```
!pip3 install torch numpy matplotlib
```

```
Requirement already satisfied: torch in
/usr/local/lib/python3.10/dist-packages (2.2.1+cu121)
Requirement already satisfied: numpy in
/usr/local/lib/python3.10/dist-packages (1.25.2)
Requirement already satisfied: matplotlib in
/usr/local/lib/python3.10/dist-packages (3.7.1)
Requirement already satisfied: filelock in
/usr/local/lib/python3.10/dist-packages (from torch) (3.14.0)
Requirement already satisfied: typing-extensions>=4.8.0 in
/usr/local/lib/python3.10/dist-packages (from torch) (4.11.0)
Requirement already satisfied: sympy in
/usr/local/lib/python3.10/dist-packages (from torch) (1.12)
Requirement already satisfied: networkx in
/usr/local/lib/python3.10/dist-packages (from torch) (3.3)
Requirement already satisfied: jinja2 in
/usr/local/lib/python3.10/dist-packages (from torch) (3.1.4)
Requirement already satisfied: fsspec in
/usr/local/lib/python3.10/dist-packages (from torch) (2023.6.0)
Collecting nvidia-cuda-nvrtc-cu12==12.1.105 (from torch)
  Using cached nvidia_cuda_nvrtc_cu12-12.1.105-py3-none-
manylinux1_x86_64.whl (23.7 MB)
Collecting nvidia-cuda-runtime-cu12==12.1.105 (from torch)
  Using cached nvidia_cuda_runtime_cu12-12.1.105-py3-none-
manylinux1_x86_64.whl (823 kB)
Collecting nvidia-cuda-cupti-cu12==12.1.105 (from torch)
  Using cached nvidia_cuda_cupti_cu12-12.1.105-py3-none-
manylinux1_x86_64.whl (14.1 MB)
Collecting nvidia-cudnn-cu12==8.9.2.26 (from torch)
  Using cached nvidia_cudnn_cu12-8.9.2.26-py3-none-
manylinux1_x86_64.whl (731.7 MB)
Collecting nvidia-cublas-cu12==12.1.3.1 (from torch)
  Using cached nvidia_cublas_cu12-12.1.3.1-py3-none-
manylinux1_x86_64.whl (410.6 MB)
Collecting nvidia-cufft-cu12==11.0.2.54 (from torch)
  Using cached nvidia_cufft_cu12-11.0.2.54-py3-none-
manylinux1_x86_64.whl (121.6 MB)
Collecting nvidia-curand-cu12==10.3.2.106 (from torch)
  Using cached nvidia_curand_cu12-10.3.2.106-py3-none-
manylinux1_x86_64.whl (56.5 MB)
Collecting nvidia-cusolver-cu12==11.4.5.107 (from torch)
  Using cached nvidia_cusolver_cu12-11.4.5.107-py3-none-
manylinux1_x86_64.whl (124.2 MB)
Collecting nvidia-cusparse-cu12==12.1.0.106 (from torch)
  Using cached nvidia_cusparse_cu12-12.1.0.106-py3-none-
manylinux1_x86_64.whl (196.0 MB)
Collecting nvidia-nccl-cu12==2.19.3 (from torch)
  Using cached nvidia_nccl_cu12-2.19.3-py3-none-manylinux1_x86_64.whl
(166.0 MB)
```

```
Collecting nvidia-nvtx-cu12==12.1.105 (from torch)
  Using cached nvidia_nvtx_cu12-12.1.105-py3-none-
manylinux1_x86_64.whl (99 kB)
Requirement already satisfied: triton==2.2.0 in
/usr/local/lib/python3.10/dist-packages (from torch) (2.2.0)
Collecting nvidia-nvjitlink-cu12 (from nvidia-cusolver-
cu12==11.4.5.107->torch)
  Using cached nvidia_nvjitlink_cu12-12.4.127-py3-none-
manylinux2014_x86_64.whl (21.1 MB)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (1.2.1)
Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (4.51.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (1.4.5)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (24.0)
Requirement already satisfied: pillow>=6.2.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (2.8.2)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7-
>matplotlib) (1.16.0)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.10/dist-packages (from jinja2->torch) (2.1.5)
Requirement already satisfied: mpmath>=0.19 in
/usr/local/lib/python3.10/dist-packages (from sympy->torch) (1.3.0)
Installing collected packages: nvidia-nvtx-cu12, nvidia-nvjitlink-
cu12, nvidia-nccl-cu12, nvidia-curand-cu12, nvidia-cufft-cu12, nvidia-
cuda-runtime-cu12, nvidia-cuda-nvrtc-cu12, nvidia-cuda-cupti-cu12,
nvidia-cublas-cu12, nvidia-cusparse-cu12, nvidia-cudnn-cu12, nvidia-
cusolver-cu12
Successfully installed nvidia-cublas-cu12-12.1.3.1 nvidia-cuda-cupti-
cu12-12.1.105 nvidia-cuda-nvrtc-cu12-12.1.105 nvidia-cuda-runtime-
cu12-12.1.105 nvidia-cudnn-cu12-8.9.2.26 nvidia-cufft-cu12-11.0.2.54
nvidia-curand-cu12-10.3.2.106 nvidia-cusolver-cu12-11.4.5.107 nvidia-
cusparse-cu12-12.1.0.106 nvidia-nccl-cu12-2.19.3 nvidia-nvjitlink-
cu12-12.4.127 nvidia-nvtx-cu12-12.1.105

## Requirements
from __future__ import unicode_literals, print_function, division
from io import open
import unicodedata
import string
import re
```

```python
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")
device
```

```
device(type='cpu')
```

```python
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```python
%cd /content/drive/MyDrive/FIT5217/Assignment2/Cooking_Dataset
```

```
/content/drive/MyDrive/FIT5217/Assignment2/Cooking_Dataset
```

```python
SOS_token = 0
EOS_token = 1


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

    def addSentence(self, sentence):
        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
```

Implementation of Baseline 1

```python
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 = self.lstm(output, hidden)
        return output, hidden

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

class DecoderRNN(nn.Module):
    def __init__(self, hidden_size, output_size, dropout_p = 0.1):
        super(DecoderRNN, self).__init__()
        self.hidden_size = hidden_size
        self.dropout = nn.Dropout(dropout_p)
        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 = self.dropout(output)
        output = F.relu(output)
        output, hidden = self.lstm(output, hidden)
        output = self.softmax(self.out(output[0]))
        return output, hidden

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

def unicodeToAscii(s):
    return ''.join(
        c for c in unicodedata.normalize('NFD', s)
        if unicodedata.category(c) != 'Mn'
    )
def normalizeString(s):
    s = unicodeToAscii(s.lower().strip())
    s = re.sub(r"([;])", r",", s)
    s = re.sub(r"[\W]+", r" ", s)
    return s
```

```
def readLangs(ingredients, recipe):
    print("Reading lines...")

    ingredients = [str(s) for s in ingredients]
    recipe = [str(s) for s in recipe]
    ingredients = [normalizeString(str(s)) for s in ingredients]
    recipe = [normalizeString(str(s)) for s in recipe]

    pairs = [[ingredients[i],recipe[i]] for i in
range(len(ingredients))]

    input_lang = Lang('Ingredients')
    output_lang = Lang('Recipe')

    return input_lang, output_lang, pairs

MAX_LENGTH = 150


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


def filterPairs(pairs):
    return [pair for pair in pairs if filterPair(pair)]

import pandas as pd
train = pd.read_csv("train.csv")
test = pd.read_csv("test.csv")
dev = pd.read_csv("dev.csv")
def prepareData(lang1, lang2):
    input_lang, output_lang, pairs = readLangs(lang1, lang2)
    print("Read %s ingredients to recipe pairs" % len(pairs))
    pairs = filterPairs(pairs)
    print("Trimmed to %s sentence pairs" % len(pairs))
    print("Counting words...")
    for pair in pairs:
        input_lang.addSentence(pair[0])
        output_lang.addSentence(pair[1])
    print("Counted words:")
    print(input_lang.name, input_lang.n_words)
    print(output_lang.name, output_lang.n_words)
    return input_lang, output_lang, pairs


input_lang, output_lang, pairs = prepareData(train.Ingredients,
train.Recipe)
print(random.choice(pairs))
```

```
Reading lines...
Read 101340 ingredients to recipe pairs
Trimmed to 85857 sentence pairs
Counting words...
Counted words:
Ingredients 15154
Recipe 29059
['1 dried ancho chile 1 ounce 1 1 2 oz sun dried tomatoes 20 packed
without oil 2 c boiling water 1 tb olive oil 1 4 c minced red onion 1
garlic minced 3 c fresh corn kernels 2 c diced zucchini 1 4 c tequila
1 tb minced fresh tarragon 1 2 ts salt 1 pork tenderloin 1 pound 3 4
ts ground cumin 1 2 ts salt 1 2 ts pepper vegetable cooking spray 8 c
torn romaine lettuce', 'remove stem and seeds from chile combine chile
tomatoes and boiling wate r cover and let stand 20 minutes drain
reserving 1 4 cup liquid chop chile and tomatoes set aside heat oil in
a large nonstick skillet over medium heat add onion and garlic saute 1
minute add reserved 1 4 cup liquid corn zucchini and tequila cook 7
minutes or until vegetables are tender and liquid nearly evaporates
spoon into a bowl stir in chopped chile chopped tomato tarragon and 1
2 teaspoon salt let cool to room temperature trim fat from pork rub
pork with cumin 1 2 teaspoon salt and pepper pre pare grill place pork
on rack coated with cooking spray cover and cook 20 minutes or until
meat thermometer registers 160 degrees turning pork occasionally 4
servings 24g protein 65g carbohydrate 37mg cholesterol ']
```

```
input_test, output_test, pairs_test = prepareData(test.Ingredients,
test.Recipe)
input_dev, output_dev, pairs_dev = prepareData(dev.Ingredients,
dev.Recipe)
print(random.choice(pairs_test))
print(random.choice(pairs_dev))
```

```
Reading lines...
Read 778 ingredients to recipe pairs
Trimmed to 676 sentence pairs
Counting words...
Counted words:
Ingredients 1859
Recipe 3065
Reading lines...
Read 797 ingredients to recipe pairs
Trimmed to 682 sentence pairs
Counting words...
Counted words:
Ingredients 1851
Recipe 3231
['3 4 lb tuna steak 1 2 c lime juice 1 2 c 4 ounces coconut milk 2 tb
olive oil salt and pepper 1 c small diced mango 2 tb small diced red
pepper 2 tb chopped fresh cilantro 2 tb toasted coconut 2 tb minced
shallots cilantro sprigs for garnish', 'dice the tuna into small
```

```
pieces place in a glass bowl cover with the lime juice and coconut
milk cover and refrigerate for 4 hours pour off the excess liquid and
toss with 1 tablespoon of the olive oil and salt and pepper to taste
in another bowl combine the mango peppers cilantro shallots coconut
and the remaining olive oil and season combine remaining ingredients
for relish begin building your parfait place 1 tablespoon of the
relish in the bottom of each glass top with 2 tablespoon of the tuna
continue until the glasses are full ending with the relish on top
chill for 1 hour garnish with more cilantro sprigs ']
['1 md onion chopped 2 cl garlic minced 2 tb butter melted 1 lg can
whole tomatos 14 1 2oz 8 oz tomato paste 1 2 c celery chopped 1 3 c
vinegar 1 4 c green pepper chopped 2 fresh celery leaves chopped 1 bay
leaf 3 tb molasses 1 1 2 ts salt 2 ts dry mustard 2 ts tabasco sauce
to taste 1 2 ts clove ground 1 2 ts allspice ground 2 sl lemon',
'saute onion and garlic in butter in a saucepan until tender stir in
remaining ingredients bring to boil reduce heat and simmer uncovered
30 minutes stir occasionally discard bay leaf and lemon slices process
through a food processer if desired use sauce for basting and as a
side dish for dipping yield 3 cups ']

def indexesFromSentence(lang, sentence):
    return [lang.word2index[word] for word in sentence.split(' ')  if
word in lang.word2index]


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)

teacher_forcing_ratio = 1
MAX_LENGTH = 150

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

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))

def trainIters(encoder, decoder, n_iters, print_every=1000,
plot_every=100, learning_rate=0.01):
    start = time.time()
    plot_losses = []
    print_loss_total = 0  # Reset every print_every
    plot_loss_total = 0  # Reset every plot_every

    encoder_optimizer = optim.Adam(encoder.parameters())
    decoder_optimizer = optim.Adam(decoder.parameters())
    training_pairs = [tensorsFromPair(random.choice(pairs))
                      for i in range(n_iters)]
    losses_val = []
    criterion = nn.NLLLoss()

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

        loss = train(input_tensor, target_tensor, encoder,
                     decoder, encoder_optimizer, decoder_optimizer,
criterion)
        print_loss_total += loss
        plot_loss_total += loss

        if iter % print_every == 0:
            print_loss_avg = print_loss_total / print_every
            print_loss_total = 0
            loss_avg_val = evaluate_val(encoder, decoder, pairs_dev)
            losses_val.append(loss_avg_val)
            print('%s (%d %d%%) Train Loss: %.4f Val Loss: %.4f' %
(timeSince(start, iter / n_iters),
                                         iter, iter / n_iters * 100,
print_loss_avg, loss_avg_val))

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

```
            plot_loss_total = 0

    showPlot(plot_losses, losses_val)

import matplotlib.pyplot as plt
plt.switch_backend('agg')
import matplotlib.ticker as ticker
import numpy as np
%matplotlib inline

def showPlot(points_train, points_val):
    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.plot(points_train)
    plt.plot(points_val)

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)
            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 evaluate_val(encoder, decoder, pairs_dev, max_length=MAX_LENGTH):
    encoder.eval()
    decoder.eval()
    criterion = nn.NLLLoss()
    val_loss_total = 0
    with torch.no_grad():
      for pair in pairs_dev:
        input_tensor = tensorFromSentence(input_lang, pair[0])
        target_tensor = tensorFromSentence(output_lang, pair[1])
        input_length = input_tensor.size(0)
        target_length = target_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

        loss = 0

        for di in range(target_length):
            decoder_output, decoder_hidden = decoder(
                decoder_input, decoder_hidden)
            topv, topi = decoder_output.data.topk(1)
            decoder_input = topi.squeeze().detach()
            loss += criterion(decoder_output, target_tensor[di])
            if decoder_input.item() == EOS_token:
                break
        val_loss_total += loss.item()/target_length
      encoder.train()
      decoder.train()
      return val_loss_total/len(pairs_dev)

hidden_size = 256
encoder1 = EncoderRNN(input_lang.n_words, hidden_size).to(device)
decoder1 = DecoderRNN(hidden_size, output_lang.n_words).to(device)

trainIters(encoder1, decoder1, 10000, print_every=100)
```

```
torch.save(encoder1.state_dict(),"encoder_baseline1.pt")
torch.save(decoder1.state_dict(),"decoder_baseline1.pt")

hidden_size = 256
encoder1 = EncoderRNN(input_lang.n_words, hidden_size)
encoder1.load_state_dict(torch.load("encoder_baseline1.pt",device))
print(encoder1.eval())
decoder1 = DecoderRNN(hidden_size, output_lang.n_words)
decoder1.load_state_dict(torch.load("decoder_baseline1.pt",device))
print(decoder1.eval())

EncoderRNN(
  (embedding): Embedding(15154, 256)
  (lstm): LSTM(256, 256)
)
DecoderRNN(
  (dropout): Dropout(p=0.1, inplace=False)
  (embedding): Embedding(29059, 256)
  (lstm): LSTM(256, 256)
  (out): Linear(in_features=256, out_features=29059, bias=True)
  (softmax): LogSoftmax(dim=1)
)
```

Implementation of Baseline 2

```
MAX_LENGTH = 150
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.attn = nn.Linear(self.hidden_size * 2, self.max_length)
        self.attn_combine = nn.Linear(self.hidden_size * 2,
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].view(1,-1)), 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 = self.lstm(output, hidden)

        output = F.log_softmax(self.out(output[0]), dim=1)

        return output, hidden, attn_weights

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

teacher_forcing_ratio = 1


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 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]

def evaluate_val_attn(encoder, decoder, pairs_dev,
max_length=MAX_LENGTH):
    encoder.eval()
    decoder.eval()
    criterion = nn.NLLLoss()
    val_loss_total = 0
    with torch.no_grad():
      for pair in pairs_dev:
        input_tensor = tensorFromSentence(input_lang, pair[0])
        target_tensor = tensorFromSentence(output_lang, pair[1])
        input_length = input_tensor.size(0)
        target_length = target_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

        loss = 0
        decoder_attentions = torch.zeros(max_length, max_length)
        for di in range(target_length):
            decoder_output, decoder_hidden, decoder_attention =
decoder(
                decoder_input, decoder_hidden, encoder_outputs)
```

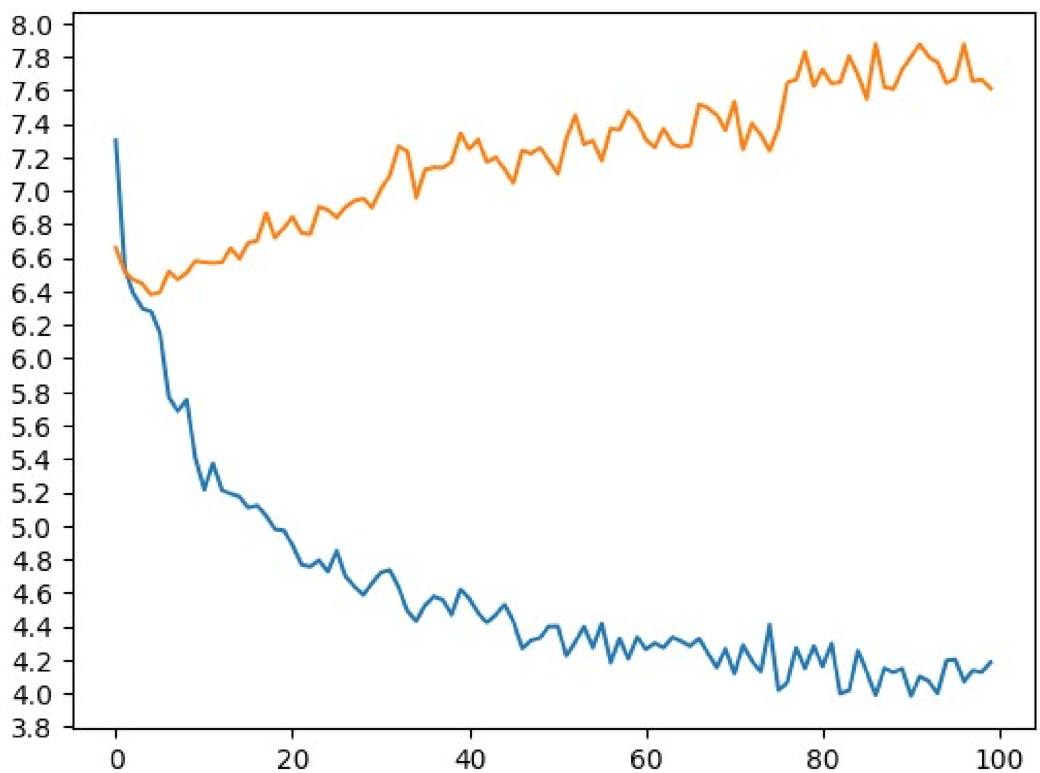
```
            decoder_attentions[di] = decoder_attention.data
            topv, topi = decoder_output.data.topk(1)
            loss += criterion(decoder_output, target_tensor[di])
            if decoder_input.item() == EOS_token:
                break
        val_loss_total += loss.item()/target_length
      encoder.train()
      decoder.train()
      return val_loss_total/len(pairs_dev)

def trainIters_attn(encoder, decoder, n_iters, print_every=1000,
plot_every=100, learning_rate=0.01):
    start = time.time()
    plot_losses = []
    print_loss_total = 0  # Reset every print_every
    plot_loss_total = 0  # Reset every plot_every

    encoder_optimizer = optim.Adam(encoder.parameters())
    decoder_optimizer = optim.Adam(decoder.parameters())
    training_pairs = [tensorsFromPair(random.choice(pairs))
                      for i in range(n_iters)]
    losses_val = []
    criterion = nn.NLLLoss()

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

        loss = train_attn(input_tensor, target_tensor, encoder,
                     decoder, encoder_optimizer, decoder_optimizer,
criterion)
        print_loss_total += loss
        plot_loss_total += loss

        if iter % print_every == 0:
            print_loss_avg = print_loss_total / print_every
            print_loss_total = 0
            loss_avg_val = evaluate_val_attn(encoder, decoder,
pairs_dev)
            losses_val.append(loss_avg_val)
            print('%s (%d %d%%) Train Loss: %.4f Val Loss: %.4f' %
(timeSince(start, iter / n_iters),
                                         iter, iter / n_iters * 100,
print_loss_avg, loss_avg_val))

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

```
    showPlot(plot_losses,losses_val)

hidden_size = 256
attn_encoder1 = EncoderRNN(input_lang.n_words, hidden_size).to(device)
attn_decoder1 = AttnDecoderRNN(hidden_size, output_lang.n_words,
dropout_p=0.1).to(device)

trainIters_attn(attn_encoder1, attn_decoder1, 10000, print_every=100)
```

```
1m 24s (- 139m 27s) (100 1%) Train Loss: 7.3001 Val Loss: 6.6600
2m 40s (- 131m 15s) (200 2%) Train Loss: 6.5311 Val Loss: 6.5148
3m 56s (- 127m 25s) (300 3%) Train Loss: 6.3845 Val Loss: 6.4688
5m 11s (- 124m 35s) (400 4%) Train Loss: 6.2978 Val Loss: 6.4439
6m 23s (- 121m 33s) (500 5%) Train Loss: 6.2787 Val Loss: 6.3791
7m 39s (- 119m 58s) (600 6%) Train Loss: 6.1472 Val Loss: 6.3959
8m 55s (- 118m 35s) (700 7%) Train Loss: 5.7664 Val Loss: 6.5187
10m 10s (- 116m 56s) (800 8%) Train Loss: 5.6841 Val Loss: 6.4691
11m 26s (- 115m 42s) (900 9%) Train Loss: 5.7517 Val Loss: 6.5107
12m 41s (- 114m 13s) (1000 10%) Train Loss: 5.4068 Val Loss: 6.5807
13m 56s (- 112m 48s) (1100 11%) Train Loss: 5.2168 Val Loss: 6.5732
15m 10s (- 111m 18s) (1200 12%) Train Loss: 5.3715 Val Loss: 6.5685
16m 25s (- 109m 57s) (1300 13%) Train Loss: 5.2129 Val Loss: 6.5731
17m 40s (- 108m 34s) (1400 14%) Train Loss: 5.1921 Val Loss: 6.6558
18m 53s (- 107m 0s) (1500 15%) Train Loss: 5.1732 Val Loss: 6.5939
20m 8s (- 105m 43s) (1600 16%) Train Loss: 5.1080 Val Loss: 6.6900
21m 23s (- 104m 28s) (1700 17%) Train Loss: 5.1217 Val Loss: 6.7019
22m 36s (- 102m 58s) (1800 18%) Train Loss: 5.0600 Val Loss: 6.8669
23m 51s (- 101m 41s) (1900 19%) Train Loss: 4.9796 Val Loss: 6.7192
25m 6s (- 100m 25s) (2000 20%) Train Loss: 4.9723 Val Loss: 6.7780
26m 22s (- 99m 12s) (2100 21%) Train Loss: 4.8865 Val Loss: 6.8426
27m 39s (- 98m 2s) (2200 22%) Train Loss: 4.7686 Val Loss: 6.7476
28m 56s (- 96m 53s) (2300 23%) Train Loss: 4.7570 Val Loss: 6.7406
30m 11s (- 95m 36s) (2400 24%) Train Loss: 4.7946 Val Loss: 6.9052
31m 26s (- 94m 18s) (2500 25%) Train Loss: 4.7279 Val Loss: 6.8854
32m 40s (- 93m 0s) (2600 26%) Train Loss: 4.8530 Val Loss: 6.8386
33m 53s (- 91m 38s) (2700 27%) Train Loss: 4.6966 Val Loss: 6.9033
35m 6s (- 90m 17s) (2800 28%) Train Loss: 4.6393 Val Loss: 6.9387
36m 21s (- 89m 1s) (2900 28%) Train Loss: 4.5873 Val Loss: 6.9528
37m 37s (- 87m 48s) (3000 30%) Train Loss: 4.6577 Val Loss: 6.8999
38m 52s (- 86m 31s) (3100 31%) Train Loss: 4.7223 Val Loss: 7.0114
40m 7s (- 85m 15s) (3200 32%) Train Loss: 4.7373 Val Loss: 7.0885
41m 22s (- 84m 0s) (3300 33%) Train Loss: 4.6323 Val Loss: 7.2668
42m 38s (- 82m 45s) (3400 34%) Train Loss: 4.4918 Val Loss: 7.2340
43m 51s (- 81m 27s) (3500 35%) Train Loss: 4.4309 Val Loss: 6.9561
45m 6s (- 80m 11s) (3600 36%) Train Loss: 4.5270 Val Loss: 7.1252
46m 20s (- 78m 54s) (3700 37%) Train Loss: 4.5786 Val Loss: 7.1387
47m 33s (- 77m 35s) (3800 38%) Train Loss: 4.5584 Val Loss: 7.1365
48m 48s (- 76m 19s) (3900 39%) Train Loss: 4.4690 Val Loss: 7.1726
50m 4s (- 75m 6s) (4000 40%) Train Loss: 4.6204 Val Loss: 7.3414
```

```
51m 20s (- 73m 52s) (4100 41%) Train Loss: 4.5673 Val Loss: 7.2485
52m 34s (- 72m 36s) (4200 42%) Train Loss: 4.4797 Val Loss: 7.3058
53m 49s (- 71m 20s) (4300 43%) Train Loss: 4.4235 Val Loss: 7.1719
55m 3s (- 70m 4s) (4400 44%) Train Loss: 4.4696 Val Loss: 7.1990
56m 16s (- 68m 46s) (4500 45%) Train Loss: 4.5295 Val Loss: 7.1281
57m 29s (- 67m 29s) (4600 46%) Train Loss: 4.4294 Val Loss: 7.0447
58m 43s (- 66m 13s) (4700 47%) Train Loss: 4.2698 Val Loss: 7.2396
59m 56s (- 64m 56s) (4800 48%) Train Loss: 4.3152 Val Loss: 7.2207
61m 10s (- 63m 40s) (4900 49%) Train Loss: 4.3316 Val Loss: 7.2560
62m 24s (- 62m 24s) (5000 50%) Train Loss: 4.3992 Val Loss: 7.1841
63m 39s (- 61m 9s) (5100 51%) Train Loss: 4.3999 Val Loss: 7.1010
64m 55s (- 59m 55s) (5200 52%) Train Loss: 4.2253 Val Loss: 7.3099
66m 10s (- 58m 40s) (5300 53%) Train Loss: 4.3083 Val Loss: 7.4516
67m 25s (- 57m 26s) (5400 54%) Train Loss: 4.3969 Val Loss: 7.2775
68m 43s (- 56m 13s) (5500 55%) Train Loss: 4.2720 Val Loss: 7.2992
70m 0s (- 55m 0s) (5600 56%) Train Loss: 4.4174 Val Loss: 7.1806
71m 16s (- 53m 46s) (5700 56%) Train Loss: 4.1823 Val Loss: 7.3711
72m 32s (- 52m 32s) (5800 57%) Train Loss: 4.3276 Val Loss: 7.3644
73m 49s (- 51m 17s) (5900 59%) Train Loss: 4.2059 Val Loss: 7.4735
75m 6s (- 50m 4s) (6000 60%) Train Loss: 4.3360 Val Loss: 7.4115
76m 25s (- 48m 51s) (6100 61%) Train Loss: 4.2644 Val Loss: 7.3062
77m 42s (- 47m 37s) (6200 62%) Train Loss: 4.2997 Val Loss: 7.2583
78m 58s (- 46m 23s) (6300 63%) Train Loss: 4.2733 Val Loss: 7.3703
80m 15s (- 45m 8s) (6400 64%) Train Loss: 4.3356 Val Loss: 7.2801
81m 31s (- 43m 54s) (6500 65%) Train Loss: 4.3103 Val Loss: 7.2614
82m 48s (- 42m 39s) (6600 66%) Train Loss: 4.2820 Val Loss: 7.2727
84m 4s (- 41m 24s) (6700 67%) Train Loss: 4.3264 Val Loss: 7.5141
85m 20s (- 40m 9s) (6800 68%) Train Loss: 4.2423 Val Loss: 7.4955
86m 36s (- 38m 54s) (6900 69%) Train Loss: 4.1556 Val Loss: 7.4532
87m 52s (- 37m 39s) (7000 70%) Train Loss: 4.2660 Val Loss: 7.3599
89m 7s (- 36m 24s) (7100 71%) Train Loss: 4.1188 Val Loss: 7.5300
90m 24s (- 35m 9s) (7200 72%) Train Loss: 4.2876 Val Loss: 7.2463
91m 41s (- 33m 54s) (7300 73%) Train Loss: 4.1927 Val Loss: 7.4017
92m 57s (- 32m 39s) (7400 74%) Train Loss: 4.1304 Val Loss: 7.3349
94m 14s (- 31m 24s) (7500 75%) Train Loss: 4.4113 Val Loss: 7.2372
95m 30s (- 30m 9s) (7600 76%) Train Loss: 4.0188 Val Loss: 7.3744
96m 45s (- 28m 54s) (7700 77%) Train Loss: 4.0664 Val Loss: 7.6450
98m 1s (- 27m 38s) (7800 78%) Train Loss: 4.2715 Val Loss: 7.6642
99m 18s (- 26m 23s) (7900 79%) Train Loss: 4.1493 Val Loss: 7.8281
100m 35s (- 25m 8s) (8000 80%) Train Loss: 4.2823 Val Loss: 7.6234
101m 51s (- 23m 53s) (8100 81%) Train Loss: 4.1613 Val Loss: 7.7219
103m 9s (- 22m 38s) (8200 82%) Train Loss: 4.2974 Val Loss: 7.6402
104m 26s (- 21m 23s) (8300 83%) Train Loss: 3.9985 Val Loss: 7.6474
105m 43s (- 20m 8s) (8400 84%) Train Loss: 4.0187 Val Loss: 7.8039
106m 59s (- 18m 52s) (8500 85%) Train Loss: 4.2543 Val Loss: 7.6917
108m 14s (- 17m 37s) (8600 86%) Train Loss: 4.1269 Val Loss: 7.5455
109m 31s (- 16m 21s) (8700 87%) Train Loss: 3.9904 Val Loss: 7.8759
110m 46s (- 15m 6s) (8800 88%) Train Loss: 4.1504 Val Loss: 7.6182
112m 1s (- 13m 50s) (8900 89%) Train Loss: 4.1253 Val Loss: 7.6070
```

```
113m 15s (- 12m 35s) (9000 90%) Train Loss: 4.1484 Val Loss: 7.7207
114m 31s (- 11m 19s) (9100 91%) Train Loss: 3.9849 Val Loss: 7.7992
115m 46s (- 10m 4s) (9200 92%) Train Loss: 4.1009 Val Loss: 7.8739
117m 2s (- 8m 48s) (9300 93%) Train Loss: 4.0733 Val Loss: 7.7989
118m 17s (- 7m 33s) (9400 94%) Train Loss: 4.0005 Val Loss: 7.7676
119m 33s (- 6m 17s) (9500 95%) Train Loss: 4.1955 Val Loss: 7.6409
120m 50s (- 5m 2s) (9600 96%) Train Loss: 4.2020 Val Loss: 7.6680
122m 6s (- 3m 46s) (9700 97%) Train Loss: 4.0699 Val Loss: 7.8762
123m 24s (- 2m 31s) (9800 98%) Train Loss: 4.1369 Val Loss: 7.6542
124m 39s (- 1m 15s) (9900 99%) Train Loss: 4.1264 Val Loss: 7.6645
125m 55s (- 0m 0s) (10000 100%) Train Loss: 4.1855 Val Loss: 7.6103

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



torch.save(attn_encoder1.state_dict(),"encoder_baseline2.pt")
torch.save(attn_decoder1.state_dict(),"decoder_baseline2.pt")

```
hidden_size = 256
attn_encoder1 = EncoderRNN(input_lang.n_words, hidden_size)
attn_encoder1.load_state_dict(torch.load("encoder_baseline2.pt",device
))
print(attn_encoder1.eval())
attn_decoder1 = AttnDecoderRNN(hidden_size, output_lang.n_words,
dropout_p=0.1)
attn_decoder1.load_state_dict(torch.load("decoder_baseline2.pt",device
```

```
))
print(attn_decoder1.eval())
```

```
EncoderRNN(
  (embedding): Embedding(15154, 256)
  (lstm): LSTM(256, 256)
)
AttnDecoderRNN(
  (embedding): Embedding(29059, 256)
  (attn): Linear(in_features=512, out_features=150, bias=True)
  (attn_combine): Linear(in_features=512, out_features=256, bias=True)
  (dropout): Dropout(p=0.1, inplace=False)
  (lstm): LSTM(256, 256)
  (out): Linear(in_features=256, out_features=29059, bias=True)
)
```

Implementation of Extension 1

```
import nltk
# Ensure NLTK data is downloaded
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
```

```
True
```

```
def unicodeToAscii(s):
    return ''.join(
        c for c in unicodedata.normalize('NFD', s)
        if unicodedata.category(c) != 'Mn'
    )
def normalizeStringIngredients(s):
    s = unicodeToAscii(s.lower().strip())
    s = re.sub(r'\b[a-zA-Z]{1,2}\b', " ", s)
    s = re.sub(r"[^a-zA-Z]+", r" ", s)
    s = re.sub(r"[\s]+", r" ", s)
    words = nltk.word_tokenize(s)
    # Tag the words with part-of-speech tags
    tagged_words = nltk.pos_tag(words)
    # Keep only words that are tagged as NN (noun, singular)
    nouns = [word for word, tag in tagged_words if tag == 'NN']
```

```
    nouns.sort()
    nouns = list(dict.fromkeys(nouns))
    # Join the nouns back into a string
    s = ' '.join(nouns)
    return s
def normalizeStringRecipe(s):
    s = unicodeToAscii(s.lower().strip())
    #s = re.sub(r"([;])", r".", s)
    s = re.sub(r"[^a-zA-Z0-9]+", r" ", s)
    #s = re.sub(r"[^a-zA-Z0-9\.,]+", r" ", s)
    s = re.sub(r"[\s]+", r" ", s)
    return s

def readLangs(ingredients, recipe):
    print("Reading lines...")

    ingredients = [str(s) for s in ingredients]
    recipe = [str(s) for s in recipe]
    ingredients = [normalizeStringIngredients(str(s)) for s in
ingredients]
    recipe = [normalizeStringRecipe(str(s)) for s in recipe]
    pairs = [[ingredients[i],recipe[i]] for i in
range(len(ingredients))]
    input_lang = Lang('Ingredients')
    output_lang = Lang('Recipe')

    return input_lang, output_lang, pairs

def indexesFromSentence(lang, sentence):
    return [lang.word2index[word] for word in sentence.split(' ')  if
word in lang.word2index]


def tensorFromSentence(lang, sentence, ingredients = False):
    indexes = indexesFromSentence(lang, sentence)
    if ingredients:
      indexes = list(set(indexes))
      indexes.sort()
    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], True)
    target_tensor = tensorFromSentence(output_lang, pair[1])
    return (input_tensor, target_tensor)

MAX_LENGTH = 150


def filterPair(p):
```

```
    return ((1 <= len(p[0].split(' ')) < MAX_LENGTH) and \
        (2 < len(p[1].split(' ')) < MAX_LENGTH) and \
        (len(p[1].split('.')) < 15))


def filterPairs(pairs):
    return [pair for pair in pairs if filterPair(pair)]
def removeDuplicates(pairs):
  pair_list = {pair[0]: pair for pair in pairs}
  return list(pair_list.values())

import pandas as pd
train = pd.read_csv("train.csv").drop_duplicates(subset='Ingredients',
keep="first")
test = pd.read_csv("test.csv")
dev = pd.read_csv("dev.csv").drop_duplicates(subset='Ingredients',
keep="first")
def prepareData(lang1, lang2):
    input_lang, output_lang, pairs = readLangs(lang1, lang2)
    print("Read %s ingredients to recipe pairs" % len(pairs))
    pairs = filterPairs(pairs)
    print("Trimmed to %s sentence pairs" % len(pairs))
    pairs = removeDuplicates(pairs)
    print("Trimmed further to %s sentence pairs" % len(pairs))
    print("Counting words...")
    for pair in pairs:
        input_lang.addSentence(pair[0])
        output_lang.addSentence(pair[1])
    print("Counted words:")
    print(input_lang.name, input_lang.n_words)
    print(output_lang.name, output_lang.n_words)
    return input_lang, output_lang, pairs


input_lang, output_lang, pairs = prepareData(train.Ingredients,
train.Recipe)
print(random.choice(pairs))
```

```
Reading lines...
Read 91263 ingredients to recipe pairs
Trimmed to 77458 sentence pairs
Trimmed further to 69515 sentence pairs
Counting words...
Counted words:
Ingredients 7987
Recipe 27697
['chicken ground mild pork pref shoulder', 'melt the pork fat in a
heavy skillet over medium high heat add the pork cubes a few at a time
stirring to brown evenly add the salt and garlic stirring well remove
from the heat and stir in the ground chile and oregano coating the
```

```
meat evenly with the spices rrb add a small amount of broth and stir
well return to the heat add a bit more broth and stir continue to add
broth a little at a time stirring until the chili is smooth then
reduce the heat and simmer uncovered for about 1 hour taste and adjust
seasonings adding the ground hot chile to taste at this point to add
remove the pot from the heat sprinkle the chile over the top and stir
well serve the chili with a bowl of freshly stewed pinto beans on the
side ']
```

```
input_dev, output_dev, pairs_dev = prepareData(dev.Ingredients,
dev.Recipe)
ingredients = [normalizeStringIngredients(str(s)) for s in
test.Ingredients]
print(random.choice(pairs_dev))
```

```
Reading lines...
Read 793 ingredients to recipe pairs
Trimmed to 679 sentence pairs
Counting words...
Counted words:
Ingredients 872
Recipe 3184
['anise chicken cinnamon dark oil rice sesame sherry soy star stick
sugar wine', 'bring them to a slow boil in a large kettle add meat or
poultry return to slow boil reduce heat to simmer and cook until done
drain meat and serve with some sauce lrb optionally thickened with
cornstarch water rrb on the side save and reuse leftover sauce which
gets richer with each use ']
```

```
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 = self.lstm(output, hidden)
        return output, hidden

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

MAX_LENGTH = 150
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.attn = nn.Linear(self.hidden_size * 2, self.max_length)
        self.attn_combine = nn.Linear(self.hidden_size * 2,
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].view(1,-1)), 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 = self.lstm(output, hidden)

        output = F.log_softmax(self.out(output[0]), dim=1)

        return output, hidden, attn_weights

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

teacher_forcing_ratio = 1


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 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]

def trainIters_attn(encoder, decoder, n_iters, print_every=1000,
plot_every=100, learning_rate=0.01):
    start = time.time()
    plot_losses = []
    print_loss_total = 0  # Reset every print_every
    plot_loss_total = 0  # Reset every plot_every

    encoder_optimizer = optim.Adam(encoder.parameters())
    decoder_optimizer = optim.Adam(decoder.parameters())
    training_pairs = [tensorsFromPair(random.choice(pairs))
                      for i in range(n_iters)]
    losses_val = []
    criterion = nn.NLLLoss()

    for iter in range(1, n_iters + 1):
```

```
        training_pair = training_pairs[iter - 1]
        input_tensor = training_pair[0]
        target_tensor = training_pair[1]

        loss = train_attn(input_tensor, target_tensor, encoder,
                     decoder, encoder_optimizer, decoder_optimizer,
criterion)
        print_loss_total += loss
        plot_loss_total += loss

        if iter % print_every == 0:
            print_loss_avg = print_loss_total / print_every
            print_loss_total = 0
            loss_avg_val = evaluate_val_attn(encoder, decoder,
pairs_dev)
            losses_val.append(loss_avg_val)
            print('%s (%d %d%%) Train Loss: %.4f Val Loss: %.4f' %
(timeSince(start, iter / n_iters),
                                         iter, iter / n_iters * 100,
print_loss_avg, loss_avg_val))

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

    showPlot(plot_losses,losses_val)

def evaluate_val_attn(encoder, decoder, pairs_dev,
max_length=MAX_LENGTH):
    encoder.eval()
    decoder.eval()
    criterion = nn.NLLLoss()
    val_loss_total = 0
    with torch.no_grad():
      for pair in pairs_dev:
        input_tensor = tensorFromSentence(input_lang, pair[0])
        target_tensor = tensorFromSentence(output_lang, pair[1])
        input_length = input_tensor.size(0)
        target_length = target_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

        loss = 0
        decoder_attentions = torch.zeros(max_length, max_length)
        for di in range(target_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)
            loss += criterion(decoder_output, target_tensor[di])
            if decoder_input.item() == EOS_token:
                break
        val_loss_total += loss.item()/target_length
      encoder.train()
      decoder.train()
      return val_loss_total/len(pairs_dev)

hidden_size = 256
attn_encoder1_1 = EncoderRNN(input_lang.n_words,
hidden_size).to(device)
attn_decoder1_1 = AttnDecoderRNN(hidden_size, output_lang.n_words,
dropout_p=0.1).to(device)

trainIters_attn(attn_encoder1_1, attn_decoder1_1, 10000,
print_every=100)

1m 6s (- 110m 4s) (100 1%) Train Loss: 7.2752 Val Loss: 6.5613
2m 12s (- 108m 19s) (200 2%) Train Loss: 6.2857 Val Loss: 6.5689
3m 19s (- 107m 20s) (300 3%) Train Loss: 6.0586 Val Loss: 6.6044
4m 22s (- 105m 11s) (400 4%) Train Loss: 5.8290 Val Loss: 6.7002
5m 28s (- 103m 54s) (500 5%) Train Loss: 5.7340 Val Loss: 6.7462
6m 33s (- 102m 51s) (600 6%) Train Loss: 5.3995 Val Loss: 6.8530
7m 39s (- 101m 39s) (700 7%) Train Loss: 5.3304 Val Loss: 6.9600
8m 44s (- 100m 28s) (800 8%) Train Loss: 5.3046 Val Loss: 7.0560
9m 49s (- 99m 19s) (900 9%) Train Loss: 5.1166 Val Loss: 7.0687
10m 54s (- 98m 6s) (1000 10%) Train Loss: 5.1842 Val Loss: 7.1094
11m 58s (- 96m 49s) (1100 11%) Train Loss: 4.9143 Val Loss: 7.1690
13m 2s (- 95m 40s) (1200 12%) Train Loss: 5.2300 Val Loss: 7.2124
14m 7s (- 94m 35s) (1300 13%) Train Loss: 4.9153 Val Loss: 7.2577
15m 15s (- 93m 40s) (1400 14%) Train Loss: 5.0950 Val Loss: 7.2263
16m 19s (- 92m 29s) (1500 15%) Train Loss: 4.7515 Val Loss: 7.3689
17m 25s (- 91m 27s) (1600 16%) Train Loss: 4.8846 Val Loss: 7.3716
18m 30s (- 90m 21s) (1700 17%) Train Loss: 4.7661 Val Loss: 7.4245
19m 34s (- 89m 9s) (1800 18%) Train Loss: 4.8931 Val Loss: 7.2050
20m 39s (- 88m 2s) (1900 19%) Train Loss: 4.6627 Val Loss: 7.3733
21m 44s (- 86m 59s) (2000 20%) Train Loss: 4.7310 Val Loss: 7.2958
22m 50s (- 85m 54s) (2100 21%) Train Loss: 4.6222 Val Loss: 7.2839
23m 55s (- 84m 48s) (2200 22%) Train Loss: 4.6167 Val Loss: 7.2494
```

```
24m 59s (- 83m 38s) (2300 23%) Train Loss: 4.6031 Val Loss: 7.5849
26m 4s (- 82m 35s) (2400 24%) Train Loss: 4.6130 Val Loss: 7.4675
27m 9s (- 81m 27s) (2500 25%) Train Loss: 4.6946 Val Loss: 7.4997
28m 15s (- 80m 24s) (2600 26%) Train Loss: 4.5815 Val Loss: 7.4985
29m 21s (- 79m 23s) (2700 27%) Train Loss: 4.6604 Val Loss: 7.7402
30m 28s (- 78m 21s) (2800 28%) Train Loss: 4.5513 Val Loss: 7.4112
31m 33s (- 77m 16s) (2900 28%) Train Loss: 4.4796 Val Loss: 7.5508
32m 40s (- 76m 13s) (3000 30%) Train Loss: 4.5954 Val Loss: 7.4798
33m 46s (- 75m 10s) (3100 31%) Train Loss: 4.2968 Val Loss: 7.5623
34m 52s (- 74m 7s) (3200 32%) Train Loss: 4.5817 Val Loss: 7.3482
35m 58s (- 73m 3s) (3300 33%) Train Loss: 4.3615 Val Loss: 7.4771
37m 3s (- 71m 56s) (3400 34%) Train Loss: 4.3465 Val Loss: 7.5116
38m 9s (- 70m 51s) (3500 35%) Train Loss: 4.4358 Val Loss: 7.4029
39m 13s (- 69m 44s) (3600 36%) Train Loss: 4.2303 Val Loss: 7.6483
40m 18s (- 68m 38s) (3700 37%) Train Loss: 4.3498 Val Loss: 7.7001
41m 23s (- 67m 32s) (3800 38%) Train Loss: 4.4259 Val Loss: 7.6612
42m 27s (- 66m 25s) (3900 39%) Train Loss: 4.3822 Val Loss: 7.6594
43m 32s (- 65m 19s) (4000 40%) Train Loss: 4.3516 Val Loss: 7.5569
44m 37s (- 64m 13s) (4100 41%) Train Loss: 4.2899 Val Loss: 7.7883
45m 43s (- 63m 8s) (4200 42%) Train Loss: 4.2288 Val Loss: 7.6963
46m 49s (- 62m 4s) (4300 43%) Train Loss: 4.2369 Val Loss: 7.8533
47m 53s (- 60m 56s) (4400 44%) Train Loss: 4.5026 Val Loss: 7.5654
48m 59s (- 59m 52s) (4500 45%) Train Loss: 4.3899 Val Loss: 7.4847
50m 3s (- 58m 45s) (4600 46%) Train Loss: 4.0396 Val Loss: 7.6060
51m 8s (- 57m 40s) (4700 47%) Train Loss: 4.0063 Val Loss: 7.9589
52m 11s (- 56m 32s) (4800 48%) Train Loss: 4.2870 Val Loss: 7.8550
53m 16s (- 55m 26s) (4900 49%) Train Loss: 4.2296 Val Loss: 7.6303
54m 22s (- 54m 22s) (5000 50%) Train Loss: 4.2629 Val Loss: 7.6901
55m 26s (- 53m 15s) (5100 51%) Train Loss: 4.1014 Val Loss: 7.7200
56m 31s (- 52m 10s) (5200 52%) Train Loss: 4.1138 Val Loss: 7.8443
57m 35s (- 51m 4s) (5300 53%) Train Loss: 4.1190 Val Loss: 7.7484
58m 40s (- 49m 58s) (5400 54%) Train Loss: 4.3133 Val Loss: 7.8088
59m 47s (- 48m 54s) (5500 55%) Train Loss: 4.2597 Val Loss: 7.6894
60m 51s (- 47m 49s) (5600 56%) Train Loss: 4.2703 Val Loss: 7.8660
61m 56s (- 46m 43s) (5700 56%) Train Loss: 4.2677 Val Loss: 7.7339
63m 1s (- 45m 38s) (5800 57%) Train Loss: 4.2790 Val Loss: 7.7219
64m 8s (- 44m 34s) (5900 59%) Train Loss: 4.0966 Val Loss: 8.0050
65m 12s (- 43m 28s) (6000 60%) Train Loss: 4.2130 Val Loss: 7.8432
66m 16s (- 42m 22s) (6100 61%) Train Loss: 4.1030 Val Loss: 7.8609
67m 21s (- 41m 16s) (6200 62%) Train Loss: 4.0942 Val Loss: 7.7924
68m 25s (- 40m 11s) (6300 63%) Train Loss: 4.1589 Val Loss: 7.8261
69m 30s (- 39m 5s) (6400 64%) Train Loss: 4.1890 Val Loss: 8.0109
70m 34s (- 38m 0s) (6500 65%) Train Loss: 4.3172 Val Loss: 7.7508
71m 39s (- 36m 54s) (6600 66%) Train Loss: 4.0506 Val Loss: 8.0995
72m 44s (- 35m 49s) (6700 67%) Train Loss: 3.9415 Val Loss: 8.0392
73m 49s (- 34m 44s) (6800 68%) Train Loss: 4.1834 Val Loss: 8.1319
74m 55s (- 33m 39s) (6900 69%) Train Loss: 4.1690 Val Loss: 8.1836
76m 1s (- 32m 34s) (7000 70%) Train Loss: 4.1460 Val Loss: 8.2588
77m 5s (- 31m 29s) (7100 71%) Train Loss: 4.0691 Val Loss: 7.9272
```

```
78m 10s (- 30m 23s) (7200 72%) Train Loss: 4.1099 Val Loss: 8.0819
79m 14s (- 29m 18s) (7300 73%) Train Loss: 4.0808 Val Loss: 8.0245
80m 19s (- 28m 13s) (7400 74%) Train Loss: 4.0050 Val Loss: 8.1040
```

```
torch.save(attn_encoder1_1.state_dict(),"encoder_extended1.pt")
torch.save(attn_decoder1_1.state_dict(),"decoder_extended1.pt")
```

```
hidden_size = 256
attn_encoder1_1 = EncoderRNN(input_lang.n_words, hidden_size)
attn_encoder1_1.load_state_dict(torch.load("encoder_extended1.pt",devi
ce))
print(attn_encoder1_1.eval())
attn_decoder1_1 = AttnDecoderRNN(hidden_size, output_lang.n_words,
dropout_p=0.1)
attn_decoder1_1.load_state_dict(torch.load("decoder_extended1.pt",devi
ce))
print(attn_decoder1_1.eval())
```

```
predictions_0 = [' '.join(evaluate(encoder1,decoder1,s,MAX_LENGTH))
for s in ingredients]
predictions_00 = ['
'.join(evaluate_attn(attn_encoder1,attn_decoder1,s,MAX_LENGTH)[0]) for
s in ingredients]
predictions_1 = ['
'.join(evaluate_attn(attn_encoder1_1,attn_decoder1_1,s,MAX_LENGTH)[0])
for s in ingredients]
```

Implementation of Extension 2

```
import gensim.downloader as api
from sklearn.metrics.pairwise import cosine_similarity

# Load pre-trained Word2Vec model
model = api.load("word2vec-google-news-300")
```

```
[==================================================] 100.0%
1662.8/1662.8MB downloaded
```

```
# Helper function to get pre-trained word embeddings
def get_pretrained_embedding(word, model, embedding_dim=300):
    if word in model.key_to_index:
        return torch.tensor(model[word], dtype=torch.float).to(device)
    else:
        return torch.zeros(embedding_dim,
dtype=torch.float).to(device)

class EncoderRNN(nn.Module):
    def __init__(self, input_size, hidden_size, pre_trained_model,
embedding_dim=300):
```

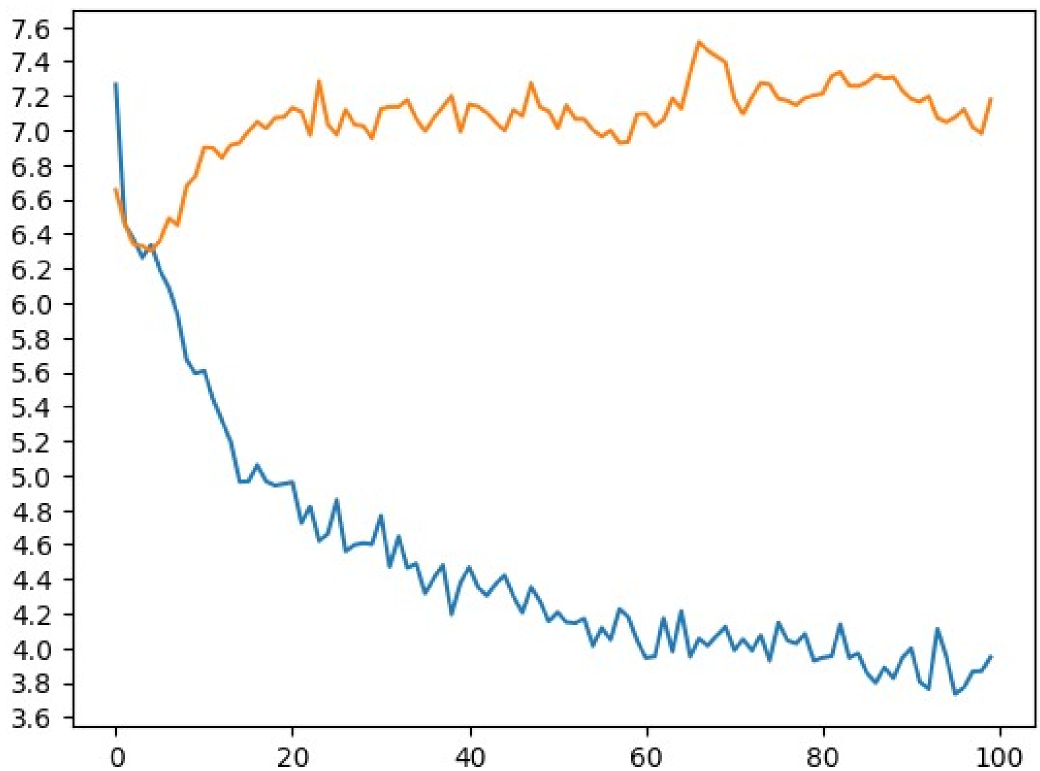
```
        super(EncoderRNN, self).__init__()
        self.hidden_size = hidden_size
        self.embedding_dim = embedding_dim

        self.embedding = nn.Embedding(input_size, hidden_size)
        # Initialize embedding weights with pre-trained embeddings

self.embedding.weight.data.copy_(self.load_pretrained_embeddings(pre_t
rained_model, input_size, hidden_size))
        self.lstm = nn.LSTM(hidden_size, hidden_size)

    def load_pretrained_embeddings(self, pre_trained_model,
input_size, hidden_size):
        embeddings = torch.zeros(input_size, hidden_size)
        for idx in range(input_size):
            word = input_lang.index2word[idx]  # idx2word should map
index to word
            embeddings[idx] = get_pretrained_embedding(word,
pre_trained_model, hidden_size)
        return embeddings

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

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

MAX_LENGTH = 150

class AttnDecoderRNN(nn.Module):
    def __init__(self, hidden_size, output_size, pre_trained_model,
dropout_p=0.1, max_length=MAX_LENGTH, embedding_dim=300):
        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_dim = embedding_dim

        self.embedding = nn.Embedding(self.output_size,
self.hidden_size)
        # Initialize embedding weights with pre-trained embeddings

self.embedding.weight.data.copy_(self.load_pretrained_embeddings(pre_t
rained_model, output_size, hidden_size))
        self.attn = nn.Linear(self.hidden_size * 2, self.max_length)
```

```
        self.attn_combine = nn.Linear(self.hidden_size * 2,
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 load_pretrained_embeddings(self, pre_trained_model,
output_size, hidden_size):
        embeddings = torch.zeros(output_size, hidden_size)
        for idx in range(output_size):
            word = output_lang.index2word[idx]
            embeddings[idx] = get_pretrained_embedding(word,
pre_trained_model, hidden_size)
        return embeddings

    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].view(1,-1)), 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 = self.lstm(output, hidden)

        output = F.log_softmax(self.out(output[0]), dim=1)

        return output, hidden, attn_weights

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

hidden_size = 300
attn_encoder2 = EncoderRNN(input_lang.n_words,
hidden_size,model).to(device)
attn_decoder2 = AttnDecoderRNN(hidden_size, output_lang.n_words,
model, dropout_p=0.1).to(device)

trainIters_attn(attn_encoder2, attn_decoder2, 10000, print_every=100)
```

```
1m 12s (- 120m 19s) (100 1%) Train Loss: 7.2649 Val Loss: 6.6552
2m 20s (- 115m 6s) (200 2%) Train Loss: 6.4589 Val Loss: 6.4540
3m 34s (- 115m 33s) (300 3%) Train Loss: 6.3694 Val Loss: 6.3400
4m 44s (- 113m 39s) (400 4%) Train Loss: 6.2623 Val Loss: 6.3296
5m 53s (- 111m 53s) (500 5%) Train Loss: 6.3363 Val Loss: 6.3020
7m 1s (- 110m 7s) (600 6%) Train Loss: 6.1884 Val Loss: 6.3629
```

```
8m 9s (- 108m 25s) (700 7%) Train Loss: 6.0874 Val Loss: 6.4898
9m 17s (- 106m 53s) (800 8%) Train Loss: 5.9268 Val Loss: 6.4509
10m 25s (- 105m 24s) (900 9%) Train Loss: 5.6686 Val Loss: 6.6783
11m 35s (- 104m 18s) (1000 10%) Train Loss: 5.5926 Val Loss: 6.7331
12m 45s (- 103m 13s) (1100 11%) Train Loss: 5.6079 Val Loss: 6.9026
13m 54s (- 102m 3s) (1200 12%) Train Loss: 5.4426 Val Loss: 6.8979
15m 3s (- 100m 47s) (1300 13%) Train Loss: 5.3186 Val Loss: 6.8408
16m 12s (- 99m 36s) (1400 14%) Train Loss: 5.1965 Val Loss: 6.9149
17m 24s (- 98m 40s) (1500 15%) Train Loss: 4.9635 Val Loss: 6.9287
18m 36s (- 97m 43s) (1600 16%) Train Loss: 4.9660 Val Loss: 6.9933
19m 46s (- 96m 35s) (1700 17%) Train Loss: 5.0616 Val Loss: 7.0500
20m 56s (- 95m 23s) (1800 18%) Train Loss: 4.9661 Val Loss: 7.0113
22m 6s (- 94m 15s) (1900 19%) Train Loss: 4.9425 Val Loss: 7.0712
23m 16s (- 93m 7s) (2000 20%) Train Loss: 4.9518 Val Loss: 7.0768
24m 27s (- 92m 2s) (2100 21%) Train Loss: 4.9614 Val Loss: 7.1313
25m 37s (- 90m 52s) (2200 22%) Train Loss: 4.7241 Val Loss: 7.1080
26m 48s (- 89m 45s) (2300 23%) Train Loss: 4.8203 Val Loss: 6.9721
28m 1s (- 88m 44s) (2400 24%) Train Loss: 4.6192 Val Loss: 7.2872
29m 10s (- 87m 32s) (2500 25%) Train Loss: 4.6636 Val Loss: 7.0343
30m 18s (- 86m 15s) (2600 26%) Train Loss: 4.8609 Val Loss: 6.9731
31m 25s (- 84m 59s) (2700 27%) Train Loss: 4.5622 Val Loss: 7.1195
32m 33s (- 83m 43s) (2800 28%) Train Loss: 4.5974 Val Loss: 7.0339
33m 43s (- 82m 33s) (2900 28%) Train Loss: 4.6077 Val Loss: 7.0255
34m 52s (- 81m 21s) (3000 30%) Train Loss: 4.6018 Val Loss: 6.9542
36m 2s (- 80m 13s) (3100 31%) Train Loss: 4.7676 Val Loss: 7.1247
37m 10s (- 79m 0s) (3200 32%) Train Loss: 4.4716 Val Loss: 7.1370
38m 19s (- 77m 48s) (3300 33%) Train Loss: 4.6483 Val Loss: 7.1366
39m 27s (- 76m 34s) (3400 34%) Train Loss: 4.4653 Val Loss: 7.1754
40m 36s (- 75m 25s) (3500 35%) Train Loss: 4.4901 Val Loss: 7.0671
41m 46s (- 74m 15s) (3600 36%) Train Loss: 4.3166 Val Loss: 6.9952
42m 54s (- 73m 2s) (3700 37%) Train Loss: 4.4051 Val Loss: 7.0713
44m 4s (- 71m 54s) (3800 38%) Train Loss: 4.4808 Val Loss: 7.1359
45m 11s (- 70m 41s) (3900 39%) Train Loss: 4.1948 Val Loss: 7.2012
46m 20s (- 69m 31s) (4000 40%) Train Loss: 4.3792 Val Loss: 6.9911
47m 29s (- 68m 20s) (4100 41%) Train Loss: 4.4689 Val Loss: 7.1514
48m 39s (- 67m 11s) (4200 42%) Train Loss: 4.3510 Val Loss: 7.1378
49m 48s (- 66m 1s) (4300 43%) Train Loss: 4.3036 Val Loss: 7.1020
50m 57s (- 64m 51s) (4400 44%) Train Loss: 4.3735 Val Loss: 7.0450
52m 5s (- 63m 39s) (4500 45%) Train Loss: 4.4202 Val Loss: 6.9986
53m 13s (- 62m 28s) (4600 46%) Train Loss: 4.3004 Val Loss: 7.1205
54m 20s (- 61m 16s) (4700 47%) Train Loss: 4.2059 Val Loss: 7.0825
55m 28s (- 60m 5s) (4800 48%) Train Loss: 4.3541 Val Loss: 7.2746
56m 40s (- 58m 59s) (4900 49%) Train Loss: 4.2721 Val Loss: 7.1347
57m 50s (- 57m 50s) (5000 50%) Train Loss: 4.1548 Val Loss: 7.1121
58m 59s (- 56m 41s) (5100 51%) Train Loss: 4.2077 Val Loss: 7.0126
60m 10s (- 55m 32s) (5200 52%) Train Loss: 4.1500 Val Loss: 7.1454
61m 20s (- 54m 23s) (5300 53%) Train Loss: 4.1441 Val Loss: 7.0653
62m 27s (- 53m 12s) (5400 54%) Train Loss: 4.1715 Val Loss: 7.0648
63m 36s (- 52m 2s) (5500 55%) Train Loss: 4.0115 Val Loss: 7.0023
```

```
64m 45s (- 50m 53s) (5600 56%) Train Loss: 4.1171 Val Loss: 6.9642
65m 53s (- 49m 42s) (5700 56%) Train Loss: 4.0465 Val Loss: 6.9998
67m 3s (- 48m 33s) (5800 57%) Train Loss: 4.2254 Val Loss: 6.9282
68m 13s (- 47m 24s) (5900 59%) Train Loss: 4.1810 Val Loss: 6.9341
69m 25s (- 46m 17s) (6000 60%) Train Loss: 4.0430 Val Loss: 7.0940
70m 34s (- 45m 7s) (6100 61%) Train Loss: 3.9428 Val Loss: 7.0969
71m 45s (- 43m 58s) (6200 62%) Train Loss: 3.9513 Val Loss: 7.0246
72m 55s (- 42m 49s) (6300 63%) Train Loss: 4.1738 Val Loss: 7.0618
74m 5s (- 41m 40s) (6400 64%) Train Loss: 3.9805 Val Loss: 7.1853
75m 14s (- 40m 31s) (6500 65%) Train Loss: 4.2144 Val Loss: 7.1262
76m 25s (- 39m 22s) (6600 66%) Train Loss: 3.9482 Val Loss: 7.3333
77m 35s (- 38m 12s) (6700 67%) Train Loss: 4.0536 Val Loss: 7.5125
78m 43s (- 37m 2s) (6800 68%) Train Loss: 4.0132 Val Loss: 7.4624
79m 53s (- 35m 53s) (6900 69%) Train Loss: 4.0717 Val Loss: 7.4283
81m 4s (- 34m 44s) (7000 70%) Train Loss: 4.1239 Val Loss: 7.3952
82m 12s (- 33m 34s) (7100 71%) Train Loss: 3.9890 Val Loss: 7.1813
83m 21s (- 32m 25s) (7200 72%) Train Loss: 4.0499 Val Loss: 7.0954
84m 29s (- 31m 14s) (7300 73%) Train Loss: 3.9863 Val Loss: 7.1934
85m 37s (- 30m 4s) (7400 74%) Train Loss: 4.0740 Val Loss: 7.2741
86m 45s (- 28m 55s) (7500 75%) Train Loss: 3.9254 Val Loss: 7.2658
87m 54s (- 27m 45s) (7600 76%) Train Loss: 4.1474 Val Loss: 7.1830
89m 2s (- 26m 35s) (7700 77%) Train Loss: 4.0441 Val Loss: 7.1708
90m 11s (- 25m 26s) (7800 78%) Train Loss: 4.0267 Val Loss: 7.1471
91m 18s (- 24m 16s) (7900 79%) Train Loss: 4.0830 Val Loss: 7.1868
92m 26s (- 23m 6s) (8000 80%) Train Loss: 3.9267 Val Loss: 7.2027
93m 34s (- 21m 57s) (8100 81%) Train Loss: 3.9436 Val Loss: 7.2130
94m 44s (- 20m 47s) (8200 82%) Train Loss: 3.9529 Val Loss: 7.3151
95m 51s (- 19m 37s) (8300 83%) Train Loss: 4.1375 Val Loss: 7.3371
97m 0s (- 18m 28s) (8400 84%) Train Loss: 3.9424 Val Loss: 7.2581
98m 9s (- 17m 19s) (8500 85%) Train Loss: 3.9703 Val Loss: 7.2559
99m 19s (- 16m 10s) (8600 86%) Train Loss: 3.8558 Val Loss: 7.2768
100m 28s (- 15m 0s) (8700 87%) Train Loss: 3.7997 Val Loss: 7.3216
101m 38s (- 13m 51s) (8800 88%) Train Loss: 3.8882 Val Loss: 7.3024
102m 46s (- 12m 42s) (8900 89%) Train Loss: 3.8258 Val Loss: 7.3091
103m 53s (- 11m 32s) (9000 90%) Train Loss: 3.9407 Val Loss: 7.2331
105m 2s (- 10m 23s) (9100 91%) Train Loss: 3.9997 Val Loss: 7.1816
106m 12s (- 9m 14s) (9200 92%) Train Loss: 3.8061 Val Loss: 7.1644
107m 20s (- 8m 4s) (9300 93%) Train Loss: 3.7623 Val Loss: 7.1985
108m 28s (- 6m 55s) (9400 94%) Train Loss: 4.1109 Val Loss: 7.0713
109m 39s (- 5m 46s) (9500 95%) Train Loss: 3.9492 Val Loss: 7.0479
110m 49s (- 4m 37s) (9600 96%) Train Loss: 3.7342 Val Loss: 7.0750
111m 56s (- 3m 27s) (9700 97%) Train Loss: 3.7718 Val Loss: 7.1238
113m 4s (- 2m 18s) (9800 98%) Train Loss: 3.8642 Val Loss: 7.0179
114m 13s (- 1m 9s) (9900 99%) Train Loss: 3.8663 Val Loss: 6.9812
115m 20s (- 0m 0s) (10000 100%) Train Loss: 3.9474 Val Loss: 7.1773

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




```
torch.save(attn_encoder2.state_dict(),"encoder_extended2.pt")
torch.save(attn_decoder2.state_dict(),"decoder_extended2.pt")

hidden_size = 300
attn_encoder2 = EncoderRNN(input_lang.n_words, hidden_size, model)
attn_encoder2.load_state_dict(torch.load("encoder_extended2.pt",device
))
print(attn_encoder2.eval())
attn_decoder2 = AttnDecoderRNN(hidden_size, output_lang.n_words,
model, dropout_p=0.1)
attn_decoder2.load_state_dict(torch.load("decoder_extended2.pt",device
))
print(attn_decoder2.eval())

predictions_2 = ['
'.join(evaluate_attn(attn_encoder2,attn_decoder2,s,MAX_LENGTH)[0]) for
s in ingredients]

def remove_last_word(string):
    words = string.split()
    return ' '.join(words[:-1])

# Apply the function to each string in the list to remove the <EOS>
taag
predictions_0 = [remove_last_word(s) for s in predictions_0]
predictions_00 = [remove_last_word(s) for s in predictions_00]
```

```
predictions_1 = [remove_last_word(s) for s in predictions_01]
predictions_2 = [remove_last_word(s) for s in predictions_2]
```

Evaluations, metrics and generating the final csv file

```
from nltk.translate.bleu_score import sentence_bleu, corpus_bleu
from nltk.translate import meteor_score
gold = "combine <sugar> and <water> in medium saucepan . Heat ,
stirring , until <sugar> dissolves , then boil 5 minutes . cool .
force <strawberries> through food mill or blend in blender or food
processor . strain to remove seeds , if desired . blend the puree and
<lemon juice> and <orange juice> into syrup . pour into freezer trays
and freeze . remove from freezer 20 minutes before serving . turn into
bowl and stir until smooth .".split()
sample = "Combine <sugar> and <water> in a medium saucepan . Heat,
stirring, until <sugar> dissolves . Bring to a boil and let simmer for
5 minutes . Remove from heat and allow to cool . In a blender or food
processor , combine <strawberries> and <cantaloupe> . Blend until
smooth . Strain the mixture to remove any seeds and fibers, if
desired. Stir the puree into the cooled syrup along with the <lemon
juice> and <orange juice> . Pour the mixture into a large bowl and
gently fold in the <vanilla ice cream> until well mixed . Freeze in a
container for at least 4 hours . Before serving , let it sit at room
temperature for 20 minutes to soften . Stir well to achieve a smooth
consistency and serve chilled .".split()
```

```
sentence_bleu([gold], sample)
```

```
0.11770400167201682
```

```
nltk.download('wordnet')
meteor_score.meteor_score([gold], sample)
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
```

```
0.5736654804270463
```

```
sum([sentence_bleu([test.Recipe[i].split()], predictions_2[i].split())
for i in range(len(test.Recipe))])/len(test.Recipe)
```

```
sum([sentence_bleu([test.Recipe[i].split()], predictions_1[i].split())
for i in range(len(test.Recipe))])/len(test.Recipe)
```

```
sum([meteor_score.meteor_score([test.Recipe[i].split()],
predictions_1[i].split()) for i in
range(len(test.Recipe))])/len(test.Recipe)
```

```
sum([meteor_score.meteor_score([test.Recipe[i].split()],
predictions_2[i].split()) for i in
range(len(test.Recipe))])/len(test.Recipe)

def extra(truth,pred):
  return sum([1 for w in pred if w not in truth])

sum([extra(test.Recipe[i].split(), predictions_1[i].split()) for i in
range(len(test.Recipe))])/len(test.Recipe)

sum([extra(test.Recipe[i].split(), predictions_2[i].split()) for i in
range(len(test.Recipe))])/len(test.Recipe)

def given(truth,pred):
  return sum([1 for w in truth if w in pred])/len(truth)
sum([given(test.Recipe[i].split(), predictions_1[i].split()) for i in
range(len(test.Recipe))])/len(test.Recipe)

sum([given(test.Recipe[i].split(), predictions_2[i].split()) for i in
range(len(test.Recipe))])/len(test.Recipe)

extra(gold,sample)

60

given(gold,sample)

0.7721518987341772

sum([given(test.Recipe[i].split(), predictions_0[i].split()) for i in
range(len(test.Recipe))])/len(test.Recipe)

0.185015224178579

sum([given(test.Recipe[i].split(), predictions_00[i].split()) for i in
range(len(test.Recipe))])/len(test.Recipe)

0.15759055848775372

sum([extra(test.Recipe[i].split(), predictions_0[i].split()) for i in
range(len(test.Recipe))])/len(test.Recipe)

19.533419023136247

sum([extra(test.Recipe[i].split(), predictions_00[i].split()) for i in
range(len(test.Recipe))])/len(test.Recipe)

12.222365038560412

sum([sentence_bleu([test.Recipe[i].split()], predictions_0[i].split())
for i in range(len(test.Recipe))])/len(test.Recipe)
```

```
/usr/local/lib/python3.10/dist-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)
/usr/local/lib/python3.10/dist-packages/nltk/translate/bleu_score.py:5
52: 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)
/usr/local/lib/python3.10/dist-packages/nltk/translate/bleu_score.py:5
52: 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)

0.0032338780771845816
```

```
sum([sentence_bleu([test.Recipe[i].split()],
predictions_00[i].split()) for i in
range(len(test.Recipe))])/len(test.Recipe)
```

```
0.0006401527443015746
```

```
sum([meteor_score.meteor_score([test.Recipe[i].split()],
predictions_0[i].split()) for i in
range(len(test.Recipe))])/len(test.Recipe)
```

```
0.082428071303489
```

```
sum([meteor_score.meteor_score([test.Recipe[i].split()],
predictions_00[i].split()) for i in
range(len(test.Recipe))])/len(test.Recipe)
```

```
0.0575463662417303
```

```
print(min(len(pair[0].split(' ')) for pair in pairs))
```

```
1
```

```
print(min(len(pair[1].split(' ')) for pair in pairs))
```

```
3
```

```
generated_33197970 = test.iloc[:, :-1]
generated_33197970['Generated Recipe - Baseline 1'] = predictions_0
```

```python
generated_33197970['Generated Recipe - Baseline 2'] = predictions_00
generated_33197970['Generated Recipe - Extended 1'] = predictions_1
generated_33197970['Generated Recipe - Extended 2'] = predictions_2

generated_33197970.to_csv('generated_33197970.csv', index=False)
```