## Colab FAQ

For some basic overview and features offered in Colab notebooks, check out: Overview of Colaboratory Features

You need to use the colab GPU for this assignment by selecting:

> **Runtime → Change runtime type → Hardware Accelerator: GPU**

## Setup PyTorch

All files will be stored at /content/csc421/a3/ folder

```
######################################################################
# Setup python environment and change the current working directory
######################################################################
!pip install Pillow
%mkdir -p ./content/csc421/a3/
%cd ./content/csc421/a3
```

```
Requirement already satisfied: Pillow in
/usr/local/lib/python3.7/dist-packages (7.1.2)
/content/content/csc421/a3
```

## Helper code

### Utility functions

```
%matplotlib inline

import os
import pdb
import argparse
import pickle as pkl
from pathlib import Path

from collections import defaultdict

import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker

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

```python
from torch.autograd import Variable

from six.moves.urllib.request import urlretrieve
import tarfile
import pickle
import sys


def get_file(
    fname, origin, untar=False, extract=False, archive_format="auto",
cache_dir="data"
):
    datadir = os.path.join(cache_dir)
    if not os.path.exists(datadir):
        os.makedirs(datadir)

    if untar:
        untar_fpath = os.path.join(datadir, fname)
        fpath = untar_fpath + ".tar.gz"
    else:
        fpath = os.path.join(datadir, fname)

    print(fpath)
    if not os.path.exists(fpath):
        print("Downloading data from", origin)

        error_msg = "URL fetch failure on {}: {} -- {}"
        try:
            try:
                urlretrieve(origin, fpath)
            except URLError as e:
                raise Exception(error_msg.format(origin, e.errno,
e.reason))
            except HTTPError as e:
                raise Exception(error_msg.format(origin, e.code,
e.msg))
        except (Exception, KeyboardInterrupt) as e:
            if os.path.exists(fpath):
                os.remove(fpath)
            raise

    if untar:
        if not os.path.exists(untar_fpath):
            print("Extracting file.")
            with tarfile.open(fpath) as archive:
                archive.extractall(datadir)
        return untar_fpath

    if extract:
```

```python
        _extract_archive(fpath, datadir, archive_format)

    return fpath


class AttrDict(dict):
    def __init__(self, *args, **kwargs):
        super(AttrDict, self).__init__(*args, **kwargs)
        self.__dict__ = self


def to_var(tensor, cuda):
    """Wraps a Tensor in a Variable, optionally placing it on the GPU.

    Arguments:
        tensor: A Tensor object.
        cuda: A boolean flag indicating whether to use the GPU.

    Returns:
        A Variable object, on the GPU if cuda==True.
    """
    if cuda:
        return Variable(tensor.cuda())
    else:
        return Variable(tensor)


def create_dir_if_not_exists(directory):
    """Creates a directory if it doesn't already exist."""
    if not os.path.exists(directory):
        os.makedirs(directory)


def save_loss_plot(train_losses, val_losses, opts):
    """Saves a plot of the training and validation loss curves."""
    plt.figure()
    plt.plot(range(len(train_losses)), train_losses)
    plt.plot(range(len(val_losses)), val_losses)
    plt.title("BS={}, nhid={}".format(opts.batch_size,
opts.hidden_size), fontsize=20)
    plt.xlabel("Epochs", fontsize=16)
    plt.ylabel("Loss", fontsize=16)
    plt.xticks(fontsize=14)
    plt.yticks(fontsize=14)
    plt.tight_layout()
    plt.savefig(os.path.join(opts.checkpoint_path, "loss_plot.pdf"))
    plt.close()
```

```
def save_loss_comparison_gru(l1, l2, o1, o2, fn, s=500):
    """Plot comparison of training and val loss curves from GRU runs.

    Arguments:
        l1: Tuple of lists containing training / val losses for model
1.
        l2: Tuple of lists containing training / val losses for model
2.
        o1: Options for model 1.
        o2: Options for model 2.
        fn: Output file name.
        s: Number of training iterations to average over.
    """
    mean_l1 = [np.mean(l1[0][i * s : (i + 1) * s]) for i in
range(len(l1[0]) // s)]
    mean_l2 = [np.mean(l2[0][i * s : (i + 1) * s]) for i in
range(len(l2[0]) // s)]

    plt.figure()

    fig, ax = plt.subplots(1, 2, figsize=(10, 4))

    ax[0].plot(range(len(mean_l1)), mean_l1, label="ds=" +
o1.data_file_name)
    ax[0].plot(range(len(mean_l2)), mean_l2, label="ds=" +
o2.data_file_name)
    ax[0].title.set_text("Train Loss | GRU Hidden Size =
{}".format(o2.hidden_size))

    # Validation losses are assumed to be by epoch
    ax[1].plot(range(len(l1[1])), l1[1], label="ds=" +
o1.data_file_name)
    ax[1].plot(range(len(l2[1])), l2[1], label="ds=" +
o2.data_file_name)
    ax[1].title.set_text("Val Loss | GRU Hidden Size =
{}".format(o2.hidden_size))

    ax[0].set_xlabel("Iterations (x{})".format(s), fontsize=10)
    ax[0].set_ylabel("Loss", fontsize=10)
    ax[1].set_xlabel("Epochs", fontsize=10)
    ax[1].set_ylabel("Loss", fontsize=10)
    ax[0].legend(loc="upper right")
    ax[1].legend(loc="upper right")

    fig.suptitle("GRU Performance by Dataset", fontsize=14)
    plt.tight_layout()
    fig.subplots_adjust(top=0.85)
    plt.legend()
```

```python
    plt_path = "./loss_plot_{}.pdf".format(fn)
    plt.savefig(plt_path)
    print(f"Plot saved to: {Path(plt_path).resolve()}")


def save_loss_comparison_by_dataset(l1, l2, l3, l4, o1, o2, o3, o4,
fn, s=500):
    """Plot comparison of training and validation loss curves from all
four
    runs in Part 3, comparing by dataset while holding hidden size
constant.

    Models within each pair (l1, l2) and (l3, l4) have the same hidden
sizes.

    Arguments:
        l1: Tuple of lists containing training / val losses for model
1.
        l2: Tuple of lists containing training / val losses for model
2.
        l3: Tuple of lists containing training / val losses for model
3.
        l4: Tuple of lists containing training / val losses for model
4.
        o1: Options for model 1.
        o2: Options for model 2.
        o3: Options for model 3.
        o4: Options for model 4.
        fn: Output file name.
        s: Number of training iterations to average over.
    """
    mean_l1 = [np.mean(l1[0][i * s : (i + 1) * s]) for i in
range(len(l1[0]) // s)]
    mean_l2 = [np.mean(l2[0][i * s : (i + 1) * s]) for i in
range(len(l2[0]) // s)]
    mean_l3 = [np.mean(l3[0][i * s : (i + 1) * s]) for i in
range(len(l3[0]) // s)]
    mean_l4 = [np.mean(l4[0][i * s : (i + 1) * s]) for i in
range(len(l4[0]) // s)]

    plt.figure()
    fig, ax = plt.subplots(2, 2, figsize=(10, 8))

    ax[0][0].plot(range(len(mean_l1)), mean_l1, label="ds=" +
o1.data_file_name)
    ax[0][0].plot(range(len(mean_l2)), mean_l2, label="ds=" +
o2.data_file_name)
    ax[0][0].title.set_text(
        "Train Loss | Model Hidden Size = {}".format(o1.hidden_size)
    )
```

```
    # Validation losses are assumed to be by epoch
    ax[0][1].plot(range(len(l1[1])), l1[1], label="ds=" +
o1.data_file_name)
    ax[0][1].plot(range(len(l2[1])), l2[1], label="ds=" +
o2.data_file_name)
    ax[0][1].title.set_text("Val Loss | Model Hidden Size =
{}".format(o1.hidden_size))

    ax[1][0].plot(range(len(mean_l3)), mean_l3, label="ds=" +
o3.data_file_name)
    ax[1][0].plot(range(len(mean_l4)), mean_l4, label="ds=" +
o4.data_file_name)
    ax[1][0].title.set_text(
        "Train Loss | Model Hidden Size = {}".format(o3.hidden_size)
    )

    ax[1][1].plot(range(len(l3[1])), l3[1], label="ds=" +
o3.data_file_name)
    ax[1][1].plot(range(len(l4[1])), l4[1], label="ds=" +
o4.data_file_name)
    ax[1][1].title.set_text("Val Loss | Model Hidden Size =
{}".format(o4.hidden_size))

    for i in range(2):
        ax[i][0].set_xlabel("Iterations (x{})".format(s), fontsize=10)
        ax[i][0].set_ylabel("Loss", fontsize=10)
        ax[i][1].set_xlabel("Epochs", fontsize=10)
        ax[i][1].set_ylabel("Loss", fontsize=10)
        ax[i][0].legend(loc="upper right")
        ax[i][1].legend(loc="upper right")

    fig.suptitle("Performance by Dataset Size", fontsize=16)
    plt.tight_layout()
    fig.subplots_adjust(top=0.9)
    plt.legend()
    plt.savefig("./loss_plot_{}.pdf".format(fn))
    plt.close()


def save_loss_comparison_by_hidden(l1, l2, l3, l4, o1, o2, o3, o4, fn,
s=500):
    """Plot comparison of training and validation loss curves from all
four
    runs in Part 3, comparing by hidden size while holding dataset
constant.

    Models within each pair (l1, l3) and (l2, l4) have the same
dataset.
```

```
    Arguments:
        l1: Tuple of lists containing training / val losses for model
1.
        l2: Tuple of lists containing training / val losses for model
2.
        l3: Tuple of lists containing training / val losses for model
3.
        l4: Tuple of lists containing training / val losses for model
4.
        o1: Options for model 1.
        o2: Options for model 2.
        o3: Options for model 3.
        o4: Options for model 4.
        fn: Output file name.
        s: Number of training iterations to average over.
    """
    mean_l1 = [np.mean(l1[0][i * s : (i + 1) * s]) for i in
range(len(l1[0]) // s)]
    mean_l2 = [np.mean(l2[0][i * s : (i + 1) * s]) for i in
range(len(l2[0]) // s)]
    mean_l3 = [np.mean(l3[0][i * s : (i + 1) * s]) for i in
range(len(l3[0]) // s)]
    mean_l4 = [np.mean(l4[0][i * s : (i + 1) * s]) for i in
range(len(l4[0]) // s)]

    plt.figure()
    fig, ax = plt.subplots(2, 2, figsize=(10, 8))

    ax[0][0].plot(range(len(mean_l1)), mean_l1, label="hid_size=" +
str(o1.hidden_size))
    ax[0][0].plot(range(len(mean_l3)), mean_l3, label="hid_size=" +
str(o3.hidden_size))
    ax[0][0].title.set_text("Train Loss | Dataset = " +
o1.data_file_name)

    # Validation losses are assumed to be by epoch
    ax[0][1].plot(range(len(l1[1])), l1[1], label="hid_size=" +
str(o1.hidden_size))
    ax[0][1].plot(range(len(l3[1])), l3[1], label="hid_size=" +
str(o3.hidden_size))
    ax[0][1].title.set_text("Val Loss | Dataset = " +
o1.data_file_name)

    ax[1][0].plot(range(len(mean_l2)), mean_l2, label="hid_size=" +
str(o2.hidden_size))
    ax[1][0].plot(range(len(mean_l4)), mean_l4, label="hid_size=" +
str(o4.hidden_size))
    ax[1][0].title.set_text("Train Loss | Dataset = " +
o3.data_file_name)
```

```
    ax[1][1].plot(range(len(l2[1])), l2[1], label="hid_size=" +
str(o2.hidden_size))
    ax[1][1].plot(range(len(l4[1])), l4[1], label="hid_size=" +
str(o4.hidden_size))
    ax[1][1].title.set_text("Val Loss | Dataset = " +
o4.data_file_name)

    for i in range(2):
        ax[i][0].set_xlabel("Iterations (x{})".format(s), fontsize=10)
        ax[i][0].set_ylabel("Loss", fontsize=10)
        ax[i][1].set_xlabel("Epochs", fontsize=10)
        ax[i][1].set_ylabel("Loss", fontsize=10)
        ax[i][0].legend(loc="upper right")
        ax[i][1].legend(loc="upper right")

    fig.suptitle("Performance by Hidden State Size", fontsize=16)
    plt.tight_layout()
    fig.subplots_adjust(top=0.9)
    plt.legend()
    plt.savefig("./loss_plot_{}.pdf".format(fn))
    plt.close()


def checkpoint(encoder, decoder, idx_dict, opts):
    """Saves the current encoder and decoder models, along with
idx_dict, which
    contains the char_to_index and index_to_char mappings, and the
start_token
    and end_token values.
    """
    with open(os.path.join(opts.checkpoint_path, "encoder.pt"), "wb")
as f:
        torch.save(encoder, f)

    with open(os.path.join(opts.checkpoint_path, "decoder.pt"), "wb")
as f:
        torch.save(decoder, f)

    with open(os.path.join(opts.checkpoint_path, "idx_dict.pkl"),
"wb") as f:
        pkl.dump(idx_dict, f)
```

## Data loader

```
def read_lines(filename):
    """Read a file and split it into lines."""
    lines = open(filename).read().strip().lower().split("\n")
    return lines
```

```
def read_pairs(filename):
    """Reads lines that consist of two words, separated by a space.

    Returns:
        source_words: A list of the first word in each line of the
file.
        target_words: A list of the second word in each line of the
file.
    """
    lines = read_lines(filename)
    source_words, target_words = [], []
    for line in lines:
        line = line.strip()
        if line:
            source, target = line.split()
            source_words.append(source)
            target_words.append(target)
    return source_words, target_words


def all_alpha_or_dash(s):
    """Helper function to check whether a string is alphabetic,
allowing dashes '-'."""
    return all(c.isalpha() or c == "-" for c in s)


def filter_lines(lines):
    """Filters lines to consist of only alphabetic characters or
dashes "-"."""
    return [line for line in lines if all_alpha_or_dash(line)]


def load_data(file_name):
    """Loads (English, Pig-Latin) word pairs, and creates mappings
from characters to indexes."""
    path = "./data/{}.txt".format(file_name)
    source_lines, target_lines = read_pairs(path)

    # Filter lines
    source_lines = filter_lines(source_lines)
    target_lines = filter_lines(target_lines)

    all_characters = set("".join(source_lines)) |
set("".join(target_lines))

    # Create a dictionary mapping each character to a unique index
    char_to_index = {
        char: index for (index, char) in
```

```
enumerate(sorted(list(all_characters)))
    }

    # Add start and end tokens to the dictionary
    start_token = len(char_to_index)
    end_token = len(char_to_index) + 1
    char_to_index["SOS"] = start_token
    char_to_index["EOS"] = end_token

    # Create the inverse mapping, from indexes to characters (used to
decode the model's predictions)
    index_to_char = {index: char for (char, index) in
char_to_index.items()}

    # Store the final size of the vocabulary
    vocab_size = len(char_to_index)

    line_pairs = list(set(zip(source_lines, target_lines)))  # Python
3

    idx_dict = {
        "char_to_index": char_to_index,
        "index_to_char": index_to_char,
        "start_token": start_token,
        "end_token": end_token,
    }

    return line_pairs, vocab_size, idx_dict


def create_dict(pairs):
    """Creates a mapping { (source_length, target_length): [list of
(source, target) pairs]
    This is used to make batches: each batch consists of two parallel
tensors, one containing
    all source indexes and the other containing all corresponding
target indexes.
    Within a batch, all the source words are the same length, and all
the target words are
    the same length.
    """
    unique_pairs = list(set(pairs))  # Find all unique (source,
target) pairs

    d = defaultdict(list)
    for (s, t) in unique_pairs:
        d[(len(s), len(t))].append((s, t))

    return d
```

## Training and evaluation code

```
def string_to_index_list(s, char_to_index, end_token):
    """Converts a sentence into a list of indexes (for each
character)."""
    return [char_to_index[char] for char in s] + [
        end_token
    ]  # Adds the end token to each index list


def translate_sentence(sentence, encoder, decoder, idx_dict, opts):
    """Translates a sentence from English to Pig-Latin, by splitting
the sentence into
    words (whitespace-separated), running the encoder-decoder model to
translate each
    word independently, and then stitching the words back together
with spaces between them.
    """
    if idx_dict is None:
        line_pairs, vocab_size, idx_dict =
load_data(opts["data_file_name"])
    return " ".join(
        [translate(word, encoder, decoder, idx_dict, opts) for word in
sentence.split()]
    )


def translate(input_string, encoder, decoder, idx_dict, opts):
    """Translates a given string from English to Pig-Latin."""

    char_to_index = idx_dict["char_to_index"]
    index_to_char = idx_dict["index_to_char"]
    start_token = idx_dict["start_token"]
    end_token = idx_dict["end_token"]

    max_generated_chars = 20
    gen_string = ""

    indexes = string_to_index_list(input_string, char_to_index,
end_token)
    indexes = to_var(
        torch.LongTensor(indexes).unsqueeze(0), opts.cuda
    )  # Unsqueeze to make it like BS = 1

    encoder_annotations, encoder_last_hidden = encoder(indexes)

    decoder_hidden = encoder_last_hidden
    decoder_input = to_var(torch.LongTensor([[start_token]]),
opts.cuda)  # For BS = 1
    decoder_inputs = decoder_input
```

```
    for i in range(max_generated_chars):
        ## slow decoding, recompute everything at each time
        decoder_outputs, attention_weights = decoder(
            decoder_inputs, encoder_annotations, decoder_hidden
        )

        generated_words = F.softmax(decoder_outputs, dim=2).max(2)[1]
        ni = generated_words.cpu().numpy().reshape(-1)  # LongTensor
of size 1
        ni = ni[-1]  # latest output token

        decoder_inputs = torch.cat([decoder_input, generated_words],
dim=1)

        if ni == end_token:
            break
        else:
            gen_string = "".join(
                [
                    index_to_char[int(item)]
                    for item in
generated_words.cpu().numpy().reshape(-1)
                ]
            )

    return gen_string


def visualize_attention(input_string, encoder, decoder, idx_dict,
opts):
    """Generates a heatmap to show where attention is focused in each
decoder step."""
    if idx_dict is None:
        line_pairs, vocab_size, idx_dict =
load_data(opts["data_file_name"])
    char_to_index = idx_dict["char_to_index"]
    index_to_char = idx_dict["index_to_char"]
    start_token = idx_dict["start_token"]
    end_token = idx_dict["end_token"]

    max_generated_chars = 20
    gen_string = ""

    indexes = string_to_index_list(input_string, char_to_index,
end_token)
    indexes = to_var(
        torch.LongTensor(indexes).unsqueeze(0), opts.cuda
    )  # Unsqueeze to make it like BS = 1
```

```
    encoder_annotations, encoder_hidden = encoder(indexes)

    decoder_hidden = encoder_hidden
    decoder_input = to_var(torch.LongTensor([[start_token]]),
opts.cuda)  # For BS = 1
    decoder_inputs = decoder_input

    produced_end_token = False

    for i in range(max_generated_chars):
        ## slow decoding, recompute everything at each time
        decoder_outputs, attention_weights = decoder(
            decoder_inputs, encoder_annotations, decoder_hidden
        )
        generated_words = F.softmax(decoder_outputs, dim=2).max(2)[1]
        ni = generated_words.cpu().numpy().reshape(-1)  # LongTensor
of size 1
        ni = ni[-1]  # latest output token

        decoder_inputs = torch.cat([decoder_input, generated_words],
dim=1)

        if ni == end_token:
            break
        else:
            gen_string = "".join(
                [
                    index_to_char[int(item)]
                    for item in
generated_words.cpu().numpy().reshape(-1)
                ]
            )

    if isinstance(attention_weights, tuple):
        ## transformer's attention mweights
        attention_weights, self_attention_weights = attention_weights

    all_attention_weights = attention_weights.data.cpu().numpy()

    for i in range(len(all_attention_weights)):
        attention_weights_matrix = all_attention_weights[i].squeeze()
        fig = plt.figure()
        ax = fig.add_subplot(111)
        cax = ax.matshow(attention_weights_matrix, cmap="bone")
        fig.colorbar(cax)

        # Set up axes
        ax.set_yticklabels([""] + list(input_string) + ["EOS"],
```

```
rotation=90)
        ax.set_xticklabels(
            [""] + list(gen_string) + (["EOS"] if produced_end_token
else [])
        )

        # Show label at every tick
        ax.xaxis.set_major_locator(ticker.MultipleLocator(1))
        ax.yaxis.set_major_locator(ticker.MultipleLocator(1))
        # Add title
        plt.xlabel("Attention weights to the source sentence in layer
{}".format(i + 1))
        plt.tight_layout()
        plt.grid("off")
        plt.show()

    return gen_string


def compute_loss(data_dict, encoder, decoder, idx_dict, criterion,
optimizer, opts):
    """Train/Evaluate the model on a dataset.

    Arguments:
        data_dict: The validation/test word pairs, organized by source
and target lengths.
        encoder: An encoder model to produce annotations for each step
of the input sequence.
        decoder: A decoder model (with or without attention) to
generate output tokens.
        idx_dict: Contains char-to-index and index-to-char mappings,
and start & end token indexes.
        criterion: Used to compute the CrossEntropyLoss for each
decoder output.
        optimizer: Train the weights if an optimizer is given. None if
only evaluate the model.
        opts: The command-line arguments.

    Returns:
        mean_loss: The average loss over all batches from data_dict.
    """
    start_token = idx_dict["start_token"]
    end_token = idx_dict["end_token"]
    char_to_index = idx_dict["char_to_index"]

    losses = []
    for key in data_dict:
        input_strings, target_strings = zip(*data_dict[key])
        input_tensors = [
```

```
            torch.LongTensor(string_to_index_list(s, char_to_index,
end_token))
            for s in input_strings
        ]
        target_tensors = [
            torch.LongTensor(string_to_index_list(s, char_to_index,
end_token))
            for s in target_strings
        ]

        num_tensors = len(input_tensors)
        num_batches = int(np.ceil(num_tensors /
float(opts.batch_size)))

        for i in range(num_batches):

            start = i * opts.batch_size
            end = start + opts.batch_size

            inputs = to_var(torch.stack(input_tensors[start:end]),
opts.cuda)
            targets = to_var(torch.stack(target_tensors[start:end]),
opts.cuda)

            # The batch size may be different in each epoch
            BS = inputs.size(0)

            encoder_annotations, encoder_hidden = encoder(inputs)

            # The last hidden state of the encoder becomes the first
hidden state of the decoder
            decoder_hidden = encoder_hidden

            start_vector = (
                torch.ones(BS).long().unsqueeze(1) * start_token
            )  # BS x 1 --> 16x1  CHECKED
            decoder_input = to_var(start_vector, opts.cuda)  # BS x 1
--> 16x1  CHECKED

            loss = 0.0

            seq_len = targets.size(1)  # Gets seq_len from BS x
seq_len

            decoder_inputs = torch.cat(
                [decoder_input, targets[:, 0:-1]], dim=1
            )  # Gets decoder inputs by shifting the targets to the
right
```

```
            decoder_outputs, attention_weights = decoder(
                decoder_inputs, encoder_annotations, decoder_hidden
            )
            decoder_outputs_flatten = decoder_outputs.view(-1,
decoder_outputs.size(2))
            targets_flatten = targets.view(-1)

            loss = criterion(decoder_outputs_flatten, targets_flatten)

            losses.append(loss.item())

            ## training if an optimizer is provided
            if optimizer:
                # Zero gradients
                optimizer.zero_grad()
                # Compute gradients
                loss.backward()
                # Update the parameters of the encoder and decoder
                optimizer.step()

    return losses


def training_loop(
    train_dict, val_dict, idx_dict, encoder, decoder, criterion,
optimizer, opts
):
    """Runs the main training loop; evaluates the model on the val set
every epoch.
        * Prints training and val loss each epoch.
        * Prints qualitative translation results each epoch using
TEST_SENTENCE
        * Saves an attention map for TEST_WORD_ATTN each epoch
        * Returns loss curves for comparison

    Arguments:
        train_dict: The training word pairs, organized by source and
target lengths.
        val_dict: The validation word pairs, organized by source and
target lengths.
        idx_dict: Contains char-to-index and index-to-char mappings,
and start & end token indexes.
        encoder: An encoder model to produce annotations for each step
of the input sequence.
        decoder: A decoder model (with or without attention) to
generate output tokens.
        criterion: Used to compute the CrossEntropyLoss for each
decoder output.
        optimizer: Implements a step rule to update the parameters of
```

```
the encoder and decoder.
        opts: The command-line arguments.

    Returns:
        losses: Lists containing training and validation loss curves.
    """

    start_token = idx_dict["start_token"]
    end_token = idx_dict["end_token"]
    char_to_index = idx_dict["char_to_index"]

    loss_log = open(os.path.join(opts.checkpoint_path,
"loss_log.txt"), "w")

    best_val_loss = 1e6
    train_losses = []
    val_losses = []

    mean_train_losses = []
    mean_val_losses = []

    early_stopping_counter = 0

    for epoch in range(opts.nepochs):

        optimizer.param_groups[0]["lr"] *= opts.lr_decay

        train_loss = compute_loss(
            train_dict, encoder, decoder, idx_dict, criterion,
optimizer, opts
        )
        val_loss = compute_loss(
            val_dict, encoder, decoder, idx_dict, criterion, None,
opts
        )

        mean_train_loss = np.mean(train_loss)
        mean_val_loss = np.mean(val_loss)

        if mean_val_loss < best_val_loss:
            checkpoint(encoder, decoder, idx_dict, opts)
            best_val_loss = mean_val_loss
            early_stopping_counter = 0
        else:
            early_stopping_counter += 1

        if early_stopping_counter > opts.early_stopping_patience:
            print(
                "Validation loss has not improved in {} epochs,
```

```
stopping early".format(
                    opts.early_stopping_patience
                )
            )
            print("Obtained lowest validation loss of:
{}".format(best_val_loss))
            return (train_losses, mean_val_losses)

        gen_string = translate_sentence(TEST_SENTENCE, encoder,
decoder, idx_dict, opts)
        print(
            "Epoch: {:3d} | Train loss: {:.3f} | Val loss: {:.3f} |
Gen: {:20s}".format(
                epoch, mean_train_loss, mean_val_loss, gen_string
            )
        )

        loss_log.write("{} {} {}\n".format(epoch, train_loss,
val_loss))
        loss_log.flush()

        train_losses += train_loss
        val_losses += val_loss

        mean_train_losses.append(mean_train_loss)
        mean_val_losses.append(mean_val_loss)

        save_loss_plot(mean_train_losses, mean_val_losses, opts)

    print("Obtained lowest validation loss of:
{}".format(best_val_loss))
    return (train_losses, mean_val_losses)


def print_data_stats(line_pairs, vocab_size, idx_dict):
    """Prints example word pairs, the number of data points, and the
vocabulary."""
    print("=" * 80)
    print("Data Stats".center(80))
    print("-" * 80)
    for pair in line_pairs[:5]:
        print(pair)
    print("Num unique word pairs: {}".format(len(line_pairs)))
    print("Vocabulary: {}".format(idx_dict["char_to_index"].keys()))
    print("Vocab size: {}".format(vocab_size))
    print("=" * 80)


def train(opts):
```

```
    line_pairs, vocab_size, idx_dict =
load_data(opts["data_file_name"])
    print_data_stats(line_pairs, vocab_size, idx_dict)

    # Split the line pairs into an 80% train and 20% val split
    num_lines = len(line_pairs)
    num_train = int(0.8 * num_lines)
    train_pairs, val_pairs = line_pairs[:num_train],
line_pairs[num_train:]

    # Group the data by the lengths of the source and target words, to
form batches
    train_dict = create_dict(train_pairs)
    val_dict = create_dict(val_pairs)


######################################################################
####
    ### Setup: Create Encoder, Decoder, Learning Criterion, and
Optimizers ###

######################################################################
####
    if opts.encoder_type == "rnn":
        encoder = GRUEncoder(
            vocab_size=vocab_size, hidden_size=opts.hidden_size,
opts=opts
        )
    elif opts.encoder_type == "transformer":
        encoder = TransformerEncoder(
            vocab_size=vocab_size,
            hidden_size=opts.hidden_size,
            num_layers=opts.num_transformer_layers,
            opts=opts,
        )
    elif opts.encoder_type == "attention":
      encoder = AttentionEncoder(
            vocab_size=vocab_size,
            hidden_size=opts.hidden_size,
            opts=opts,
        )
    else:
        raise NotImplementedError

    if opts.decoder_type == "rnn":
        decoder = RNNDecoder(vocab_size=vocab_size,
hidden_size=opts.hidden_size)
    elif opts.decoder_type == "rnn_attention":
        decoder = RNNAttentionDecoder(
            vocab_size=vocab_size,
```

```
            hidden_size=opts.hidden_size,
            attention_type=opts.attention_type,
        )
    elif opts.decoder_type == "transformer":
        decoder = TransformerDecoder(
            vocab_size=vocab_size,
            hidden_size=opts.hidden_size,
            num_layers=opts.num_transformer_layers,
        )
    elif opts.encoder_type == "attention":
      decoder = AttentionDecoder(
            vocab_size=vocab_size,
            hidden_size=opts.hidden_size,
        )
    else:
        raise NotImplementedError

    #### setup checkpoint path
    model_name = "h{}-bs{}-{}-{}".format(
        opts.hidden_size, opts.batch_size, opts.decoder_type,
opts.data_file_name
    )
    opts.checkpoint_path = model_name
    create_dir_if_not_exists(opts.checkpoint_path)
    ####

    if opts.cuda:
        encoder.cuda()
        decoder.cuda()
        print("Moved models to GPU!")

    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(
        list(encoder.parameters()) + list(decoder.parameters()),
lr=opts.learning_rate
    )

    try:
        losses = training_loop(
            train_dict, val_dict, idx_dict, encoder, decoder,
criterion, optimizer, opts
        )
    except KeyboardInterrupt:
        print("Exiting early from training.")
        return encoder, decoder, losses

    return encoder, decoder, losses
```

```
def print_opts(opts):
    """Prints the values of all command-line arguments."""
    print("=" * 80)
    print("Opts".center(80))
    print("-" * 80)
    for key in opts.__dict__:
        print("{:>30}: {:<30}".format(key,
opts.__dict__[key]).center(80))
    print("=" * 80)
```

### Download dataset

```
######################################################################
# Download Translation datasets
######################################################################
data_fpath = get_file(
    fname="pig_latin_small.txt",
    origin="http://www.cs.toronto.edu/~jba/pig_latin_small.txt",
    untar=False,
)

data_fpath = get_file(
    fname="pig_latin_large.txt",
    origin="http://www.cs.toronto.edu/~jba/pig_latin_large.txt",
    untar=False,
)
```

```
data/pig_latin_small.txt
Downloading data from
http://www.cs.toronto.edu/~jba/pig_latin_small.txt
data/pig_latin_large.txt
Downloading data from
http://www.cs.toronto.edu/~jba/pig_latin_large.txt
```

## Part 1: Neural machine translation (NMT)

In this section, you will implement a Gated Recurrent Unit (GRU) cell, a common type of recurrent neural network (RNN). The GRU cell is a simplification of the Long Short-Term Memory cell. Therefore, we have provided you with an implemented LSTM cell (`MyLSTMCell`), which you can reference when completing `MyGRUCell`.

```
class MyLSTMCell(nn.Module):
    def __init__(self, input_size, hidden_size):
        super(MyLSTMCell, self).__init__()

        self.input_size = input_size
        self.hidden_size = hidden_size

        self.Wif = nn.Linear(input_size, hidden_size)
        self.Whf = nn.Linear(hidden_size, hidden_size)
```

```
        self.Wii = nn.Linear(input_size, hidden_size)
        self.Whi = nn.Linear(hidden_size, hidden_size)

        self.Wic = nn.Linear(input_size, hidden_size)
        self.Whc = nn.Linear(hidden_size, hidden_size)

        self.Wio = nn.Linear(input_size, hidden_size)
        self.Who = nn.Linear(hidden_size, hidden_size)

    def forward(self, x, h_prev, c_prev):
        """Forward pass of the LSTM computation for one time step.

        Arguments
            x: batch_size x input_size
            h_prev: batch_size x hidden_size
            c_prev: batch_size x hidden_size

        Returns:
            h_new: batch_size x hidden_size
            c_new: batch_size x hidden_size
        """

        f = torch.sigmoid(self.Wif(x) + self.Whf(h_prev))
        i = torch.sigmoid(self.Wii(x) + self.Whi(h_prev))

        c = torch.tanh(self.Wic(x) + self.Whc(h_prev))
        o = torch.sigmoid(self.Wio(x) + self.Who(h_prev))

        c_new = f * c_prev + i * c
        h_new = o * torch.tanh(c_new)

        return h_new, c_new
```

### Step 1: GRU Cell

Please implement the `MyGRUCell` class defined in the next cell.

```
class MyGRUCell(nn.Module):
    def __init__(self, input_size, hidden_size):
        super(MyGRUCell, self).__init__()

        self.input_size = input_size
        self.hidden_size = hidden_size

        # Input linear layers
        self.Wiz = nn.Linear(input_size, hidden_size)
        self.Wir = nn.Linear(input_size, hidden_size)
        self.Wih = nn.Linear(input_size, hidden_size)
```

```
        # Hidden linear layers
        self.Whz = nn.Linear(hidden_size, hidden_size)
        self.Whr = nn.Linear(hidden_size, hidden_size)
        self.Whh = nn.Linear(hidden_size, hidden_size)

    def forward(self, x, h_prev):
        """Forward pass of the GRU computation for one time step.

        Arguments
            x: batch_size x input_size
            h_prev: batch_size x hidden_size

        Returns:
            h_new: batch_size x hidden_size
        """

        z = torch.sigmoid(self.Wiz(x) + self.Whz(h_prev))
        r = torch.sigmoid(self.Wir(x) + self.Whr(h_prev))
        g = torch.tanh(self.Wih(x) + r * (self.Whh(h_prev * r)))
        h_new = (1 - z) * h_prev + z * g
        return h_new
```

### Step 2: GRU Encoder

The following cells use your `MyGRUCell` implementation to build a recurrent encoder and decoder. Please read the implementations to understand what they do and run the cells before proceeding.

```
class GRUEncoder(nn.Module):
    def __init__(self, vocab_size, hidden_size, opts):
        super(GRUEncoder, self).__init__()

        self.vocab_size = vocab_size
        self.hidden_size = hidden_size
        self.opts = opts

        self.embedding = nn.Embedding(vocab_size, hidden_size)
        self.gru = MyGRUCell(hidden_size, hidden_size)

    def forward(self, inputs):
        """Forward pass of the encoder RNN.

        Arguments:
            inputs: Input token indexes across a batch for all time
steps in the sequence. (batch_size x seq_len)

        Returns:
            annotations: The hidden states computed at each step of
```

```
the input sequence. (batch_size x seq_len x hidden_size)
            hidden: The final hidden state of the encoder, for each
sequence in a batch. (batch_size x hidden_size)
        """

        batch_size, seq_len = inputs.size()
        hidden = self.init_hidden(batch_size)

        encoded = self.embedding(inputs)  # batch_size x seq_len x
hidden_size
        annotations = []

        for i in range(seq_len):
            x = encoded[:, i, :]  # Get the current time step, across
the whole batch
            hidden = self.gru(x, hidden)
            annotations.append(hidden)

        annotations = torch.stack(annotations, dim=1)
        return annotations, hidden

    def init_hidden(self, bs):
        """Creates a tensor of zeros to represent the initial hidden
states
        of a batch of sequences.

        Arguments:
            bs: The batch size for the initial hidden state.

        Returns:
            hidden: An initial hidden state of all zeros. (batch_size
x hidden_size)
        """
        return to_var(torch.zeros(bs, self.hidden_size),
self.opts.cuda)

class RNNDecoder(nn.Module):
    def __init__(self, vocab_size, hidden_size):
        super(RNNDecoder, self).__init__()
        self.vocab_size = vocab_size
        self.hidden_size = hidden_size

        self.embedding = nn.Embedding(vocab_size, hidden_size)
        self.rnn = MyGRUCell(input_size=hidden_size,
hidden_size=hidden_size)
        self.out = nn.Linear(hidden_size, vocab_size)

    def forward(self, inputs, annotations, hidden_init):
        """Forward pass of the non-attentional decoder RNN.
```

```
        Arguments:
            inputs: Input token indexes across a batch. (batch_size x
seq_len)
            annotations: This is not used here. It just maintains
consistency with the
                    interface used by the AttentionDecoder class.
            hidden_init: The hidden states from the last step of
encoder, across a batch. (batch_size x hidden_size)

        Returns:
            output: Un-normalized scores for each token in the
vocabulary, across a batch for all the decoding time steps.
(batch_size x decoder_seq_len x vocab_size)
            None
        """
        batch_size, seq_len = inputs.size()
        embed = self.embedding(inputs)  # batch_size x seq_len x
hidden_size

        hiddens = []
        h_prev = hidden_init

        for i in range(seq_len):
            x = embed[
                :, i, :
            ]  # Get the current time step input tokens, across the
whole batch
            h_prev = self.rnn(x, h_prev)  # batch_size x hidden_size
            hiddens.append(h_prev)

        hiddens = torch.stack(hiddens, dim=1)  # batch_size x seq_len
x hidden_size

        output = self.out(hiddens)  # batch_size x seq_len x
vocab_size
        return output, None
```

### Step 3: Training and Analysis

Train the encoder-decoder model to perform English --> Pig Latin translation. We will start by training on the smaller dataset.

```
TEST_SENTENCE = "the air conditioning is working"

rnn_args_s = AttrDict()
args_dict = {
    "data_file_name": "pig_latin_small",
    "cuda": True,
    "nepochs": 50,
```

```
    "checkpoint_dir": "checkpoints",
    "learning_rate": 0.005,
    "lr_decay": 0.99,
    "early_stopping_patience": 20,
    "batch_size": 64,
    "hidden_size": 32,
    "encoder_type": "rnn",  # options: rnn / transformer
    "decoder_type": "rnn",  # options: rnn / rnn_attention /
transformer
    "attention_type": "",   # options: additive / scaled_dot
}
rnn_args_s.update(args_dict)

print_opts(rnn_args_s)
rnn_encode_s, rnn_decoder_s, rnn_losses_s = train(rnn_args_s)

translated = translate_sentence(
    TEST_SENTENCE, rnn_encode_s, rnn_decoder_s, None, rnn_args_s
)
print("source:\t\t{} \ntranslated:\t{}".format(TEST_SENTENCE,
translated))
```

```
======================================================================
==========
                                      Opts

----------------------------------------------------------------------
----------
                         data_file_name: pig_latin_small

                                   cuda: 1

                                nepochs: 50

                         checkpoint_dir: checkpoints

                          learning_rate: 0.005

                               lr_decay: 0.99

                early_stopping_patience: 20

                             batch_size: 64

                            hidden_size: 32

                           encoder_type: rnn

                           decoder_type: rnn
```

```
attention_type:

================================================================================
================================================================================
                                        Data Stats

--------------------------------------------------------------------------------
('folded', 'oldedfay')
('supposition', 'uppositionsay')
('advancing', 'advancingway')
('reconcile', 'econcileray')
('leagued', 'eaguedlay')
Num unique word pairs: 3198
Vocabulary: dict_keys(['-', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h',
'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v',
'w', 'x', 'y', 'z', 'SOS', 'EOS'])
Vocab size: 29
================================================================================
Moved models to GPU!
Epoch:   0 | Train loss: 2.297 | Val loss: 2.022 | Gen: ingay-onsay-
onsay-on ingay-onsay-onsay-on ingsay-onsay-onsay-o ingay-onsay-onsay-
on ingsay-onsay-onsay-o
Epoch:   1 | Train loss: 1.891 | Val loss: 1.827 | Gen: eway alway
onssay ingsay onsay
Epoch:   2 | Train loss: 1.715 | Val loss: 1.728 | Gen: eway away-
onday oonsay isteray oonsay
Epoch:   3 | Train loss: 1.601 | Val loss: 1.682 | Gen: eday away-
onday onday-onday isteray onday-onday
Epoch:   4 | Train loss: 1.517 | Val loss: 1.644 | Gen: eway away-
onday onday-onday-onday iway otay-onday
Epoch:   5 | Train loss: 1.443 | Val loss: 1.597 | Gen: eway away-
ingsay-onday onday-onday-onday iway otay-onday
Epoch:   6 | Train loss: 1.370 | Val loss: 1.564 | Gen: eway away-otay
ondeday-otay-otay iway otay-onday-atersay
Epoch:   7 | Train loss: 1.312 | Val loss: 1.548 | Gen: eway away-
away-away-aters otay-ingsay-onday-at iway otay
Epoch:   8 | Train loss: 1.261 | Val loss: 1.550 | Gen: eway aiway-
ingsay-oday oncingsay-oday iway otay-away-awlay
Epoch:   9 | Train loss: 1.216 | Val loss: 1.493 | Gen: eway away-
ingsray-outedwa oncitersay-oday isway ondingsray-oteray
Epoch:  10 | Train loss: 1.167 | Val loss: 1.471 | Gen: eway away-
ingsray-outedwa oncinessay-ingway-in isway oringsay-oday
Epoch:  11 | Train loss: 1.132 | Val loss: 1.517 | Gen: eway aindedway
oncienceringsray-ote isway onmouredway
Epoch:  12 | Train loss: 1.113 | Val loss: 1.499 | Gen: eway iway-
```

```
indway-awlay oncienceday isway ortisedway
Epoch:  13 | Train loss: 1.076 | Val loss: 1.509 | Gen: eway aindway
oncionway-away-awlay isway ortinedway
Epoch:  14 | Train loss: 1.040 | Val loss: 1.468 | Gen: eway anway-
ingsray-ousedw oncienceringway isway omporedway
Epoch:  15 | Train loss: 1.011 | Val loss: 1.491 | Gen: eathay aindway
onciencedway isway orksterway
Epoch:  16 | Train loss: 0.985 | Val loss: 1.466 | Gen: eway amay-
ingsray-oughtay onciencedway isway omprerestencedway
Epoch:  17 | Train loss: 0.965 | Val loss: 1.442 | Gen: eway
aingencedway onciencedway isway onmonsingway-oughtay
Epoch:  18 | Train loss: 0.940 | Val loss: 1.433 | Gen: easedway
aingenay onciecepay-outeway isway offineway
Epoch:  19 | Train loss: 0.937 | Val loss: 1.410 | Gen: eway amay
onciencedway isway offinencedway
Epoch:  20 | Train loss: 0.916 | Val loss: 1.449 | Gen: eatsay
aindeway oncondingway-ietient isway offrineway
Epoch:  21 | Train loss: 0.902 | Val loss: 1.360 | Gen: ehay aingeray
ongingday-andway-oda isway offinentray
Epoch:  22 | Train loss: 0.877 | Val loss: 1.348 | Gen: eway aingeray
onciencedway isway orforificationsway
Epoch:  23 | Train loss: 0.852 | Val loss: 1.351 | Gen: ehay aingenay
onciencedway isway offinessay
Epoch:  24 | Train loss: 0.854 | Val loss: 1.358 | Gen: ehay aingsray
ongray-iecepay isway ofrsway
Epoch:  25 | Train loss: 0.830 | Val loss: 1.325 | Gen: eway aingray
ongingday isway offourshay
Epoch:  26 | Train loss: 0.814 | Val loss: 1.325 | Gen: ehay aingray
oncientray-otedway isway orforfientway
Epoch:  27 | Train loss: 0.794 | Val loss: 1.337 | Gen: ehay aingay
oncingway-oday isway orforingway
Epoch:  28 | Train loss: 0.780 | Val loss: 1.348 | Gen: ehay aingray
ongrandingway isway offouredway
Epoch:  29 | Train loss: 0.763 | Val loss: 1.329 | Gen: ehay aingray
ongringalingway isway orfinedway
Epoch:  30 | Train loss: 0.749 | Val loss: 1.387 | Gen: ehay aingay
oncingday-aturednay isway onkersay-ondway-awla
Epoch:  31 | Train loss: 0.746 | Val loss: 1.356 | Gen: ehay aingray
ongrindedway isway orfourshay
Epoch:  32 | Train loss: 0.739 | Val loss: 1.404 | Gen: ehay aingray
oncingday-ousedway isway onkneway-iecepay
Epoch:  33 | Train loss: 0.737 | Val loss: 1.367 | Gen: ehay ay
ongingday-otecay isway onknersay
Epoch:  34 | Train loss: 0.722 | Val loss: 1.354 | Gen: ehay aingray
oncingday-otecay isway orkway-aturednay
Epoch:  35 | Train loss: 0.705 | Val loss: 1.350 | Gen: ehay aingray
ongingdray-otecay isway onknersway
Epoch:  36 | Train loss: 0.694 | Val loss: 1.393 | Gen: ehay aingray
onginglay-iecepay isway onksay-ondemnationsd
Epoch:  37 | Train loss: 0.698 | Val loss: 1.391 | Gen: ehay ay
```

```
oncingday-otecay isway orkneway
Epoch:  38 | Train loss: 0.688 | Val loss: 1.383 | Gen: ehay arway
ongingday-ousedway isway onkersedway
Epoch:  39 | Train loss: 0.675 | Val loss: 1.361 | Gen: ehay ay
ongingday-odesway isway orkway-ightnay
Epoch:  40 | Train loss: 0.672 | Val loss: 1.361 | Gen: ehay ay
oncinglay-iecepay isway onknersay
Epoch:  41 | Train loss: 0.663 | Val loss: 1.417 | Gen: ehay arway
oncivicationray isway onksay-ortificationm
Epoch:  42 | Train loss: 0.676 | Val loss: 1.396 | Gen: ehay ay
ongingrandway isway orkway-ightnay
Epoch:  43 | Train loss: 0.674 | Val loss: 1.383 | Gen: ehay anay
ongingday isway orkneway
Epoch:  44 | Train loss: 0.681 | Val loss: 1.353 | Gen: ehay aingray
ongingnationday isway orkshedway
Epoch:  45 | Train loss: 0.669 | Val loss: 1.364 | Gen: ehay ay
ongingday-indway-yba isway orkway-indway-ybay
Validation loss has not improved in 20 epochs, stopping early
Obtained lowest validation loss of: 1.3249081877561717
source:         the air conditioning is working
translated:     ehay aingray ongingday-odway isway onkuningway
```

Next, we train on the larger dataset. This experiment investigates if increasing dataset size improves model generalization on the validation set.

For a fair comparison, the number of iterations (not number of epochs) for each run should be similar. This is done in a quick and dirty way by adjusting the batch size so approximately the same number of batches is processed per epoch.

```python
TEST_SENTENCE = "the air conditioning is working"

rnn_args_l = AttrDict()
args_dict = {
    "data_file_name": "pig_latin_large",
    "cuda": True,
    "nepochs": 50,
    "checkpoint_dir": "checkpoints",
    "learning_rate": 0.005,
    "lr_decay": 0.99,
    "early_stopping_patience": 10,
    "batch_size": 512,
    "hidden_size": 32,
    "encoder_type": "rnn",  # options: rnn / transformer
    "decoder_type": "rnn",  # options: rnn / rnn_attention /
transformer
    "attention_type": "",   # options: additive / scaled_dot
}
rnn_args_l.update(args_dict)

print_opts(rnn_args_l)
```

```
rnn_encode_l, rnn_decoder_l, rnn_losses_l = train(rnn_args_l)

translated = translate_sentence(
    TEST_SENTENCE, rnn_encode_l, rnn_decoder_l, None, rnn_args_l
)
print("source:\t\t{} \ntranslated:\t{}".format(TEST_SENTENCE,
translated))
```

```
======================================================================
==========
                                      Opts

----------------------------------------------------------------------
----------
                         data_file_name: pig_latin_large

                                   cuda: 1

                                nepochs: 50

                         checkpoint_dir: checkpoints

                          learning_rate: 0.005

                               lr_decay: 0.99

                early_stopping_patience: 10

                             batch_size: 512

                            hidden_size: 32

                           encoder_type: rnn

                           decoder_type: rnn

                         attention_type:

======================================================================
==========
======================================================================
==========
                                   Data Stats

----------------------------------------------------------------------
----------
('ford', 'ordfay')
('eq', 'eqway')
('needs', 'eedsnay')
```

```
('frontline', 'ontlinefray')
('labor', 'aborlay')
Num unique word pairs: 22402
Vocabulary: dict_keys(['-', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h',
'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v',
'w', 'x', 'y', 'z', 'SOS', 'EOS'])
Vocab size: 29
======================================================================
==========
Moved models to GPU!
Epoch:   0 | Train loss: 2.331 | Val loss: 2.083 | Gen: eray-ay-ay ay-
ay estay-ay-eray-ay-ay ay-ay-ay eray-eray-ay-ay
Epoch:   1 | Train loss: 1.893 | Val loss: 1.919 | Gen: esay-esay-ay-
esay-ay away ongay-ingay-ay-ay-ay atersay-ay-esay-ay-e oteray-edway
Epoch:   2 | Train loss: 1.720 | Val loss: 1.814 | Gen: edway away
ontay-ingay-ingay-in atersay-onsay-onsay otingray-ingay-inway
Epoch:   3 | Train loss: 1.599 | Val loss: 1.728 | Gen: edway away
ontingay-inway istay-onsay omontay-inghay
Epoch:   4 | Train loss: 1.491 | Val loss: 1.666 | Gen: eday away
ontingay-otingay-oth issay oomurationtay-intera
Epoch:   5 | Train loss: 1.408 | Val loss: 1.640 | Gen: edway away
ontinglay-inway issay omulintationcay
Epoch:   6 | Train loss: 1.355 | Val loss: 1.579 | Gen: edway away
ontay-inway-inway istay oomurationgray
Epoch:   7 | Train loss: 1.294 | Val loss: 1.546 | Gen: edway away
ontinglay istay orivintationcay
Epoch:   8 | Train loss: 1.221 | Val loss: 1.482 | Gen: edtay away
ontinglay issway orivingray
Epoch:   9 | Train loss: 1.151 | Val loss: 1.458 | Gen: edtay aitionay
ontinglay-imemay-ime istay orivinicationday
Epoch:  10 | Train loss: 1.122 | Val loss: 1.504 | Gen: eday-imetay
away ontinglay-imemay-ime issway orivingmeray
Epoch:  11 | Train loss: 1.094 | Val loss: 1.427 | Gen: etay-edway
aitionway ontinglay istay orivermentay-inedway
Epoch:  12 | Train loss: 1.037 | Val loss: 1.375 | Gen: edtay away
ontinglingway issway orkingedmay
Epoch:  13 | Train loss: 0.993 | Val loss: 1.362 | Gen: edtay away
ontinglingday istay orkingdereway
Epoch:  14 | Train loss: 0.971 | Val loss: 1.391 | Gen: eteday away
ontionday issway orkingdereway
Epoch:  15 | Train loss: 0.967 | Val loss: 1.344 | Gen: edway away
ontininglingway issway orkingedfay
Epoch:  16 | Train loss: 0.939 | Val loss: 1.404 | Gen: edtay away
ontinglay-emingnay issway ormingderedway
Epoch:  17 | Train loss: 0.922 | Val loss: 1.352 | Gen: edtay ailyway
ontinglyfay issway orikedway
Epoch:  18 | Train loss: 0.876 | Val loss: 1.255 | Gen: edtay away
ontinglay-etrovingwa issway orkingdereway
Epoch:  19 | Train loss: 0.830 | Val loss: 1.280 | Gen: edtay away
ontingdray issway oridemnay
```

```
Epoch:  20 | Train loss: 0.818 | Val loss: 1.336 | Gen: edtay airicay
ontingingderentay issway orkingdereway
Epoch:  21 | Train loss: 0.823 | Val loss: 1.242 | Gen: edtay away
ontingdrouway isway orkingdray
Epoch:  22 | Train loss: 0.810 | Val loss: 1.333 | Gen: edtay away
ontinglay-omemnay issway orkingderay
Epoch:  23 | Train loss: 0.830 | Val loss: 1.325 | Gen: edtay away
ontingday isway orkinglay
Epoch:  24 | Train loss: 0.799 | Val loss: 1.248 | Gen: edtay away
ondingnay issway orkingday
Epoch:  25 | Train loss: 0.764 | Val loss: 1.204 | Gen: edtyway away
onnondinglay-elitywa issway orkingdereway
Epoch:  26 | Train loss: 0.744 | Val loss: 1.209 | Gen: edtay away
onningnedway isway orkingdray
Epoch:  27 | Train loss: 0.724 | Val loss: 1.140 | Gen: edtay airway
ontinglidegray isway orkingday
Epoch:  28 | Train loss: 0.698 | Val loss: 1.159 | Gen: ethay away
ontinglidegray isway orkingdray
Epoch:  29 | Train loss: 0.686 | Val loss: 1.160 | Gen: ethay airway
ontinondinglay issway oridemenglay
Epoch:  30 | Train loss: 0.677 | Val loss: 1.225 | Gen: ethay airway
onininglinglytay isway orkingedway
Epoch:  31 | Train loss: 0.688 | Val loss: 1.234 | Gen: edtay airway
ondingtay issway oridementway
Epoch:  32 | Train loss: 0.705 | Val loss: 1.199 | Gen: ethay airway
onininglay-elitytay isway orindimentway
Epoch:  33 | Train loss: 0.695 | Val loss: 1.171 | Gen: ethay airway
ontingionday-iecepay issway orkingdray
Epoch:  34 | Train loss: 0.657 | Val loss: 1.227 | Gen: edtay airway
onningedway isway oridemnnway
Epoch:  35 | Train loss: 0.650 | Val loss: 1.216 | Gen: ethay aryway
oninondingderay-inem isway orkingedfay
Epoch:  36 | Train loss: 0.674 | Val loss: 1.255 | Gen: edtay airway
ononiondingway issway orkionedgray
Epoch:  37 | Train loss: 0.651 | Val loss: 1.143 | Gen: ehtay airway
oninidentray issway orkingeday
Epoch:  38 | Train loss: 0.604 | Val loss: 1.140 | Gen: ehtay airway
onningderobay issway orkingday
Epoch:  39 | Train loss: 0.579 | Val loss: 1.131 | Gen: ehtay airway
ondiningctercay isway orkingday
Epoch:  40 | Train loss: 0.578 | Val loss: 1.174 | Gen: ehtay airway
ondinondingway issway orkingdray
Epoch:  41 | Train loss: 0.569 | Val loss: 1.155 | Gen: ehtay airway
oningonday-inedway isway orkingday
Epoch:  42 | Train loss: 0.583 | Val loss: 1.237 | Gen: edtay airway
oninondingway-eway isway orkingway
Epoch:  43 | Train loss: 0.601 | Val loss: 1.160 | Gen: ehtay airway
oninglay-imetetay isway orkingday
Epoch:  44 | Train loss: 0.569 | Val loss: 1.109 | Gen: ehtay airway
onninondingday isway orkingdray
```

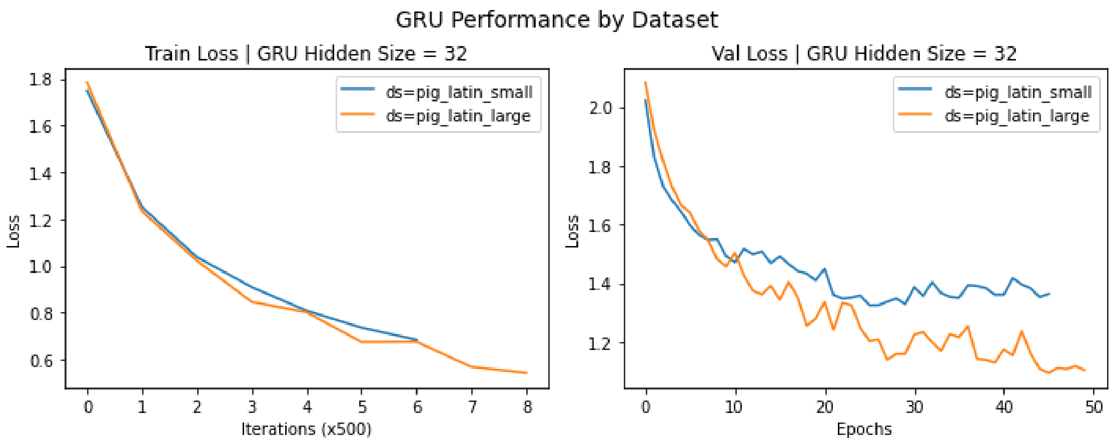
```
Epoch:  45 | Train loss: 0.547 | Val loss: 1.095 | Gen: ehtay airway ondinicgancecay isway orkingday
Epoch:  46 | Train loss: 0.527 | Val loss: 1.112 | Gen: ehtay airway ondiningctorhay isway orkingday
Epoch:  47 | Train loss: 0.516 | Val loss: 1.108 | Gen: ehtay airway ondiningctorhay isway orkingday
Epoch:  48 | Train loss: 0.510 | Val loss: 1.118 | Gen: ehtay airway ondinicgabloway isway orkingday
Epoch:  49 | Train loss: 0.506 | Val loss: 1.104 | Gen: ehtay airway ondioningdray isway orkingday
Obtained lowest validation loss of: 1.095056235182042
source:         the air conditioning is working
translated:     ehtay airway ondioningdray isway orkingday
```

The code below plots the training and validation losses of each model, as a function of the number of gradient descent iterations. Are there significant differences in the validation performance of each model? (see follow-up questions in handout)

```
save_loss_comparison_gru(rnn_losses_s, rnn_losses_l, rnn_args_s, rnn_args_l, "gru")
```

```
Plot saved to: /content/content/csc421/a3/loss_plot_gru.pdf
```

```
<Figure size 432x288 with 0 Axes>
```




### Question 1

Overall, the validation loss for both models are similar. However, `pig_latin_small` has a higher validation loss. Therefore, `pig_latin_large` performs better than `pig_latin_small`.

For code, refer to Step 1.

Select best performing model, and try translating different sentences by changing the variable TEST_SENTENCE. Identify a failure mode and briefly describe it (see follow-up questions in handout).

```
best_encoder = rnn_encode_l  # Replace with rnn_encode_s or
rnn_encode_l
best_decoder = rnn_decoder_l  # Replace with rnn_decoder_s or
rnn_decoder_l
best_args = rnn_args_l  # Replace with rnn_args_s or rnn_args_l

TEST_SENTENCE = "the air conditioning is working"
translated = translate_sentence(
    TEST_SENTENCE, best_encoder, best_decoder, None, best_args
)
print("source:\t\t{} \ntranslated:\t{}".format(TEST_SENTENCE,
translated))
```

```
source:         the air conditioning is working
translated:     ehtay airway ondioningdray isway orkingday
```

### Question 2

failed input & output pairs:

[conditioning, ondioningdray] [working, orkingday]

In the TEST_SENTENCE, 2 words failed: conditioning and working. This shows that the model cannot work with words ending in "ing" very well as some characters are missing after translation.

### Question 3

Number of parameters of the LSTM encoder is $4HVDK+4H^2$ and number of parameters of GRU encoder is $3HVDK+3H^2$.

## Part 2: Attention mechanisms

### Step 1: Additive attention

In the next cell, the additive attention mechanism has been implemented for you. Please take a momement to read through it and understand what it is doing. See the assignment handouts for details.

```
class AdditiveAttention(nn.Module):
    def __init__(self, hidden_size):
        super(AdditiveAttention, self).__init__()

        self.hidden_size = hidden_size

        # A two layer fully-connected network
        # hidden_size * 2 --> hidden_size, ReLU, hidden_size --> 1
        self.attention_network = nn.Sequential(
            nn.Linear(hidden_size * 2, hidden_size),
```

```
            nn.ReLU(),
            nn.Linear(hidden_size, 1),
        )

        self.softmax = nn.Softmax(dim=1)

    def forward(self, queries, keys, values):
        """The forward pass of the additive attention mechanism.

        Arguments:
            queries: The current decoder hidden state. (batch_size x
hidden_size)
            keys: The encoder hidden states for each step of the input
sequence. (batch_size x seq_len x hidden_size)
            values: The encoder hidden states for each step of the
input sequence. (batch_size x seq_len x hidden_size)

        Returns:
            context: weighted average of the values (batch_size x 1 x
hidden_size)
            attention_weights: Normalized attention weights for each
encoder hidden state. (batch_size x seq_len x 1)

            The attention_weights must be a softmax weighting over the
seq_len annotations.
        """
        batch_size = keys.size(0)
        expanded_queries = queries.view(batch_size, -1,
self.hidden_size).expand_as(
            keys
        )
        concat_inputs = torch.cat([expanded_queries, keys], dim=2)
        unnormalized_attention = self.attention_network(concat_inputs)
        attention_weights = self.softmax(unnormalized_attention)
        context = torch.bmm(attention_weights.transpose(2, 1), values)
        return context, attention_weights
```

### Step 2: RNN + additive attention

In the next cell, a modification of our `RNNDecoder` that makes use of an additive attention mechanism as been implemented for your. Please take a momement to read through it and understand what it is doing. See the assignment handouts for details.

```
class RNNAttentionDecoder(nn.Module):
    def __init__(self, vocab_size, hidden_size,
attention_type="scaled_dot"):
        super(RNNAttentionDecoder, self).__init__()
        self.vocab_size = vocab_size
        self.hidden_size = hidden_size
```

```
        self.embedding = nn.Embedding(vocab_size, hidden_size)

        self.rnn = MyGRUCell(input_size=hidden_size * 2,
hidden_size=hidden_size)
        if attention_type == "additive":
            self.attention =
AdditiveAttention(hidden_size=hidden_size)
        elif attention_type == "scaled_dot":
            self.attention =
ScaledDotAttention(hidden_size=hidden_size)

        self.out = nn.Linear(hidden_size, vocab_size)

    def forward(self, inputs, annotations, hidden_init):
        """Forward pass of the attention-based decoder RNN.

        Arguments:
            inputs: Input token indexes across a batch for all the
time step. (batch_size x decoder_seq_len)
            annotations: The encoder hidden states for each step of
the input.
                         sequence. (batch_size x seq_len x
hidden_size)
            hidden_init: The final hidden states from the encoder,
across a batch. (batch_size x hidden_size)

        Returns:
            output: Un-normalized scores for each token in the
vocabulary, across a batch for all the decoding time steps.
(batch_size x decoder_seq_len x vocab_size)
            attentions: The stacked attention weights applied to the
encoder annotations (batch_size x encoder_seq_len x decoder_seq_len)
        """

        batch_size, seq_len = inputs.size()
        embed = self.embedding(inputs)  # batch_size x seq_len x
hidden_size

        hiddens = []
        attentions = []
        h_prev = hidden_init

        for i in range(seq_len):
            embed_current = embed[
                :, i, :
            ]  # Get the current time step, across the whole batch
            context, attention_weights = self.attention(
                h_prev, annotations, annotations
            )  # batch_size x 1 x hidden_size
```

```
            embed_and_context = torch.cat(
                [embed_current, context.squeeze(1)], dim=1
            )  # batch_size x (2*hidden_size)
            h_prev = self.rnn(embed_and_context, h_prev)  # batch_size x hidden_size

            hiddens.append(h_prev)
            attentions.append(attention_weights)

        hiddens = torch.stack(hiddens, dim=1)  # batch_size x seq_len x hidden_size
        attentions = torch.cat(attentions, dim=2)  # batch_size x seq_len x seq_len

        output = self.out(hiddens)  # batch_size x seq_len x vocab_size
        return output, attentions
```

## Step 3: Training and analysis (with additive attention)

Now, run the following cell to train our recurrent encoder-decoder model with additive attention. How does it perform compared to the recurrent encoder-decoder model without attention?

```
TEST_SENTENCE = "the air conditioning is working"

rnn_attn_args = AttrDict()
args_dict = {
    "data_file_name": "pig_latin_small",
    "cuda": True,
    "nepochs": 50,
    "checkpoint_dir": "checkpoints",
    "learning_rate": 0.005,
    "lr_decay": 0.99,
    "early_stopping_patience": 10,
    "batch_size": 64,
    "hidden_size": 64,
    "encoder_type": "rnn",            # options: rnn / transformer
    "decoder_type": "rnn_attention",  # options: rnn / rnn_attention / transformer
    "attention_type": "additive",     # options: additive / scaled_dot
}
rnn_attn_args.update(args_dict)

print_opts(rnn_attn_args)
rnn_attn_encoder, rnn_attn_decoder, rnn_attn_losses = train(rnn_attn_args)

translated = translate_sentence(
    TEST_SENTENCE, rnn_attn_encoder, rnn_attn_decoder, None,
```

```
rnn_attn_args
)
print("source:\t\t{} \ntranslated:\t{}".format(TEST_SENTENCE,
translated))

======================================================================
==========
                                      Opts

----------------------------------------------------------------------
----------
                         data_file_name: pig_latin_small

                                   cuda: 1

                                nepochs: 50

                         checkpoint_dir: checkpoints

                          learning_rate: 0.005

                               lr_decay: 0.99

                early_stopping_patience: 10

                             batch_size: 64

                            hidden_size: 64

                           encoder_type: rnn

                           decoder_type: rnn_attention

                         attention_type: additive

======================================================================
==========
======================================================================
==========
                                   Data Stats

----------------------------------------------------------------------
----------
('piqued', 'iquedpay')
('learnt', 'earntlay')
('acquainted', 'acquaintedway')
('terms', 'ermstay')
('steady', 'eadystay')
Num unique word pairs: 3198
```

```
Vocabulary: dict_keys(['-', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h',
'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v',
'w', 'x', 'y', 'z', 'SOS', 'EOS'])
Vocab size: 29
======================================================================
==========
Moved models to GPU!
Epoch:   0 | Train loss: 1.956 | Val loss: 1.843 | Gen: otay-odgay-
oday-oday illlay-illlay-illlay intinday-intinday-in isssay-ilssay-
ilssay ingway-ingway-ingway
Epoch:   1 | Train loss: 1.429 | Val loss: 1.589 | Gen: ethay arway
ondingstingstay-ings isssay-isssay-isssay orongway-ingway-ingw
Epoch:   2 | Train loss: 1.135 | Val loss: 1.416 | Gen: athehtay arway
ondincationday issay oongringray
Epoch:   3 | Train loss: 0.958 | Val loss: 1.350 | Gen: elfay-uethay
away ondincingnincingnay isway ovedingway
Epoch:   4 | Train loss: 0.767 | Val loss: 1.092 | Gen: ethay-uenthay-
ehthay arirway onditingnay isway orfingngnay
Epoch:   5 | Train loss: 0.631 | Val loss: 1.028 | Gen: ethay ariway
ondincingingingingin isway orfingnay
Epoch:   6 | Train loss: 0.544 | Val loss: 1.125 | Gen: ethay airway
omitiondingsay-ition isway orfingsay-ingsay-ing
Epoch:   7 | Train loss: 0.482 | Val loss: 0.934 | Gen: eteway arway
onditingncay isway orefingnay
Epoch:   8 | Train loss: 0.398 | Val loss: 0.837 | Gen: ethay airway
onditingingcay isway orfingningway
Epoch:   9 | Train loss: 0.366 | Val loss: 0.930 | Gen: ethay away
ondcay isway orkingnay
Epoch:  10 | Train loss: 0.317 | Val loss: 0.862 | Gen: eway airway
onditionitionday isway orfingway
Epoch:  11 | Train loss: 0.240 | Val loss: 0.620 | Gen: ethay airway
onditionday isway orkingnay
Epoch:  12 | Train loss: 0.179 | Val loss: 0.540 | Gen: ethtay airway
onditiningcay isway orkingway
Epoch:  13 | Train loss: 0.168 | Val loss: 0.664 | Gen: ehtay away
onditingway iway orfingway
Epoch:  14 | Train loss: 0.171 | Val loss: 0.594 | Gen: ethay airway
onditiongcay isway orfingway
Epoch:  15 | Train loss: 0.150 | Val loss: 0.583 | Gen: ethay airway
onditionminingcay isway orkingway
Epoch:  16 | Train loss: 0.117 | Val loss: 0.443 | Gen: ethay airway
onditiongcay isway orkingway
Epoch:  17 | Train loss: 0.080 | Val loss: 0.402 | Gen: ethay airway
onditiondcay isway orkingway
Epoch:  18 | Train loss: 0.062 | Val loss: 0.390 | Gen: ehthay airway
onditionicaningway isway orkingway
Epoch:  19 | Train loss: 0.046 | Val loss: 0.422 | Gen: ethay airway
onditionday isway orkingway
Epoch:  20 | Train loss: 0.041 | Val loss: 0.358 | Gen: ethay airway
onditiongway way orkingway
```

```
Epoch:  21 | Train loss: 0.035 | Val loss: 0.404 | Gen: ethay airway
onditiongcay isway orkingway
Epoch:  22 | Train loss: 0.037 | Val loss: 0.361 | Gen: ethay airway
onditioncay isway orkingway
Epoch:  23 | Train loss: 0.024 | Val loss: 0.373 | Gen: ethay airway
onditioningcay isway orkingway
Epoch:  24 | Train loss: 0.016 | Val loss: 0.326 | Gen: ethay airway
onditionicningcay isway orkingway
Epoch:  25 | Train loss: 0.019 | Val loss: 0.433 | Gen: ethay airway
onditioncay isway orkingway
Epoch:  26 | Train loss: 0.038 | Val loss: 0.447 | Gen: ethay airway
onditiniingcay isway orkingway
Epoch:  27 | Train loss: 0.064 | Val loss: 0.664 | Gen: ethay airway
onditingcay isway orkingway
Epoch:  28 | Train loss: 0.290 | Val loss: 0.904 | Gen: ekthay-outhay
airwway onditicininingcay isway onkingway-ingway
Epoch:  29 | Train loss: 0.264 | Val loss: 0.705 | Gen: ethay arway
onitingcay issay orfingugway
Epoch:  30 | Train loss: 0.186 | Val loss: 0.554 | Gen: ethay airway
onitiningcay isway orkingingway
Epoch:  31 | Train loss: 0.099 | Val loss: 0.444 | Gen: ethay airway
onditinicingcay isway orfingway
Epoch:  32 | Train loss: 0.065 | Val loss: 0.492 | Gen: ethay airway
onditioncay isway orfingway
Epoch:  33 | Train loss: 0.067 | Val loss: 0.421 | Gen: ethay airway
onditioncay isway orkingway
Epoch:  34 | Train loss: 0.036 | Val loss: 0.389 | Gen: ethay airway
onditioncay isway orkingway
Validation loss has not improved in 10 epochs, stopping early
Obtained lowest validation loss of: 0.32645468270549405
source:         the air conditioning is working
translated:     ethay airway onditiningcay isway orkingway
```

```
TEST_SENTENCE = "the air conditioning is working"
translated = translate_sentence(
    TEST_SENTENCE, rnn_attn_encoder, rnn_attn_decoder, None,
rnn_attn_args
)
print("source:\t\t{} \ntranslated:\t{}".format(TEST_SENTENCE,
translated))
```

```
source:         the air conditioning is working
translated:     ethay airway onditiningcay isway orkingway
```

### Step 4: Implement scaled dot-product attention

In the next cell, you will implement the scaled dot-product attention mechanism. See the assignment handouts for details.

```
class ScaledDotAttention(nn.Module):
    def __init__(self, hidden_size):
```

```
        super(ScaledDotAttention, self).__init__()

        self.hidden_size = hidden_size

        self.Q = nn.Linear(hidden_size, hidden_size)
        self.K = nn.Linear(hidden_size, hidden_size)
        self.V = nn.Linear(hidden_size, hidden_size)
        self.softmax = nn.Softmax(dim=1)
        self.scaling_factor = torch.rsqrt(
            torch.tensor(self.hidden_size, dtype=torch.float)
        )

    def forward(self, queries, keys, values):
        """The forward pass of the scaled dot attention mechanism.

        Arguments:
            queries: The current decoder hidden state, 2D or 3D
tensor. (batch_size x (k) x hidden_size)
            keys: The encoder hidden states for each step of the input
sequence. (batch_size x seq_len x hidden_size)
            values: The encoder hidden states for each step of the
input sequence. (batch_size x seq_len x hidden_size)

        Returns:
            context: weighted average of the values (batch_size x k x
hidden_size)
            attention_weights: Normalized attention weights for each
encoder hidden state. (batch_size x seq_len x k)

            The output must be a softmax weighting over the seq_len
annotations.
        """

        # ------------
        # FILL THIS IN
        # ------------

        if len(queries.size()) == 2:
          queries = queries.unsqueeze(1)
          queries = queries.permute(0, 2, 1) # (batch_size, k,
hidden_size)

        q = self.Q(queries).permute(0, 2, 1) # (batch_size,
hidden_size, k)
        k = self.K(keys) # (batch_size, seq_len, hidden_size)
        v = self.V(values)

        # (batch_size x seq_len x k) = (batch_size x seq_len x
hidden_size) @ (batch_size x hidden_size x k)
```

```
        unnormalized_attention = torch.bmm(k * self.scaling_factor, q)
        attention_weights = self.softmax(unnormalized_attention)

        # (batch_size x k x hidden_size) = (batch_size x k x seq_len)
@ (batch_size x seq_len x hidden_size)
        context = torch.bmm(attention_weights.permute(0, 2, 1), v)
        return context, attention_weights
```

### Step 5: Implement causal dot-product Attention

Now, implement the casual scaled dot-product attention mechanism. It will be very similar to your implementation for `ScaledDotAttention`. The additional step is to mask out the attention to future timesteps so this attention mechanism can be used in a decoder. See the assignment handouts for details.

```
class CausalScaledDotAttention(nn.Module):
    def __init__(self, hidden_size):
        super(CausalScaledDotAttention, self).__init__()

        self.hidden_size = hidden_size
        self.neg_inf = torch.tensor(-1e7)

        self.Q = nn.Linear(hidden_size, hidden_size)
        self.K = nn.Linear(hidden_size, hidden_size)
        self.V = nn.Linear(hidden_size, hidden_size)
        self.softmax = nn.Softmax(dim=1)
        self.scaling_factor = torch.rsqrt(
            torch.tensor(self.hidden_size, dtype=torch.float)
        )

    def forward(self, queries, keys, values):
        """The forward pass of the scaled dot attention mechanism.

        Arguments:
            queries: The current decoder hidden state, 2D or 3D
tensor. (batch_size x (k) x hidden_size)
            keys: The encoder hidden states for each step of the input
sequence. (batch_size x seq_len x hidden_size)
            values: The encoder hidden states for each step of the
input sequence. (batch_size x seq_len x hidden_size)

        Returns:
            context: weighted average of the values (batch_size x k x
hidden_size)
            attention_weights: Normalized attention weights for each
encoder hidden state. (batch_size x seq_len x k)

            The output must be a softmax weighting over the seq_len
annotations.
        """
```

```
        # ------------
        # FILL THIS IN
        # ------------

        if len(queries.size()) == 2:
          queries = queries.unsqueeze(1)
          queries = queries.permute(0, 2, 1) # (batch_size, k,
hidden_size)

        q = self.Q(queries).permute(0, 2, 1) # (batch_size,
hidden_size, k)
        k = self.K(keys) # (batch_size, seq_len, hidden_size)
        v = self.V(values)

        # (batch_size x seq_len x k) = (batch_size x seq_len x
hidden_size) @ (batch_size x hidden_size x k)
        unnormalized_attention = torch.bmm(k * self.scaling_factor, q)

        mask = torch.ones((unnormalized_attention.size()[0],
unnormalized_attention.size()[-1], unnormalized_attention.size()[-2]),
device=torch.device('cuda:0'))
        mask = torch.tril(mask * self.neg_inf)

        attention_weights =
self.softmax(torch.bmm(unnormalized_attention, mask))
        context = torch.bmm(attention_weights.permute(0, 2, 1), v)
        return context, attention_weights
```

### Step 6: Attention encoder and decoder

The following cells provide an implementation of an encoder and decoder that use a single `ScaledDotAttention` block. Please read through them to understand what they are doing.

```
class AttentionEncoder(nn.Module):
    def __init__(self, vocab_size, hidden_size, opts):
        super(AttentionEncoder, self).__init__()

        self.vocab_size = vocab_size
        self.hidden_size = hidden_size
        self.opts = opts

        self.embedding = nn.Embedding(vocab_size, hidden_size)

        self.self_attention = ScaledDotAttention(
                    hidden_size=hidden_size,
                )

        self.attention_mlp = nn.Sequential(
```

```
                                nn.Linear(hidden_size, hidden_size),
                                nn.ReLU(),
                              )

    def forward(self, inputs):
        """Forward pass of the encoder scaled dot attention.

        Arguments:
            inputs: Input token indexes across a batch for all time
steps in the sequence. (batch_size x seq_len)

        Returns:
            annotations: The hidden states computed at each step of
the input sequence. (batch_size x seq_len x hidden_size)
            None: Used to conform to standard encoder return
signature.
        """
        batch_size, seq_len = inputs.size()

        encoded = self.embedding(inputs)  # batch_size x seq_len x
hidden_size

        annotations = encoded
        new_annotations, self_attention_weights = self.self_attention(
            annotations, annotations, annotations
        )  # batch_size x seq_len x hidden_size
        residual_annotations = annotations + new_annotations
        new_annotations = self.attention_mlp(residual_annotations)
        annotations = residual_annotations + new_annotations

        return annotations, None

class AttentionDecoder(nn.Module):
    def __init__(self, vocab_size, hidden_size):
        super(AttentionDecoder, self).__init__()
        self.vocab_size = vocab_size
        self.hidden_size = hidden_size

        self.embedding = nn.Embedding(vocab_size, hidden_size)

        self.self_attention = CausalScaledDotAttention(
                                hidden_size=hidden_size,
                                )

        self.decoder_attention = ScaledDotAttention(
                                  hidden_size=hidden_size,
                                  )

        self.attention_mlp = nn.Sequential(
                                nn.Linear(hidden_size, hidden_size),
```

```
                                nn.ReLU(),
                              )

        self.out = nn.Linear(hidden_size, vocab_size)


    def forward(self, inputs, annotations, hidden_init):
        """Forward pass of the attention-based decoder RNN.

        Arguments:
            inputs: Input token indexes across a batch for all the
time step. (batch_size x decoder_seq_len)
            annotations: The encoder hidden states for each step of
the input.
                         sequence. (batch_size x seq_len x
hidden_size)
            hidden_init: Not used in the transformer decoder
        Returns:
            output: Un-normalized scores for each token in the
vocabulary, across a batch for all the decoding time steps.
(batch_size x decoder_seq_len x vocab_size)
            attentions: The stacked attention weights applied to the
encoder annotations (batch_size x encoder_seq_len x decoder_seq_len)
        """

        batch_size, seq_len = inputs.size()
        embed = self.embedding(inputs)  # batch_size x seq_len x
hidden_size

        encoder_attention_weights_list = []
        self_attention_weights_list = []
        contexts = embed
        new_contexts, self_attention_weights = self.self_attention(
            contexts, contexts, contexts
        )  # batch_size x seq_len x hidden_size
        residual_contexts = contexts + new_contexts
        new_contexts, encoder_attention_weights =
self.decoder_attention(
            residual_contexts, annotations, annotations
        )  # batch_size x seq_len x hidden_size
        residual_contexts = residual_contexts + new_contexts
        new_contexts = self.attention_mlp(residual_contexts)
        contexts = residual_contexts + new_contexts


encoder_attention_weights_list.append(encoder_attention_weights)
        self_attention_weights_list.append(self_attention_weights)

        output = self.out(contexts)
```

```
        encoder_attention_weights =
torch.stack(encoder_attention_weights_list)
        self_attention_weights =
torch.stack(self_attention_weights_list)

        return output, (encoder_attention_weights,
self_attention_weights)
```

### Step 7: Training and analysis (single scaled dot-product attention block)

Now, train the following model, with an encoder and decoder each composed a single `ScaledDotAttention` block.

```
TEST_SENTENCE = "the air conditioning is working"

attention_args_s = AttrDict()
args_dict = {
    "data_file_name": "pig_latin_small",
    "cuda": True,
    "nepochs": 100,
    "checkpoint_dir": "checkpoints",
    "learning_rate": 5e-4,
    "early_stopping_patience": 100,
    "lr_decay": 0.99,
    "batch_size": 64,
    "hidden_size": 32,
    "encoder_type": "attention",
    "decoder_type": "attention",  # options: rnn / rnn_attention /
attention / transformer
}
attention_args_s.update(args_dict)
print_opts(attention_args_s)

attention_encoder_s, attention_decoder_s, attention_losses_s =
train(attention_args_s)

translated = translate_sentence(
    TEST_SENTENCE, attention_encoder_s, attention_decoder_s, None,
attention_args_s
)
print("source:\t\t{} \ntranslated:\t{}".format(TEST_SENTENCE,
translated))
```

```
======================================================================
==========
                                      Opts

----------------------------------------------------------------------
----------
                         data_file_name: pig_latin_small
```

```
                    cuda: 1

                 nepochs: 100

          checkpoint_dir: checkpoints

           learning_rate: 0.0005

 early_stopping_patience: 100

                lr_decay: 0.99

              batch_size: 64

             hidden_size: 32

            encoder_type: attention

            decoder_type: attention

======================================================================
==========
======================================================================
==========
                                 Data Stats

----------------------------------------------------------------------
----------
('fully', 'ullyfay')
('determining', 'eterminingday')
('confounded', 'onfoundedcay')
('attempting', 'attemptingway')
('darling', 'arlingday')
Num unique word pairs: 3198
Vocabulary: dict_keys(['-', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h',
'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v',
'w', 'x', 'y', 'z', 'SOS', 'EOS'])
Vocab size: 29
======================================================================
==========
Moved models to GPU!
Epoch:   0 | Train loss: 3.047 | Val loss: 2.526 | Gen: ay ay ay ay ay

Epoch:   1 | Train loss: 2.329 | Val loss: 2.239 | Gen:
ayayayayayayayayayayay ay ay ay ay
Epoch:   2 | Train loss: 2.123 | Val loss: 2.099 | Gen:
ayayayayayayayayayayay ay iny ay ongayy
Epoch:   3 | Train loss: 1.975 | Val loss: 2.007 | Gen: eaay ay
```

```
onsiniiniiniiniiniin ay ongayy
Epoch:   4 | Train loss: 1.877 | Val loss: 1.924 | Gen: ay ay
onsnnnnnnnnnnnnnnnnn ay ongmoEOSooooononooiyoo
Epoch:   5 | Train loss: 1.800 | Val loss: 1.871 | Gen: ay ay
onsnnnnnnnnnnnnnnnnn ay ongrwayoogoEOSEOSynnmmoo
Epoch:   6 | Train loss: 1.751 | Val loss: 1.855 | Gen: ay ay
onsnnnnnnnnnnnnnnnnn ay ongongongongongongon
Epoch:   7 | Train loss: 1.719 | Val loss: 1.811 | Gen: ay ay
onsnnnnnnnnnnnnnnnnn ay oorray
Epoch:   8 | Train loss: 1.653 | Val loss: 1.762 | Gen: ay ay
ongnnnnnnnnnnnnnnnnn issssssssssssssssss oorray
Epoch:   9 | Train loss: 1.615 | Val loss: 1.712 | Gen: ay ay
ongingingingingingon issssssssssssssssss oorray
Epoch:  10 | Train loss: 1.565 | Val loss: 1.670 | Gen: ay ay
ongioggiogiogiogiogo issssssssssssssssss oorray
Epoch:  11 | Train loss: 1.526 | Val loss: 1.663 | Gen: ay ay
ongingongongongongon issssssssssssssssss ooorwEOSy
Epoch:  12 | Train loss: 1.504 | Val loss: 1.642 | Gen: ay
arararararararararar ongingongongongongon issssssssssssssssss
ooornEOSy
Epoch:  13 | Train loss: 1.475 | Val loss: 1.634 | Gen: ay
arararararararararar ongingongongongongon issssssssssssssssss oorray
Epoch:  14 | Train loss: 1.453 | Val loss: 1.616 | Gen: ay
arararararararararar onginEOSinnnnnnnnnnnnn issssssssssssssssss
iraryy
Epoch:  15 | Train loss: 1.455 | Val loss: 1.604 | Gen: ay
arararararararararar ongingingingingingin isisisisisisisisisis inayyy
Epoch:  16 | Train loss: 1.419 | Val loss: 1.576 | Gen: ay
arararararararararar ongingingingingingin isisisisisisisisisis inayyy
Epoch:  17 | Train loss: 1.416 | Val loss: 1.567 | Gen: ay
arararararararararar ongingingingingingin iswsy inayyy
Epoch:  18 | Train loss: 1.397 | Val loss: 1.554 | Gen: ay
arararararararararar ongingingingingingin iswsy inayyy
Epoch:  19 | Train loss: 1.406 | Val loss: 1.582 | Gen: ay
arararararararararar ongingingingingingin issssssssssssssssss irayay
Epoch:  20 | Train loss: 1.387 | Val loss: 1.554 | Gen: ay
arwarwarwarwarwarwar ongwngiay isisisisisisisisisis irgnyyy
Epoch:  21 | Train loss: 1.373 | Val loss: 1.542 | Gen: ay
arararararararararar ingwnyiny isisisisisisisisisis inayyy
Epoch:  22 | Train loss: 1.358 | Val loss: 1.525 | Gen: ay
arararararararararar ongingiay isisisisisisisisisis inayyy
Epoch:  23 | Train loss: 1.339 | Val loss: 1.504 | Gen: ay
arararararararararar ongwnywngwnywny isisisisisisisisisis inayyy
Epoch:  24 | Train loss: 1.333 | Val loss: 1.531 | Gen: ay
arwarwarwarwarwarwar ingwnywngwnywny isisisisisisisisisis inayyy
Epoch:  25 | Train loss: 1.340 | Val loss: 1.510 | Gen: ay
arwarwarwarwarwarwar ingiay isisisisisisisisisis irayay
Epoch:  26 | Train loss: 1.314 | Val loss: 1.484 | Gen: ay
arwarwarwarwarwarwar ingiay isisisisisisisisisis irrr
Epoch:  27 | Train loss: 1.285 | Val loss: 1.470 | Gen: ay
```

```
arararararararararar ingingingingingingin isisisisisisisisisis irayay
Epoch:  28 | Train loss: 1.279 | Val loss: 1.476 | Gen: ay
arwarwarwarwarwarwar ingiay isisisisisisisisisis irayay
Epoch:  29 | Train loss: 1.262 | Val loss: 1.472 | Gen: ay
arwarwarwarwarwarwar indiay isisisisisisisisisis irayay
Epoch:  30 | Train loss: 1.266 | Val loss: 1.485 | Gen: ay ay
ingingingingingingyinginyww iswsy irayay
Epoch:  31 | Train loss: 1.265 | Val loss: 1.465 | Gen: ay
arwarwarwarwarwarwar ingwwywwyiwywny iswsy irayay
Epoch:  32 | Train loss: 1.250 | Val loss: 1.472 | Gen: ay
arwarwarwarwarwarwar ingingingingingingww isisisisisisisisisis irayay
Epoch:  33 | Train loss: 1.248 | Val loss: 1.450 | Gen: ay
arwarwarwarwarwarwar ingyagEOSngwngEOSngy isisisisisisisisisis irayay
Epoch:  34 | Train loss: 1.243 | Val loss: 1.461 | Gen: ay ay ingway
isisisisisisisisisis irayay
Epoch:  35 | Train loss: 1.245 | Val loss: 1.492 | Gen: ay ay ingway
isisisisisisisisisis orayay
Epoch:  36 | Train loss: 1.268 | Val loss: 1.482 | Gen: ay ay ingway
iswsy irayay
Epoch:  37 | Train loss: 1.231 | Val loss: 1.472 | Gen: ay
arwarwarwarwarwarwar ingway isisisisisisisisisis irayay
Epoch:  38 | Train loss: 1.237 | Val loss: 1.482 | Gen: ay ay ingway
isisisisisisisisisis irayay
Epoch:  39 | Train loss: 1.229 | Val loss: 1.469 | Gen: ay
arwarwarwarwarwarwar ingway issssssssssssssssss irayay
Epoch:  40 | Train loss: 1.231 | Val loss: 1.448 | Gen: ay
arwarwarwarwarwarwar ingway isisisisisisisisisis irayay
Epoch:  41 | Train loss: 1.238 | Val loss: 1.460 | Gen: ay ay ingway
iswsy irayay
Epoch:  42 | Train loss: 1.238 | Val loss: 1.472 | Gen: ay ay ongway
iswsy iraray
Epoch:  43 | Train loss: 1.214 | Val loss: 1.454 | Gen: ay
arwarwarwarwarwarwar ongway iswsy ingr
Epoch:  44 | Train loss: 1.206 | Val loss: 1.449 | Gen: ay
arwarwarwarwarwarwar ongway iswsy orgway
Epoch:  45 | Train loss: 1.246 | Val loss: 1.447 | Gen: ethayEOSety ay
ongway isway owgway
Epoch:  46 | Train loss: 1.197 | Val loss: 1.438 | Gen: etayyEOSey
arwarwarwarwarwarwar ongway isway owgway
Epoch:  47 | Train loss: 1.177 | Val loss: 1.434 | Gen: ethy
aywaywayEOSarEOSayEOSarEOSar ongway isisisisisisisisisis owgway
Epoch:  48 | Train loss: 1.187 | Val loss: 1.432 | Gen: etayyEOSey
arwarwarwarwarwarwar ongway isisisisisisisisisis owgway
Epoch:  49 | Train loss: 1.187 | Val loss: 1.436 | Gen: ethay
arwarwarwarwarwarwar ongway isisisisisisisisisis orarEOSy
Epoch:  50 | Train loss: 1.242 | Val loss: 1.476 | Gen: ethayy arEOSay
ongway isisisisisisisisisis orarEOSy
Epoch:  51 | Train loss: 1.273 | Val loss: 1.483 | Gen: ethayy
arwarwarwarwarwarwar ongway issssssssssssssssss orayEOSy
Epoch:  52 | Train loss: 1.269 | Val loss: 1.470 | Gen: ethayy aaEOSay
```

```
ingwayy isisisisisisisisisis iwaaynyy
Epoch:  53 | Train loss: 1.254 | Val loss: 1.464 | Gen:
ethwhwhwhwhwhwhwhwhh aaEOSay ingwayy isssssssssssssssss iwaaynyy
Epoch:  54 | Train loss: 1.252 | Val loss: 1.466 | Gen:
ethwhwhwhwhwhwhwhwhh aaEOSay ongwaayEOSayyy isssssssssssssssss
inayayy
Epoch:  55 | Train loss: 1.246 | Val loss: 1.461 | Gen:
ethwhwhwhwhwhwhwhwhh iwarwarwwawaawaarwar inggway isssssssssssssssss
owgway
Epoch:  56 | Train loss: 1.223 | Val loss: 1.458 | Gen:
ethwhwhwhwhwhwhwhwhh arwarwarwarwarwarwar inggway isssssssssssssssss
owgway
Epoch:  57 | Train loss: 1.220 | Val loss: 1.463 | Gen:
ethwhwhwhwhwhwhwhwhh arwarwarwarwarwarwar inggway isssssssssssssssss
owgway
Epoch:  58 | Train loss: 1.221 | Val loss: 1.461 | Gen:
ethwhwhwhwhwhwhwhwhh arwarwarwarwarwarwar ingwway isssssssssssssssss
owgway
Epoch:  59 | Train loss: 1.214 | Val loss: 1.451 | Gen:
ethwhwhwhwhwhwhwhwhh arwarwarwarwarwarwar ingwway isssssssssssssssss
owgway
Epoch:  60 | Train loss: 1.203 | Val loss: 1.457 | Gen:
ethwhwhwhwhwhwhwhwhh arwarwarwarwarwarwar ingwway isssssssssssssssss
iwaray
Epoch:  61 | Train loss: 1.202 | Val loss: 1.448 | Gen:
ethwhwhwhwhwhwhwhwhh arwarwarwarwarwarwar ingwway isssssssssssssssss
owgway
Epoch:  62 | Train loss: 1.192 | Val loss: 1.444 | Gen:
ethwhwhwhwhwhwhwhwhh ayEOSay inggway isssssssssssssssss owgway
Epoch:  63 | Train loss: 1.187 | Val loss: 1.434 | Gen:
ethwhwhwhwhwhwhwhwhh arwarwarwarwarwarwar inggway isssssssssssssssss
owgway
Epoch:  64 | Train loss: 1.166 | Val loss: 1.410 | Gen: ethwy
arwirwrrirrirrrrirr ingwway isisisisisisisisisis owgway
Epoch:  65 | Train loss: 1.146 | Val loss: 1.388 | Gen: ethy
arEOSarEOSarEOSarEOSarEOSarEOSar ingwway isway owgway
Epoch:  66 | Train loss: 1.134 | Val loss: 1.385 | Gen: ethy
arrrrrrrrrrrrrrrrrr ingwway isway owgway
Epoch:  67 | Train loss: 1.135 | Val loss: 1.378 | Gen: ethy ay
ingwway isway owgway
Epoch:  68 | Train loss: 1.140 | Val loss: 1.390 | Gen: ethy ay
ingwway isway owgway
Epoch:  69 | Train loss: 1.128 | Val loss: 1.391 | Gen: ethy ay
ingwway isway owgway
Epoch:  70 | Train loss: 1.166 | Val loss: 1.417 | Gen: ethy ay
ingwway isway owgway
Epoch:  71 | Train loss: 1.163 | Val loss: 1.413 | Gen: ethy
arrrrrrrrrrrrrrrrrr ingwway isway owgway
Epoch:  72 | Train loss: 1.162 | Val loss: 1.422 | Gen: ethy ay
ingwayy isway owgry
```

```
Epoch:  73 | Train loss: 1.155 | Val loss: 1.408 | Gen: ethy
arrrrrrrrrrrrrrrrrrr ingwayy isway owgry
Epoch:  74 | Train loss: 1.134 | Val loss: 1.411 | Gen: ethy ay ingayy
isway ingry
Epoch:  75 | Train loss: 1.145 | Val loss: 1.418 | Gen: ethy
arrrrrrrrrrrrrrrrrrr ingayy isway ingry
Epoch:  76 | Train loss: 1.144 | Val loss: 1.407 | Gen: ethy
arrrrrrrrrrrrrrrrrrr ingayy isway ingry
Epoch:  77 | Train loss: 1.123 | Val loss: 1.384 | Gen: ethy
arrrrrrrrrrrrrrrrrrr ingayy isway ingry
Epoch:  78 | Train loss: 1.103 | Val loss: 1.373 | Gen: ethy
arrrrrrrrrrrrrrrrrrr ingwayEOSngyy isssssssssssssssssss ingry
Epoch:  79 | Train loss: 1.110 | Val loss: 1.405 | Gen: ethy ay ingayy
isssssssssssssssssss ingry
Epoch:  80 | Train loss: 1.133 | Val loss: 1.372 | Gen: ethy
arrrrrrrrrrrrrrrrrrr ingwayEOSngyy isssssssssssssssssss iogwwy
Epoch:  81 | Train loss: 1.101 | Val loss: 1.374 | Gen: ethy
arEOSaraaraaraararywyw ingwayEOSngyy isssssssssssssssssss ingwagy
Epoch:  82 | Train loss: 1.092 | Val loss: 1.361 | Gen: ethy
arEOSaraaraaraararywyw ingwayEOSngyy isway ingwagy
Epoch:  83 | Train loss: 1.083 | Val loss: 1.363 | Gen: ethy
arEOSaraaraaraararywyw ingwayEOSngyyyy isway ingwaay
Epoch:  84 | Train loss: 1.077 | Val loss: 1.361 | Gen: ethy
arrrrrrrrrrrrrrrrrrr ingwayEOSngyyyy isway ingwaay
Epoch:  85 | Train loss: 1.078 | Val loss: 1.360 | Gen: ethy
arEOSaraaraaraararywyw ingwayEOSngyyyy isway ingwagy
Epoch:  86 | Train loss: 1.071 | Val loss: 1.350 | Gen: ethy
arEOSaraaraaraararywyw ingwayEOSngyyyy isway ingwagy
Epoch:  87 | Train loss: 1.067 | Val loss: 1.347 | Gen: ethy
arrrrrrrrrrrrrrrrrrr ingwayEOSngyy isway ingrwgy
Epoch:  88 | Train loss: 1.069 | Val loss: 1.355 | Gen: ethy
arrrrrrrrrrrrrrrrrrr ingwayEOSngyyyy isway ingrnay
Epoch:  89 | Train loss: 1.066 | Val loss: 1.351 | Gen: ethy
arrrrrrrrrrrrrrrrrrr inddyy isway ingrnay
Epoch:  90 | Train loss: 1.066 | Val loss: 1.349 | Gen: ay
aywaywayiyywyywyywww ingwayEOSngy isway ingray
Epoch:  91 | Train loss: 1.061 | Val loss: 1.342 | Gen: ay
ayEOSayaiywiywyywyywww ingwayEOSngy isway ingray
Epoch:  92 | Train loss: 1.057 | Val loss: 1.338 | Gen: ay
ayEOSayaayaayaayayywyw ingwayEOSngy isway ingray
Epoch:  93 | Train loss: 1.057 | Val loss: 1.331 | Gen: ay
arrrrrrrrrrrrrrrrrrr ingwayEOSngy isway ingray
Epoch:  94 | Train loss: 1.066 | Val loss: 1.337 | Gen: ethy
arEOSaraaraayaayayywyw ongwayEOSngy isway ingray
Epoch:  95 | Train loss: 1.051 | Val loss: 1.324 | Gen: ethy
arEOSaraayaayaayayywyw ingwayEOSngy isway ingray
Epoch:  96 | Train loss: 1.038 | Val loss: 1.323 | Gen: ethy
arEOSaraaraaraararywyw ingway isway ingray
Epoch:  97 | Train loss: 1.037 | Val loss: 1.329 | Gen: ethy
arEOSaraaraaraararywyw ingway isway ingray
```

```
Epoch:  98 | Train loss: 1.056 | Val loss: 1.326 | Gen: ethy
arrrrrrrrrrrrrrrrrr ingway isway ingray
Epoch:  99 | Train loss: 1.054 | Val loss: 1.358 | Gen: ethy
arEOSaraaraaraararywyw ingway isway ingray
Obtained lowest validation loss of: 1.3230563144426088
source:          the air conditioning is working
translated:      ethy arEOSaraaraaraararywyw ingway isway ingray
```

### Step 8: Transformer encoder and decoder

The following cells provide an implementation of the transformer encoder and decoder that use your `ScaledDotAttention` and `CausalScaledDotAttention`. Please read through them to understand what they are doing.

```
class TransformerEncoder(nn.Module):
    def __init__(self, vocab_size, hidden_size, num_layers, opts):
        super(TransformerEncoder, self).__init__()

        self.vocab_size = vocab_size
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.opts = opts

        self.embedding = nn.Embedding(vocab_size, hidden_size)

        self.self_attentions = nn.ModuleList(
            [
                ScaledDotAttention(
                    hidden_size=hidden_size,
                )
                for i in range(self.num_layers)
            ]
        )
        self.attention_mlps = nn.ModuleList(
            [
                nn.Sequential(
                    nn.Linear(hidden_size, hidden_size),
                    nn.ReLU(),
                )
                for i in range(self.num_layers)
            ]
        )

        self.positional_encodings = self.create_positional_encodings()

    def forward(self, inputs):
        """Forward pass of the encoder RNN.

        Arguments:
            inputs: Input token indexes across a batch for all time
```

```
steps in the sequence. (batch_size x seq_len)

        Returns:
            annotations: The hidden states computed at each step of
the input sequence. (batch_size x seq_len x hidden_size)
            None: Used to conform to standard encoder return
signature.
            None: Used to conform to standard encoder return
signature.
        """
        batch_size, seq_len = inputs.size()

        encoded = self.embedding(inputs)  # batch_size x seq_len x
hidden_size

        # Add positinal embeddings from
self.create_positional_encodings. (a'la
https://arxiv.org/pdf/1706.03762.pdf, section 3.5)
        encoded = encoded + self.positional_encodings[:seq_len]

        annotations = encoded
        for i in range(self.num_layers):
            new_annotations, self_attention_weights =
self.self_attentions[i](
                annotations, annotations, annotations
            )  # batch_size x seq_len x hidden_size
            residual_annotations = annotations + new_annotations
            new_annotations = self.attention_mlps[i]
(residual_annotations)
            annotations = residual_annotations + new_annotations

        # Transformer encoder does not have a last hidden or cell
layer.
        return annotations, None
        # return annotations, None, None
    def create_positional_encodings(self, max_seq_len=1000):
        """Creates positional encodings for the inputs.

        Arguments:
            max_seq_len: a number larger than the maximum string
length we expect to encounter during training

        Returns:
            pos_encodings: (max_seq_len, hidden_dim) Positional
encodings for a sequence with length max_seq_len.
        """
        pos_indices = torch.arange(max_seq_len)[..., None]
        dim_indices = torch.arange(self.hidden_size // 2)[None, ...]
        exponents = (2 * dim_indices).float() / (self.hidden_size)
```

```python
        trig_args = pos_indices / (10000**exponents)
        sin_terms = torch.sin(trig_args)
        cos_terms = torch.cos(trig_args)

        pos_encodings = torch.zeros((max_seq_len, self.hidden_size))
        pos_encodings[:, 0::2] = sin_terms
        pos_encodings[:, 1::2] = cos_terms

        if self.opts.cuda:
            pos_encodings = pos_encodings.cuda()

        return pos_encodings

class TransformerDecoder(nn.Module):
    def __init__(self, vocab_size, hidden_size, num_layers):
        super(TransformerDecoder, self).__init__()
        self.vocab_size = vocab_size
        self.hidden_size = hidden_size

        self.embedding = nn.Embedding(vocab_size, hidden_size)
        self.num_layers = num_layers

        self.self_attentions = nn.ModuleList(
            [
                CausalScaledDotAttention(
                    hidden_size=hidden_size,
                )
                for i in range(self.num_layers)
            ]
        )
        self.encoder_attentions = nn.ModuleList(
            [
                ScaledDotAttention(
                    hidden_size=hidden_size,
                )
                for i in range(self.num_layers)
            ]
        )
        self.attention_mlps = nn.ModuleList(
            [
                nn.Sequential(
                    nn.Linear(hidden_size, hidden_size),
                    nn.ReLU(),
                )
                for i in range(self.num_layers)
            ]
        )
        self.out = nn.Linear(hidden_size, vocab_size)

        self.positional_encodings = self.create_positional_encodings()
```

```
    def forward(self, inputs, annotations, hidden_init):
        """Forward pass of the attention-based decoder RNN.

        Arguments:
            inputs: Input token indexes across a batch for all the
time step. (batch_size x decoder_seq_len)
            annotations: The encoder hidden states for each step of
the input.
                         sequence. (batch_size x seq_len x
hidden_size)
            hidden_init: Not used in the transformer decoder
        Returns:
            output: Un-normalized scores for each token in the
vocabulary, across a batch for all the decoding time steps.
(batch_size x decoder_seq_len x vocab_size)
            attentions: The stacked attention weights applied to the
encoder annotations (batch_size x encoder_seq_len x decoder_seq_len)
        """

        batch_size, seq_len = inputs.size()
        embed = self.embedding(inputs)  # batch_size x seq_len x
hidden_size

        embed = embed + self.positional_encodings[:seq_len]

        encoder_attention_weights_list = []
        self_attention_weights_list = []
        contexts = embed
        for i in range(self.num_layers):
            new_contexts, self_attention_weights =
self.self_attentions[i](
                contexts, contexts, contexts
            )  # batch_size x seq_len x hidden_size
            residual_contexts = contexts + new_contexts
            new_contexts, encoder_attention_weights =
self.encoder_attentions[i](
                residual_contexts, annotations, annotations
            )  # batch_size x seq_len x hidden_size
            residual_contexts = residual_contexts + new_contexts
            new_contexts = self.attention_mlps[i](residual_contexts)
            contexts = residual_contexts + new_contexts


encoder_attention_weights_list.append(encoder_attention_weights)
            self_attention_weights_list.append(self_attention_weights)

        output = self.out(contexts)
        encoder_attention_weights =
```

```
torch.stack(encoder_attention_weights_list)
        self_attention_weights =
torch.stack(self_attention_weights_list)

        return output, (encoder_attention_weights,
self_attention_weights)

    def create_positional_encodings(self, max_seq_len=1000):
        """Creates positional encodings for the inputs.

        Arguments:
            max_seq_len: a number larger than the maximum string
length we expect to encounter during training

        Returns:
            pos_encodings: (max_seq_len, hidden_dim) Positional
encodings for a sequence with length max_seq_len.
        """
        pos_indices = torch.arange(max_seq_len)[..., None]
        dim_indices = torch.arange(self.hidden_size // 2)[None, ...]
        exponents = (2 * dim_indices).float() / (self.hidden_size)
        trig_args = pos_indices / (10000**exponents)
        sin_terms = torch.sin(trig_args)
        cos_terms = torch.cos(trig_args)

        pos_encodings = torch.zeros((max_seq_len, self.hidden_size))
        pos_encodings[:, 0::2] = sin_terms
        pos_encodings[:, 1::2] = cos_terms

        pos_encodings = pos_encodings.cuda()

        return pos_encodings
```

## Step 9: Training and analysis (with scaled dot-product attention)

Now we will train a (simplified) transformer encoder-decoder model.

First, we train our smaller model on the small dataset. Use this model to answer Question 4 in the handout.

```
TEST_SENTENCE = "the air conditioning is working"

trans32_args_s = AttrDict()
args_dict = {
    "data_file_name": "pig_latin_small",
    "cuda": True,
    "nepochs": 100,
    "checkpoint_dir": "checkpoints",
    "learning_rate": 5e-4,
    "early_stopping_patience": 100,
```

```
    "lr_decay": 0.99,
    "batch_size": 64,
    "hidden_size": 32,
    "encoder_type": "transformer",
    "decoder_type": "transformer",  # options: rnn / rnn_attention /
transformer
    "num_transformer_layers": 4,
}
trans32_args_s.update(args_dict)
print_opts(trans32_args_s)

trans32_encoder_s, trans32_decoder_s, trans32_losses_s =
train(trans32_args_s)

translated = translate_sentence(
    TEST_SENTENCE, trans32_encoder_s, trans32_decoder_s, None,
trans32_args_s
)
print("source:\t\t{} \ntranslated:\t{}".format(TEST_SENTENCE,
translated))
```

```
======================================================================
==========
                                      Opts

----------------------------------------------------------------------
----------
                         data_file_name: pig_latin_small

                                   cuda: 1

                                nepochs: 100

                         checkpoint_dir: checkpoints

                          learning_rate: 0.0005

                early_stopping_patience: 100

                               lr_decay: 0.99

                             batch_size: 64

                            hidden_size: 32

                           encoder_type: transformer

                           decoder_type: transformer
```

```
                 num_transformer_layers: 4

======================================================================
==========
======================================================================
==========
                                    Data Stats

----------------------------------------------------------------------
----------
('whisper', 'isperwhay')
('prayers', 'ayerspray')
('sketch', 'etchskay')
('disinclination', 'isinclinationday')
('tranquilize', 'anquilizetray')
Num unique word pairs: 3198
Vocabulary: dict_keys(['-', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h',
'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v',
'w', 'x', 'y', 'z', 'SOS', 'EOS'])
Vocab size: 29
======================================================================
==========
Moved models to GPU!
Epoch:   0 | Train loss: 3.680 | Val loss: 2.709 | Gen: ayayay ay
ayayayy ay ay
Epoch:   1 | Train loss: 2.436 | Val loss: 2.318 | Gen: iyy iwy iy
itititoty if
Epoch:   2 | Train loss: 2.102 | Val loss: 2.074 | Gen: ay iay
itcEOSEOScy irrrrrrrrrrrrwwy iowywwnyyy
Epoch:   3 | Train loss: 1.907 | Val loss: 1.954 | Gen: ay iyy
iaannayay iwiyyyEOSyay ofofgwwwwy
Epoch:   4 | Train loss: 1.781 | Val loss: 1.868 | Gen: ay iay aanyyyy
iway onangwgy
Epoch:   5 | Train loss: 1.691 | Val loss: 1.812 | Gen: ay iwyy intai-
y isisisisisisisisEOSy inoiyy
Epoch:   6 | Train loss: 1.636 | Val loss: 1.744 | Gen: ay iway
ongaaningiggyyy iway ongogoahy
Epoch:   7 | Train loss: 1.571 | Val loss: 1.761 | Gen: ay iway
insncaayyiy-y-ydiiy iswy onhohhygk
Epoch:   8 | Train loss: 1.494 | Val loss: 1.737 | Gen: eway ayay
onggggagiy iswwy onkkkkkkkkk
Epoch:   9 | Train loss: 1.449 | Val loss: 1.630 | Gen: eyEOSy iwwy
ongggnniiga-n--yaoyi isway onhkkhkkk
Epoch:  10 | Train loss: 1.396 | Val loss: 1.593 | Gen: ewEOSy iwwy
ongna-aiy iswsy ongngngniEOSgknhnn
Epoch:  11 | Train loss: 1.351 | Val loss: 1.689 | Gen: ebay iway
ongay--ynaayEOSEOSyyEOSiai iwwwy onhhohhhhhhy
Epoch:  12 | Train loss: 1.340 | Val loss: 1.579 | Gen: esay irwy
oogyyayyyyyyy iswwwwy onhnonnnrrroooonohrr
Epoch:  13 | Train loss: 1.305 | Val loss: 1.580 | Gen: ey iririrwry
```

```
ingay-iiaay is ongkgngigkEOSay
Epoch:  14 | Train loss: 1.304 | Val loss: 1.489 | Gen: eay aywy oogay
iwwy onokonokikwwy
Epoch:  15 | Train loss: 1.240 | Val loss: 1.605 | Gen: eyy ay ayayy
iwwy orahEOSaaay
Epoch:  16 | Train loss: 1.207 | Val loss: 1.482 | Gen: eay ay ongny-
iiy isway okayykakEOSkEOSiiiayy
Epoch:  17 | Train loss: 1.157 | Val loss: 1.464 | Gen: eay ayEOSiy
ondwyEOSyEOSy isaay irayahayaaaEOSyiyyy
Epoch:  18 | Train loss: 1.140 | Val loss: 1.475 | Gen: eay ayEOSiy
oogay iwwy onioiyyyhnwyyhiiyygh
Epoch:  19 | Train loss: 1.112 | Val loss: 1.461 | Gen: eay ayEOSiriwy
ongiyyiy isaay ohgghahagyyyyhyy
Epoch:  20 | Train loss: 1.096 | Val loss: 1.448 | Gen: eay ayEOSiy
ongny-ony iwwy onggaaagagaggggEOSwwy
Epoch:  21 | Train loss: 1.076 | Val loss: 1.451 | Gen: ehy aywywyyy
ongay iswwy ikkkkkkkkkyhhEOSEOShaaaa
Epoch:  22 | Train loss: 1.054 | Val loss: 1.414 | Gen: ettty ayEOSiry
ongigggayaagay isay okiiiiignrnwhhkk
Epoch:  23 | Train loss: 1.049 | Val loss: 1.402 | Gen: ety aywayEOSy
inaanniy isway okkkkiairrk
Epoch:  24 | Train loss: 1.020 | Val loss: 1.443 | Gen: eththay
ayiirirry ingny-i-gy iway ohghhhghhhiyhy
Epoch:  25 | Train loss: 1.006 | Val loss: 1.418 | Gen: etty
aywayEOSiy ongny isway okkgikyyygyiiykiyygy
Epoch:  26 | Train loss: 0.991 | Val loss: 1.399 | Gen: etwy
iririr--irry ingngngngiy iwiy ivivagEOSEOSEOSgyyy
Epoch:  27 | Train loss: 0.971 | Val loss: 1.416 | Gen: ththtay ayway
ongngngngnggynyy iwwyy okvgigaEOSkEOSEOSaEOSyaEOSkyy
Epoch:  28 | Train loss: 0.960 | Val loss: 1.402 | Gen: etwy arryyriyy
ongannyy iswwy okgghgwwyaawy
Epoch:  29 | Train loss: 0.930 | Val loss: 1.352 | Gen: tthahEOSy
airiy ingninnniy iwwy oghnhhrrgggy
Epoch:  30 | Train loss: 1.349 | Val loss: 1.549 | Gen: ehay ayay
iiinihhhay iway okgkgggyyy
Epoch:  31 | Train loss: 1.169 | Val loss: 1.512 | Gen: ehathy arway
ongngdy iway ogggaaagyy
Epoch:  32 | Train loss: 1.120 | Val loss: 1.478 | Gen: thhhhayy
ayaawwy ongniniiiiy isayEOSEOSsy ovonwwwywwhhohry
Epoch:  33 | Train loss: 1.137 | Val loss: 1.401 | Gen: ehaay ayEOSy
ongaayayiy isway ovingggy
Epoch:  34 | Train loss: 1.044 | Val loss: 1.379 | Gen: ehathay ay
ondniwgndy isway okikgwkkEOSyy
Epoch:  35 | Train loss: 0.985 | Val loss: 1.371 | Gen: ehathay ayEOSy
ongnnynny isway ovingggy
Epoch:  36 | Train loss: 0.960 | Val loss: 1.369 | Gen: ehathay ay
ondwnyaiyyy isway okingwgy
Epoch:  37 | Train loss: 0.931 | Val loss: 1.355 | Gen: ehathay ay
oninnyoiyny isway okingwgy
Epoch:  38 | Train loss: 0.919 | Val loss: 1.352 | Gen: ehhthay ay
```

```
ingnnydny isway okiinny
Epoch:  39 | Train loss: 0.904 | Val loss: 1.354 | Gen: ehathay ay
ondiiy isway okikgwwyEOSyy
Epoch:  40 | Train loss: 0.941 | Val loss: 1.357 | Gen: ehathay arEOSy
ondindnny isway oviiway
Epoch:  41 | Train loss: 1.302 | Val loss: 1.574 | Gen: ewwwwy iway
ondondiiy iway okgaaaaEOSwy
Epoch:  42 | Train loss: 1.175 | Val loss: 1.430 | Gen: ettwtwEOSy
iray ondnndddgdgdy isway owwnaEOSyy
Epoch:  43 | Train loss: 1.101 | Val loss: 1.419 | Gen: ettttty irayay
ontingindiigggy isway ogwaagggy
Epoch:  44 | Train loss: 1.059 | Val loss: 1.388 | Gen: etttway iray
ondindindinddyy isway ogiaaggyy
Epoch:  45 | Train loss: 1.021 | Val loss: 1.376 | Gen: etttway iray
ondindindinddndnnny isway okwwnnyy
Epoch:  46 | Train loss: 0.997 | Val loss: 1.366 | Gen: etttway iray
ondindindindanEOSnnny isway okwwgnyy
Epoch:  47 | Train loss: 0.978 | Val loss: 1.370 | Gen: etttway iray
ondindindinddndnnny isway ogwwgnyy
Epoch:  48 | Train loss: 0.964 | Val loss: 1.370 | Gen: etttway aray
ondindindindiyEOSEOSy isway ogwwgwyy
Epoch:  49 | Train loss: 0.949 | Val loss: 1.362 | Gen: etttay iray
ondindindidy isway ogiwnnyy
Epoch:  50 | Train loss: 0.934 | Val loss: 1.361 | Gen: etttway irwy
ondindindinay iway ogwaaayy
Epoch:  51 | Train loss: 0.920 | Val loss: 1.352 | Gen: ettttay irwyy
ondindindinay isway ogwwggyy
Epoch:  52 | Train loss: 0.912 | Val loss: 1.359 | Gen: ettttay irwyy
ondindindinay iway oggkwayy
Epoch:  53 | Train loss: 0.900 | Val loss: 1.357 | Gen: ettttay iriyy
ondindindinwy isway oggkwayy
Epoch:  54 | Train loss: 0.895 | Val loss: 1.349 | Gen: ettttay irayy
ondindindinyy isway oggkwayy
Epoch:  55 | Train loss: 0.884 | Val loss: 1.343 | Gen: etthey irayy
ondindindinay isway oggkwhay
Epoch:  56 | Train loss: 0.872 | Val loss: 1.332 | Gen: etthey iriy
ondindindindinidy isway oggggay
Epoch:  57 | Train loss: 0.866 | Val loss: 1.330 | Gen: etthey arwy
ondindindinay isway okwwgwyy
Epoch:  58 | Train loss: 0.858 | Val loss: 1.329 | Gen: ettttay iriy
ondindindinay isway oggawwaygy
Epoch:  59 | Train loss: 0.850 | Val loss: 1.339 | Gen: etthey iriry
ondindindnnay isway okwwgwyy
Epoch:  60 | Train loss: 0.840 | Val loss: 1.332 | Gen: etthey irway
ondindindinay isway owwwrwyywy
Epoch:  61 | Train loss: 0.831 | Val loss: 1.334 | Gen: etthey arwry
ondindindinay isway okwwgwyy
Epoch:  62 | Train loss: 0.823 | Val loss: 1.335 | Gen: etthey
irwayEOSy ondindindnnay isway ogkkwawEOSay
Epoch:  63 | Train loss: 0.819 | Val loss: 1.339 | Gen: etthay arway
```

```
ondindindiaay isway okgiawyy
Epoch:  64 | Train loss: 1.251 | Val loss: 1.672 | Gen: ewwy iraywwy oninininingny iwwyyy ogginy
Epoch:  65 | Train loss: 1.122 | Val loss: 1.443 | Gen: etway ieay ongngny iwway ogkwy
Epoch:  66 | Train loss: 1.017 | Val loss: 1.429 | Gen: eway awaway ondniny iwway ogiy
Epoch:  67 | Train loss: 0.970 | Val loss: 1.385 | Gen: eway iywy ongagayEOSy isway ogiy
Epoch:  68 | Train loss: 0.940 | Val loss: 1.368 | Gen: etwy aewy ongnggday isway okkkgy
Epoch:  69 | Train loss: 0.920 | Val loss: 1.353 | Gen: ethay aeEOSaEOSEOSy onddiidy isway ogkwngy
Epoch:  70 | Train loss: 0.901 | Val loss: 1.357 | Gen: ethey irwy ondididinEOSdy isway ogknnnyy
Epoch:  71 | Train loss: 0.890 | Val loss: 1.366 | Gen: ethey aewaEOSay oniniddndEOSy isway ogkigggEOSy
Epoch:  72 | Train loss: 0.880 | Val loss: 1.353 | Gen: ethey arwy onininingEOSy isway ogkigggEOSy
Epoch:  73 | Train loss: 0.865 | Val loss: 1.350 | Gen: ethey iywy ongngggay isway ogignwyy
Epoch:  74 | Train loss: 0.852 | Val loss: 1.345 | Gen: ethey iywy ondndndndady isway ogkigggEOSy
Epoch:  75 | Train loss: 0.844 | Val loss: 1.353 | Gen: ethey irwy ondniidndwy isway ogknnyaiy
Epoch:  76 | Train loss: 0.836 | Val loss: 1.347 | Gen: ethey iywy ongngdgy isway ogkigggEOSy
Epoch:  77 | Train loss: 0.827 | Val loss: 1.342 | Gen: ethey iywy onindnnndway isway okkkay
Epoch:  78 | Train loss: 0.819 | Val loss: 1.333 | Gen: ethey iywy ondniddy isway ogkigggEOSy
Epoch:  79 | Train loss: 0.813 | Val loss: 1.339 | Gen: ethey iywy oniningnday isway ogknnyaky
Epoch:  80 | Train loss: 0.807 | Val loss: 1.334 | Gen: ethty aywy onddiddy isway oginnwwy
Epoch:  81 | Train loss: 0.797 | Val loss: 1.337 | Gen: ethey aywy oniningddwyy isway oginnwgy
Epoch:  82 | Train loss: 0.791 | Val loss: 1.332 | Gen: ethey aywy oniningddwyy isway ogiigwwy
Epoch:  83 | Train loss: 0.784 | Val loss: 1.334 | Gen: ethey aywy ongngdindwy isway ogggagay
Epoch:  84 | Train loss: 0.805 | Val loss: 1.367 | Gen: etwy aywy ongdy isway ogwaaway
Epoch:  85 | Train loss: 0.832 | Val loss: 1.340 | Gen: ethty aywy onddin-idyy isway ogkkkaEOSwy
Epoch:  86 | Train loss: 0.810 | Val loss: 1.333 | Gen: etway aywy ongiiignnyny isway ogkwaagay
Epoch:  87 | Train loss: 0.799 | Val loss: 1.336 | Gen: etway aywy ongiinginyny isway ogkwaagay
Epoch:  88 | Train loss: 0.788 | Val loss: 1.320 | Gen: ethty aywy
```

```
onddiiinnyny isway ogkwaagay
Epoch:  89 | Train loss: 0.778 | Val loss: 1.325 | Gen: etway aywy
ongiingnnyny isway ogkwaagay
Epoch:  90 | Train loss: 0.771 | Val loss: 1.317 | Gen: etway aywy
ongny isway owwwwagay
Epoch:  91 | Train loss: 0.766 | Val loss: 1.319 | Gen: etway aywy
ongdiiinyyny isway ogkingway
Epoch:  92 | Train loss: 0.758 | Val loss: 1.320 | Gen: ethety aywy
ongny isway ogkingway
Epoch:  93 | Train loss: 0.761 | Val loss: 1.303 | Gen: etway aywy
ongniggoyyny isway oggiggway
Epoch:  94 | Train loss: 0.748 | Val loss: 1.315 | Gen: ethey aiaay
ongngdyy isway oggiggway
Epoch:  95 | Train loss: 0.743 | Val loss: 1.311 | Gen: etway aywy
oniiinindaEOSy isway oggigwway
Epoch:  96 | Train loss: 0.732 | Val loss: 1.305 | Gen: ethty aiaay
ongnigddndEOSy isway oggiggway
Epoch:  97 | Train loss: 0.727 | Val loss: 1.306 | Gen: ethty aywy
ondiininyyny isway oggigwway
Epoch:  98 | Train loss: 0.719 | Val loss: 1.294 | Gen: ethty iywy
ongny isway ogkiggway
Epoch:  99 | Train loss: 0.716 | Val loss: 1.308 | Gen: etway aywy
ondiinanyyny isway ogkinwway
Obtained lowest validation loss of: 1.2940169721841812
source:         the air conditioning is working
translated:     etway aywy ondiinanyyny isway ogkinwway
```

```
TEST_SENTENCE = "the air conditioning is working"
translated = translate_sentence(
    TEST_SENTENCE, trans32_encoder_s, trans32_decoder_s, None,
trans32_args_s
)
print("source:\t\t{} \ntranslated:\t{}".format(TEST_SENTENCE,
translated))
```

```
source:         the air conditioning is working
translated:     etway aywy ondiinanyyny isway ogkinwway
```

In the following cells, we investigate the effects of increasing model size and dataset size on the training / validation curves and generalization of the Transformer. We will increase hidden size to 64, and also increase dataset size. Include the best achieved validation loss in your report.

```
TEST_SENTENCE = "the air conditioning is working"

trans32_args_l = AttrDict()
args_dict = {
    "data_file_name": "pig_latin_large",  # Increased data set size
    "cuda": True,
    "nepochs": 100,
    "checkpoint_dir": "checkpoints",
```

```
    "learning_rate": 5e-4,
    "early_stopping_patience": 10,
    "lr_decay": 0.99,
    "batch_size": 512,
    "hidden_size": 32,
    "encoder_type": "transformer",
    "decoder_type": "transformer",  # options: rnn / rnn_attention /
transformer
    "num_transformer_layers": 3,
}
trans32_args_l.update(args_dict)
print_opts(trans32_args_l)

trans32_encoder_l, trans32_decoder_l, trans32_losses_l =
train(trans32_args_l)

translated = translate_sentence(
    TEST_SENTENCE, trans32_encoder_l, trans32_decoder_l, None,
trans32_args_l
)
print("source:\t\t{} \ntranslated:\t{}".format(TEST_SENTENCE,
translated))
```

```
======================================================================
==========
                                      Opts

----------------------------------------------------------------------
----------
                         data_file_name: pig_latin_large

                                   cuda: 1

                                nepochs: 100

                         checkpoint_dir: checkpoints

                          learning_rate: 0.0005

                early_stopping_patience: 10

                               lr_decay: 0.99

                             batch_size: 512

                            hidden_size: 32

                           encoder_type: transformer
```

```
                          decoder_type: transformer

                   num_transformer_layers: 3

======================================================================
==========
======================================================================
==========
                                    Data Stats

----------------------------------------------------------------------
----------
('brass', 'assbray')
('limo', 'imolay')
('nb', 'nbay')
('race', 'aceray')
('clarify', 'arifyclay')
Num unique word pairs: 22402
Vocabulary: dict_keys(['-', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h',
'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v',
'w', 'x', 'y', 'z', 'SOS', 'EOS'])
Vocab size: 29
======================================================================
==========
Moved models to GPU!
Epoch:   0 | Train loss: 3.143 | Val loss: 2.665 | Gen: ayEOSy
ayayayayayayayayayayay ayay ayay
Epoch:   1 | Train loss: 2.474 | Val loss: 2.392 | Gen: ete-t-
ayayayay oooooooooooooooooooo ay oooooooooooooooooooo
Epoch:   2 | Train loss: 2.255 | Val loss: 2.250 | Gen: ayay aay
oooooooooooooooooooo ay oooooooooooooooooooo
Epoch:   3 | Train loss: 2.117 | Val loss: 2.146 | Gen: - aaay inty ay
oooooooooooonoooooon
Epoch:   4 | Train loss: 2.016 | Val loss: 2.112 | Gen: ay ay inddy ay
aaEOSaEOSaEOSEOSy
Epoch:   5 | Train loss: 1.939 | Val loss: 2.024 | Gen: o- ay innnny
ay onnnnnnnnnnnnnnnnnnn
Epoch:   6 | Train loss: 1.856 | Val loss: 1.959 | Gen: adhtyyEOSy ay
innnntayy ay onnnnnnnnnnnnnnnnnnn
Epoch:   7 | Train loss: 1.791 | Val loss: 1.912 | Gen:
ayhtEOSttyyyEOSyy ay innndtayy ay onnnnnnnnnnnnnnnnnnn
Epoch:   8 | Train loss: 1.737 | Val loss: 1.873 | Gen: ay ay
innntaayy ay onnnnnnnnnnnyEOSyaay
Epoch:   9 | Train loss: 1.690 | Val loss: 1.854 | Gen: ay ay inddddy
ay onnoonnoy
Epoch:  10 | Train loss: 1.685 | Val loss: 1.914 | Gen: ay
ayayayayayayayayayay indyy-yy isayay onnnnnnnnnnnnonnny
Epoch:  11 | Train loss: 1.674 | Val loss: 1.823 | Gen: ay aray intty
iy ongrgggay
Epoch:  12 | Train loss: 1.620 | Val loss: 1.789 | Gen: ay ay indcy iy
```

```
ongrgggry
Epoch:  13 | Train loss: 1.578 | Val loss: 1.744 | Gen: ay aray
intinttnttnnay iy ongrgggay
Epoch:  14 | Train loss: 2.129 | Val loss: 2.063 | Gen: aay ararsy
ondaEOSy ayayayEOSEOSy inyay
Epoch:  15 | Train loss: 1.833 | Val loss: 1.937 | Gen: ----- ayarh-
aaEOSyy onddyaay ayyEOSy usyay
Epoch:  16 | Train loss: 1.755 | Val loss: 1.883 | Gen: --- arsytsy
onc sssssssssssssssssss onc
Epoch:  17 | Train loss: 1.707 | Val loss: 1.850 | Gen: ----- arstssy
onddyy asyEOSyy onc
Epoch:  18 | Train loss: 1.669 | Val loss: 1.826 | Gen: ------ arsyssy
onddyy ayyEOSy ongpay
Epoch:  19 | Train loss: 1.635 | Val loss: 1.794 | Gen: ehhtttt-
arayssy ongccy ayyEOSy ongy
Epoch:  20 | Train loss: 1.606 | Val loss: 1.772 | Gen: ----- arayssy
intiy ayyEOSy ongy
Epoch:  21 | Train loss: 1.626 | Val loss: 1.781 | Gen:
athhhhhtthhhhhhhhhty arssssssssssshshssss ingcy ay onay
Epoch:  22 | Train loss: 1.607 | Val loss: 1.753 | Gen: ay
arssssssssssshssssss ingcy aay onay
Epoch:  23 | Train loss: 1.578 | Val loss: 1.750 | Gen: ay ararEOSy
inaay ay ongyEOSyy
Epoch:  24 | Train loss: 1.554 | Val loss: 1.711 | Gen: ay
arEOSriiasryyyy inaay ay ogEOSaEOSEOSEOSEOSEOSy
Epoch:  25 | Train loss: 1.533 | Val loss: 1.714 | Gen: ay ayEOSraaayy
inaay ay ongray
Epoch:  26 | Train loss: 1.514 | Val loss: 1.677 | Gen: ay aysyyyy
inaay isay ongayEOSyEOSEOSy
Epoch:  27 | Train loss: 1.496 | Val loss: 1.682 | Gen: ay arayEOSy
inaay isay ongaEOSyy
Epoch:  28 | Train loss: 1.489 | Val loss: 1.632 | Gen: ay arayEOSy
inaay aaay ongayEOSEOSy
Epoch:  29 | Train loss: 1.469 | Val loss: 1.651 | Gen: ayEOStEOSyEOSy
araray ingay asay ongiay
Epoch:  30 | Train loss: 1.452 | Val loss: 1.612 | Gen: ayEOStEOSyEOSy
ararEOSy ingay asay odraay
Epoch:  31 | Train loss: 1.437 | Val loss: 1.634 | Gen:
ehhhhhhhhhhhhhhhhhhh arararEOSrayayy ingay asay ongyy
Epoch:  32 | Train loss: 1.421 | Val loss: 1.594 | Gen: ayEOStEOSyEOSy
ararEOSy ingay asay odraay
Epoch:  33 | Train loss: 1.407 | Val loss: 1.619 | Gen:
ehhhhhhhhhhhhhhhhhhh arayEOSyayy ingay ay ongyy
Epoch:  34 | Train loss: 1.427 | Val loss: 1.592 | Gen:
ehhhhhhhhhhhhhhhyEOSyh aray ongcayyEOSy sy oggrgay
Epoch:  35 | Train loss: 1.401 | Val loss: 1.611 | Gen: - aray
ongEOSay ay ongy
Epoch:  36 | Train loss: 1.382 | Val loss: 1.583 | Gen: ehhhhayEOSy
aray ondingy ay ooorgay
Epoch:  37 | Train loss: 1.367 | Val loss: 1.580 | Gen:
```

```
ehhhhhhhhhhhhhhhhhhhh aray ondingy ay ooorgay
Epoch:  38 | Train loss: 1.356 | Val loss: 1.567 | Gen: ehhhhyyEOSy
aray ondingy ay oogry
Epoch:  39 | Train loss: 1.345 | Val loss: 1.567 | Gen:
ethtthhEOSyyyEOSEOSEOShhhayy aray onciy ay oogry
Epoch:  40 | Train loss: 1.335 | Val loss: 1.538 | Gen: adhEOSyy aray
ondingcy ay ongry
Epoch:  41 | Train loss: 1.329 | Val loss: 1.565 | Gen:
ehhhhhhhhhhhhhhhhhhhh aray onciy ay ogggrrgy
Epoch:  42 | Train loss: 1.320 | Val loss: 1.518 | Gen: ethhtEOSay
aray ondingcy ay ongry
Epoch:  43 | Train loss: 1.309 | Val loss: 1.537 | Gen:
ehhhhhhhhhhhhhhhhhhha araray onciy ay ogggrrgy
Epoch:  44 | Train loss: 1.296 | Val loss: 1.508 | Gen: ethhyay aray
oncingcy ay oonrgry
Epoch:  45 | Train loss: 1.285 | Val loss: 1.520 | Gen:
ehhhhhhhhhhhhhhhhhhhy aray onciy ay ogggrrgy
Epoch:  46 | Train loss: 1.276 | Val loss: 1.494 | Gen: ahhhhaay aray
ongcngciy ay oggggrgy
Epoch:  47 | Train loss: 1.266 | Val loss: 1.504 | Gen:
ehhhhhhhhhhhhhhhhhhhy aray onciy ay oggggggy
Epoch:  48 | Train loss: 1.257 | Val loss: 1.483 | Gen: ahhhhaay aray
ongcngciy ay oggggrgy
Epoch:  49 | Train loss: 1.250 | Val loss: 1.494 | Gen:
ettttayEOSyyyEOSyhEOShhaay araray onciy ay oggggggy
Epoch:  50 | Train loss: 1.369 | Val loss: 1.535 | Gen: ewhhawy away
oncingcy isy ogggggggyyy
Epoch:  51 | Train loss: 1.319 | Val loss: 1.499 | Gen: ewhhhaay aray
oncongcy isy oggggggaEOSy
Epoch:  52 | Train loss: 1.296 | Val loss: 1.477 | Gen: ewhthaay aray
onccndcky isy oggggggaEOSy
Epoch:  53 | Train loss: 1.279 | Val loss: 1.465 | Gen:
ewhyhaytyhhyyhhhhy aray ongcngggy isy opgrry
Epoch:  54 | Train loss: 1.266 | Val loss: 1.464 | Gen: ewhtawy aray
ongcngggy isy oggggggaEOSy
Epoch:  55 | Train loss: 1.256 | Val loss: 1.448 | Gen: ethhahy aray
ongcngggy isy opgrry
Epoch:  56 | Train loss: 1.243 | Val loss: 1.439 | Gen: ethtawy aray
ongcngggy iay ogggggggyyy
Epoch:  57 | Train loss: 1.233 | Val loss: 1.432 | Gen: ethtahay
araray ongcngggy iay ogggggggyay
Epoch:  58 | Train loss: 1.223 | Val loss: 1.428 | Gen: eththaay
araray ongcngggy isy oggiggggyy
Epoch:  59 | Train loss: 1.214 | Val loss: 1.418 | Gen: ethtahay
araray ongcngggy isy odshrrss
Epoch:  60 | Train loss: 1.204 | Val loss: 1.417 | Gen: eththaay
araway ondindcdy iay oggiiggyy
Epoch:  61 | Train loss: 1.197 | Val loss: 1.409 | Gen: ethtaway
araray ongcngggy isy odgrrrss
Epoch:  62 | Train loss: 1.189 | Val loss: 1.406 | Gen: eththayaay
```

```
araway ondinidnccy iay oggiiggyy
Epoch:  63 | Train loss: 1.181 | Val loss: 1.400 | Gen: eththaayEOSy
araray ongingtty iay odgrry
Epoch:  64 | Train loss: 1.174 | Val loss: 1.401 | Gen: eththayaay
araray ontinidnndy iay ogrigggyy
Epoch:  65 | Train loss: 1.168 | Val loss: 1.400 | Gen: eththaayEOSy
araray ongingtty iay opgrggy
Epoch:  66 | Train loss: 1.199 | Val loss: 1.428 | Gen: eththayaay
aray ontcittiigEOScy iay opsry
Epoch:  67 | Train loss: 1.668 | Val loss: 1.810 | Gen: thththy ayway
oy ieysssay o
Epoch:  68 | Train loss: 1.564 | Val loss: 1.650 | Gen:
thththtthhththththaa awaywy onEOSnEOSyyy iay o
Epoch:  69 | Train loss: 1.464 | Val loss: 1.597 | Gen:
ethththththththhhty aray ociiiyy iay onny
Epoch:  70 | Train loss: 1.414 | Val loss: 1.562 | Gen:
eththththtthhhhhhhy aray ontioyiy iay onEOSy
Epoch:  71 | Train loss: 1.378 | Val loss: 1.542 | Gen:
ethththththaytthhhay aray ontioy iay oniy
Epoch:  72 | Train loss: 1.354 | Val loss: 1.528 | Gen:
tthththththhhhhhhy aray ontioy iay oniy
Epoch:  73 | Train loss: 1.333 | Val loss: 1.513 | Gen:
ethththththhhhhhhy aray ontiny isy oniy
Validation loss has not improved in 10 epochs, stopping early
Obtained lowest validation loss of: 1.3996964582690485
source:         the air conditioning is working
translated:     ehhthththththaEOShhhhy aray ontiny isy onay
```

```python
TEST_SENTENCE = "the air conditioning is working"

trans64_args_s = AttrDict()
args_dict = {
    "data_file_name": "pig_latin_small",
    "cuda": True,
    "nepochs": 50,
    "checkpoint_dir": "checkpoints",
    "learning_rate": 5e-4,
    "early_stopping_patience": 20,
    "lr_decay": 0.99,
    "batch_size": 64,
    "hidden_size": 64,  # Increased model size
    "encoder_type": "transformer",
    "decoder_type": "transformer",  # options: rnn / rnn_attention /
transformer
    "num_transformer_layers": 3,
}
trans64_args_s.update(args_dict)
print_opts(trans64_args_s)

trans64_encoder_s, trans64_decoder_s, trans64_losses_s =
```

```
train(trans64_args_s)

translated = translate_sentence(
    TEST_SENTENCE, trans64_encoder_s, trans64_decoder_s, None,
trans64_args_s
)
print("source:\t\t{} \ntranslated:\t{}".format(TEST_SENTENCE,
translated))
```

```
======================================================================
==========
                                      Opts

----------------------------------------------------------------------
----------
                         data_file_name: pig_latin_small

                                   cuda: 1

                                nepochs: 50

                         checkpoint_dir: checkpoints

                          learning_rate: 0.0005

                early_stopping_patience: 20

                               lr_decay: 0.99

                             batch_size: 64

                            hidden_size: 64

                           encoder_type: transformer

                           decoder_type: transformer

                 num_transformer_layers: 3

======================================================================
==========
======================================================================
==========
                                   Data Stats

----------------------------------------------------------------------
----------
('whisper', 'isperwhay')
('prayers', 'ayerspray')
```

```
('sketch', 'etchskay')
('disinclination', 'isinclinationday')
('tranquilize', 'anquilizetray')
Num unique word pairs: 3198
Vocabulary: dict_keys(['-', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h',
'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v',
'w', 'x', 'y', 'z', 'SOS', 'EOS'])
Vocab size: 29
======================================================================
==========
Moved models to GPU!
Epoch:   0 | Train loss: 2.446 | Val loss: 2.080 | Gen: iway ttattt
iliaiii iwaay inay
Epoch:   1 | Train loss: 1.731 | Val loss: 1.782 | Gen: ihhy
rrrrrrrrrrrrrrrrrr ongoygayEOSoogy iway ongngggggEOSnnEOSfooogg
Epoch:   2 | Train loss: 1.496 | Val loss: 1.720 | Gen: ehhhyyy
ondwnanrrwdEOSy iway ingray
Epoch:   3 | Train loss: 1.357 | Val loss: 1.690 | Gen: ihhhh iiaai
odffffffffcffffccfff isisssssswswsaaaass igigiiiggy
Epoch:   4 | Train loss: 1.266 | Val loss: 1.458 | Gen: ewhy  ongngay
issay ongngay
Epoch:   5 | Train loss: 1.178 | Val loss: 1.486 | Gen: eway iway
ondindangnggddndyayn iway ongngrayEOSygngwEOSyny
Epoch:   6 | Train loss: 1.101 | Val loss: 1.331 | Gen: eway awwy
ondiiayy iwaay ongngnny
Epoch:   7 | Train loss: 1.060 | Val loss: 1.432 | Gen: ft
atayyEOSEOSraray ontittintitiiataay iwaay onngkngy
Epoch:   8 | Train loss: 1.038 | Val loss: 1.365 | Gen: ethay iway
ondiinnyyy iwaay oksksggggygggay
Epoch:   9 | Train loss: 0.990 | Val loss: 1.232 | Gen: eay away
ondindnn-EOSngyyay isway okkiiggiiy
Epoch:  10 | Train loss: 0.892 | Val loss: 1.274 | Gen:
etetetetetetetetetet aaray onditnnanngnannnyy iaayyyaayyyyayyiiiii
okkkgyykkyyyy
Epoch:  11 | Train loss: 0.851 | Val loss: 1.265 | Gen: ethay away
onniiiiigiy iway okiigiaay
Epoch:  12 | Train loss: 0.807 | Val loss: 1.182 | Gen: ethay away
ondnny iway okkkiiky
Epoch:  13 | Train loss: 0.737 | Val loss: 1.117 | Gen: ethay away
ondctnnay iway oksiiyjjnnynyny
Epoch:  14 | Train loss: 0.681 | Val loss: 1.072 | Gen: ethay away
oninninngy isway okwggggEOStttty
Epoch:  15 | Train loss: 0.645 | Val loss: 1.120 | Gen:
eteteteteteteteteteee awaway ondtitiiinEOSy isway oknnaasaay
Epoch:  16 | Train loss: 0.621 | Val loss: 1.069 | Gen: ethay iwaway
ondctinay isway okkkaggwgyEOSEOSy
Epoch:  17 | Train loss: 0.593 | Val loss: 1.155 | Gen: ethay awaway
onitiningny isway okiigkay
Epoch:  18 | Train loss: 0.552 | Val loss: 1.065 | Gen: etethay
irayEOSy ondttingcatyyay isway okinwnwgay
```

```
Epoch:  19 | Train loss: 0.572 | Val loss: 1.061 | Gen: eteteay iraway
ondtitiaanay isway orikgiiay
Epoch:  20 | Train loss: 0.540 | Val loss: 1.002 | Gen: ehay
awawayEOSwayyyy ondctitgnawwy isway okiaiggggy
Epoch:  21 | Train loss: 0.489 | Val loss: 0.965 | Gen: ethay iwaway
ondtiinainy isway okkigwnay
Epoch:  22 | Train loss: 0.449 | Val loss: 0.944 | Gen: ethay iwaway
ondtittngny isway okkiggwgy
Epoch:  23 | Train loss: 0.424 | Val loss: 0.958 | Gen: ethay
awaaayEOSyy onatiitnany isway okkingway
Epoch:  24 | Train loss: 0.525 | Val loss: 1.089 | Gen: etetay iriyay
oniitininiiggiyy isway okkiiggkky
Epoch:  25 | Train loss: 0.576 | Val loss: 0.967 | Gen: etethay irwyy
indtittdnwcnccaiy isway okrknnkjkjy
Epoch:  26 | Train loss: 0.489 | Val loss: 0.870 | Gen: ethay irayy
ondititintotoay isway okiningway
Epoch:  27 | Train loss: 0.425 | Val loss: 0.909 | Gen: ethay irway
onditinininnnEOSy isway orkingwayEOSy
Epoch:  28 | Train loss: 0.392 | Val loss: 0.824 | Gen: ethay iray
ondtitiniicgy isway okingwgway
Epoch:  29 | Train loss: 0.339 | Val loss: 0.805 | Gen: ethay iriaEOSy
onditininingnny isway okingwgyay
Epoch:  30 | Train loss: 0.311 | Val loss: 0.777 | Gen: ethay iray
onditionoooy isway okingway
Epoch:  31 | Train loss: 0.293 | Val loss: 0.762 | Gen: ethay irayy
ondititiningcayy isway okingwgway
Epoch:  32 | Train loss: 0.286 | Val loss: 0.934 | Gen: ethay iray
onditiningay isway okwakway
Epoch:  33 | Train loss: 0.329 | Val loss: 0.835 | Gen: ethay iraay
onditiionngoy isway okikgkngway
Epoch:  34 | Train loss: 0.315 | Val loss: 0.855 | Gen: ehay iraay
onditinininnccy isway okkingway
Epoch:  35 | Train loss: 0.352 | Val loss: 0.804 | Gen: ethay arawy
onditionioiy isway okikiwgyy
Epoch:  36 | Train loss: 0.265 | Val loss: 0.726 | Gen: ethay irway
onditioniongaay isway okikiayEOSEOSy
Epoch:  37 | Train loss: 0.219 | Val loss: 0.754 | Gen: ethay irway
onditioningcay isway okinggway
Epoch:  38 | Train loss: 0.236 | Val loss: 0.723 | Gen: ethyy irray
onditionioncay isway okingwway
Epoch:  39 | Train loss: 0.223 | Val loss: 0.816 | Gen: ethyy arway
onditiononoocyaEOSy isway okingway
Epoch:  40 | Train loss: 0.239 | Val loss: 0.727 | Gen: ethyy ariray
ondititingincy isway orkingway
Epoch:  41 | Train loss: 0.194 | Val loss: 0.686 | Gen: ethyy arwaay
ondititionggay isway orkingway
Epoch:  42 | Train loss: 0.158 | Val loss: 0.731 | Gen: ethyy ariaay
onditioioiiiayy isway orkingway
Epoch:  43 | Train loss: 0.154 | Val loss: 0.685 | Gen: ethyy irrwy
onditioionoicayEOSy isway orkingway
```

```
Epoch:  44 | Train loss: 0.148 | Val loss: 0.779 | Gen: ethhy ariryay
onnditiningcaay isway orkingway
Epoch:  45 | Train loss: 0.154 | Val loss: 0.723 | Gen: ethyy iiiay
onnditioioncny isway orkingway
Epoch:  46 | Train loss: 0.127 | Val loss: 0.749 | Gen: ethyy ariwyay
onddiiionoonay isway orkingway
Epoch:  47 | Train loss: 0.133 | Val loss: 0.671 | Gen: ethyy arrwaay
onniitoooonoocy isway orkingway
Epoch:  48 | Train loss: 0.126 | Val loss: 0.724 | Gen: eehyy arrwyay
onnditioniончy isway orkingway
Epoch:  49 | Train loss: 0.134 | Val loss: 0.790 | Gen: ethyy airway
ondditiiioaay isway orkingway
Obtained lowest validation loss of: 0.6713013056861726
source:         the air conditioning is working
translated:     ethyy airway ondditiiioaay isway orkingway
```

```
TEST_SENTENCE = "the air conditioning is working"

trans64_args_l = AttrDict()
args_dict = {
    "data_file_name": "pig_latin_large",  # Increased data set size
    "cuda": True,
    "nepochs": 50,
    "checkpoint_dir": "checkpoints",
    "learning_rate": 5e-4,
    "early_stopping_patience": 20,
    "lr_decay": 0.99,
    "batch_size": 512,
    "hidden_size": 64,  # Increased model size
    "encoder_type": "transformer",
    "decoder_type": "transformer",  # options: rnn / rnn_attention /
transformer
    "num_transformer_layers": 3,
}
trans64_args_l.update(args_dict)
print_opts(trans64_args_l)

trans64_encoder_l, trans64_decoder_l, trans64_losses_l =
train(trans64_args_l)

translated = translate_sentence(
    TEST_SENTENCE, trans64_encoder_l, trans64_decoder_l, None,
trans64_args_l
)
print("source:\t\t{} \ntranslated:\t{}".format(TEST_SENTENCE,
translated))
```

```
======================================================================
==========
                                      Opts
```

```
--------------------------------------------------------------------------------
                       data_file_name: pig_latin_large

                                 cuda: 1

                              nepochs: 50

                       checkpoint_dir: checkpoints

                        learning_rate: 0.0005

              early_stopping_patience: 20

                             lr_decay: 0.99

                           batch_size: 512

                          hidden_size: 64

                         encoder_type: transformer

                         decoder_type: transformer

               num_transformer_layers: 3

================================================================================
================================================================================
                                   Data Stats

--------------------------------------------------------------------------------
('brass', 'assbray')
('limo', 'imolay')
('nb', 'nbay')
('race', 'aceray')
('clarify', 'arifyclay')
Num unique word pairs: 22402
Vocabulary: dict_keys(['-', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h',
'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v',
'w', 'x', 'y', 'z', 'SOS', 'EOS'])
Vocab size: 29
================================================================================
Moved models to GPU!
Epoch:   0 | Train loss: 2.732 | Val loss: 2.294 | Gen: ey ay
```

```
iaaaaaaaaaadaaaayy inginiin nyy
Epoch:   1 | Train loss: 2.024 | Val loss: 2.081 | Gen: t ay intntay
int igy
Epoch:   2 | Train loss: 1.806 | Val loss: 1.884 | Gen: e ay
ontnnoyoooaEOS-oyEOSEOSoon isisaaaassssssyy inay
Epoch:   3 | Train loss: 1.660 | Val loss: 1.756 | Gen: etey ay
ontintintintintiy isisisisisisiway ingyy
Epoch:   4 | Train loss: 1.571 | Val loss: 1.673 | Gen: etay ay
ondadaadddnnnnddy isssssssssssssssssss ingggy
Epoch:   5 | Train loss: 1.499 | Val loss: 1.612 | Gen: etayhy ay
ontiiiiiy isississsssiiiiiissi ingaaay
Epoch:   6 | Train loss: 1.434 | Val loss: 1.550 | Gen: etay
ararrrrrrrrrrrrrarra indayEOSayEOSEOSywyayy isssssssssssssssssss ongiy
Epoch:   7 | Train loss: 1.315 | Val loss: 1.473 | Gen: eththt ay
ontintintintintintin isayEOSEOSyEOSaaay ingiyy
Epoch:   8 | Train loss: 1.331 | Val loss: 1.484 | Gen: eay ay
ongoogoogonnniaanyyn isayyyyy y
Epoch:   9 | Train loss: 1.852 | Val loss: 2.158 | Gen: ttway ay
ommmmmmmmmmmmommammmm iay owaywaywaywaywaywayw
Epoch:  10 | Train loss: 1.629 | Val loss: 1.872 | Gen: ey ay
ondiiddyddydydddyiyn iay onaaygy-y
Epoch:  11 | Train loss: 1.518 | Val loss: 1.690 | Gen: ay ay inniniy
iy y
Epoch:  12 | Train loss: 1.425 | Val loss: 1.562 | Gen: etway ay
ondin-inEOSinEOSinEOSnnind iy ay
Epoch:  13 | Train loss: 1.358 | Val loss: 1.510 | Gen: ey ay
ondindindindinaindin iy ay
Epoch:  14 | Train loss: 1.320 | Val loss: 1.503 | Gen: twaay ay
ondindgnEOSinEOSinnininind iy yy
Epoch:  15 | Train loss: 1.282 | Val loss: 1.468 | Gen: eway ay
ondindindindindindin isaay ay
Epoch:  16 | Train loss: 1.258 | Val loss: 1.466 | Gen: eway ay
ondindindindindindin isway ay
Epoch:  17 | Train loss: 1.233 | Val loss: 1.428 | Gen: eway ay
ondindindindindindin isway ay
Epoch:  18 | Train loss: 1.220 | Val loss: 1.420 | Gen: eway ay
ondindiddincincincin isaay ay
Epoch:  19 | Train loss: 1.199 | Val loss: 1.387 | Gen: eway aray
ondindindindway iway ay
Epoch:  20 | Train loss: 1.188 | Val loss: 1.407 | Gen: eway ay
ondindincincincincin isway oy
Epoch:  21 | Train loss: 1.164 | Val loss: 1.370 | Gen: eway irrrrray
ondindindindway isayy oy
Epoch:  22 | Train loss: 1.154 | Val loss: 1.364 | Gen: eway ay
ondindindincincincin isway oy
Epoch:  23 | Train loss: 1.127 | Val loss: 1.325 | Gen: eway irrrray
ondindindindwny isayy oway
Epoch:  24 | Train loss: 1.108 | Val loss: 1.315 | Gen: eway ay
ondindindindwnnEOSnndn isway oway
Epoch:  25 | Train loss: 1.092 | Val loss: 1.334 | Gen: eway irryy
```

```
ondiddiddindidyEOSnycc isayy owiy
Epoch:  26 | Train loss: 1.081 | Val loss: 1.319 | Gen: eway aray
oncincindincwacwincn isway owiy
Epoch:  27 | Train loss: 1.068 | Val loss: 1.269 | Gen: eway irrraay
ondindindiddday isway owiy
Epoch:  28 | Train loss: 1.069 | Val loss: 1.325 | Gen: ewty aray
oncincindincwncinnin isway orgiggay
Epoch:  29 | Train loss: 1.061 | Val loss: 1.261 | Gen: eway arrray
ondindandiniindnEOScni isayy owiy
Epoch:  30 | Train loss: 1.055 | Val loss: 1.314 | Gen: eththtty aray
ondindindindincindnc isayy oniy
Epoch:  31 | Train loss: 1.028 | Val loss: 1.247 | Gen: eththaay
arrray ondiciid-y isayy oniy
Epoch:  32 | Train loss: 1.009 | Val loss: 1.272 | Gen: eththtwyy ay
ondindindinciyacay isayy okiy
Epoch:  33 | Train loss: 0.998 | Val loss: 1.247 | Gen: eththtay
iraraEOSEOSiirriirry oncinciccccccccciiii isayEOSiy oniy
Epoch:  34 | Train loss: 0.989 | Val loss: 1.246 | Gen: ethththay
array onccccicinnwncinnin isayy okiy
Epoch:  35 | Train loss: 0.966 | Val loss: 1.185 | Gen: eththaay
arrray onginginainny isayy oknnkky
Epoch:  36 | Train loss: 0.974 | Val loss: 1.379 | Gen: eththtty aray
ongingiggingngggnegg isway okngaay
Epoch:  37 | Train loss: 0.985 | Val loss: 1.334 | Gen: eththaay aray
oncicczcaaEOScnycyy isisEOSiy owiy
Epoch:  38 | Train loss: 0.975 | Val loss: 1.296 | Gen: ethtay araray
ongingdggiinoyyiygid isway okiy
Epoch:  39 | Train loss: 0.974 | Val loss: 1.277 | Gen: eththtay
araray oncincindiicwdy issssyiiiiissyy owiy
Epoch:  40 | Train loss: 0.943 | Val loss: 1.157 | Gen: eththaay aray
ongingnndndwnnoingen isway orinniy
Epoch:  41 | Train loss: 0.917 | Val loss: 1.173 | Gen: ethththtty
arrrarraaarararararaar ondincindwEOSy isayyy oriingay
Epoch:  42 | Train loss: 0.913 | Val loss: 1.232 | Gen: eththaay
arrray ongingingingingingnc isway orkingway
Epoch:  43 | Train loss: 0.894 | Val loss: 1.190 | Gen: eththaay
irrayyEOSiay oncincinEOSiaEOSineinccc isway orkingy
Epoch:  44 | Train loss: 0.882 | Val loss: 1.179 | Gen: eththaay array
ongnngonnnnnnicyiiwn isway orkngwyEOSyy
Epoch:  45 | Train loss: 0.864 | Val loss: 1.133 | Gen: eththaay
arrray ondiniiiwwyywyyy isayyy orkingway
Epoch:  46 | Train loss: 0.851 | Val loss: 1.179 | Gen: eththaay array
ondindiinwicacny isway orkrgwyEOSyy
Epoch:  47 | Train loss: 0.842 | Val loss: 1.129 | Gen: eththaay
arrray onddnddngw-cconeccei issway orkingwayy
Epoch:  48 | Train loss: 0.833 | Val loss: 1.145 | Gen: ethtay arrray
ondingooaaaEOSwnniinii isway oriingway
Epoch:  49 | Train loss: 0.823 | Val loss: 1.115 | Gen: eththaay
arrray ondinddny isswy oriingway
Obtained lowest validation loss of: 1.115473040552051
```

```
source:          the air conditioning is working
translated:      eththaay arrray ondinddny isswy oriingway
```

The following cell generates two loss plots. In the first plot, we compare the effects of increasing dataset size. In the second plot, we compare the effects of increasing model size. Include both plots in your report, and include your analysis of the results.

```
save_loss_comparison_by_dataset(
    trans32_losses_s,
    trans32_losses_l,
    trans64_losses_s,
    trans64_losses_l,
    trans32_args_s,
    trans32_args_l,
    trans64_args_s,
    trans64_args_l,
    "trans_by_dataset",
)
save_loss_comparison_by_hidden(
    trans32_losses_s,
    trans32_losses_l,
    trans64_losses_s,
    trans64_losses_l,
    trans32_args_s,
    trans32_args_l,
    trans64_args_s,
    trans64_args_l,
    "trans_by_hidden",
)
```

```
<Figure size 432x288 with 0 Axes>
```

```
<Figure size 432x288 with 0 Axes>
```

# Scaled Dot Product Attention

## Part 2.1: Additive Attention

It takes around four and a half minutes to run the traning loops in Part 1 but it takes aroud three and a half minutes to run the traning loop in Part 2. Additive attention trains faster as it converges faster, which triggers early stopping. Therefore, we would not need to go through all 50 epoches like the GRUs.

(More comments regarding training time between `RNNAttention` and `ScaledDotAttention` can be found in Part 2.2 Question 3)

## Part 2.2: Scaled Dot Product Attention

### Question 1 & 2

Code for `ScaledDotProduct` and `CausalScaledDotProduct` can be found in Step 4 and Step 5 respectively.

### Question 3

The model trained using `ScaledDotAttention` performs worse than `RNNAttention`. However, it took much less time to run 100 epoches of `ScaledDotAttention` than 35 epoches of `RNNAttention`. This reveals that `ScaledDotAttention` is more computationally efficient and space-efficient. `ScaledDotAttention` is also easier to implement as it uses highly optimized matrix multiplication (whereas additive attention uses a two layer fully-connected network). This could be the reason that `RNNAttention` outperforms `ScaledDotAttention`.

### Question 4

We chose to represent word positions in this manner because we want the positions to be bounded and unique for each time step. Another advantage of using sine and cosine is that we could retreive information regarding the relative positions between 2 words through linear transformation (matrix).

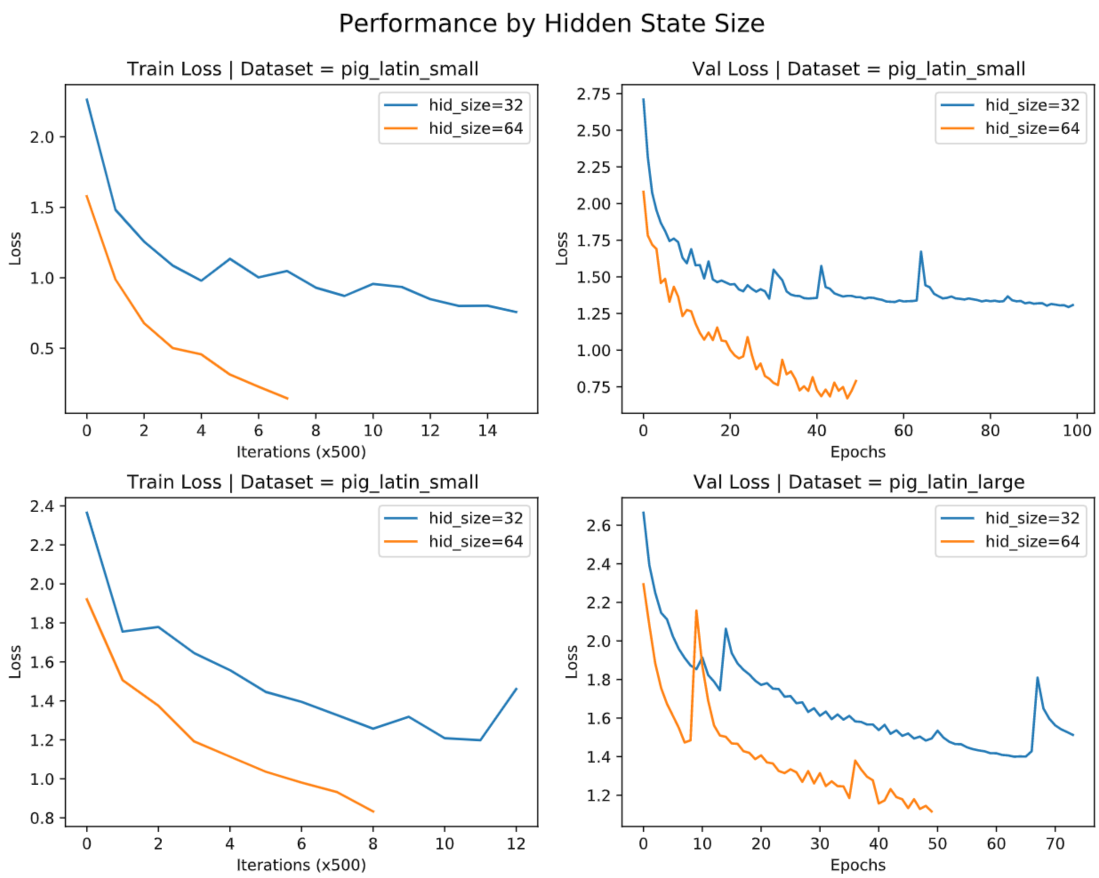
Although one-hot encoding can be used for positional encoding, one hot encoding is a long list/array where every bit is 1 or 0. Flipping bits for a long array becomes a waste of space when you could utilize floats to represent postions.

### Question 5

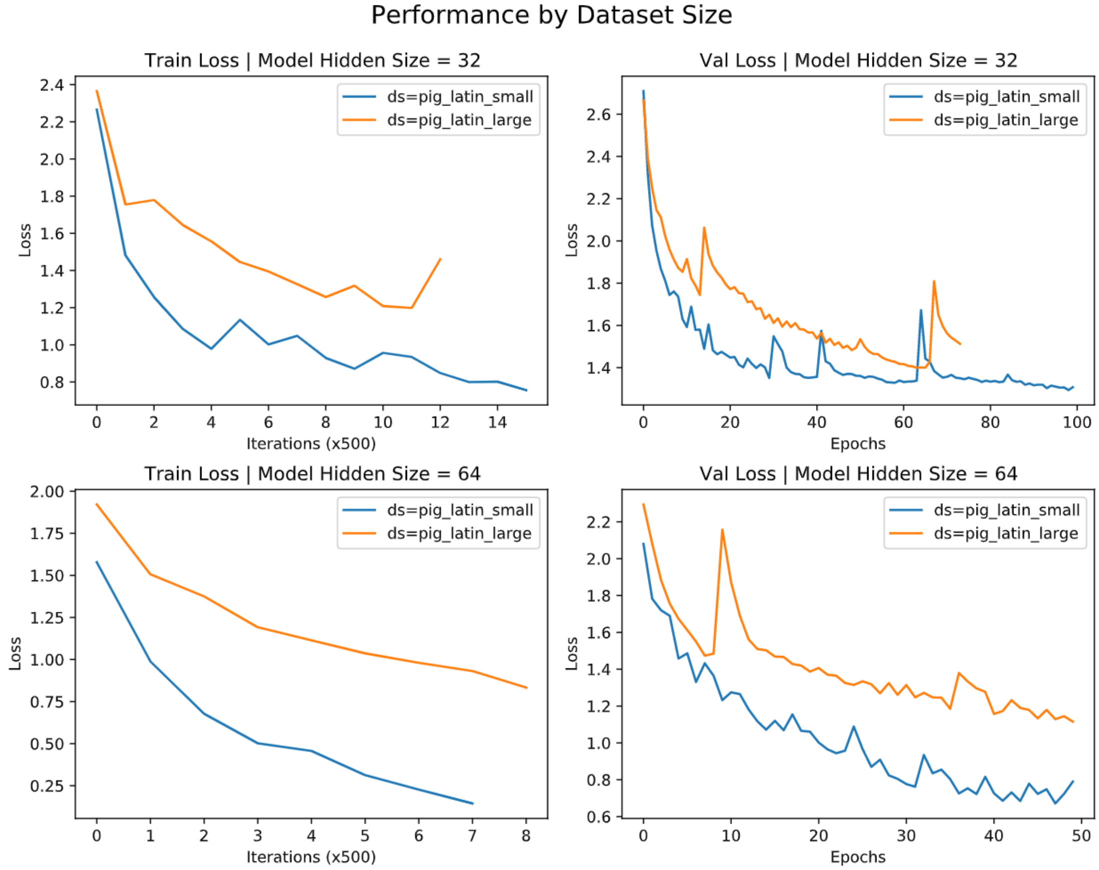
The `TransformerEncoder` and `TransformerDecoder` obtains the lowest validation accuracy of 1.02. It performs better than single block `Attention` tjat obtains the lowest validation accuracy of 1.32. However, it is worse than `RNNAttention` model. which obtains the lowest validation accuracy of about 0.33. Additionally, both Transformer and single-layer attention are trained using all 100 epoches but the RNN Attention model reached early stopping condition. Therefore, RNN Attention is the best model among all these 3 models.

## Question 6

### *Loss Curve for `save_loss_comparison_by_hidden`*



### *Loss Curve for `save_loss_comparison_by_dataset`*




### *Lowest Validation Loss*

- small datasize, small hidden/model size: 1.29
- small datasize, large hidden/model size: 0.67
- large datasize, small hidden/model size: 1.39
- large datasize, large hidden/model size: 1.12

### *Analysis*

When the datasize remains, large model size lowers validation loss. When the model size is constant, smaller datasize leads to lower validation loss. This bahavior adheres to the intuition that large hidden layers in Transformers have the ability to extract and detect patterns, which is why larger model size decreases validation loss given the same datasize. If the size of the hidden layer is fixed, the model can achieve better result with small dataset as the Transformer is not overwhelmed by the number of training examples.

## Colab FAQ

For some basic overview and features offered in Colab notebooks, check out: Overview of Colaboratory Features

You need to use the colab GPU for this assignment by selecting:

**Runtime** → **Change runtime type** → **Hardware Accelerator: GPU**

## Part 4: Fine-tuning pretrained language models

Acknowledgement: This notebook is based on the code from https://mccormickml.com/2019/07/22/BERT-fine-tuning/. Credit to Chris McCormick and Nick Ryan.

### Background

Fine-tuning BERT on our task of interest takes some setup. Although these steps are done for you, please take a moment to look through them and make sure you understand their purpose.

Install the HuggingFace Transformers package that contains the pretrained BERT models.

```
!pip install --upgrade transformers
```

```
Collecting transformers
  Downloading transformers-4.17.0-py3-none-any.whl (3.8 MB)
ent already satisfied: packaging>=20.0 in
/usr/local/lib/python3.7/dist-packages (from transformers) (21.3)
Requirement already satisfied: filelock in
/usr/local/lib/python3.7/dist-packages (from transformers) (3.6.0)
Requirement already satisfied: tqdm>=4.27 in
/usr/local/lib/python3.7/dist-packages (from transformers) (4.63.0)
Requirement already satisfied: numpy>=1.17 in
/usr/local/lib/python3.7/dist-packages (from transformers) (1.21.5)
Collecting sacremoses
  Downloading sacremoses-0.0.49-py3-none-any.whl (895 kB)
ent already satisfied: importlib-metadata in
/usr/local/lib/python3.7/dist-packages (from transformers) (4.11.2)
Collecting pyyaml>=5.1
  Downloading PyYAML-6.0-cp37-cp37m-
manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_12_x86_64.manylinux
2010_x86_64.whl (596 kB)
ent already satisfied: requests in /usr/local/lib/python3.7/dist-
packages (from transformers) (2.23.0)
Collecting tokenizers!=0.11.3,>=0.11.1
  Downloading tokenizers-0.11.6-cp37-cp37m-
```

```
manylinux_2_12_x86_64.manylinux2010_x86_64.whl (6.5 MB)
ent already satisfied: regex!=2019.12.17 in
/usr/local/lib/python3.7/dist-packages (from transformers)
(2019.12.20)
Requirement already satisfied: typing-extensions>=3.7.4.3 in
/usr/local/lib/python3.7/dist-packages (from huggingface-
hub<1.0,>=0.1.0->transformers) (3.10.0.2)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in
/usr/local/lib/python3.7/dist-packages (from packaging>=20.0-
>transformers) (3.0.7)
Requirement already satisfied: zipp>=0.5 in
/usr/local/lib/python3.7/dist-packages (from importlib-metadata-
>transformers) (3.7.0)
Requirement already satisfied: idna<3,>=2.5 in
/usr/local/lib/python3.7/dist-packages (from requests->transformers)
(2.10)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1
in /usr/local/lib/python3.7/dist-packages (from requests-
>transformers) (1.24.3)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.7/dist-packages (from requests->transformers)
(2021.10.8)
Requirement already satisfied: chardet<4,>=3.0.2 in
/usr/local/lib/python3.7/dist-packages (from requests->transformers)
(3.0.4)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-
packages (from sacremoses->transformers) (1.15.0)
Requirement already satisfied: click in /usr/local/lib/python3.7/dist-
packages (from sacremoses->transformers) (7.1.2)
Requirement already satisfied: joblib in
/usr/local/lib/python3.7/dist-packages (from sacremoses->transformers)
(1.1.0)
Installing collected packages: pyyaml, tokenizers, sacremoses,
huggingface-hub, transformers
  Attempting uninstall: pyyaml
    Found existing installation: PyYAML 3.13
    Uninstalling PyYAML-3.13:
      Successfully uninstalled PyYAML-3.13
Successfully installed huggingface-hub-0.4.0 pyyaml-6.0 sacremoses-
0.0.49 tokenizers-0.11.6 transformers-4.17.0
```

Set the random seeds for reproducibility.

```
import os
import random

import numpy as np
import torch

SEED = 42
```

```
torch.manual_seed(SEED)
torch.cuda.manual_seed_all(SEED)
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False
np.random.seed(SEED)
random.seed(SEED)
os.environ['PYTHONHASHSEED'] = str(SEED)
```

Run the following cells to download the verbal arithmetic dataset from the CSC413 webpage and load it into a `DataFrame`

```
!pip install wget
```

```
Collecting wget
  Downloading wget-3.2.zip (10 kB)
Building wheels for collected packages: wget
  Building wheel for wget (setup.py) ... e=wget-3.2-py3-none-any.whl
size=9675
sha256=680307cee4380a314ad0fff12fe37a5ef705fe6130a3dc9de21fedd745aa816
1
  Stored in directory:
/root/.cache/pip/wheels/a1/b6/7c/0e63e34eb06634181c63adacca38b79ff8f35
c37e3c13e3c02
Successfully built wget
Installing collected packages: wget
Successfully installed wget-3.2
```

```
import wget
import os

print('Downloading verbal arithmetic dataset')

# The URL for the dataset zip file.
url = 'https://csc413-uoft.github.io/2021/assets/misc/'

# Download the file (if we haven't already)
if not os.path.exists('./PA03_data_20_train.csv'):
  wget.download(url + 'PA03_data_20_train.csv',
'./PA03_data_20_train.csv')
  print('Done downloading training data')
else:
  print('Already downloaded training data')

if not os.path.exists('./PA03_data_20_test.csv'):
  wget.download(url + 'PA03_data_20_test.csv',
'./PA03_data_20_test.csv')
  print('Done downloading test data')
else:
  print('Already downloaded test data')
```

```
Downloading verbal arithmetic dataset
Done downloading training data
Done downloading test data
```

```python
import pandas as pd

df = pd.read_csv("./PA03_data_20_train.csv", header=0, names=["index",
"input", "label"])

print("Number of data points: ", df.shape[0])
sampled = df.sample(10)
# Display 10 random rows from the data.
df.sample(10)
```

```
Number of data points:  640
```

| | index | input | label |
|---|---|---|---|
| 20 | 264 | thirteen plus four | 2 |
| 180 | 712 | fifteen minus twelve | 2 |
| 270 | 327 | sixteen plus seven | 2 |
| 221 | 58 | two plus eighteen | 2 |
| 542 | 646 | twelve minus six | 2 |
| 143 | 626 | eleven minus six | 2 |
| 247 | 759 | seventeen minus nineteen | 0 |
| 467 | 691 | fourteen minus eleven | 2 |
| 283 | 372 | eighteen plus twelve | 2 |
| 259 | 274 | thirteen plus fourteen | 2 |

### Tokenizer

To feed our text to BERT, it must be split into tokens, and then these tokens must be mapped to their index in the tokenizer vocabulary. For this we can use the `AutoTokenizer` from the `transformers` library.

As mentioned in the assignment handout, we will use MathBERT, which uses the same architecture as BERT, but has been pretrained on text from pre-kindergarten, high-school, and college graduate level mathematical content.

```python
from transformers import AutoTokenizer

bert_tokenizer = AutoTokenizer.from_pretrained('tbs17/MathBERT',
do_lower_case=True)
```

{"version_major":2,"version_minor":0,"model_id":"b7c00bcb2523480cb2681c8816b0a647"}

{"version_major":2,"version_minor":0,"model_id":"d1c5814c3ed34dffb45960558fd2c71e"}

{"version_major":2,"version_minor":0,"model_id":"425cc192d58b4e269ad6f6c95bdfbdb8"}

{"version_major":2,"version_minor":0,"model_id":"bb7dd10304d64f6ebe8f8b4ff7621660"}

```
inputs = df.input.values
labels = df.label.values
print("Train data size ", len(inputs))
print('* Original:  ', inputs[0])
# Print the sentence split into tokens.
print('* Tokenized: ', bert_tokenizer.tokenize(inputs[0]))
# Print the sentence mapped to token ids.
print('* Token IDs: ',
bert_tokenizer.convert_tokens_to_ids(bert_tokenizer.tokenize(inputs[0]
)))
```

```
Train data size  640
* Original:   five minus twelve
* Tokenized:  ['five', 'minus', 'twelve']
* Token IDs:  [2274, 15718, 4376]
```

### Formatting the inputs

In order to use BERT for fine-tuning, we need to format the inputs in a way that matches the inputs of the pretraining step. In short, we need to:

1. Add special tokens to the start and end of each sentence.
2. Pad & truncate all sentences to a single constant length.
3. Explicitly differentiate real tokens from padding tokens with the "attention mask".

#### *Special Tokens*

**`[SEP]`**

At the end of every sentence, we need to append the special `[SEP]` token.

This token is an artifact of two-sentence tasks, where BERT is given two separate sentences and asked to determine something (e.g., can the answer to the question in sentence A be found in sentence B?).

**`[CLS]`**

For classification tasks, we must prepend the special `[CLS]` token to the beginning of every sentence.

This token has special significance. BERT consists of 12 Transformer layers. Each transformer takes in a list of token embeddings, and produces the same number of embeddings on the output.

On the output of the final transformer, *only the first embedding (corresponding to the [CLS] token) is used by the classifier*.

> "The first token of every sequence is always a special classification token ([CLS]). The final hidden state corresponding to this token is used as the aggregate sequence representation for classification tasks." (from the BERT paper)

Also, because BERT is trained to only use this [CLS] token for classification, we know that the model has been motivated to encode everything it needs for the classification step into that single 768-value embedding vector.

*Sentence Length & Attention Mask*

The sentences in our dataset obviously have varying lengths, so how does BERT handle this?

BERT has two constraints:

1. All sentences must be padded or truncated to a single, fixed length.
2. The maximum sentence length is 512 tokens.

Padding is done with a special `[PAD]` token, which is at index 0 in the BERT vocabulary.

The "Attention Mask" is simply an array of 0s and 1s indicating which tokens are padding and which aren't.

In our dataset, all sentences have three word tokens. However, we set the max length of sentence to 7 in this example to show what paddings will be in real world applications.

```
# Set the maximum sequence length.
MAX_LEN = 7

# Print BERTs special PAD token and its index in the vocabulary
print(f'Padding token: "{bert_tokenizer.pad_token}", ID:
{bert_tokenizer.pad_token_id}')
```

```
Padding token: "[PAD]", ID: 0
```

Luckily, the `BertTokenizer` object from the transformers library makes it easy to preprocess our input text correctly

```
tokenized_inputs = bert_tokenizer(
    inputs.tolist(),          # Input text
    add_special_tokens=True,  # add '[CLS]' and '[SEP]'
    padding='max_length',     # pad to a length specified by the
max_length
    max_length=MAX_LEN,       # truncate all sentences longer than
max_length
    return_tensors='pt',      # return everything we need as PyTorch
tensors
)

input_ids = tokenized_inputs['input_ids']
attention_masks = tokenized_inputs['attention_mask']
```

```
# Print sentence 0, now as a list of IDs.
print('Original: ', tokenized_inputs['input_ids'][0])
print('* Token IDs:', tokenized_inputs['attention_mask'][0])
print('* Tokenized:',
bert_tokenizer.decode(tokenized_inputs['input_ids'][0]))
print('* Attention_mask', tokenized_inputs['attention_mask'][0])
```

```
Original:  tensor([  101,  2274, 15718,  4376,   102,     0,     0])
* Token IDs: tensor([1, 1, 1, 1, 1, 0, 0])
* Tokenized: [CLS] five minus twelve [SEP] [PAD] [PAD]
* Attention_mask tensor([1, 1, 1, 1, 1, 0, 0])
```

### Training & Validation Split

Let's divide up our data into a train set (80%) and a validation set (20%).

We'also create an iterator for our dataset using the torch `DataLoader` class. This helps save on memory during training because, unlike a for loop, with an iterator the entire dataset does not need to be loaded into memory.

```
from sklearn.model_selection import train_test_split
import torch
from torch.utils.data import TensorDataset, DataLoader, RandomSampler,
SequentialSampler

def train_valid_split(input_ids, attention_masks, labels,
batch_size=32):
    # Use 80% for training and 20% for validation.
    train_inputs, validation_inputs,  train_masks, validation_masks,
train_labels, validation_labels = train_test_split(
        input_ids, attention_masks, labels, random_state=SEED,
test_size=0.2, stratify=labels
    )

    print('example train_input:    ', train_inputs[0])
    print('example attention_mask: ', train_masks[0])

    train_labels = torch.tensor(train_labels)
    validation_labels = torch.tensor(validation_labels)

    # Create the DataLoader for our training set.
    train_data = TensorDataset(train_inputs, train_masks,
train_labels)
    train_dataloader = DataLoader(train_data, shuffle=True,
batch_size=batch_size)

    # Create the DataLoader for our validation set.
    validation_data = TensorDataset(validation_inputs,
validation_masks, validation_labels)
```

```
    validation_dataloader = DataLoader(validation_data, shuffle=False,
batch_size=batch_size)

    return train_dataloader, validation_dataloader

bert_train_dataloader, bert_validation_dataloader = train_valid_split(
    input_ids=input_ids,
    attention_masks=attention_masks,
    labels=labels,
    batch_size=32
)

example train_input:     tensor([  101, 11977, 15718,  4376,   102,
0,     0])
example attention_mask:  tensor([1, 1, 1, 1, 1, 0, 0])
```

## Questions

### Question 1: Add a classifier to BERT [1pts]

Here, we will add a simple classifier to the BertModel provided by the Transformers library.

Your tasks are:

1. In `__init__`, add a linear classifier that will map BERTs `[CLS]` token representation to the unnormalized output probabilities for each class (`logits`).
2. In `forward`, pass BERTs `[CLS]` token representation to this new classifier to produce the `logits`.

In total, you won't have to write more than three new lines of code. See the comments in the code for help!

```
from transformers import BertModel
import torch.nn as nn

class BertForSentenceClassification(BertModel):
    def __init__(self, config):
        super().__init__(config)

        ##### START YOUR CODE HERE #####
        # Add a linear classifier that map BERTs [CLS] token
representation to the unnormalized
        # output probabilities for each class (logits).
        # Notes:
        #  * See the documentation for torch.nn.Linear
        #  * You do not need to add a softmax, as this is included in
the loss function
        #  * The size of BERTs token representation can be accessed at
config.hidden_size
        #  * The number of output classes can be accessed at
```

```
config.num_labels
        self.classifier = torch.nn.Linear(config.hidden_size,
config.num_labels)
        ##### END YOUR CODE HERE #####
        self.loss = torch.nn.CrossEntropyLoss()


    def forward(self, labels=None, **kwargs):
        outputs = super().forward(**kwargs)
        ##### START YOUR CODE HERE #####
        # Pass BERTs [CLS] token representation to this new classifier
to produce the logits.
        # Notes:
        #  * The [CLS] token representation can be accessed at
outputs.pooler_output
        cls_token_repr = outputs.pooler_output
        logits = self.classifier(cls_token_repr)
        ##### END YOUR CODE HERE #####
        if labels is not None:
            outputs = (logits, self.loss(logits, labels))
        else:
            outputs = (logits,)
        return outputs
```

### Question 2: Fine-tune BERT [0pts]

In this section, we will instantiate our pretrained BERT model + the new classifier, and train both on our verbal arithmetic dataset for a few epochs.

As mentioned in the assignment handout, we will use MathBERT, which uses the same architecture as BERT, but has been pretrained on text from pre-kindergarten, high-school, and college graduate level mathematical content.

> Although the code is written for you, please read it first to understand what it is doing. Additionally, running this code and making sure the model can be fine-tuned helps you check your implementation from Question 1. **Note**: This may print a warning: *"Some weights of the model checkpoint at..."* which you can ignore.

```
mathbert = BertForSentenceClassification.from_pretrained(
    "tbs17/MathBERT",  # the name of the pretrained model
    num_labels=3,      # the number of classes in our downstream task
)
```


```
{"version_major":2,"version_minor":0,"model_id":"9ec178a6d4114b6c9be92
f4c90a1eb71"}
```


```
Some weights of the model checkpoint at tbs17/MathBERT were not used
when initializing BertForSentenceClassification:
['cls.predictions.transform.LayerNorm.bias', 'cls.predictions.bias',
'cls.seq_relationship.weight', 'cls.predictions.decoder.bias',
'cls.predictions.transform.dense.weight',
```

```
'cls.predictions.transform.LayerNorm.weight',
'cls.seq_relationship.bias', 'cls.predictions.transform.dense.bias',
'cls.predictions.decoder.weight']
- This IS expected if you are initializing
BertForSentenceClassification from the checkpoint of a model trained
on another task or with another architecture (e.g. initializing a
BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing
BertForSentenceClassification from the checkpoint of a model that you
expect to be exactly identical (initializing a
BertForSequenceClassification model from a
BertForSequenceClassification model).
Some weights of BertForSentenceClassification were not initialized
from the model checkpoint at tbs17/MathBERT and are newly initialized:
['bert.classifier.bias', 'bert.classifier.weight']
You should probably TRAIN this model on a down-stream task to be able
to use it for predictions and inference.
```

The following cell prints information about the models parameters

```python
# Model parameters visualization
params = list(mathbert.named_parameters())

print('The BERT model has {:} different named parameters.\
n'.format(len(params)))

print('==== Embedding Layer ====\n')

for p in params[0:5]:
    print("{:<55} {:>12}".format(p[0], str(tuple(p[1].size()))))

print('\n==== First Transformer Layer ====\n')

for p in params[5:21]:
    print("{:<55} {:>12}".format(p[0], str(tuple(p[1].size()))))

print('\n==== Output Layer ====\n')

for p in params[-4:]:
    print("{:<55} {:>12}".format(p[0], str(tuple(p[1].size()))))
```

```
The BERT model has 201 different named parameters.

==== Embedding Layer ====

embeddings.word_embeddings.weight                         (30522, 768)
embeddings.position_embeddings.weight                       (512, 768)
embeddings.token_type_embeddings.weight                       (2, 768)
embeddings.LayerNorm.weight                                     (768,)
embeddings.LayerNorm.bias                                       (768,)
```

```
==== First Transformer Layer ====

encoder.layer.0.attention.self.query.weight               (768, 768)
encoder.layer.0.attention.self.query.bias                     (768,)
encoder.layer.0.attention.self.key.weight                 (768, 768)
encoder.layer.0.attention.self.key.bias                       (768,)
encoder.layer.0.attention.self.value.weight               (768, 768)
encoder.layer.0.attention.self.value.bias                     (768,)
encoder.layer.0.attention.output.dense.weight             (768, 768)
encoder.layer.0.attention.output.dense.bias                   (768,)
encoder.layer.0.attention.output.LayerNorm.weight             (768,)
encoder.layer.0.attention.output.LayerNorm.bias               (768,)
encoder.layer.0.intermediate.dense.weight                (3072, 768)
encoder.layer.0.intermediate.dense.bias                      (3072,)
encoder.layer.0.output.dense.weight                      (768, 3072)
encoder.layer.0.output.dense.bias                             (768,)
encoder.layer.0.output.LayerNorm.weight                       (768,)
encoder.layer.0.output.LayerNorm.bias                         (768,)

==== Output Layer ====

pooler.dense.weight                                       (768, 768)
pooler.dense.bias                                             (768,)
classifier.weight                                           (3, 768)
classifier.bias                                                 (3,)
```

The next cell defines fairly standard train and evaluation loops in PyTorch

```
from torch.optim import AdamW
import time
import datetime
from transformers import get_linear_schedule_with_warmup
from tqdm import tqdm

def flat_accuracy(preds, labels):
    pred_flat = np.argmax(preds, axis=1).flatten()
    labels_flat = labels.flatten()
    return np.sum(pred_flat == labels_flat) / len(labels_flat)

def format_time(elapsed):
    elapsed_rounded = int(round((elapsed)))
    return str(datetime.timedelta(seconds=elapsed_rounded))

def get_optimizer_and_scheduler(model, total_steps, lr=2e-5,
weight_decay=0.01):
    # Apply weight decay to all parameters beside the biases or
LayerNorm weights
    no_decay = ['bias', 'LayerNorm.weight']
    optimizer_grouped_parameters = [
```

```
        {
            'params': [p for n, p in model.named_parameters() if not
any(nd in n for nd in no_decay)],
            'weight_decay': weight_decay},
        {
            'params': [p for n, p in model.named_parameters() if
any(nd in n for nd in no_decay)],
            'weight_decay': 0.0
        }
    ]
    optimizer = AdamW(model.parameters(), lr=lr)
    scheduler = get_linear_schedule_with_warmup(
        optimizer,
        # Warmup learning rate for first 10% of training steps
        num_warmup_steps=int(0.10 * total_steps),
        num_training_steps=total_steps,
    )
    return optimizer, scheduler

def train_model(model, epochs, train_dataloader,
validation_dataloader):
    # Use GPU, if available
    device = torch.device("cuda" if torch.cuda.is_available() else
"cpu")
    model = model.to(device)

    # Setup optimizer and LR scheduler
    total_steps = len(train_dataloader) * epochs
    optimizer, scheduler = get_optimizer_and_scheduler(
        model, total_steps, lr=5e-5, weight_decay=0.01
    )

    loss_values = []
    eval_accs = []

    for epoch in range(0, epochs):
        t0 = time.time()

        total_loss = 0
        model.train()

        with tqdm(train_dataloader, unit="batch") as train_pbar:
            for batch in train_pbar:
                train_pbar.set_description(f"Training (epoch {epoch +
1})")
                b_input_ids = batch[0].to(device)
                b_input_mask = batch[1].to(device)
                b_labels = batch[2].to(device)
```

```
            model.zero_grad()

            # Perform a forward pass (evaluate the model on this
training batch).
            # This will return the loss because we have provided
the `labels`.
            outputs = model(
                input_ids=b_input_ids,
                attention_mask=b_input_mask,
                labels=b_labels
            )

            # The call to `model` always returns a tuple, so we
need to pull the
            # loss value out of the tuple.
            _, loss = outputs

            # Accumulate the training loss over all of the batches
so that we can
            # calculate the average loss at the end. `loss` is a
Tensor containing a
            # single value; the `.item()` function just returns
the Python value
            # from the tensor.
            total_loss += loss.item()

            # Perform a backward pass to calculate the gradients.
            loss.backward()

            # Clip the norm of the gradients to 1.0.
            # This is to help prevent the "exploding gradients"
problem.
            torch.nn.utils.clip_grad_norm_(model.parameters(),
1.0)

            # Update parameters and take a step using the computed
gradient.
            # The optimizer dictates the "update rule"--how the
parameters are
            # modified based on their gradients, the learning
rate, etc.
            optimizer.step()

            # Update the learning rate.
            scheduler.step()

        # Calculate the average loss over the training data.
        avg_train_loss = total_loss / len(train_dataloader)
```

```
        # Store the loss value for plotting the learning curve.
        loss_values.append(avg_train_loss)

        print("  * Average training loss:
{0:.2f}".format(avg_train_loss))
        print("  * Training epoch took:
{:}".format(format_time(time.time() - t0)))

        print("Running Validation...")

        t0 = time.time()
        model.eval()

        eval_loss, eval_accuracy = 0, 0
        nb_eval_steps, nb_eval_examples = 0, 0

        # Evaluate data for one epoch
        for batch in validation_dataloader:
            batch = tuple(t.to(device) for t in batch)
            b_input_ids, b_input_mask, b_labels = batch

            with torch.no_grad():
                # Forward pass, calculate logit predictions.
                # This will return the logits rather than the loss
because we have
                # not provided labels.
                # token_type_ids is the same as the "segment ids",
which
                # differentiates sentence 1 and 2 in 2-sentence tasks.
                outputs = model(
                    input_ids=b_input_ids,
                    attention_mask=b_input_mask
                )

            # Get the "logits" output by the model. The "logits" are
the output
            # values prior to applying an activation function like the
softmax.
            logits = outputs[0]
            # Move logits and labels to CPU
            logits = logits.detach().cpu().numpy()
            label_ids = b_labels.to('cpu').numpy()
            # Calculate the accuracy for this batch of test sentences.
            tmp_eval_accuracy = flat_accuracy(logits, label_ids)
            # Accumulate the total accuracy.
            eval_accuracy += tmp_eval_accuracy
            # Track the number of batches
            nb_eval_steps += 1
```

```
        avg_eval_acc = eval_accuracy/nb_eval_steps
        print("  * Accuracy: {0:.2f}".format(avg_eval_acc))
        print("  * Validation took:
{:}".format(format_time(time.time() - t0)))
        eval_accs.append(avg_eval_acc)
    print("Training complete!")
    return loss_values, eval_accs
```

Finally, run the following cell to fine-tune the model

```
# About 2-3 seconds per epoch using GPU
mathbert_loss_vals, mathbert_eval_accs = train_model(
    model=mathbert,
    epochs=3,
    train_dataloader=bert_train_dataloader,
    validation_dataloader=bert_validation_dataloader
)
```

```
Training (epoch 1): 100%|██████████| 16/16 [00:03<00:00,  4.78batch/s]

  * Average training loss: 0.68
  * Training epoch took: 0:00:03
Running Validation...
  * Accuracy: 0.85
  * Validation took: 0:00:00

Training (epoch 2): 100%|██████████| 16/16 [00:03<00:00,  5.02batch/s]

  * Average training loss: 0.37
  * Training epoch took: 0:00:03
Running Validation...
  * Accuracy: 0.91
  * Validation took: 0:00:00

Training (epoch 3): 100%|██████████| 16/16 [00:03<00:00,  4.99batch/s]

  * Average training loss: 0.22
  * Training epoch took: 0:00:03
Running Validation...
  * Accuracy: 0.90
  * Validation took: 0:00:00
Training complete!
```

Once the model is trained, we can plot some performance metrics

```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns

def plot_loss_and_acc(loss_vals, eval_accs):
    sns.set(style='darkgrid')
    sns.set(font_scale=1.5)
```

```
    plt.rcParams["figure.figsize"] = (12,6)
    fig, ax1 = plt.subplots(1,1)
    ax1.plot(loss_vals, 'b-o', label = 'training loss')
    ax2 = ax1.twinx()
    ax2.plot(eval_accs, 'y-o', label = 'validation accuracy')
    ax2.set_title("Training loss and validation accuracy")
    ax2.set_xlabel("Epoch")
    ax1.set_ylabel("Loss", color='b')
    ax2.set_ylabel("Accuracy", color='y')
    ax1.tick_params(axis='y', rotation=0, labelcolor='b' )
    ax2.tick_params(axis='y', rotation=0, labelcolor='y' )
    plt.show()
```

```
plot_loss_and_acc(mathbert_loss_vals, mathbert_eval_accs)
```


### Question 3: Freezing the pretrained weights [0.5pts]

Now, lets try training the model again, except this time we will *not* fine-tune BERTs weights (we sometimes say these weights are "frozen"). To do this, we will only compute gradients for the classifiers parameters.

> We can do this in pytorch by setting the `requires_grad` attribute to `False` for all parameters beside the classifiers.

Run the following cells to instantiate the model and train only the classifier. Then answer the follow-up questions in the assignment handout.

> **Note**: This may print a warning: *"Some weights of the model checkpoint at..."* which you can ignore.

```
mathbert_frozen = BertForSentenceClassification.from_pretrained(
    "tbs17/MathBERT",  # the name of the pretrained model
    num_labels=3,      # the number of classes in our downstream task
)
```

```
Some weights of the model checkpoint at tbs17/MathBERT were not used
when initializing BertForSentenceClassification:
['cls.predictions.transform.LayerNorm.bias', 'cls.predictions.bias',
'cls.seq_relationship.weight', 'cls.predictions.decoder.bias',
'cls.predictions.transform.dense.weight',
'cls.predictions.transform.LayerNorm.weight',
'cls.seq_relationship.bias', 'cls.predictions.transform.dense.bias',
'cls.predictions.decoder.weight']
- This IS expected if you are initializing
BertForSentenceClassification from the checkpoint of a model trained
on another task or with another architecture (e.g. initializing a
BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing
BertForSentenceClassification from the checkpoint of a model that you
expect to be exactly identical (initializing a
BertForSequenceClassification model from a
BertForSequenceClassification model).
Some weights of BertForSentenceClassification were not initialized
from the model checkpoint at tbs17/MathBERT and are newly initialized:
['bert.classifier.bias', 'bert.classifier.weight']
You should probably TRAIN this model on a down-stream task to be able
to use it for predictions and inference.
```

```python
for name, param in mathbert_frozen.named_parameters():
     # Only compute gradients for parameters of our
     # newly added classifier. BERT will not be trained.
     if 'classifier' not in name:
          param.requires_grad = False

# About 1 second per epoch on GPU
mathbert_frozen_loss_vals, mathbert_frozen_eval_accs = train_model(
    model=mathbert_frozen,
    epochs=3,
    train_dataloader=bert_train_dataloader,
    validation_dataloader=bert_validation_dataloader
)
```

```
Training (epoch 1): 100%|██████████| 16/16 [00:00<00:00, 16.33batch/s]

  * Average training loss: 1.36
  * Training epoch took: 0:00:01
Running Validation...
  * Accuracy: 0.05
  * Validation took: 0:00:00

Training (epoch 2): 100%|██████████| 16/16 [00:00<00:00, 17.70batch/s]

  * Average training loss: 1.19
  * Training epoch took: 0:00:01
Running Validation...
```

```
  * Accuracy: 0.12
  * Validation took: 0:00:00

Training (epoch 3): 100%|██████████| 16/16 [00:00<00:00, 17.73batch/s]

  * Average training loss: 1.11
  * Training epoch took: 0:00:01
Running Validation...
  * Accuracy: 0.30
  * Validation took: 0:00:00
Training complete!
```

```
plot_loss_and_acc(mathbert_frozen_loss_vals,
mathbert_frozen_eval_accs)
```


*Response to Question 3*

It takes about 3 seconds to train when using fine-tuning and 1 second to train when BERTs weights are frozen. As fine-turning without freezing the pre-trained weights has more weights, it would take more time to compute and updates all the weight parameters.

However, as no layers are frozon in fine-tuning, all the paramters are adjusted for this single sentence classification task. Therefore, the validation accuracy with fine-tuning is much higher than training with pretrained weights frozen.

### Question 4: Effect of pretraining data [0.5pts]

Now, let's try fine-tuning the model again, except this time we will use BERTweets pretrained weights. BERTweets uses the same architecture as BERT (and MathBERT), but has been pretrained on 100s of millions of *tweets*.

Run the following cells to instantiate our model with BERTweets pretrained weights and fine-tune it. Then answer the follow-up questions in the assignment handout.

**Note**: This may print a warning: *"You are using a model of type..."* which you can ignore.

```
bertweet = BertForSentenceClassification.from_pretrained(
    "vinai/bertweet-base",  # the name of the pretrained model
    num_labels=3,           # the number of classes in our downstream
task
)
```

{"version_major":2,"version_minor":0,"model_id":"38f5446897ef411bb1553
398468ba266"}

```
You are using a model of type roberta to instantiate a model of type
bert. This is not supported for all configurations of models and can
yield errors.
```

{"version_major":2,"version_minor":0,"model_id":"d502f19f09e5417fadf95
b5d58687fcc"}

```
Some weights of the model checkpoint at vinai/bertweet-base were not
used when initializing BertForSentenceClassification:
['roberta.encoder.layer.10.intermediate.dense.weight',
'roberta.encoder.layer.9.intermediate.dense.weight',
'roberta.encoder.layer.10.attention.output.LayerNorm.weight',
'roberta.encoder.layer.2.attention.output.dense.weight',
'roberta.encoder.layer.2.output.dense.bias',
'roberta.encoder.layer.9.attention.self.value.weight',
'roberta.encoder.layer.11.attention.output.LayerNorm.bias',
'roberta.encoder.layer.6.output.LayerNorm.bias',
'roberta.encoder.layer.3.attention.output.LayerNorm.bias',
'lm_head.decoder.weight',
'roberta.encoder.layer.3.output.LayerNorm.weight',
'roberta.encoder.layer.3.output.dense.weight',
'roberta.encoder.layer.1.output.LayerNorm.bias',
'roberta.encoder.layer.9.output.dense.bias',
'roberta.encoder.layer.11.attention.self.key.bias',
'roberta.encoder.layer.3.output.LayerNorm.bias',
'roberta.encoder.layer.9.intermediate.dense.bias',
'roberta.encoder.layer.5.intermediate.dense.bias',
'roberta.encoder.layer.2.attention.output.dense.bias',
'roberta.encoder.layer.0.attention.self.value.weight',
'roberta.encoder.layer.2.output.LayerNorm.weight',
'roberta.encoder.layer.5.output.dense.bias',
'roberta.encoder.layer.5.output.LayerNorm.bias',
'roberta.encoder.layer.7.attention.self.key.weight',
'roberta.encoder.layer.7.attention.self.value.weight',
'roberta.encoder.layer.11.attention.self.query.weight',
'roberta.encoder.layer.9.attention.output.dense.weight',
'roberta.encoder.layer.2.output.dense.weight',
'roberta.encoder.layer.2.attention.self.value.bias',
'roberta.encoder.layer.3.attention.output.LayerNorm.weight',
```

```
'roberta.encoder.layer.9.attention.output.LayerNorm.bias',
'roberta.encoder.layer.1.attention.self.value.weight',
'roberta.encoder.layer.9.attention.self.key.bias',
'roberta.encoder.layer.3.intermediate.dense.weight',
'roberta.encoder.layer.5.output.LayerNorm.weight',
'roberta.encoder.layer.3.output.dense.bias',
'roberta.embeddings.token_type_embeddings.weight',
'roberta.encoder.layer.1.attention.output.dense.weight',
'roberta.encoder.layer.3.attention.output.dense.bias',
'roberta.encoder.layer.6.attention.self.value.weight',
'roberta.encoder.layer.4.attention.self.query.weight',
'roberta.encoder.layer.9.attention.output.dense.bias',
'roberta.encoder.layer.10.attention.self.query.weight',
'roberta.encoder.layer.1.attention.self.value.bias',
'roberta.encoder.layer.10.attention.output.dense.bias',
'roberta.pooler.dense.weight',
'roberta.encoder.layer.7.output.LayerNorm.bias',
'roberta.encoder.layer.1.intermediate.dense.weight',
'roberta.encoder.layer.0.output.dense.bias',
'roberta.encoder.layer.1.output.dense.weight',
'roberta.encoder.layer.5.attention.self.key.bias',
'roberta.encoder.layer.6.output.LayerNorm.weight',
'roberta.encoder.layer.6.attention.output.dense.bias',
'roberta.encoder.layer.0.attention.self.key.weight',
'roberta.encoder.layer.9.attention.self.key.weight',
'roberta.encoder.layer.11.output.dense.bias',
'roberta.encoder.layer.7.attention.output.LayerNorm.bias',
'roberta.encoder.layer.10.output.dense.weight',
'roberta.encoder.layer.2.attention.self.value.weight',
'roberta.encoder.layer.1.attention.output.dense.bias',
'roberta.encoder.layer.11.output.LayerNorm.weight',
'roberta.encoder.layer.4.attention.self.value.bias',
'roberta.encoder.layer.7.intermediate.dense.weight',
'roberta.encoder.layer.4.attention.self.query.bias',
'roberta.encoder.layer.11.attention.self.value.weight',
'roberta.encoder.layer.2.attention.output.LayerNorm.bias',
'roberta.encoder.layer.0.attention.output.dense.bias',
'roberta.encoder.layer.7.attention.self.value.bias',
'roberta.encoder.layer.3.attention.self.key.bias',
'roberta.encoder.layer.0.output.dense.weight',
'roberta.encoder.layer.4.intermediate.dense.weight',
'roberta.encoder.layer.6.output.dense.bias',
'roberta.encoder.layer.2.intermediate.dense.weight',
'roberta.encoder.layer.4.attention.output.dense.weight',
'roberta.encoder.layer.8.attention.output.dense.weight',
'roberta.encoder.layer.4.attention.self.value.weight',
'roberta.encoder.layer.0.attention.self.key.bias',
'roberta.encoder.layer.8.attention.self.value.bias',
'roberta.encoder.layer.8.output.LayerNorm.weight',
'roberta.encoder.layer.10.attention.self.value.weight',
```

```
'roberta.encoder.layer.8.attention.self.key.bias',
'roberta.encoder.layer.8.attention.output.LayerNorm.bias',
'roberta.encoder.layer.8.intermediate.dense.weight',
'lm_head.layer_norm.weight',
'roberta.encoder.layer.6.attention.output.LayerNorm.weight',
'roberta.encoder.layer.8.output.LayerNorm.bias',
'roberta.encoder.layer.6.attention.self.query.bias',
'roberta.encoder.layer.6.attention.output.dense.weight',
'roberta.encoder.layer.10.attention.self.key.weight',
'roberta.encoder.layer.5.attention.output.LayerNorm.bias',
'roberta.encoder.layer.6.intermediate.dense.bias',
'roberta.encoder.layer.5.attention.self.value.bias',
'roberta.encoder.layer.10.output.LayerNorm.weight',
'roberta.encoder.layer.11.output.dense.weight',
'roberta.encoder.layer.3.attention.self.value.weight',
'roberta.encoder.layer.1.attention.output.LayerNorm.weight',
'roberta.encoder.layer.10.output.dense.bias',
'roberta.encoder.layer.2.intermediate.dense.bias',
'roberta.encoder.layer.4.intermediate.dense.bias',
'roberta.encoder.layer.6.intermediate.dense.weight',
'roberta.encoder.layer.4.attention.self.key.bias',
'roberta.encoder.layer.5.intermediate.dense.weight',
'roberta.encoder.layer.7.output.dense.weight',
'roberta.encoder.layer.10.intermediate.dense.bias',
'roberta.encoder.layer.0.output.LayerNorm.weight',
'roberta.encoder.layer.4.output.LayerNorm.bias',
'roberta.encoder.layer.7.attention.output.dense.bias',
'roberta.encoder.layer.3.attention.self.query.bias',
'roberta.encoder.layer.4.attention.output.LayerNorm.weight',
'roberta.encoder.layer.1.attention.output.LayerNorm.bias',
'roberta.encoder.layer.8.attention.output.dense.bias',
'roberta.encoder.layer.9.attention.self.value.bias',
'roberta.encoder.layer.7.attention.self.query.bias',
'roberta.pooler.dense.bias', 'roberta.embeddings.LayerNorm.weight',
'roberta.encoder.layer.5.attention.self.value.weight',
'roberta.encoder.layer.0.output.LayerNorm.bias',
'roberta.encoder.layer.2.attention.self.query.bias',
'roberta.embeddings.position_ids',
'roberta.encoder.layer.1.intermediate.dense.bias',
'roberta.encoder.layer.11.attention.output.dense.bias',
'roberta.encoder.layer.8.output.dense.bias',
'roberta.encoder.layer.0.attention.output.LayerNorm.bias',
'roberta.encoder.layer.5.attention.output.LayerNorm.weight',
'roberta.encoder.layer.11.intermediate.dense.bias',
'roberta.encoder.layer.1.attention.self.key.weight',
'roberta.encoder.layer.4.attention.output.LayerNorm.bias',
'roberta.encoder.layer.7.output.LayerNorm.weight',
'roberta.encoder.layer.6.attention.self.key.weight',
'roberta.encoder.layer.8.attention.self.query.weight',
'lm_head.dense.bias',
```

```
'roberta.encoder.layer.6.attention.output.LayerNorm.bias',
'roberta.encoder.layer.3.attention.self.query.weight',
'roberta.encoder.layer.11.attention.output.LayerNorm.weight',
'roberta.encoder.layer.2.output.LayerNorm.bias',
'roberta.encoder.layer.6.attention.self.query.weight',
'roberta.embeddings.LayerNorm.bias',
'roberta.encoder.layer.11.intermediate.dense.weight',
'roberta.encoder.layer.7.attention.output.dense.weight',
'roberta.encoder.layer.9.attention.output.LayerNorm.weight',
'roberta.encoder.layer.10.attention.output.dense.weight',
'lm_head.layer_norm.bias',
'roberta.encoder.layer.4.attention.self.key.weight',
'roberta.encoder.layer.4.attention.output.dense.bias',
'roberta.encoder.layer.4.output.dense.weight',
'roberta.encoder.layer.2.attention.output.LayerNorm.weight',
'roberta.encoder.layer.1.attention.self.query.weight',
'roberta.encoder.layer.5.attention.self.key.weight',
'roberta.encoder.layer.0.attention.output.LayerNorm.weight',
'roberta.encoder.layer.1.output.LayerNorm.weight',
'roberta.encoder.layer.6.attention.self.value.bias',
'roberta.encoder.layer.0.attention.self.value.bias',
'roberta.encoder.layer.7.intermediate.dense.bias',
'roberta.encoder.layer.9.output.LayerNorm.bias',
'roberta.encoder.layer.10.attention.output.LayerNorm.bias',
'roberta.encoder.layer.6.output.dense.weight',
'roberta.encoder.layer.4.output.LayerNorm.weight',
'roberta.encoder.layer.11.attention.self.value.bias',
'roberta.encoder.layer.6.attention.self.key.bias',
'roberta.encoder.layer.5.attention.output.dense.bias',
'roberta.encoder.layer.9.attention.self.query.bias',
'roberta.encoder.layer.10.output.LayerNorm.bias',
'roberta.encoder.layer.3.attention.output.dense.weight',
'roberta.encoder.layer.1.attention.self.query.bias',
'roberta.encoder.layer.0.attention.self.query.bias',
'roberta.encoder.layer.8.attention.self.query.bias',
'roberta.encoder.layer.5.attention.output.dense.weight',
'roberta.encoder.layer.11.attention.self.query.bias',
'roberta.encoder.layer.7.attention.output.LayerNorm.weight',
'roberta.encoder.layer.8.intermediate.dense.bias',
'roberta.encoder.layer.2.attention.self.key.weight',
'roberta.encoder.layer.5.attention.self.query.weight',
'roberta.encoder.layer.7.output.dense.bias',
'roberta.encoder.layer.8.attention.self.key.weight',
'roberta.encoder.layer.11.output.LayerNorm.bias',
'roberta.encoder.layer.0.attention.output.dense.weight',
'roberta.encoder.layer.9.output.LayerNorm.weight',
'roberta.encoder.layer.1.output.dense.bias',
'roberta.encoder.layer.4.output.dense.bias',
'roberta.encoder.layer.5.output.dense.weight',
'roberta.encoder.layer.10.attention.self.key.bias',
```

```
'roberta.encoder.layer.8.output.dense.weight',
'roberta.encoder.layer.5.attention.self.query.bias',
'roberta.encoder.layer.0.attention.self.query.weight',
'roberta.encoder.layer.10.attention.self.value.bias',
'roberta.encoder.layer.9.attention.self.query.weight',
'roberta.encoder.layer.11.attention.self.key.weight',
'roberta.encoder.layer.0.intermediate.dense.weight',
'roberta.embeddings.position_embeddings.weight',
'roberta.encoder.layer.7.attention.self.key.bias',
'roberta.encoder.layer.7.attention.self.query.weight',
'lm_head.dense.weight', 'roberta.embeddings.word_embeddings.weight',
'roberta.encoder.layer.3.intermediate.dense.bias',
'roberta.encoder.layer.2.attention.self.key.bias',
'roberta.encoder.layer.3.attention.self.key.weight',
'roberta.encoder.layer.8.attention.output.LayerNorm.weight',
'roberta.encoder.layer.2.attention.self.query.weight',
'roberta.encoder.layer.9.output.dense.weight',
'roberta.encoder.layer.8.attention.self.value.weight',
'roberta.encoder.layer.3.attention.self.value.bias',
'roberta.encoder.layer.11.attention.output.dense.weight',
'roberta.encoder.layer.1.attention.self.key.bias', 'lm_head.bias',
'roberta.encoder.layer.0.intermediate.dense.bias',
'roberta.encoder.layer.10.attention.self.query.bias',
'lm_head.decoder.bias']
- This IS expected if you are initializing
BertForSentenceClassification from the checkpoint of a model trained
on another task or with another architecture (e.g. initializing a
BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing
BertForSentenceClassification from the checkpoint of a model that you
expect to be exactly identical (initializing a
BertForSequenceClassification model from a
BertForSequenceClassification model).
Some weights of BertForSentenceClassification were not initialized
from the model checkpoint at vinai/bertweet-base and are newly
initialized: ['encoder.layer.11.output.dense.weight',
'encoder.layer.0.output.LayerNorm.bias',
'encoder.layer.3.output.LayerNorm.bias',
'encoder.layer.11.output.LayerNorm.bias',
'encoder.layer.1.attention.self.key.bias',
'encoder.layer.6.attention.output.dense.weight',
'encoder.layer.3.attention.output.LayerNorm.weight',
'embeddings.word_embeddings.weight',
'encoder.layer.5.attention.output.LayerNorm.weight',
'encoder.layer.11.attention.output.dense.weight',
'encoder.layer.10.attention.self.key.weight',
'encoder.layer.4.attention.self.key.weight',
'embeddings.LayerNorm.bias',
'embeddings.token_type_embeddings.weight',
'encoder.layer.0.attention.self.key.weight',
```

```
'encoder.layer.1.attention.output.LayerNorm.weight',
'encoder.layer.11.attention.self.value.weight',
'encoder.layer.9.output.dense.weight',
'encoder.layer.8.intermediate.dense.weight',
'encoder.layer.0.attention.self.key.bias',
'encoder.layer.4.output.LayerNorm.weight',
'encoder.layer.2.attention.self.value.bias',
'encoder.layer.3.attention.self.query.weight',
'encoder.layer.7.attention.self.key.weight',
'encoder.layer.8.attention.output.LayerNorm.bias',
'encoder.layer.8.attention.self.key.bias',
'encoder.layer.0.attention.output.LayerNorm.bias',
'encoder.layer.0.attention.self.query.weight',
'encoder.layer.2.intermediate.dense.bias',
'encoder.layer.3.output.LayerNorm.weight',
'encoder.layer.8.attention.output.dense.bias',
'encoder.layer.7.attention.self.value.weight',
'encoder.layer.1.output.LayerNorm.bias',
'encoder.layer.7.output.dense.weight',
'encoder.layer.11.intermediate.dense.weight',
'encoder.layer.11.output.LayerNorm.weight', 'pooler.dense.weight',
'encoder.layer.1.intermediate.dense.weight',
'encoder.layer.4.attention.output.LayerNorm.bias',
'encoder.layer.6.attention.self.value.bias',
'encoder.layer.6.output.dense.bias',
'encoder.layer.7.attention.self.query.bias',
'encoder.layer.2.attention.self.query.bias',
'encoder.layer.0.attention.output.dense.weight',
'encoder.layer.6.attention.output.LayerNorm.bias',
'encoder.layer.10.output.LayerNorm.bias',
'encoder.layer.8.attention.output.dense.weight',
'encoder.layer.2.attention.self.key.weight',
'encoder.layer.11.attention.self.query.weight',
'encoder.layer.2.attention.output.LayerNorm.bias',
'encoder.layer.1.attention.self.query.weight',
'encoder.layer.10.intermediate.dense.bias',
'encoder.layer.11.attention.self.key.bias',
'encoder.layer.10.intermediate.dense.weight',
'encoder.layer.11.attention.self.key.weight',
'encoder.layer.0.attention.self.value.bias',
'encoder.layer.5.output.LayerNorm.bias',
'encoder.layer.3.output.dense.weight',
'encoder.layer.4.attention.self.value.weight',
'encoder.layer.11.attention.output.dense.bias',
'encoder.layer.8.intermediate.dense.bias',
'encoder.layer.0.attention.output.LayerNorm.weight',
'encoder.layer.1.output.dense.weight',
'encoder.layer.8.attention.self.key.weight',
'encoder.layer.4.output.dense.weight',
'encoder.layer.1.attention.self.query.bias',
```

```
'encoder.layer.10.attention.output.LayerNorm.bias',
'encoder.layer.5.attention.self.key.bias',
'encoder.layer.3.attention.output.dense.weight',
'encoder.layer.1.attention.self.key.weight',
'encoder.layer.3.attention.output.LayerNorm.bias',
'encoder.layer.3.attention.self.value.weight',
'encoder.layer.6.output.LayerNorm.bias',
'encoder.layer.8.attention.output.LayerNorm.weight',
'encoder.layer.4.attention.self.query.weight',
'encoder.layer.0.intermediate.dense.weight',
'encoder.layer.4.output.dense.bias',
'encoder.layer.8.attention.self.query.bias',
'encoder.layer.2.output.dense.bias',
'encoder.layer.7.attention.output.LayerNorm.weight',
'encoder.layer.6.intermediate.dense.bias',
'encoder.layer.2.attention.self.key.bias',
'encoder.layer.10.output.dense.weight',
'encoder.layer.2.attention.output.LayerNorm.weight',
'encoder.layer.7.attention.output.dense.bias',
'encoder.layer.9.output.LayerNorm.weight',
'encoder.layer.2.attention.self.value.weight',
'encoder.layer.8.attention.self.value.bias',
'encoder.layer.4.attention.output.dense.bias',
'encoder.layer.3.intermediate.dense.weight',
'encoder.layer.0.attention.self.value.weight',
'encoder.layer.3.attention.self.query.bias',
'encoder.layer.5.intermediate.dense.bias',
'encoder.layer.11.attention.output.LayerNorm.bias',
'encoder.layer.8.output.LayerNorm.bias',
'encoder.layer.1.attention.self.value.bias',
'encoder.layer.4.intermediate.dense.weight',
'encoder.layer.0.attention.output.dense.bias',
'encoder.layer.5.attention.self.value.bias',
'encoder.layer.6.attention.self.query.bias',
'encoder.layer.5.attention.self.value.weight',
'encoder.layer.6.attention.self.value.weight',
'encoder.layer.7.attention.output.LayerNorm.bias',
'encoder.layer.9.attention.output.dense.bias',
'encoder.layer.10.output.LayerNorm.weight',
'encoder.layer.9.attention.self.query.bias',
'encoder.layer.0.intermediate.dense.bias',
'encoder.layer.1.output.dense.bias',
'encoder.layer.5.output.dense.weight',
'encoder.layer.7.attention.self.key.bias',
'encoder.layer.9.output.LayerNorm.bias',
'encoder.layer.2.attention.self.query.weight',
'encoder.layer.7.attention.output.dense.weight',
'encoder.layer.5.attention.self.query.weight',
'encoder.layer.5.output.LayerNorm.weight',
'encoder.layer.5.attention.output.LayerNorm.bias',
```

```
'encoder.layer.11.attention.self.value.bias',
'encoder.layer.10.attention.self.value.weight',
'encoder.layer.9.attention.output.LayerNorm.bias',
'encoder.layer.2.output.LayerNorm.weight',
'encoder.layer.3.attention.self.key.weight',
'encoder.layer.5.attention.output.dense.bias',
'encoder.layer.5.output.dense.bias',
'encoder.layer.8.output.dense.weight', 'embeddings.LayerNorm.weight',
'encoder.layer.6.attention.self.key.weight',
'encoder.layer.7.output.LayerNorm.weight',
'encoder.layer.4.attention.self.key.bias',
'embeddings.position_embeddings.weight',
'encoder.layer.6.attention.self.key.bias',
'encoder.layer.9.output.dense.bias',
'encoder.layer.10.attention.self.query.weight',
'encoder.layer.4.attention.self.query.bias',
'encoder.layer.4.attention.output.dense.weight',
'encoder.layer.8.output.LayerNorm.weight', 'classifier.weight',
'encoder.layer.5.attention.output.dense.weight',
'encoder.layer.0.output.dense.weight',
'encoder.layer.3.attention.output.dense.bias',
'encoder.layer.2.attention.output.dense.weight',
'encoder.layer.7.attention.self.value.bias',
'encoder.layer.11.attention.output.LayerNorm.weight',
'encoder.layer.6.attention.output.dense.bias',
'encoder.layer.0.output.LayerNorm.weight',
'encoder.layer.10.attention.output.dense.weight',
'encoder.layer.0.attention.self.query.bias',
'encoder.layer.0.output.dense.bias',
'encoder.layer.6.attention.self.query.weight',
'encoder.layer.10.attention.self.value.bias',
'encoder.layer.10.attention.output.dense.bias',
'encoder.layer.9.attention.self.value.bias',
'encoder.layer.11.intermediate.dense.bias',
'encoder.layer.1.attention.self.value.weight',
'encoder.layer.7.output.dense.bias',
'encoder.layer.7.output.LayerNorm.bias',
'encoder.layer.9.attention.output.LayerNorm.weight',
'encoder.layer.4.output.LayerNorm.bias',
'encoder.layer.8.output.dense.bias',
'encoder.layer.1.output.LayerNorm.weight',
'encoder.layer.6.attention.output.LayerNorm.weight',
'encoder.layer.2.output.dense.weight',
'encoder.layer.5.intermediate.dense.weight',
'encoder.layer.6.output.LayerNorm.weight',
'encoder.layer.9.attention.self.key.weight',
'encoder.layer.1.attention.output.dense.bias',
'encoder.layer.9.attention.self.key.bias',
'encoder.layer.9.attention.self.query.weight',
'encoder.layer.10.attention.self.query.bias',
```

```
'encoder.layer.8.attention.self.query.weight',
'encoder.layer.3.intermediate.dense.bias',
'encoder.layer.9.intermediate.dense.weight',
'encoder.layer.11.output.dense.bias',
'encoder.layer.1.attention.output.LayerNorm.bias',
'encoder.layer.9.intermediate.dense.bias',
'encoder.layer.10.attention.output.LayerNorm.weight',
'encoder.layer.11.attention.self.query.bias',
'encoder.layer.8.attention.self.value.weight',
'encoder.layer.7.intermediate.dense.weight',
'encoder.layer.6.output.dense.weight',
'encoder.layer.2.attention.output.dense.bias',
'encoder.layer.9.attention.output.dense.weight',
'encoder.layer.6.intermediate.dense.weight',
'encoder.layer.10.output.dense.bias', 'pooler.dense.bias',
'encoder.layer.4.attention.self.value.bias',
'encoder.layer.4.attention.output.LayerNorm.weight',
'encoder.layer.5.attention.self.query.bias',
'encoder.layer.4.intermediate.dense.bias',
'encoder.layer.3.attention.self.key.bias',
'encoder.layer.3.output.dense.bias', 'classifier.bias',
'encoder.layer.5.attention.self.key.weight',
'encoder.layer.1.attention.output.dense.weight',
'encoder.layer.2.output.LayerNorm.bias',
'encoder.layer.7.intermediate.dense.bias',
'encoder.layer.10.attention.self.key.bias',
'encoder.layer.7.attention.self.query.weight',
'encoder.layer.3.attention.self.value.bias',
'encoder.layer.1.intermediate.dense.bias',
'encoder.layer.2.intermediate.dense.weight',
'encoder.layer.9.attention.self.value.weight']
You should probably TRAIN this model on a down-stream task to be able
to use it for predictions and inference.
```

BERTweets has its own tokenizer, so we have to repeat the data loading process

```
from transformers import AutoTokenizer

bertweet_tokenizer = AutoTokenizer.from_pretrained('vinai/bertweet-
base', do_lower_case=True)

tokenized_inputs = bertweet_tokenizer(
    inputs.tolist(),
    add_special_tokens=True,
    padding='max_length',
    max_length=MAX_LEN,
    return_tensors='pt',
)

bert_train_dataloader, bert_validation_dataloader = train_valid_split(
```

```
    input_ids=tokenized_inputs['input_ids'],
    attention_masks=tokenized_inputs['attention_mask'],
    labels=labels,
    batch_size=32
)
```

{"version_major":2,"version_minor":0,"model_id":"5923335ee9754c179af07b12d5224fc1"}

{"version_major":2,"version_minor":0,"model_id":"7ec021c2f46842f4a0d8976367b4f35b"}

```
emoji is not installed, thus not converting emoticons or emojis into
text. Please install emoji: pip3 install emoji
Special tokens have been added in the vocabulary, make sure the
associated word embeddings are fine-tuned or trained.

example train_input:     tensor([    0, 57641, 12309, 15103,     2,
1,     1])
example attention_mask:  tensor([1, 1, 1, 1, 1, 0, 0])
```

```
# About 2-3 seconds per epoch on GPU
bertweet_loss_vals, bertweet__eval_accs = train_model(
    model=bertweet,
    epochs=3,
    train_dataloader=bert_train_dataloader,
    validation_dataloader=bert_validation_dataloader
)
```

```
Training (epoch 1): 100%|██████████| 16/16 [00:03<00:00,  4.63batch/s]

  * Average training loss: 0.70
  * Training epoch took: 0:00:03
Running Validation...
  * Accuracy: 0.73
  * Validation took: 0:00:00

Training (epoch 2): 100%|██████████| 16/16 [00:03<00:00,  4.30batch/s]

  * Average training loss: 0.47
  * Training epoch took: 0:00:04
Running Validation...
  * Accuracy: 0.75
  * Validation took: 0:00:00

Training (epoch 3): 100%|██████████| 16/16 [00:04<00:00,  3.96batch/s]

  * Average training loss: 0.46
  * Training epoch took: 0:00:04
Running Validation...
  * Accuracy: 0.76
  * Validation took: 0:00:00
Training complete!
```

```
plot_loss_and_acc(bertweet_loss_vals, bertweet__eval_accs)
```


*Response to Question 4*

When fine-tuning BERT with BERTweet, its validation accuracy is lower than fine-tuning than MathBERT. Fine-tuning with MathBERT performs better because it contains words and formulas, which flows better logically. Therefore, BERT can extract more revelant information from MathBERT than English Tweets (BERTweet).

### Question 5: Inspect models predictions [0pts]

In the following cell, we have provided a function that allows you to inspect the models predictions. Given an input, e.g. `"three minus two minus two"`, it will return a trained models prediction i.e. `"negative"`, `"zero"`, or `"positive"`.

Compare the performance of `mathbert`, `mathbert_frozen` and `bertweet`. Try a few unseen examples of arithmetic questions using all models. Can you find examples where one model clearly outperforms the others? Can you find examples where all models perform poorly?

```
def what_is(input, model, tokenizer):
    # Use GPU, if available
    device = torch.device("cuda" if torch.cuda.is_available() else
"cpu")
    model = model.to(device)

    # Get map of human readable outputs
    index_to_sentiment_map = {0: "negative", 1: "zero", 2: "positive"}

    tokenized_inputs = tokenizer(
        input,                      # Input text
        add_special_tokens=True,    # add '[CLS]' and '[SEP]'
        padding='max_length',       # pad to a length specified by the
max_length
```

```
        max_length=MAX_LEN,       # truncate all sentences longer than
max_length
        return_tensors='pt',      # return everything we need as
PyTorch tensors
    )

    input_ids = tokenized_inputs['input_ids'].to(device)
    attention_masks = tokenized_inputs['attention_mask'].to(device)

    with torch.no_grad():
        outputs = model(input_ids=input_ids,
attention_mask=attention_masks)
        logits = outputs[0]
        logits = logits.detach().cpu().numpy()
        print(index_to_sentiment_map[np.argmax(logits, axis=1)[0]])
```

```
what_is("three minus five", model=mathbert, tokenizer=bert_tokenizer)
```

```
negative
```

```
what_is("three minus five", model=mathbert_frozen,
tokenizer=bert_tokenizer)
```

```
zero
```

```
what_is("three minus five", model=bertweet, tokenizer=bert_tokenizer)
```

```
positive
```

## Colab FAQ

For some basic overview and features offered in Colab notebooks, check out: Overview of Colaboratory Features

You need to use the colab GPU for this assignment by selecting:

**Runtime → Change runtime type → Hardware Accelerator: GPU**

## Part 4: Connecting Text and Images with CLIP

Acknowledgement: This notebook is based on the code from https://colab.research.google.com/github/openai/clip/blob/master/notebooks/Interacting_with_CLIP.ipynb. Credit to OpenAI.

## Section I: Interacting with CLIP

This is a self-contained notebook that shows how to download and run CLIP models, calculate the similarity between arbitrary image and text inputs, and perform zero-shot image classifications. The next cells will install the clip package and its dependencies, and check if PyTorch 1.7.1 or later is installed.

```
! pip install ftfy regex tqdm
! pip install git+https://github.com/openai/CLIP.git
```

```
Collecting ftfy
  Downloading ftfy-6.1.1-py3-none-any.whl (53 kB)
ent already satisfied: regex in /usr/local/lib/python3.7/dist-packages
(2019.12.20)
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-
packages (4.63.0)
Requirement already satisfied: wcwidth>=0.2.5 in
/usr/local/lib/python3.7/dist-packages (from ftfy) (0.2.5)
Installing collected packages: ftfy
Successfully installed ftfy-6.1.1
Collecting git+https://github.com/openai/CLIP.git
  Cloning https://github.com/openai/CLIP.git to /tmp/pip-req-build-
ic_9stf7
  Running command git clone -q https://github.com/openai/CLIP.git
/tmp/pip-req-build-ic_9stf7
Requirement already satisfied: ftfy in /usr/local/lib/python3.7/dist-
packages (from clip==1.0) (6.1.1)
Requirement already satisfied: regex in /usr/local/lib/python3.7/dist-
packages (from clip==1.0) (2019.12.20)
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-
packages (from clip==1.0) (4.63.0)
```

```
Requirement already satisfied: torch in /usr/local/lib/python3.7/dist-packages (from clip==1.0) (1.10.0+cu111)
Requirement already satisfied: torchvision in /usr/local/lib/python3.7/dist-packages (from clip==1.0) (0.11.1+cu111)
Requirement already satisfied: wcwidth>=0.2.5 in /usr/local/lib/python3.7/dist-packages (from ftfy->clip==1.0) (0.2.5)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packages (from torch->clip==1.0) (3.10.0.2)
Requirement already satisfied: pillow!=8.3.0,>=5.3.0 in /usr/local/lib/python3.7/dist-packages (from torchvision->clip==1.0) (7.1.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from torchvision->clip==1.0) (1.21.5)
Building wheels for collected packages: clip
  Building wheel for clip (setup.py) ... e=clip-1.0-py3-none-any.whl size=1369221 sha256=2a4f078a68b65e2692c4f9487733386c8364e755604554c16e5ef0b0cd509fbc
  Stored in directory: /tmp/pip-ephem-wheel-cache-5_13w4i1/wheels/fd/b9/c3/5b4470e35ed76e174bff77c92f91da82098d5e35fd5bc8cdac
Successfully built clip
Installing collected packages: clip
Successfully installed clip-1.0
```

```
import numpy as np
import torch

print("Torch version:", torch.__version__)
torch_version = torch.__version__.split(".")
assert (int(torch_version[0]) == 1 and int(torch_version[1]) >=7) or int(torch_version[0]) > 1, "PyTorch 1.7.1 or later is required"
```

```
Torch version: 1.10.0+cu111
```

### Loading the model

`clip.available_models()` will list the names of available CLIP models.

```
import clip

clip.available_models()
```

```
['RN50',
 'RN101',
 'RN50x4',
 'RN50x16',
 'RN50x64',
 'ViT-B/32',
```

```
 'ViT-B/16',
 'ViT-L/14']
```

```
model, preprocess = clip.load("ViT-B/32")
model.cuda().eval()
input_resolution = model.visual.input_resolution
context_length = model.context_length
vocab_size = model.vocab_size

print("Model parameters:", f"{np.sum([int(np.prod(p.shape)) for p in
model.parameters()]):,}")
print("Input resolution:", input_resolution)
print("Context length:", context_length)
print("Vocab size:", vocab_size)
```

```
100%|███████████████████████████████████████| 338M/338M [00:04<00:00,
77.8MiB/s]
```

```
Model parameters: 151,277,313
Input resolution: 224
Context length: 77
Vocab size: 49408
```

### Image Preprocessing

We resize the input images and center-crop them to conform with the image resolution that the model expects. Before doing so, we will normalize the pixel intensity using the dataset mean and standard deviation.

The second return value from `clip.load()` contains a torchvision `Transform` that performs this preprocessing.

```
preprocess
```

```
Compose(
    Resize(size=224, interpolation=bicubic, max_size=None,
antialias=None)
    CenterCrop(size=(224, 224))
    <function _convert_image_to_rgb at 0x7f8774a089e0>
    ToTensor()
    Normalize(mean=(0.48145466, 0.4578275, 0.40821073),
std=(0.26862954, 0.26130258, 0.27577711))
)
```

### Text Preprocessing

We use a case-insensitive tokenizer, which can be invoked using `clip.tokenize()`. By default, the outputs are padded to become 77 tokens long, which is what the CLIP models expects.

```
clip.tokenize("Hello World!")
```

```
tensor([[49406,  3306,  1002,   256, 49407,     0,     0,     0,
0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,
0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,
0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,
0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,
0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,
0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,
0,     0,
             0,     0,     0,     0,     0,     0,     0]])
```

### Setting up input images and texts

We are going to feed 8 example images and their textual descriptions to the model, and compare the similarity between the corresponding features.

The tokenizer is case-insensitive, and we can freely give any suitable textual descriptions.

```
import os
import skimage
import IPython.display
import matplotlib.pyplot as plt
from PIL import Image
import numpy as np

from collections import OrderedDict
import torch

%matplotlib inline
%config InlineBackend.figure_format = 'retina'

# images in skimage to use and their textual descriptions
descriptions = {
    "page": "a page of text about segmentation",
    "chelsea": "a facial photo of a tabby cat",
    "astronaut": "a portrait of an astronaut with the American flag",
    "rocket": "a rocket standing on a launchpad",
    "motorcycle_right": "a red motorcycle standing in a garage",
    "camera": "a person looking at a camera on a tripod",
    "horse": "a black-and-white silhouette of a horse",
    "coffee": "a cup of coffee on a saucer"
}

original_images = []
images = []
texts = []
```

```
plt.figure(figsize=(16, 5))

for filename in [filename for filename in os.listdir(skimage.data_dir)
if filename.endswith(".png") or filename.endswith(".jpg")]:
    name = os.path.splitext(filename)[0]
    if name not in descriptions:
        continue

    image = Image.open(os.path.join(skimage.data_dir,
filename)).convert("RGB")
    #print(image.__dict__.keys())
    #print(image._size)
    #image_sequence = image.getdata()
    #image_array = np.array(image_sequence)
    #print(image_array.shape)

    plt.subplot(2, 4, len(images) + 1)
    plt.imshow(image)
    plt.title(f"{filename}\n{descriptions[name]}")
    plt.xticks([])
    plt.yticks([])

    original_images.append(image)
    images.append(preprocess(image))
    texts.append(descriptions[name])

plt.tight_layout()
```


### Building features

We normalize the images, tokenize each text input, and run the forward pass of the model to get the image and text features.

```
image_input = torch.tensor(np.stack(images)).cuda()
text_tokens = clip.tokenize(["This is " + desc for desc in
texts]).cuda()
```

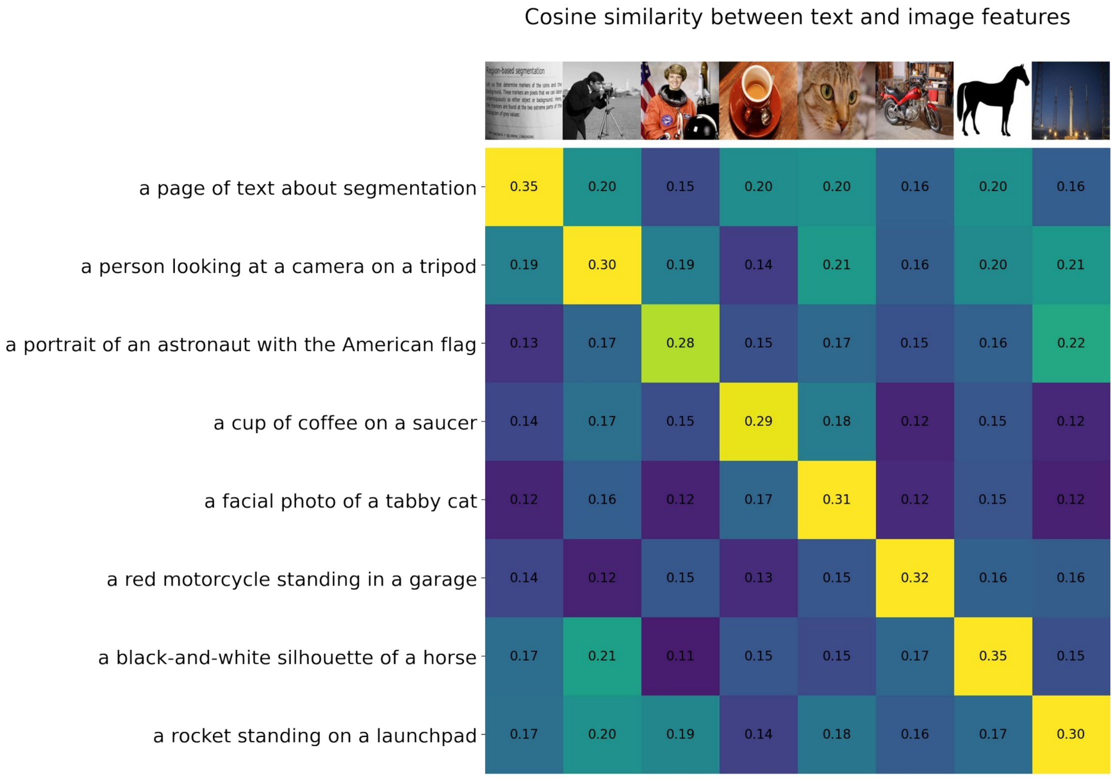
```
with torch.no_grad():
    image_features = model.encode_image(image_input).float()
    text_features = model.encode_text(text_tokens).float()
```

### Calculating cosine similarity

We normalize the features and calculate the dot product of each pair.

```
image_features /= image_features.norm(dim=-1, keepdim=True)
text_features /= text_features.norm(dim=-1, keepdim=True)
similarity = text_features.cpu().numpy() @
image_features.cpu().numpy().T

count = len(descriptions)

plt.figure(figsize=(20, 14))
plt.imshow(similarity, vmin=0.1, vmax=0.3)
# plt.colorbar()
plt.yticks(range(count), texts, fontsize=18)
plt.xticks([])
for i, image in enumerate(original_images):
    plt.imshow(image, extent=(i - 0.5, i + 0.5, -1.6, -0.6),
origin="lower")
for x in range(similarity.shape[1]):
    for y in range(similarity.shape[0]):
        plt.text(x, y, f"{similarity[y, x]:.2f}", ha="center",
va="center", size=12)

for side in ["left", "top", "right", "bottom"]:
  plt.gca().spines[side].set_visible(False)

plt.xlim([-0.5, count - 0.5])
plt.ylim([count + 0.5, -2])

plt.title("Cosine similarity between text and image features",
size=20)
```

```
Text(0.5, 1.0, 'Cosine similarity between text and image features')
```




## Zero-Shot Image Classification

You can classify images using the cosine similarity (times 100) as the logits to the softmax operation.

```
from torchvision.datasets import CIFAR100

cifar100 = CIFAR100(os.path.expanduser("~/.cache"),
transform=preprocess, download=True)
```

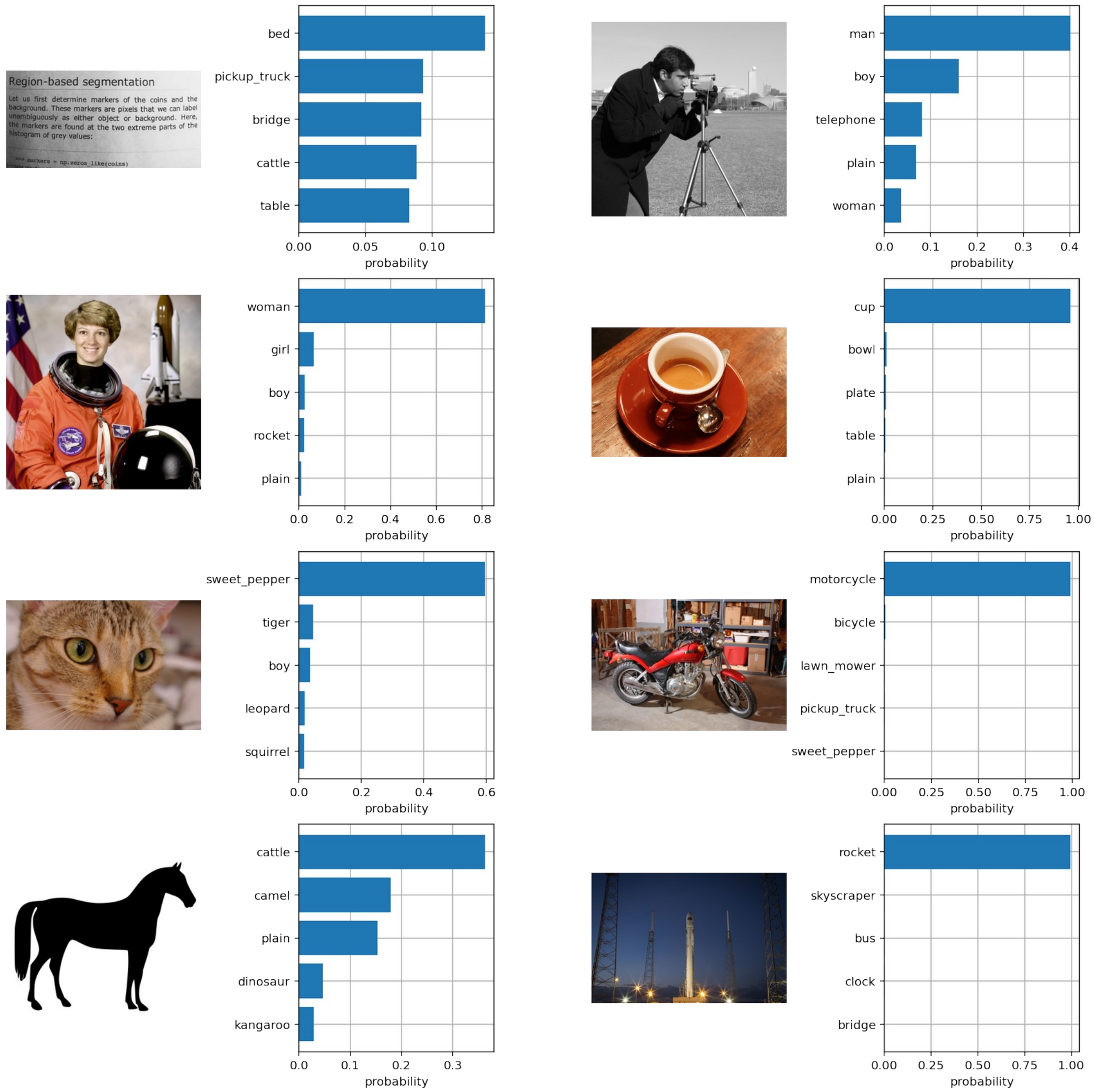
```
Downloading https://www.cs.toronto.edu/~kriz/cifar-100-python.tar.gz
to /root/.cache/cifar-100-python.tar.gz
```

{"version_major":2,"version_minor":0,"model_id":"dea5e0c9f5014e4a9c101
5cab538d1e9"}

```
Extracting /root/.cache/cifar-100-python.tar.gz to /root/.cache
```

```
text_descriptions = [f"This is a photo of a  {label}" for label in
cifar100.classes]
text_tokens = clip.tokenize(text_descriptions).cuda()

with torch.no_grad():
    text_features = model.encode_text(text_tokens).float()
    text_features /= text_features.norm(dim=-1, keepdim=True)
```

```
text_probs = (100.0 * image_features @ text_features.T).softmax(dim=-
1)
top_probs, top_labels = text_probs.cpu().topk(5, dim=-1)

plt.figure(figsize=(16, 16))

for i, image in enumerate(original_images):
    plt.subplot(4, 4, 2 * i + 1)
    plt.imshow(image)
    plt.axis("off")

    plt.subplot(4, 4, 2 * i + 2)
    y = np.arange(top_probs.shape[-1])
    plt.grid()
    plt.barh(y, top_probs[i])
    plt.gca().invert_yaxis()
    plt.gca().set_axisbelow(True)
    plt.yticks(y, [cifar100.classes[index] for index in
top_labels[i].numpy()])
    plt.xlabel("probability")

plt.subplots_adjust(wspace=0.5)
plt.show()
```




## Section II: Now let's do a Scavenger Hunt!

We want you to figure out what caption best describes the image below. We will run your caption against images in ImageNet and display the image with the highest network probability. The goal is that your caption paired with the image below will give the highest network output.

We will download a subset of ImageNet called Tiny ImageNet. Tiny ImageNet has only 200 classes, with each class having 500 traininig images, 50 validation images and 50 test images.

```
! git clone https://github.com/seshuad/IMagenet
```

```
Cloning into 'IMagenet'...
remote: Enumerating objects: 120594, done.ote: Total 120594 (delta 0), reused 0 (delta 0), pack-reused 120594
```

In order to reduce time and memory consumption, we will only consider the first 1000 images in the test set as the possible search space.

```
import os
img_paths = []
for rootdir, subdir, filenames in os.walk("IMagenet/tiny-imagenet-200/test/images"):
  for file_ in sorted(filenames)[:1000]:
    img_paths.append(os.path.join(rootdir, file_))
```

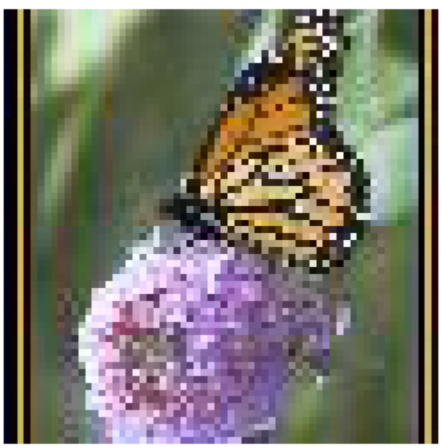
```
'''
TO DO: change caption below to produce target image
'''

# caption = "butterfly on a/one flower"
caption = "butterfly on purple flower"

# the search process can be found at the bottom of this file
```

Now, we will run the model for the first 1000 images in the Tiny ImageNet test set. We will display the image that produces the highest network probability with your written caption

```
original_images = []
images = []
```

```python
for img_path in img_paths:

    image = Image.open(img_path).convert("RGB")
    original_images.append(image)
    images.append(preprocess(image))

image_input = torch.tensor(np.stack(images)).cuda()
with torch.no_grad():
    image_features = model.encode_image(image_input).float()
image_features /= image_features.norm(dim=-1, keepdim=True)

text_tokens = clip.tokenize(caption).cuda()

with torch.no_grad():
    text_features = model.encode_text(text_tokens).float()
    text_features /= text_features.norm(dim=-1, keepdim=True)

text_probs = (100.0 * image_features @
text_features.T).softmax(dim=0).cpu().detach().numpy()
highest_prob = np.argmax(text_probs)
plt.axis('off')
plt.imshow(original_images[highest_prob])
```

```
<matplotlib.image.AxesImage at 0x7f875b003250>
```



### Search Process

To generate the image, I used `caption = "butterfly on purple flower"`.

After trying `butterfly on a flower` and `butterfly on one flower` (failed attempts), I realized I need to include more details of the image (which is try I tried specifying the color of the flower and it worked).