stir in the orange juice and lemon juice and bring t
o a boil reduce heat to low and simmer for minutes or until the mixture thickens slightly thickened and bubbly stirring constan
tly remove from heat and stir in the orange juice and lemon juice and lemon juice and bring to a boil stirring
```

Implementation of Extension 2: Sequence-to-Sequence model with attention, text-preprocessing, pretrained word-embedding with word2vec and Neural Text Generation with Predicate Logic Constraints

1. word2vec library import

In [41]:
```
import gensim.downloader as api

word2vec_model_name = "word2vec-google-news-300"
word2vec_model = api.load(word2vec_model_name)
```

2. word embedding function

In [42]:
```
embedding_dim = 300
def load_embeddings(word2vec_model, vocab_lang):
    embeddings = np.zeros((vocab_lang.n_words, embedding_dim))

    for word in (word2vec_model.index_to_key): # for every word in word2vec
        if word in vocab_lang.word2index: # if the word is in the dictionary given
            embeddings[vocab_lang.word2index[word]] = word2vec_model[word] # load the word embedding
    embeddings = torch.from_numpy(embeddings).float()
    return embeddings
```

3. Load the preprocess data and word embeddings

In [43]:
```
input_lang, output_lang, pairs, dev_pairs, test_pairs, my_test_indices  = prepareData(True)

ingredient_embedding = load_embeddings(word2vec_model, input_lang)
recipe_embedding = load_embeddings(word2vec_model, output_lang)

print("---------------------------------------------------------------------")
print("max ingredients sentence length: ", input_lang.max_sentence)
print("min ingredients sentence length: ", input_lang.min_sentence)
print("max recipe sentence length: ", output_lang.max_sentence)
print("min recipe sentence length: ", output_lang.min_sentence)
```

```
Reading lines...
Read 100925 train sentence pairs
Counting train words...
Trimmed to 87770 train sentence pairs
Counted words:
Ingredients 14479
Recipe 27381
Read 793 dev sentence pairs
Counting dev words...
Trimmed to 695 dev sentence pairs
Read 773 test sentence pairs
Counting test words...
Trimmed to 685 test sentence pairs
---------------------------------------------------------------------
max ingredients sentence length:  144
min ingredients sentence length:  1
max recipe sentence length:  149
min recipe sentence length:  1
```

4. Encoder with pretrained embedding

In [44]:
```
class EncoderRNN(nn.Module):
    def __init__(self, input_size, embedding_size, hidden_size, pretrained_embeddings=None):
        super(EncoderRNN, self).__init__()
        self.hidden_size = hidden_size
        self.embedding_size = embedding_size

        # Load pretrained embedding
        self.embedding = nn.Embedding.from_pretrained(pretrained_embeddings)

        self.lstm = nn.LSTM(embedding_size, hidden_size)

    def forward(self, input, hidden):
        embedded = self.embedding(input).view(1, 1, -1)
        output = embedded
        output, (hidden, cell_state) = self.lstm(output, hidden)
        return output, (hidden, cell_state)

    def initHidden(self):
        return (torch.zeros(1, 1, self.hidden_size, device=device),
                torch.zeros(1, 1, self.hidden_size, device=device))
```

5. Decoder with attention and pretrained embedding

In [45]:
```
class AttnDecoderRNN(nn.Module):
    def __init__(self, embedding_size, hidden_size, output_size, dropout_p=0.1, max_length=MAX_LENGTH, pretrained_embeddings=Non
        super(AttnDecoderRNN, self).__init__()
        self.hidden_size = hidden_size
        self.embedding_size = embedding_size
        self.output_size = output_size
        self.dropout_p = dropout_p
        self.max_length = max_length

        # Load pretrained embedding
        self.embedding = nn.Embedding.from_pretrained(pretrained_embeddings)

        self.attn = nn.Linear(self.hidden_size + self.embedding_size, self.max_length)
        self.attn_combine = nn.Linear(self.hidden_size + self.embedding_size, self.hidden_size)
        self.dropout = nn.Dropout(self.dropout_p)
        self.lstm = nn.LSTM(self.hidden_size, self.hidden_size)
        self.out = nn.Linear(self.hidden_size, self.output_size)

    def forward(self, input, hidden, encoder_outputs):
        embedded = self.embedding(input).view(1, 1, -1)
        embedded = self.dropout(embedded)

        attn_weights = F.softmax(
            self.attn(torch.cat((embedded[0], hidden[0][0]), 1)), dim=1)
        attn_applied = torch.bmm(attn_weights.unsqueeze(0),
                                 encoder_outputs.unsqueeze(0))

        output = torch.cat((embedded[0], attn_applied[0]), 1)
        output = self.attn_combine(output).unsqueeze(0)

        output = F.relu(output)
        output, (hidden, cell_state) = self.lstm(output, hidden)

        output = F.log_softmax(self.out(output[0]), dim=1)
        return output, (hidden, cell_state), attn_weights

    def initHidden(self):
        return (torch.zeros(1, 1, self.hidden_size, device=device),
                torch.zeros(1, 1, self.hidden_size, device=device))
```

6. Extension 2 Mechanism application (Added in train function)

In [46]:
```
teacher_forcing_ratio = 1.0


def train_attn_ext2(input_word, input_tensor, target_tensor, encoder, decoder, encoder_optimizer, decoder_optimizer,
                    criterion, max_length=MAX_LENGTH):
    encoder_hidden = encoder.initHidden()

    encoder_optimizer.zero_grad()
    decoder_optimizer.zero_grad()

    input_length = input_tensor.size(0)
    target_length = target_tensor.size(0)

    encoder_outputs = torch.zeros(max_length, encoder.hidden_size, device=device)

    loss = 0

    for ei in range(input_length):
        encoder_output, encoder_hidden = encoder(
            input_tensor[ei], encoder_hidden)
        encoder_outputs[ei] = encoder_output[0, 0]

    decoder_input = torch.tensor([[SOS_token]], device=device)

    decoder_hidden = encoder_hidden

    use_teacher_forcing = True if random.random() < teacher_forcing_ratio else False

    if use_teacher_forcing:
        # Teacher forcing: Feed the target as the next input
        for di in range(target_length):
            decoder_output, decoder_hidden, decoder_attention = decoder(
                decoder_input, decoder_hidden, encoder_outputs)
            loss += criterion(decoder_output, target_tensor[di])
            decoder_input = target_tensor[di]  # Teacher forcing

    else:
        # Without teacher forcing: use its own predictions as the next input
        for di in range(target_length):
            decoder_output, decoder_hidden, decoder_attention = decoder(
                decoder_input, decoder_hidden, encoder_outputs)
            topv, topi = decoder_output.topk(1)
            decoder_input = topi.squeeze().detach()  # detach from history as input

            penalty = 0
            compensation = 0
```

```
            current_loss = criterion(decoder_output, target_tensor[di])

            # **EXTENSION 2 newly applied mechanism**
            _, topi_data = decoder_output.data.topk(1)
            current_word_guess = output_lang.index2word[topi_data.item()]
            # if the word is is the ingredients dictionary
            if (current_word_guess in input_lang.word2index):
                # but it is not in the given ingredients, apply penalise
                if (current_word_guess not in input_word):
                    penalty = current_loss*1.5
                # if it is in the given ingredients, give compensation
                else:
                    compensation = current_loss*0.5

            loss += (current_loss + penalty - compensation)
            if decoder_input.item() == EOS_token:
                break


    loss.backward()

    encoder_optimizer.step()
    decoder_optimizer.step()

    return loss.item() / target_length


def validate_attn_ext2(input_tensor, target_tensor, encoder, decoder, criterion, max_length=MAX_LENGTH):
    encoder_hidden = encoder.initHidden()

    input_length = input_tensor.size(0)
    target_length = target_tensor.size(0)

    encoder_outputs = torch.zeros(max_length, encoder.hidden_size, device=device)

    loss = 0

    for ei in range(input_length):
        encoder_output, encoder_hidden = encoder(
            input_tensor[ei], encoder_hidden)
        encoder_outputs[ei] = encoder_output[0, 0]

    decoder_input = torch.tensor([[SOS_token]], device=device)

    decoder_hidden = encoder_hidden
```

```
    for di in range(target_length):
        decoder_output, decoder_hidden, decoder_attention = decoder(
            decoder_input, decoder_hidden, encoder_outputs)
        topv, topi = decoder_output.topk(1)
        decoder_input = topi.squeeze().detach()  # detach from history as input

        loss += criterion(decoder_output, target_tensor[di])
        if decoder_input.item() == EOS_token:
            break

    return loss.item() / target_length


def trainIters_attn_ext2(encoder, decoder, n_iters, print_every=1000, plot_every=100, learning_rate=0.001):
    start = time.time()
    plot_losses = []
    print_loss_total = 0  # Reset every print_every
    plot_loss_total = 0  # Reset every plot_every

    # validating
    plot_valid_losses = []
    print_valid_loss_total = 0  # Reset every print_every
    plot_valid_loss_total = 0  # Reset every plot_every

    encoder_optimizer = optim.Adam(encoder.parameters(), lr=learning_rate)
    decoder_optimizer = optim.Adam(decoder.parameters(), lr=learning_rate)

    training_pairs = []
    input_words = []
    # convert pairs to tensors
    for i in range(n_iters):
        current_input_word, current_training_pair = tensorsFromPairWithWords(random.choice(pairs))
        training_pairs.append(current_training_pair)
        input_words.append(current_input_word)

    validating_pairs = [tensorsFromPair(random.choice(dev_pairs))
                        for i in range(n_iters)]    # validating set size
    criterion = nn.NLLLoss()

    for iter in range(1, n_iters + 1):
        # training
        training_pair = training_pairs[iter - 1]
        input_tensor = training_pair[0]
        target_tensor = training_pair[1]
        input_word = input_words[iter - 1]
```

```
        # validating
        validating_pair = validating_pairs[iter - 1]
        valid_input_tensor = validating_pair[0]
        valid_target_tensor = validating_pair[1]

        # training
        loss = train_attn_ext2(input_word, input_tensor, target_tensor, encoder,
                     decoder, encoder_optimizer, decoder_optimizer, criterion)
        print_loss_total += loss
        plot_loss_total += loss
        # validating
        valid_loss = validate_attn_ext2(valid_input_tensor, valid_target_tensor, encoder, decoder, criterion)
        print_valid_loss_total += valid_loss
        plot_valid_loss_total += valid_loss

        if iter % print_every == 0:
            print_loss_avg = print_loss_total / print_every
            print_loss_total = 0

            print_valid_loss_avg = print_valid_loss_total / print_every
            print_valid_loss_total = 0
            print('%s (%d %d%%) Train Loss: %.4f | Validation Loss: %.4f' % (timeSince(start, iter / n_iters),
                                         iter, iter / n_iters * 100, print_loss_avg, print_valid_loss_avg))

        if iter % plot_every == 0:
            plot_loss_avg = plot_loss_total / plot_every
            plot_valid_loss_avg = plot_valid_loss_total / plot_every
            plot_losses.append(plot_loss_avg)
            plot_valid_losses.append(plot_valid_loss_avg)
            plot_loss_total = 0
            plot_valid_loss_total = 0

    return plot_losses, plot_valid_losses
```

7. Train for extension 2

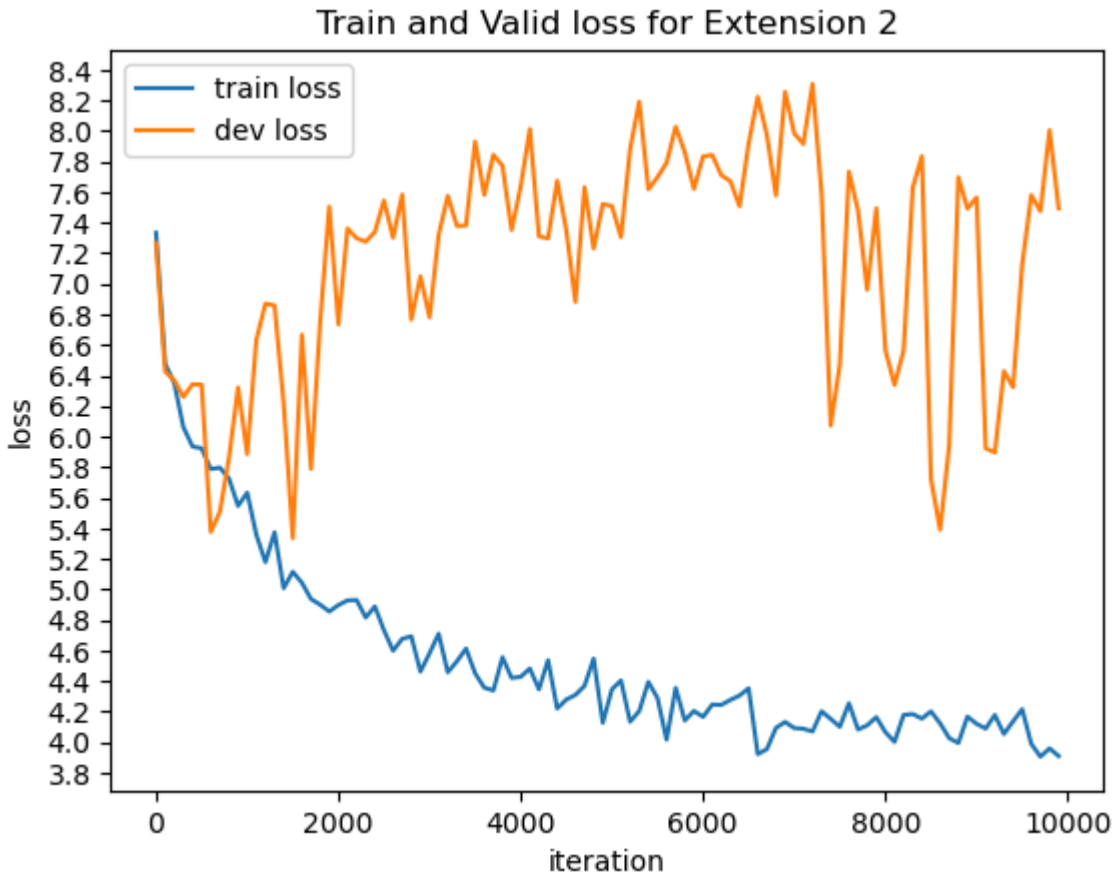
In [47]:
```
hidden_size = 256
n_iters = 10000
embedding_size = 300
extension2_encoder = EncoderRNN(input_lang.n_words, embedding_size, hidden_size,
                                            pretrained_embeddings=ingredient_embedding).to(device)

extension2_decoder = AttnDecoderRNN(embedding_size, hidden_size, output_lang.n_words,
                                                dropout_p=0.1, pretrained_embeddings=recipe_embedding).to(device)

extension2_plot_losses, extension2_plot_valid_losses = trainIters_attn_ext2(extension2_encoder,
                                                                            extension2_decoder, n_iters, print_every=1000)
```

```
2m 18s (- 20m 46s) (1000 10%) Train Loss: 6.0937 | Validation Loss: 6.2064
4m 28s (- 17m 54s) (2000 20%) Train Loss: 5.1404 | Validation Loss: 6.4505
6m 40s (- 15m 35s) (3000 30%) Train Loss: 4.7626 | Validation Loss: 7.2249
8m 53s (- 13m 20s) (4000 40%) Train Loss: 4.5018 | Validation Loss: 7.4911
11m 6s (- 11m 6s) (5000 50%) Train Loss: 4.3649 | Validation Loss: 7.4543
13m 20s (- 8m 53s) (6000 60%) Train Loss: 4.2493 | Validation Loss: 7.7500
15m 32s (- 6m 39s) (7000 70%) Train Loss: 4.1698 | Validation Loss: 7.8512
17m 41s (- 4m 25s) (8000 80%) Train Loss: 4.1323 | Validation Loss: 7.4004
19m 45s (- 2m 11s) (9000 90%) Train Loss: 4.1102 | Validation Loss: 6.7175
21m 48s (- 0m 0s) (10000 100%) Train Loss: 4.0561 | Validation Loss: 6.9814
```

8. Visualisation for extension 2 model

In [48]:
```
showPlot(n_iters, extension2_plot_losses, extension2_plot_valid_losses, "Train and Valid loss for Extension 2")
```

```
<Figure size 640x480 with 0 Axes>
```



9. Evaluation metrics using test set, update the csv file

In [49]:
```
extension2_test_loss, extension2_given_items, extension2_extra_items, extension2_bleu, extension2_meteor, extension2_prediction_
    extension2_encoder, extension2_decoder)
print("extension 2 - Test loss: {}, Avg % given: {}, Avg extra: {}, BLEU: {}, METEOR: {}".format(
    extension2_test_loss, extension2_given_items, extension2_extra_items, extension2_bleu, extension2_meteor))

update_csv_recipe(extension2_prediction_words, "Generated Recipe - Extended 2", my_test_indices)
```

```
C:\Users\ngyij\anaconda3\Lib\site-packages\nltk\translate\bleu_score.py:552: UserWarning:
The hypothesis contains 0 counts of 4-gram overlaps.
Therefore the BLEU score evaluates to 0, independently of
how many N-gram overlaps of lower order it contains.
Consider using lower n-gram order or use SmoothingFunction()
  warnings.warn(_msg)
C:\Users\ngyij\anaconda3\Lib\site-packages\nltk\translate\bleu_score.py:552: UserWarning:
The hypothesis contains 0 counts of 3-gram overlaps.
Therefore the BLEU score evaluates to 0, independently of
how many N-gram overlaps of lower order it contains.
Consider using lower n-gram order or use SmoothingFunction()
  warnings.warn(_msg)
C:\Users\ngyij\anaconda3\Lib\site-packages\nltk\translate\bleu_score.py:552: UserWarning:
The hypothesis contains 0 counts of 2-gram overlaps.
Therefore the BLEU score evaluates to 0, independently of
how many N-gram overlaps of lower order it contains.
Consider using lower n-gram order or use SmoothingFunction()
  warnings.warn(_msg)

extension 2 - Test loss: 7.958194672039772, Avg % given: 0.5518248175182482, Avg extra: 10.932846715328466, BLEU: 0.00971664354
7929478, METEOR: 0.14579308536623606
```

10. Randomly evaluate 5 examples from prediction for test set

In [50]:
```
evaluateRandomly_attn(extension2_encoder, extension2_decoder)
```

> thyme chopped lasagna noodles mushroom wild olive oil fontina cheese cloves garlic minced onion thinly sliced butter marsala wine italian parsley chopped parmesan cheese mozzarella cheese flour salt pepper milk nutmeg
= preheat oven to f in a large skillet heat olive oil on med high heat add garlic and onions and saute to minutes add mushrooms parsley and thyme saute minutes stir in wine simmer for minutes and set aside in saucepan make a roux with butter flour and nut meg stir in broth and milk cook until sauce thickens stir in fontina cheese in bottom of x baking dish arrange noodles spread w ith mushroom mixture and of the sauce repeat finishing with sauce sprinkle with mozzarella and parmesan cover with foil and bak e minutes remove foil bake more minutes
< in a large bowl combine the flour and salt and pepper in a large bowl add the eggs and stir in the flour and salt add the flo ur and salt and pepper to taste add the rest of the ingredients and the mixture and the mixture and the mixture and the mixture and the mixture and the mixture and the mixture and the mixture and the mixture is smooth and elastic and the mixture is smooth and elastic and about minutes rrb add the rest of the ingredients and the mixture and the mixture and the mixture and the mixture and the mixtu re and the mixture and the mixture is smooth and elastic and the mixture is smooth and elastic and about minutes rrb add the re st of the ingredients and the mixture and the mixture and the mixture and the mixture and the mixture and the mixture and the m ixture is smooth

> chicken grated parmesan cheese bread crumbs black pepper taste salt taste margarine melted fresh parsley chopped garlic clove s chopped
= chickens into serving sized pieces place this mixture in a shallow bowl place cup of melted margarine in a shallow dish prehe at oven to degrees have ready a shallow baking pan dip chicken pieces first in the margarine then into the bread crumb mixture place in baking pan in a single layer drizzle rest of melted margarine evenly over chicken bake for minutes turning only once
< in a large skillet over medium heat add the onion and garlic and cook until the onion is tender add the onion and saute for m inutes add the garlic and cook for minutes add the tomatoes and cook for minutes add the rice and stir fry for minutes add the remaining ingredients and cook for minutes add the rice and the sauce and cook for minutes add the rice and stir fry for minute s add the remaining ingredients and stir until the mixture is well blended add the remaining ingredients and cook for minutes a dd the remaining ingredients and cook for minutes add the rice and stir fry for minutes add the remaining ingredients and cook for minutes add the rice and the sauce and cook for minutes add the remaining ingredients and cook for minutes add the rice and stir fry for minutes add the remaining ingredients and cook

> lemon juice plain yogurt curry powder chicken legs packed brown sugar
= trim off any fat and excess skin from chicken cut at joint into thighs and drumsticks in small saucepan lsb see tip below rsb heat curry powder with ts vegetable oil over medium heat for minutes or until bubbling pour into large bowl add yogurt lemon ju ice and sugar whisk to blend well add chicken turning to coat marinate at room temperature for minutes save on time and dish wa shing by microwaving the oil and curry powder in the large bowl until bubbling place chicken fleshy side up on foil lined bakin g sheet brush with remaining yogurt marinade bake in f c oven for minutes or until browned and juices run clear when chicken is pierced serve with rice chutney and slices of radish and cucumber tossed with yogurt and seasoned with a touch of mint and caye nne
< in a large bowl combine the flour and salt and pepper in a large bowl and add the rest of the ingredients and the mixture and the mixture and the mixture and the mixture and the mixture and the mixture and the mixture and the mixture is smooth and elastic and about min utes rrb add the rest of the ingredients and the mixture and the mixture and the mixture and the mixture and the mixture is smo oth and elastic to the mixture is reached in a large bowl combine the flour and salt add the rest of the mixture and the mixtur e and the mixture and the mixture and the mixture and the mixture and the mixture is smooth and elastic fold in the flour and t he butter and the mixture add the egg whites and the egg whites and the egg whites and the egg whites add the egg whites and th e

> truffle peelings quail salt mushrooms dried minced onion chicken broth minced parsley fresh bread crumbs cayenne pepper flour oil

```
= salt pepper quail inside and out combine mushrooms bread crumbs salt cayenne pepper and truffles saute in butter stuff quail
with this mixture make roux by browning flour in oil add stock onions and parsley to browned flour then pour over quail which h
ave been put into baking pan bake hour at basting frequently
< in a large skillet over medium heat add the onion and garlic and cook until the onion is tender add the onion and saute for m
inutes add the garlic and cook for minutes add the onion and cook until the onion is tender add the garlic and cook until the m
ixture is golden add the rice and stir fry for minutes add the remaining ingredients and stir until the mixture is smooth and t
he mixture is reached to the mixture is reached in the same listed of the same of the sauce and the cooking cooking for minutes
add the remaining ingredients and cook for minutes add the rice and stir fry for minutes add the remaining ingredients and cook
for minutes add the rice and stir fry for minutes add the remaining ingredients and cook for minutes add the remaining ingredie
nts and stir until the mixture is well blended

> hot vegetable stock hot water garnish raisins toasted sesame fresh watercress sprigs unsalted peanuts dry crushed red pepper
teaspoon you prefer spicy flavoring peanut butter crunchy rice vinegar safflower oil cloves garlic minced szechuwan peanut dres
sing stalks celery cut julienne strips fresh bean sprouts soy sauce cucumber thinly sliced broccoli florets cubed firm tofu cub
es carrots shredded italian plum tomatoes cut thin wedges
= in a large bowl gently toss together salad ingredients in a small bowl combine dressing ingredients pour over salad and toss
again top with garnish for best quality toss together just before serving to cook bean thread immerse it in boiling water for a
bout minutes to soften drain rinse with cool water and cut into inch strands
< in a large skillet over medium heat add the onion and garlic and cook until the onion is tender add the onion and saute for m
inutes add the garlic and cook for minutes add the tomatoes and the stock and bring to a boil and cook for minutes add the toma
toes and cook for minutes add the tomatoes and the stock and bring to a boil and cook for minutes add the rice and stir to cook
for minutes add the rice and stir fry for minutes add the remaining ingredients and cook for minutes add the rice and the sauce
and cook for minutes add the remaining ingredients and stir until the mixture is absorbed and the sauce is too thick and the sa
uce is too thick and the mixture is too thick and about minutes rrb the mixture is cooking <EOS>
```

11. Evaluate and save the prediction for evaluation_sentence

In [51]:
```
extension2_output_sentence = evaluateSpecific_attn(extension2_encoder, extension2_decoder, evaluation_sentence, True)
```

```
> sugar lemon juice water orange juice strawberries
< combine all ingredients in a saucepan bring to a boil and simmer for minutes or until the mixture is tender and the mixture i
s reached <EOS>
```

Train and valid loss of 4 models

In [52]:
```
x_range = [i for i in range(0, n_iters-1, 100)]
plt.figure()
fig, ax = plt.subplots()
loc = ticker.MultipleLocator(base=0.2)
ax.yaxis.set_major_locator(loc)
plt.xlabel('iteration')
plt.ylabel('loss')
plt.plot(x_range, baseline1_plot_losses, label='train loss of Baseline 1', linestyle=':', color='red')
plt.plot(x_range, baseline1_plot_valid_losses, label='dev loss of Baseline 1', color='red')

plt.plot(x_range, baseline2_plot_losses, label='train loss of Baseline 2', linestyle=':', color='orange')
plt.plot(x_range, baseline2_plot_valid_losses, label='dev loss of Baseline 2', color='orange')

plt.plot(x_range, extension1_plot_losses, label='train loss of Extension 1', linestyle=':', color='green')
plt.plot(x_range, extension1_plot_valid_losses, label='dev loss of Extension 1', color='green')

plt.plot(x_range, extension2_plot_losses, label='train loss of Extension 2', linestyle=':', color='blue')
plt.plot(x_range, extension2_plot_valid_losses, label='dev loss of Extension 2', color='blue')
plt.legend(loc='upper left', bbox_to_anchor=(1, 1))
plt.title("Train and Valid loss for 4 different models")
```

Out[52]:
```
Text(0.5, 1.0, 'Train and Valid loss for 4 different models')

<Figure size 640x480 with 0 Axes>
```


Train and Valid loss for 4 different models
loss
iteration
0
2000
4000
6000
8000
10000
train loss of Baseline 1
dev loss of Baseline 1
train loss of Baseline 2
dev loss of Baseline 2
train loss of Extension 1
dev loss of Extension 1
train loss of Extension 2
dev loss of Extension 2

Benchmarks for Metrics Table

In [53]:
```
ingredient_sample = "2 c sugar, 1/4 c lemon juice, 1 c water, 1/3 c orange juice, 8 c strawberries"
gold_recipe_sample = "combine sugar and water in medium saucepan . Heat , stirring , until sugar dissolves , then boil 5 minutes
generated_recipe_sample = "Combine sugar and water in a medium saucepan . Heat, stirring, until sugar dissolves . Bring to a boi

input_lang, output_lang, _, _, _, _ = prepareData(True)

ingredient_sample = seperated_ingredient(normalizeString(ingredient_sample))

# return the calculated bleu and meteor score
given_avg, extra_avg, bleu_4_score, my_meteor_score = bleu_meteor(ingredient_sample,
                                                                  generated_recipe_sample, gold_recipe_sample)


print("BLEU: {}, METEOR: {}, Avg extra: {}, Avg % given: {}".format(bleu_4_score, my_meteor_score, extra_avg, given_avg))
```

```
Reading lines...
Read 100925 train sentence pairs
Counting train words...
Trimmed to 87770 train sentence pairs
Counted words:
Ingredients 14479
Recipe 27381
Read 793 dev sentence pairs
Counting dev words...
Trimmed to 695 dev sentence pairs
Read 773 test sentence pairs
Counting test words...
Trimmed to 685 test sentence pairs
BLEU: 0.18894258771930728, METEOR: 0.590432393180899, Avg extra: 2, Avg % given: 1.0
```