HW 10

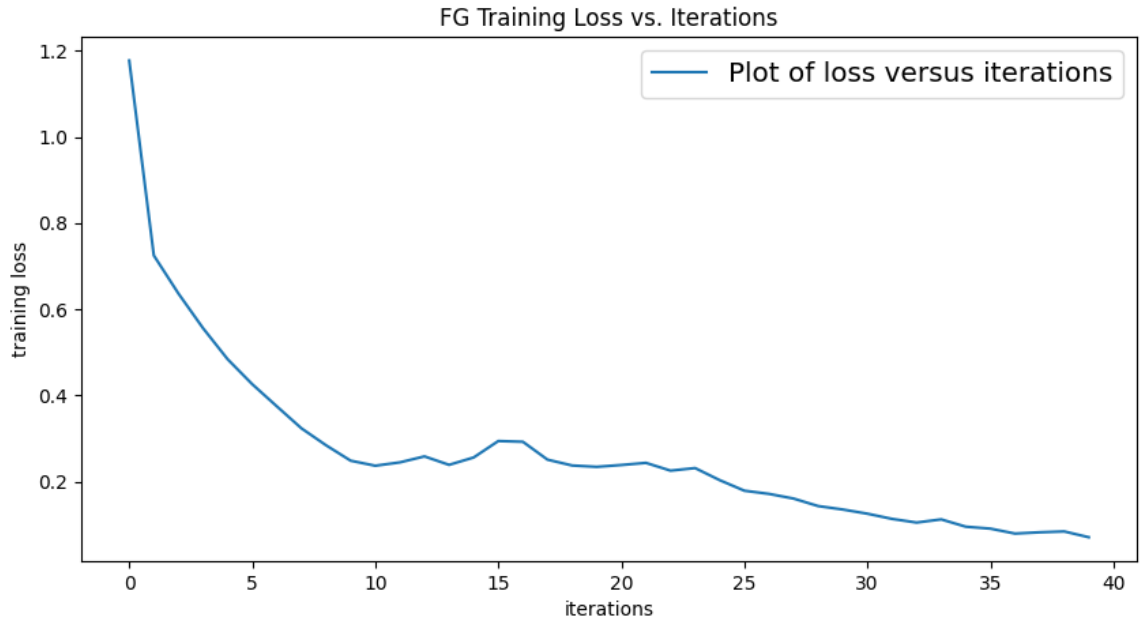
ECE 60146

Logan Dihel

1: Loss Plots

1.1: FG

The loss plot for the First Generation transformer is shown below. You can see that the loss initially decreased rapidly, but near epoch 15/40, the loss increased before finally decreasing even lower than the local minimum achieved near epoch 10/40.



1.2: PreLN

The loss plot for the PreLN transformer is shown below. This network quickly decreased the loss over the first few epochs, but after 5/40 epochs, the learning rate was about constant for the remainder of the training. The final loss value is much higher than that of the FG network, and because the learning rate is essentially constant, it is likely that the PreLN needed more than 40 epochs to finish training.

2: Five outputs from the networks

2.1: FG

Here are five outputs produced by the FG network, summarized in a table. Four of the 5 translations are precisely correct, if you don't count the SOS token which should be at the beginning of the response.

Input (English)	Ground Truth (Spanish)	Output	Output (Without SOS/EOS)
sri lanka is a beautiful island	sri lanka es una hermosa isla	EOS sri lanka es una hermosa isla EOS EOS EOS	sri lanka es una hermosa isla
do you travel a lot	viajáis mucho	EOS viajáis mucho EOS EOS EOS EOS EOS EOS EOS	viajáis mucho
i forgot i owed you money	olvidé que te debía dinero	EOS olvidé que que debía dinero EOS EOS EOS EOS	olvidé que que debía dinero
i guess it is true	supongo que es verdad	EOS supongo que es verdad EOS EOS EOS EOS EOS	supongo que es verdad
i meet a lot of people	conozco a mucha gente	EOS conozco a mucha gente EOS EOS EOS EOS EOS	conozco a mucha gente

2.2: PreLN

Here are five outputs produced by the PreLN network, summarized in a table. The PreLN does not construct full sentences, indicating traning was not finished. However, unlike the output of the FG network, the PreLN network put the SOS and EOS tokens in the correct location.

Input (English)	Ground Truth (Spanish)	Output (With SOS/EOS)	Output (Without SOS/EOS)

Input (English)	Ground Truth (Spanish)	Output (With SOS/EOS)	Output (Without SOS/EOS)
sri lanka is a beautiful island	sri lanka es una hermosa isla	SOS el es EOS EOS EOS EOS EOS EOS EOS	el es
do you travel a lot	viajáis mucho	SOS te EOS EOS EOS EOS EOS EOS EOS EOS	te
i forgot i owed you money	olvidé que te debía dinero	SOS me EOS EOS EOS EOS EOS EOS EOS EOS	me
i guess it is true	supongo que es verdad	SOS lo es EOS EOS EOS EOS EOS EOS EOS	lo es
i meet a lot of people	conozco a mucha gente	SOS me EOS EOS EOS EOS EOS EOS EOS EOS	me

3: Stop Word Removal

In order to remove the stop words, I just parsed the output text files from the networks and using the `re` module to grab the inputs in english/spanish without the EOS/SOS. However, for the output of the networks, I just needed to replace SOS and EOS tokens with an empty string. I chose to just write a function to process the entire text file, parsing out the result in an array of arrays, which I can use to create the tables in section 2, and computing the Levenshtein distance between the `Ground Truth (Spanish)` and `Output (Without SOS/EOS)`. In fact, this was the only piece of code I wrote for the entire homework assignment, aside from modifying the file path pointing to the downloaded training dataset for the `seq2seq` python scripts.

```
import re
import numpy as np

def process_lines(lines):

    result = []
    i = 0

    for line in lines:

        if line.startswith("The input sentence pair:"):
            english, spanish = re.findall(r"\['SOS (.*?) EOS'\]", line)
            english = english.strip()
            spanish = spanish.strip()
            result.append([english, spanish])

        elif line.startswith("The translation produced by"):
            # print(line)
            translation = line.split(": ")[1].strip()
            result[i].append(translation)
            # remove all EOS and SOS
            translation = translation.replace("SOS", "").replace("EOS",
```

```
"").strip()
            result[i].append(translation)
            i += 1

    return result

# this code was provided from the HW10 instructions
def levenshtein_distance(str1, str2):
    if len(str1) < len(str2):
        str1, str2 = str2, str1  # Ensure str1 is the longer string

    len_str1, len_str2 = len(str1), len(str2)

    # Initialize two rows for dynamic programming
    previous_row = list(range(len_str2 + 1))
    current_row = [0] * (len_str2 + 1)

    for i in range(1, len_str1 + 1):
        current_row[0] = i
        for j in range(1, len_str2 + 1):
            cost = 0 if str1[i - 1] == str2[j - 1] else 1
            current_row[j] = min(
                previous_row[j] + 1,       # Deletion
                current_row[j - 1] + 1,    # Insertion
                previous_row[j - 1] + cost # Substitution
            )
        previous_row, current_row = current_row, previous_row

    return previous_row[-1]

# this code was modified based off the HW10 instructions
def report_statistics(distances):
    print("| Mean | Median | Std Dev | Max | Min |")
    print("| --- | --- | --- | --- | --- |")
    print(f"| {np.mean(distances):.2f} | {np.median(distances):.2f} |
{np.std(distances):.2f} | {np.max(distances):.2f} | {np.min(distances):.2f}
|")

FG_file_path =
'/home/ldihel/Desktop/ece60146/HW10/FG/translations_with_FG_40.txt'
PreLN_file_path =
'/home/ldihel/Desktop/ece60146/HW10/PreLN/translations_with_PreLN_40.txt'


for file_path in [FG_file_path, PreLN_file_path]:

    print("\n\nOpening file:", file_path)

    with open(file_path, 'r') as fg_file:
        fg_lines = fg_file.readlines()
        result = process_lines(fg_lines)

        print(f"{len(result)} translations found.")
```

```
        print("| Input (English) | Input (Spanish) | Output (With SOS/EOS)
| Output (Without SOS/EOS) |")
        print("| --- | --- | --- | --- |")
        for item in result[:5]:
            print(f"| {item[0]} | {item[1]} | {item[2]} | {item[3]} |")

        # Calculate Levenshtein distances for all translations
        distances = []
        for item in result:
            dist = levenshtein_distance(item[1], item[3])
            distances.append(dist)

        print("Statistics:")
        report_statistics(distances)

        # number of perfect translations
        perfect_count = sum(1 for item in result if item[1] == item[3])
        print(f"Number of perfect translations: {perfect_count} out of
{len(result)}")
```

4: Levenschtein Metrics 2x5 Table

I calculated the levenschtein distance between 500 outputs of the two networks: FG and PreLN. The results are summarized in the table below.

Model	Mean	Median	Std Dev	Max	Min	Perfect Translations
FG	4.94	4.00	5.49	34.00	0.00	187 out of 500
PreLN	22.52	22.00	8.12	48.00	5.00	0 out of 500

As you can clearly see from the results above, the FG network greatly out-performed the PreLN network. In fact, 187/500 of the FG translations were perfect (ignoring the SOS/EOS tokens), while no translations from the PreLN were perfect. The median score for the FG is lower than its average, which makes sense considering that there were 187 scores of zero. The mean and median scores for the PreLN were about 5 times greater than the mean and median scores for the FG. In fact, the FG had better scores across every single metric above: mean, median, standard deviation, max, min, and number of perfect translations. However, the standard deviation score for FG is a little bit skewed because 187/500 translations were perfect, so the deviation amoungst that subset of responses is zero. The max score for the FG is larger than the minimum score for the PreLN, indicating that the FG is not uniformly better than the PreLN. Overall, we conclude that the FG transformer preformed better than the PreLN transformer over the same dataset for the same number of epochs (40). The reasoning for this is that adding in the Pre-Layer-Normalization makes the network take longer to learn. The 40 epochs used the train the PreLN model are insufficient to fully train the network.

5: Code

For completeness, I am including the two scripts I ran from DLStudio to train and test the networks. There is a total of 3 lines that I modified from these scripts.

5.1: seq2seq_with_transformerFG.py

The only modification made in this script was to change `dataroot` to a location on my computer.

```
#!/usr/bin/env python

##  seq2seq_with_transformerFG.py

"""
    This script is for experimenting with TransformerFG.

    For an introduction to TransformerFG, read the large comment block
associated
    with the definition of this class in the Transformers co-class of
DLStudio.

    Also read the doc block associated with the other transformer class,
TransformerPreLN,
    for the difference between TransformerFG and TransformerPreLN.

    To run this example, you will need to have installed at least one of
the following
    two English-to-Spanish translation dataset archives:

          en_es_xformer_8_10000.tar.gz

          en_es_xformer_8_90000.tar.gz

    The first consists of 10,000 pairs of English-Spanish sentences and the
second
    90,0000 such pairs.

    The maximum number of words in any sentence, English or Spanish, is 8.
When you
    include the sentence delimiter tokens SOS and EOS, that makes for a max
length of
    10 for the sentences.


    RECOMMENDATION:

       I recommend that you first try to run this script with exactly the
settings
       that are currently in the script:

              1.  Use the smaller debugging dataset for a faster turn-
around from
                  the code:

                         en_es_xformer_8_10000.tar.gz

              2.  Use the option
```

```
                        masking = True

              3.  epochs =  40

       Note that with the smaller dataset, you will only get one training
iteration
       per epoch, assuming you are using a batch-size of 50.

       Subsequently, try running the script for the larger dataset.

"""

import random
import numpy
import torch
import os, sys

seed = 0
random.seed(seed)
torch.manual_seed(seed)
torch.cuda.manual_seed(seed)
numpy.random.seed(seed)
torch.backends.cudnn.deterministic=True
torch.backends.cudnn.benchmarks=False
os.environ['PYTHONHASHSEED'] = str(seed)


##  watch -d -n 0.5 nvidia-smi

from DLStudio import *
from Transformers import *

# dataroot = "./data/"
#dataroot = "/home/kak/TextDatasets/en_es_corpus_xformer/"
#dataroot = "/mnt/cloudNAS3/Avi/TextDatasets/en_es_corpus_xformer/"
dataroot = '/home/ldihel/Desktop/ece60146/data/HW10/'

data_archive =  "en_es_xformer_8_10000.tar.gz"                      ## for
debugging only
#data_archive =  "en_es_xformer_8_90000.tar.gz"


max_seq_length = 10

embedding_size = 256
#embedding_size = 128
#embedding_size = 64

num_basic_encoders = num_basic_decoders = num_atten_heads = 4
#num_basic_encoders = num_basic_decoders = num_atten_heads = 2

#optimizer_params = {'beta1' : 0.9,  'beta2': 0.98,  'epsilon' : 1e-9}
optimizer_params = {'beta1' : 0.9,  'beta2': 0.98,  'epsilon' : 1e-6}
```

```
num_warmup_steps = 4000

masking = True                      ## for better results
#masking = False

dls = DLStudio(
                dataroot = dataroot,
                path_saved_model = {"encoder_FG" : "./saved_encoder_FG",
                                    "decoder_FG" : "./saved_decoder_FG",
                                    "embeddings_generator_en_FG" :
"./saved_embeddings_generator_en_FG",
                                    "embeddings_generator_es_FG" :
"./saved_embeddings_generator_es_FG",
                                   },
                batch_size = 50,
                use_gpu = True,
                epochs = 40,
              )

xformer = TransformerFG(
                        dl_studio = dls,
                        dataroot = dataroot,
                        data_archive = data_archive,
                        max_seq_length = max_seq_length,
                        embedding_size = embedding_size,
                        save_checkpoints = True,
                        num_warmup_steps = num_warmup_steps,
                        optimizer_params = optimizer_params,
          )

master_encoder = TransformerFG.MasterEncoder(
                                    dls,
                                    xformer,
                                    num_basic_encoders = num_basic_encoders,
                                    num_atten_heads = num_atten_heads,
                 )


master_decoder = TransformerFG.MasterDecoderWithMasking(
                                    dls,
                                    xformer,
                                    num_basic_decoders = num_basic_decoders,
                                    num_atten_heads = num_atten_heads,
                                    masking = masking
                 )


number_of_learnable_params_in_encoder = sum(p.numel() for p in
master_encoder.parameters() if p.requires_grad)
print("\n\nThe number of learnable parameters in the Master Encoder: %d" %
number_of_learnable_params_in_encoder)

number_of_learnable_params_in_decoder = sum(p.numel() for p in
```

```
master_decoder.parameters() if p.requires_grad)
print("\nThe number of learnable parameters in the Master Decoder: %d" %
number_of_learnable_params_in_decoder)

if masking:

xformer.run_code_for_training_TransformerFG(dls,master_encoder,master_decod
er,display_train_loss=True,

checkpoints_dir="checkpoints_with_masking_FG")
else:

xformer.run_code_for_training_TransformerFG(dls,master_encoder,master_decod
er,display_train_loss=True,

checkpoints_dir="checkpoints_no_masking_FG")

#import pymsgbox
#response = pymsgbox.confirm("Finished training.  Start evaluation?")

#if response == "OK":
xformer.run_code_for_evaluating_TransformerFG(master_encoder,
master_decoder, 'myoutput.txt')
```

5.2: seq2seq_with_transformerPreLN.py

The only modifications made to this script are changing `dataroot` to a suitable location on my computer and changing `epochs` from 60 to 40.

```
#!/usr/bin/env python

##  seq2seq_with_transformerPreLN.py

"""
    This script is for experimenting with TransformerPreLN.

    For an introduction to TransformerPreLN, read the large comment block
associated
    with the definition of this class in the Transformers co-class of
DLStudio.
    That introduction explains the difference between TransformerPreLN and
    TransformerFG.

    To run this example, you will need to have installed at least one of
the following
    two English-to-Spanish translation dataset archives:

          en_es_xformer_8_10000.tar.gz

          en_es_xformer_8_90000.tar.gz
```

```
    The first consists of 10,000 pairs of English-Spanish sentences and the
second of
    90,0000.

    The maximum number of words in any sentence, English or Spanish, is 8.
When you
    include the sentence delimiter tokens SOS and EOS, that makes for a max
length of
    10 for the sentences.

    *** RECOMMENDATION ***:

       I recommend that you first try to run this script with exactly the
settings
       that are currently in the script:

              1.  Use the smaller debugging dataset for a faster turn-
around from
                  the code:

                         en_es_xformer_8_10000.tar.gz

              2.  Use the option

                         masking = True

              3.  epochs =  60

       Note that with the smaller dataset, you will only get one training
iteration
       per epoch, assuming you are using a batch-size of 50.

       Subsequently, try running the script for the larger dataset.
"""

import random
import numpy
import torch
import os, sys

seed = 0
random.seed(seed)
torch.manual_seed(seed)
torch.cuda.manual_seed(seed)
numpy.random.seed(seed)
torch.backends.cudnn.deterministic=True
torch.backends.cudnn.benchmarks=False
os.environ['PYTHONHASHSEED'] = str(seed)


##  watch -d -n 0.5 nvidia-smi

from DLStudio import *
from Transformers import *
```

```
# dataroot = "./data/"
#dataroot = "/home/kak/TextDatasets/en_es_corpus_xformer/"
#dataroot = "/mnt/cloudNAS3/Avi/TextDatasets/en_es_corpus_xformer/"
dataroot = '/home/ldihel/Desktop/ece60146/data/HW10/'

data_archive =  "en_es_xformer_8_10000.tar.gz"                 ##  for
debugging only
#data_archive =  "en_es_xformer_8_90000.tar.gz"

max_seq_length = 10

embedding_size = 256
#embedding_size = 128
#embedding_size = 64

num_basic_encoders = num_basic_decoders = num_atten_heads = 4
#num_basic_encoders = num_basic_decoders = num_atten_heads = 2

masking = True                    ## For better results
#masking = False

dls = DLStudio(
                dataroot = dataroot,
                path_saved_model = {"encoder_PreLN" :
"./saved_encoder_PreLN",
                                    "decoder_PreLN" :
"./saved_decoder_PreLN",
                                    "embeddings_generator_en_PreLN" :
"./saved_embeddings_generator_en_PreLN",
                                    "embeddings_generator_es_PreLN" :
"./saved_embeddings_generator_es_PreLN",
                                   },
                learning_rate =  1e-5,
                batch_size = 50,
                use_gpu = True,
                epochs = 40,
      )

xformer = TransformerPreLN(
                            dl_studio = dls,
                            dataroot = dataroot,
                            save_checkpoints = True,
                            data_archive = data_archive,
                            max_seq_length = max_seq_length,
                            embedding_size = embedding_size,
          )

master_encoder = TransformerPreLN.MasterEncoder(
                                    dls,
                                    xformer,
                                    num_basic_encoders = num_basic_encoders,
                                    num_atten_heads = num_atten_heads,
```

```
                    )


master_decoder = TransformerPreLN.MasterDecoderWithMasking(
                                    dls,
                                    xformer,
                                    num_basic_decoders = num_basic_decoders,
                                    num_atten_heads = num_atten_heads,
                                    masking = masking,
                    )


number_of_learnable_params_in_encoder = sum(p.numel() for p in
master_encoder.parameters() if p.requires_grad)
print("\n\nThe number of learnable parameters in the Master Encoder: %d" %
number_of_learnable_params_in_encoder)

number_of_learnable_params_in_decoder = sum(p.numel() for p in
master_decoder.parameters() if p.requires_grad)
print("\n\nThe number of learnable parameters in the Master Decoder: %d" %
number_of_learnable_params_in_decoder)

if masking:

xformer.run_code_for_training_TransformerPreLN(dls,master_encoder,master_de
coder,display_train_loss=True,

checkpoints_dir="checkpoints_with_masking_PreLN")
else:

xformer.run_code_for_training_TransformerPreLN(dls,master_encoder,master_de
coder,display_train_loss=True,

checkpoints_dir="checkpoints_no_masking_PreLN")

#import pymsgbox
#response = pymsgbox.confirm("Finished training.  Start evaluation?")
#if response == "OK":

xformer.run_code_for_evaluating_TransformerPreLN(master_encoder,
master_decoder)
```

5.3: Code for generating tables

See section 3 for the code I used to generate all of the tables throughout this document. I tried to write clean, reusable code because the output of the two networks was the same, structurally.