README

May 15, 2024

###**CSE 256 Programming Assignment 2**

Ke Xu (kex005@ucsd.edu)

Note: To ensure the consistency and reproducibility of my experiments, all computational tasks are conducted on a **Tesla T4 GPU** provided by the **Google Colab** platform.

[1]:
```
# This mounts your Google Drive to the Colab VM.
from google.colab import drive
drive.mount('/content/drive')

# TODO: Enter the foldername in your Drive where you have saved the unzipped
# assignment folder, e.g. 'cse256/assignments/PA2/'
FOLDERNAME = None
FOLDERNAME = 'CSE256PAs/PA2'
assert FOLDERNAME is not None, "[!] Enter the foldername."

# Now that we've mounted your Drive, this ensures that
# the Python interpreter of the Colab VM can load
# python files from within it.
import sys
sys.path.append('/content/drive/My Drive/{}'.format(FOLDERNAME))

# This is later used to use the IMDB reviews
%cd /content/drive/My\ Drive/$FOLDERNAME/
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
/content/drive/My Drive/CSE256PAs/PA2
```

[2]:
```
!nvidia-smi
```

```
Tue May 14 23:19:30 2024
+-----------------------------------------------------------------------------
--------+
| NVIDIA-SMI 535.104.05             Driver Version: 535.104.05   CUDA Version:
12.2     |
|-----------------------------------------+----------------------+-------------
--------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile
```

```
Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util
Compute M. |
|                                         |                      |
MIG M. |
|=========================================+======================+==============
========|
|   0  Tesla T4                       Off | 00000000:00:04.0 Off |
0 |
| N/A   43C    P8               9W /  70W |      0MiB / 15360MiB |      0%
Default |
|                                         |                      |
N/A |
+-----------------------------------------+----------------------+--------------
--------+

+-------------------------------------------------------------------------------
--------+
| Processes:
|
|  GPU   GI   CI        PID   Type   Process name                            GPU
Memory |
|        ID   ID
Usage      |
|===============================================================================
========|
|  No running processes found
|
+-------------------------------------------------------------------------------
--------+
```

[3]:
```
!pip install torch
!pip install matplotlib
```

```
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages
(2.2.1+cu121)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-
packages (from torch) (3.14.0)
Requirement already satisfied: typing-extensions>=4.8.0 in
/usr/local/lib/python3.10/dist-packages (from torch) (4.11.0)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages
(from torch) (1.12)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-
packages (from torch) (3.3)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages
(from torch) (3.1.4)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages
(from torch) (2023.6.0)
```

```
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.1.105 in
/usr/local/lib/python3.10/dist-packages (from torch) (12.1.105)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.1.105 in
/usr/local/lib/python3.10/dist-packages (from torch) (12.1.105)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.1.105 in
/usr/local/lib/python3.10/dist-packages (from torch) (12.1.105)
Requirement already satisfied: nvidia-cudnn-cu12==8.9.2.26 in
/usr/local/lib/python3.10/dist-packages (from torch) (8.9.2.26)
Requirement already satisfied: nvidia-cublas-cu12==12.1.3.1 in
/usr/local/lib/python3.10/dist-packages (from torch) (12.1.3.1)
Requirement already satisfied: nvidia-cufft-cu12==11.0.2.54 in
/usr/local/lib/python3.10/dist-packages (from torch) (11.0.2.54)
Requirement already satisfied: nvidia-curand-cu12==10.3.2.106 in
/usr/local/lib/python3.10/dist-packages (from torch) (10.3.2.106)
Requirement already satisfied: nvidia-cusolver-cu12==11.4.5.107 in
/usr/local/lib/python3.10/dist-packages (from torch) (11.4.5.107)
Requirement already satisfied: nvidia-cusparse-cu12==12.1.0.106 in
/usr/local/lib/python3.10/dist-packages (from torch) (12.1.0.106)
Requirement already satisfied: nvidia-nccl-cu12==2.19.3 in
/usr/local/lib/python3.10/dist-packages (from torch) (2.19.3)
Requirement already satisfied: nvidia-nvtx-cu12==12.1.105 in
/usr/local/lib/python3.10/dist-packages (from torch) (12.1.105)
Requirement already satisfied: triton==2.2.0 in /usr/local/lib/python3.10/dist-
packages (from torch) (2.2.0)
Requirement already satisfied: nvidia-nvjitlink-cu12 in
/usr/local/lib/python3.10/dist-packages (from nvidia-cusolver-
cu12==11.4.5.107->torch) (12.4.127)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.10/dist-packages (from jinja2->torch) (2.1.5)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-
packages (from sympy->torch) (1.3.0)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-
packages (3.7.1)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (1.2.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-
packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (4.51.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (1.4.5)
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.10/dist-
packages (from matplotlib) (1.25.2)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (24.0)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-
packages (from matplotlib) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in
```

```
/usr/local/lib/python3.10/dist-packages (from matplotlib) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-
packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
```

[5]:
```
!pip install nltk
import nltk
nltk.download('punkt')
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.10/dist-packages
(3.8.1)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages
(from nltk) (8.1.7)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages
(from nltk) (1.4.2)
Requirement already satisfied: regex>=2021.8.3 in
/usr/local/lib/python3.10/dist-packages (from nltk) (2023.12.25)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages
(from nltk) (4.66.4)
```

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

[5]: True

[5]:
```
!python3 PA2_code/main.py
```

```
Loading data and creating tokenizer …
Vocabulary size is 5755
Total trainable parameters for encoder_model: 575187
Epoch 1, Loss: 1.0766, Test Accuracy: 33.33%
Epoch 2, Loss: 1.0313, Test Accuracy: 39.87%
Epoch 3, Loss: 0.9583, Test Accuracy: 52.67%
Epoch 4, Loss: 0.8579, Test Accuracy: 63.07%
Epoch 5, Loss: 0.7481, Test Accuracy: 64.13%
Epoch 6, Loss: 0.6201, Test Accuracy: 75.33%
Epoch 7, Loss: 0.5273, Test Accuracy: 74.93%
Epoch 8, Loss: 0.4050, Test Accuracy: 78.93%
Epoch 9, Loss: 0.3376, Test Accuracy: 81.73%
Epoch 10, Loss: 0.2595, Test Accuracy: 80.93%
Epoch 11, Loss: 0.1951, Test Accuracy: 84.27%
Epoch 12, Loss: 0.1376, Test Accuracy: 85.60%
Epoch 13, Loss: 0.1050, Test Accuracy: 84.40%
Epoch 14, Loss: 0.1049, Test Accuracy: 85.73%
Epoch 15, Loss: 0.0984, Test Accuracy: 84.27%
Input tensor shape: torch.Size([1, 32])
Number of attention maps: 4
Figure(640x480)
```

```
Figure(640x480)
Figure(640x480)
Figure(640x480)
Total trainable parameters for decoder_model: 942459
Step 100 Train Perplexity: 498.0761413574219
Step 200 Train Perplexity: 323.67205810546875
Step 300 Train Perplexity: 211.1782684326172
Step 400 Train Perplexity: 151.31732177734375
Step 500 Train Perplexity: 119.29962158203125
LM Training Loss: 5.771663032643264
Input tensor shape: torch.Size([1, 32])
Number of attention maps: 4
Figure(640x480)
Figure(640x480)
Figure(640x480)
Figure(640x480)
Step 500 Obama Perplexity: 329.2257995605469
Step 500 H. Bush Perplexity: 376.1542663574219
Step 500 W. Bush Perplexity: 447.5278625488281
```

###**Part 3 Exploration**

[1]:
```
# This mounts your Google Drive to the Colab VM.
from google.colab import drive
drive.mount('/content/drive')

# TODO: Enter the foldername in your Drive where you have saved the unzipped
# assignment folder, e.g. 'cse256/assignments/PA2/'
FOLDERNAME = None
FOLDERNAME = 'CSE256PAs/PA2'
assert FOLDERNAME is not None, "[!] Enter the foldername."

# Now that we've mounted your Drive, this ensures that
# the Python interpreter of the Colab VM can load
# python files from within it.
import sys
sys.path.append('/content/drive/My Drive/{}'.format(FOLDERNAME))

# This is later used to use the IMDB reviews
%cd /content/drive/My\ Drive/$FOLDERNAME/
```

```
Mounted at /content/drive
/content/drive/My Drive/CSE256PAs/PA2
```

[2]:
```
!pip install torch
!pip install nltk
import nltk
nltk.download('punkt')
```

```
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages
(2.2.1+cu121)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-
packages (from torch) (3.14.0)
Requirement already satisfied: typing-extensions>=4.8.0 in
/usr/local/lib/python3.10/dist-packages (from torch) (4.11.0)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages
(from torch) (1.12)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-
packages (from torch) (3.3)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages
(from torch) (3.1.4)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages
(from torch) (2023.6.0)
Collecting nvidia-cuda-nvrtc-cu12==12.1.105 (from torch)
  Using cached nvidia_cuda_nvrtc_cu12-12.1.105-py3-none-manylinux1_x86_64.whl
(23.7 MB)
Collecting nvidia-cuda-runtime-cu12==12.1.105 (from torch)
  Using cached nvidia_cuda_runtime_cu12-12.1.105-py3-none-manylinux1_x86_64.whl
(823 kB)
Collecting nvidia-cuda-cupti-cu12==12.1.105 (from torch)
  Using cached nvidia_cuda_cupti_cu12-12.1.105-py3-none-manylinux1_x86_64.whl
(14.1 MB)
Collecting nvidia-cudnn-cu12==8.9.2.26 (from torch)
  Using cached nvidia_cudnn_cu12-8.9.2.26-py3-none-manylinux1_x86_64.whl (731.7
MB)
Collecting nvidia-cublas-cu12==12.1.3.1 (from torch)
  Using cached nvidia_cublas_cu12-12.1.3.1-py3-none-manylinux1_x86_64.whl (410.6
MB)
Collecting nvidia-cufft-cu12==11.0.2.54 (from torch)
  Using cached nvidia_cufft_cu12-11.0.2.54-py3-none-manylinux1_x86_64.whl (121.6
MB)
Collecting nvidia-curand-cu12==10.3.2.106 (from torch)
  Using cached nvidia_curand_cu12-10.3.2.106-py3-none-manylinux1_x86_64.whl
(56.5 MB)
Collecting nvidia-cusolver-cu12==11.4.5.107 (from torch)
  Using cached nvidia_cusolver_cu12-11.4.5.107-py3-none-manylinux1_x86_64.whl
(124.2 MB)
Collecting nvidia-cusparse-cu12==12.1.0.106 (from torch)
  Using cached nvidia_cusparse_cu12-12.1.0.106-py3-none-manylinux1_x86_64.whl
(196.0 MB)
Collecting nvidia-nccl-cu12==2.19.3 (from torch)
  Using cached nvidia_nccl_cu12-2.19.3-py3-none-manylinux1_x86_64.whl (166.0 MB)
Collecting nvidia-nvtx-cu12==12.1.105 (from torch)
  Using cached nvidia_nvtx_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (99 kB)
Requirement already satisfied: triton==2.2.0 in /usr/local/lib/python3.10/dist-
packages (from torch) (2.2.0)
Collecting nvidia-nvjitlink-cu12 (from nvidia-cusolver-cu12==11.4.5.107->torch)
```

```
  Using cached nvidia_nvjitlink_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl
(21.1 MB)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.10/dist-packages (from jinja2->torch) (2.1.5)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-
packages (from sympy->torch) (1.3.0)
Installing collected packages: nvidia-nvtx-cu12, nvidia-nvjitlink-cu12, nvidia-
nccl-cu12, nvidia-curand-cu12, nvidia-cufft-cu12, nvidia-cuda-runtime-cu12,
nvidia-cuda-nvrtc-cu12, nvidia-cuda-cupti-cu12, nvidia-cublas-cu12, nvidia-
cusparse-cu12, nvidia-cudnn-cu12, nvidia-cusolver-cu12
Successfully installed nvidia-cublas-cu12-12.1.3.1 nvidia-cuda-cupti-
cu12-12.1.105 nvidia-cuda-nvrtc-cu12-12.1.105 nvidia-cuda-runtime-cu12-12.1.105
nvidia-cudnn-cu12-8.9.2.26 nvidia-cufft-cu12-11.0.2.54 nvidia-curand-
cu12-10.3.2.106 nvidia-cusolver-cu12-11.4.5.107 nvidia-cusparse-cu12-12.1.0.106
nvidia-nccl-cu12-2.19.3 nvidia-nvjitlink-cu12-12.4.127 nvidia-nvtx-cu12-12.1.105
Requirement already satisfied: nltk in /usr/local/lib/python3.10/dist-packages
(3.8.1)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages
(from nltk) (8.1.7)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages
(from nltk) (1.4.2)
Requirement already satisfied: regex>=2021.8.3 in
/usr/local/lib/python3.10/dist-packages (from nltk) (2023.12.25)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages
(from nltk) (4.66.4)

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

[2]: True

[3]:
```python
# Architecture Exploration
import torch
import torch.nn as nn
from torch import optim
from torch.utils.data import DataLoader
from torch.nn.utils.rnn import pad_sequence
import os

from PA2_code.tokenizer import SimpleTokenizer
from PA2_code.dataset import SpeechesClassificationDataset,␣
 ↪LanguageModelingDataset
from PA2_code.transformer import TransformerEncoder, FeedforwardClassifier,␣
 ↪SpeechSegmentModel
from PA2_code.transformer import TransformerDecoderWithDisentangledAttention,␣
 ↪TransformerDecoderWithSparseAttention
from PA2_code.utilities import Utilities
```

[4]:

```
## Decoder - TransformerDecoderWithSparseAttention

seed = 42

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

""" Hyperparameters to use for training to roughly match
the numbers mentioned in the assignment description """
batch_size = 16  # Number of independent sequences  we will process in parallel
block_size = 32  # Maximum context length for predictions
learning_rate = 1e-3  # Learning rate for the optimizer
n_embd = 64  # Embedding dimension
n_head = 2  # Number of attention heads
n_layer = 4  # Number of transformer layers


eval_interval = 100  # How often to evaluate train and test perplexity during␣
 ↪training
max_iters = 500 # For language modeling, we can process all the batches for the␣
 ↪entire dataset, but that takes a while, so we'll limit it to 500 iterations.␣
 ↪For batch size of 16 and block size of  32, this is roughly, this is  500 *␣
 ↪16 * 32 = 256000 tokens, SOTA LMs are trained on trillions of tokens, so␣
 ↪this is a very small dataset.
eval_iters = 200  # Number of iterations to evaluate perplexity on the test set


## classifier training hyperparameters. It is a simple 1 hidden layer␣
 ↪feedforward network, with input
## size of 64, hidden size of 50 and output size of 3.

n_input = 64  # Input size for the classifier, should match the embedding size␣
 ↪of the transformer
n_hidden = 100  # Hidden size for the classifier
n_output = 3  # Output size for the classifier, we have 3 classes
epochs_CLS = 15 # epochs for classifier training

def load_texts(directory):
    """
    This function loads all texts from the specified directory, ignoring any␣
 ↪files with "test" in their name. The text is used for "training" the␣
 ↪tokenizer. Since our tokenizer is simple, we don't need to do any training,␣
 ↪but we still need to ignore the test data.
    """

    texts = []
    files = os.listdir(directory)
    for filename in files:
```

```
        if "test" in filename:  ## don't "read test files"
            continue
        with open(os.path.join(directory, filename), 'r', encoding='utf-8') as␣
↪file:
            texts.append(file.read())
    return texts

def collate_batch(batch):
    """ Collate a batch of data into a single tensor with padding."""
    data, labels = zip(*batch)  # Separate the data and labels
    # Pad sequences to the fixed length
    padded_sequences = pad_sequence(data, batch_first=True, padding_value=0)
    padded_sequences = padded_sequences[:, :block_size]  # Truncate if longer
    # Add padding if shorter
    padded_sequences = torch.nn.functional.pad(padded_sequences, (0, max(0,␣
↪block_size - padded_sequences.shape[1])), "constant", 0)
    labels = torch.stack(labels)
    return padded_sequences, labels

def compute_perplexity(decoderLMmodel, data_loader, criterion, eval_iters=100):
    """ Compute the perplexity of the decoderLMmodel on the data in data_loader.
    Make sure to use the cross entropy loss for the decoderLMmodel.
    """
    decoderLMmodel.eval()
    losses= []
    with torch.no_grad():
        for X, Y in data_loader:
            X, Y = X.to(device), Y.to(device)
            outputs = decoderLMmodel(X) # your model should be computing the␣
↪cross entropy loss
            loss = criterion(outputs.view(-1, outputs.size(-1)), Y.view(-1))
            losses.append(loss.item())
            if len(losses) >= eval_iters: break

    losses = torch.tensor(losses)
    mean_loss = losses.mean()
    perplexity = torch.exp(mean_loss).item()  # Calculate perplexity as␣
↪exp(mean loss)

    decoderLMmodel.train()
    return perplexity

def main():

    print("Loading data and creating tokenizer ...")
    texts = load_texts('speechesdataset')
```

```
    tokenizer = SimpleTokenizer(' '.join(texts)) # create a tokenizer from the␣
↪data
    print("Vocabulary size is", tokenizer.vocab_size)

    inputfile = "speechesdataset/train_LM.txt"
    inputfile_obama = "speechesdataset/test_LM_obama.txt"
    inputfile_wbush = "speechesdataset/test_LM_wbush.txt"
    inputfile_hbush = "speechesdataset/test_LM_hbush.txt"

    with open(inputfile, 'r', encoding='utf-8') as f:
        lmtrainText = f.read()
    train_LM_dataset = LanguageModelingDataset(tokenizer, lmtrainText, ␣
↪block_size)
    train_LM_loader = DataLoader(train_LM_dataset, batch_size=batch_size,␣
↪shuffle=True)

    with open(inputfile_obama, 'r', encoding='utf-8') as f:
        obamatestText = f.read()
    test_obama_dataset = LanguageModelingDataset(tokenizer, obamatestText, ␣
↪block_size)
    test_obama_loader = DataLoader(test_obama_dataset, batch_size=batch_size,␣
↪shuffle=True)

    with open(inputfile_wbush, 'r', encoding='utf-8') as f:
        wbushtestText = f.read()
    test_wbush_dataset = LanguageModelingDataset(tokenizer, wbushtestText, ␣
↪block_size)
    test_wbush_loader = DataLoader(test_wbush_dataset, batch_size=batch_size,␣
↪shuffle=True)

    with open(inputfile_hbush, 'r', encoding='utf-8') as f:
        hbushtestText = f.read()
    test_hbush_dataset = LanguageModelingDataset(tokenizer, hbushtestText, ␣
↪block_size)
    test_hbush_loader = DataLoader(test_hbush_dataset, batch_size=batch_size,␣
↪shuffle=True)

    # for the language modeling task, you will iterate over the training data␣
↪for a fixed number of iterations like this:
    decoder_model = TransformerDecoderWithSparseAttention(n_layer, n_embd,␣
↪n_head, tokenizer.vocab_size).to(device)
    print("####### Decoder: TransformerDecoderWithSparseAttention #######")
    num_decoder_parameters = sum(p.numel() for p in decoder_model.parameters())
    print(f"Total trainable parameters for decoder_model:␣
↪{num_decoder_parameters}")
    optimizer = optim.Adam(decoder_model.parameters(), lr=learning_rate)
```

```
    criterion = nn.CrossEntropyLoss()

    decoder_model.train()
    for i, (xb, yb) in enumerate(train_LM_loader):
        if i >= max_iters:
            break
        xb, yb = xb.to(device), yb.to(device)
        # LM training code here
        optimizer.zero_grad()           # Reset gradients to zero for each batch
        outputs = decoder_model(xb)      # Forward pass
        outputs = outputs.view(-1, outputs.size(-1))
        yb = yb.view(-1)
        loss = criterion(outputs, yb)     # Compute loss
        loss.backward()                 # Backpropagate the loss
        optimizer.step()                # Update the model parameters

        if (i + 1) % 100 == 0:
            print(f"Step {i + 1} Train Perplexity:␣
↪{compute_perplexity(decoder_model, train_LM_loader, criterion)}")

    print(f"Step 500 Obama Perplexity: {compute_perplexity(decoder_model,␣
↪test_obama_loader, criterion)}")
    print(f"Step 500 H. Bush Perplexity: {compute_perplexity(decoder_model,␣
↪test_hbush_loader, criterion)}")
    print(f"Step 500 W. Bush Perplexity: {compute_perplexity(decoder_model,␣
↪test_wbush_loader, criterion)}")

if __name__ == "__main__":
    main()
```

```
Loading data and creating tokenizer …
Vocabulary size is 5755
####### Decoder: TransformerDecoderWithSparseAttention #######
Total trainable parameters for decoder_model: 975227
Step 100 Train Perplexity: 319.5060119628906
Step 200 Train Perplexity: 137.37399291992188
Step 300 Train Perplexity: 75.67100524902344
Step 400 Train Perplexity: 44.10769271850586
Step 500 Train Perplexity: 29.1240177154541
Step 500 Obama Perplexity: 120.5716781616211
Step 500 H. Bush Perplexity: 137.09829711914062
Step 500 W. Bush Perplexity: 166.67526245117188
```

[5]:

```
# This mounts your Google Drive to the Colab VM.
from google.colab import drive
drive.mount('/content/drive')
```

```
# TODO: Enter the foldername in your Drive where you have saved the unzipped
# assignment folder, e.g. 'cse256/assignments/PA2/'
FOLDERNAME = None
FOLDERNAME = 'CSE256PAs/PA2'
assert FOLDERNAME is not None, "[!] Enter the foldername."

# Now that we've mounted your Drive, this ensures that
# the Python interpreter of the Colab VM can load
# python files from within it.
import sys
sys.path.append('/content/drive/My Drive/{}'.format(FOLDERNAME))

# This is later used to use the IMDB reviews
%cd /content/drive/My\ Drive/$FOLDERNAME/
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
/content/drive/My Drive/CSE256PAs/PA2
```

[6]:

```
# Performance Improvement - Learning Rate Scheduler
import torch
import torch.nn as nn
from torch import optim
from torch.utils.data import DataLoader
from torch.nn.utils.rnn import pad_sequence
import os

from PA2_code.tokenizer import SimpleTokenizer
from PA2_code.dataset import SpeechesClassificationDataset,␣
 ↪LanguageModelingDataset
from PA2_code.transformer import TransformerEncoder, FeedforwardClassifier,␣
 ↪SpeechSegmentModel
from PA2_code.transformer import SpeechSegmentModel, TransformerDecoder
from PA2_code.utilities import Utilities
from torch.optim.lr_scheduler import StepLR
```

[7]:

```
seed = 42

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

""" Hyperparameters to use for training to roughly match
the numbers mentioned in the assignment description """
batch_size = 16  # Number of independent sequences  we will process in parallel
block_size = 32  # Maximum context length for predictions
learning_rate = 1e-3  # Learning rate for the optimizer
n_embd = 64  # Embedding dimension
n_head = 2  # Number of attention heads
```

```
n_layer = 4  # Number of transformer layers


eval_interval = 100  # How often to evaluate train and test perplexity during␣
↪training
max_iters = 500 # For language modeling, we can process all the batches for the␣
↪entire dataset, but that takes a while, so we'll limit it to 500 iterations.␣
↪For batch size of 16 and block size of  32, this is roughly, this is  500 *␣
↪16 * 32 = 256000 tokens, SOTA LMs are trained on trillions of tokens, so␣
↪this is a very small dataset.
eval_iters = 200  # Number of iterations to evaluate perplexity on the test set


## classifier training hyperparameters. It is a simple 1 hidden layer␣
↪feedforward network, with input
## size of 64, hidden size of 50 and output size of 3.

n_input = 64  # Input size for the classifier, should match the embedding size␣
↪of the transformer
n_hidden = 100  # Hidden size for the classifier
n_output = 3  # Output size for the classifier, we have 3 classes
epochs_CLS = 15 # epochs for classifier training

def load_texts(directory):
    """
    This function loads all texts from the specified directory, ignoring any␣
↪files with "test" in their name. The text is used for "training" the␣
↪tokenizer. Since our tokenizer is simple, we don't need to do any training,␣
↪but we still need to ignore the test data.
    """

    texts = []
    files = os.listdir(directory)
    for filename in files:
        if "test" in filename:  ## don't "read test files"
            continue
        with open(os.path.join(directory, filename), 'r', encoding='utf-8') as␣
↪file:
            texts.append(file.read())
    return texts

def collate_batch(batch):
    """ Collate a batch of data into a single tensor with padding."""
    data, labels = zip(*batch)  # Separate the data and labels
    # Pad sequences to the fixed length
    padded_sequences = pad_sequence(data, batch_first=True, padding_value=0)
```

```python
    padded_sequences = padded_sequences[:, :block_size]  # Truncate if longer
    # Add padding if shorter
    padded_sequences = torch.nn.functional.pad(padded_sequences, (0, max(0,␣
↪block_size - padded_sequences.shape[1])), "constant", 0)
    labels = torch.stack(labels)
    return padded_sequences, labels

def compute_classifier_accuracy(classifier, data_loader):
    """ Compute the accuracy of the classifier on the data in data_loader."""
    classifier.eval()
    total_correct = 0
    total_samples = 0
    with torch.no_grad():
        for X, Y in data_loader:
            X, Y = X.to(device), Y.to(device)
            outputs = classifier(X)
            _, predicted = torch.max(outputs.data, 1)
            total_correct += (predicted == Y).sum().item()
            total_samples += Y.size(0)
        accuracy = (100 * total_correct / total_samples)
        classifier.train()
        return accuracy

import math

def main():
    print("Loading data and creating tokenizer ...")
    texts = load_texts('speechesdataset')
    tokenizer = SimpleTokenizer(' '.join(texts)) # create a tokenizer from the␣
↪data
    print("Vocabulary size is", tokenizer.vocab_size)

    train_CLS_dataset = SpeechesClassificationDataset(tokenizer,␣
↪"speechesdataset/train_CLS.tsv")
    train_CLS_loader = DataLoader(train_CLS_dataset, batch_size=batch_size,␣
↪collate_fn=collate_batch, shuffle=True)
    test_CLS_dataset = SpeechesClassificationDataset(tokenizer,␣
↪"speechesdataset/test_CLS.tsv")
    test_CLS_loader = DataLoader(test_CLS_dataset, batch_size=batch_size,␣
↪collate_fn=collate_batch, shuffle=True)

    # Model
    encoder = TransformerEncoder(n_embd=n_embd, n_head=n_head, n_layer=n_layer,␣
↪vocab_size=tokenizer.vocab_size)
    classifier = FeedforwardClassifier(n_input=n_input, n_hidden=n_hidden,␣
↪n_output=n_output)
```

```python
    encoder_model = SpeechSegmentModel(encoder, classifier).to(device)

    num_encoder_parameters = sum(p.numel() for p in encoder_model.parameters())
    print(f"Total trainable parameters for encoder_model: 
 ↪{num_encoder_parameters}")

    # Optimizer and loss function
    optimizer = optim.Adam(encoder_model.parameters(), lr=learning_rate)
    criterion = nn.CrossEntropyLoss()

    # Define the Cosine Annealing learning rate scheduler
    print("##### Learning Rate Scheduler #####")
    scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, 
 ↪T_max=epochs_CLS, eta_min=1e-6)

    encoder_model.train()
    for epoch in range(epochs_CLS):
        total_loss = 0
        for xb, yb in train_CLS_loader:
            xb, yb = xb.to(device), yb.to(device)
            optimizer.zero_grad()  # Reset gradients to zero for each batch
            output = encoder_model(xb)  # Forward pass
            loss = criterion(output, yb)  # Compute loss
            loss.backward()  # Backpropagate the loss
            optimizer.step()  # Update the model parameters
            total_loss += loss.item()

        # Step through the learning rate scheduler
        scheduler.step()

        # Calculate average loss and test accuracy for the epoch
        average_loss = total_loss / len(train_CLS_loader)
        test_accuracy = compute_classifier_accuracy(encoder_model, 
 ↪test_CLS_loader)
        print(f'Epoch {epoch + 1}, Loss: {average_loss:.4f}, Test Accuracy: 
 ↪{test_accuracy:.2f}%, Learning Rate: {scheduler.get_last_lr()[0]:.6f}')

if __name__ == "__main__":
    main()
```

```
Loading data and creating tokenizer …
Vocabulary size is 5755
Total trainable parameters for encoder_model: 575187
Epoch 1, Loss: 1.0787, Test Accuracy: 33.33%, Learning Rate: 0.000989
Epoch 2, Loss: 1.0542, Test Accuracy: 44.93%, Learning Rate: 0.000957
Epoch 3, Loss: 1.0026, Test Accuracy: 49.87%, Learning Rate: 0.000905
Epoch 4, Loss: 0.9114, Test Accuracy: 50.40%, Learning Rate: 0.000835
```

```
Epoch 5, Loss: 0.7922, Test Accuracy: 63.20%, Learning Rate: 0.000750
Epoch 6, Loss: 0.6751, Test Accuracy: 69.87%, Learning Rate: 0.000655
Epoch 7, Loss: 0.5311, Test Accuracy: 77.33%, Learning Rate: 0.000553
Epoch 8, Loss: 0.3927, Test Accuracy: 79.87%, Learning Rate: 0.000448
Epoch 9, Loss: 0.3020, Test Accuracy: 80.80%, Learning Rate: 0.000346
Epoch 10, Loss: 0.2324, Test Accuracy: 83.33%, Learning Rate: 0.000251
Epoch 11, Loss: 0.1807, Test Accuracy: 84.40%, Learning Rate: 0.000166
Epoch 12, Loss: 0.1338, Test Accuracy: 85.07%, Learning Rate: 0.000096
Epoch 13, Loss: 0.1108, Test Accuracy: 85.33%, Learning Rate: 0.000044
Epoch 14, Loss: 0.0997, Test Accuracy: 85.20%, Learning Rate: 0.000012
Epoch 15, Loss: 0.0937, Test Accuracy: 85.33%, Learning Rate: 0.000001
```

Data Analysis

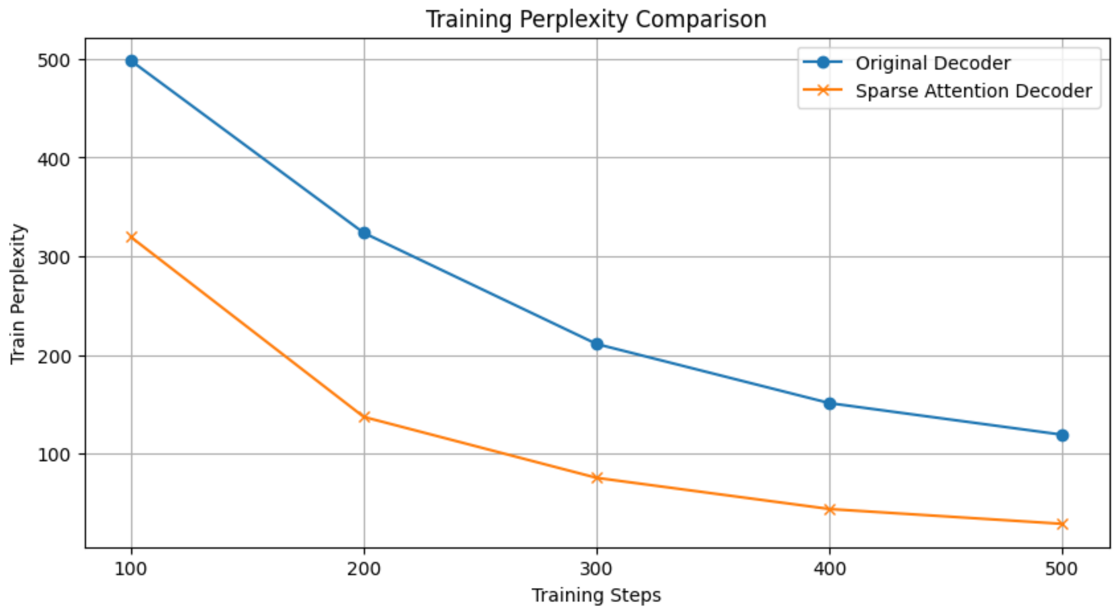
[1]:
```python
import matplotlib.pyplot as plt

# Data for the original decoder and sparse attention decoder
steps = [100, 200, 300, 400, 500]
original_train_perplexity = [498.0761, 323.6721, 211.1783, 151.3173, 119.2996]
sparse_train_perplexity = [319.506, 137.374, 75.671, 44.108, 29.124]

# Creating the plot
plt.figure(figsize=(10, 5))
plt.plot(steps, original_train_perplexity, marker='o', label='Original Decoder')
plt.plot(steps, sparse_train_perplexity, marker='x', label='Sparse Attention␣
↪Decoder')

# Adding titles and labels
plt.title('Training Perplexity Comparison')
plt.xlabel('Training Steps')
plt.ylabel('Train Perplexity')
plt.xticks(steps)
plt.legend()
plt.grid(True)

# Displaying the plot
plt.show()
```



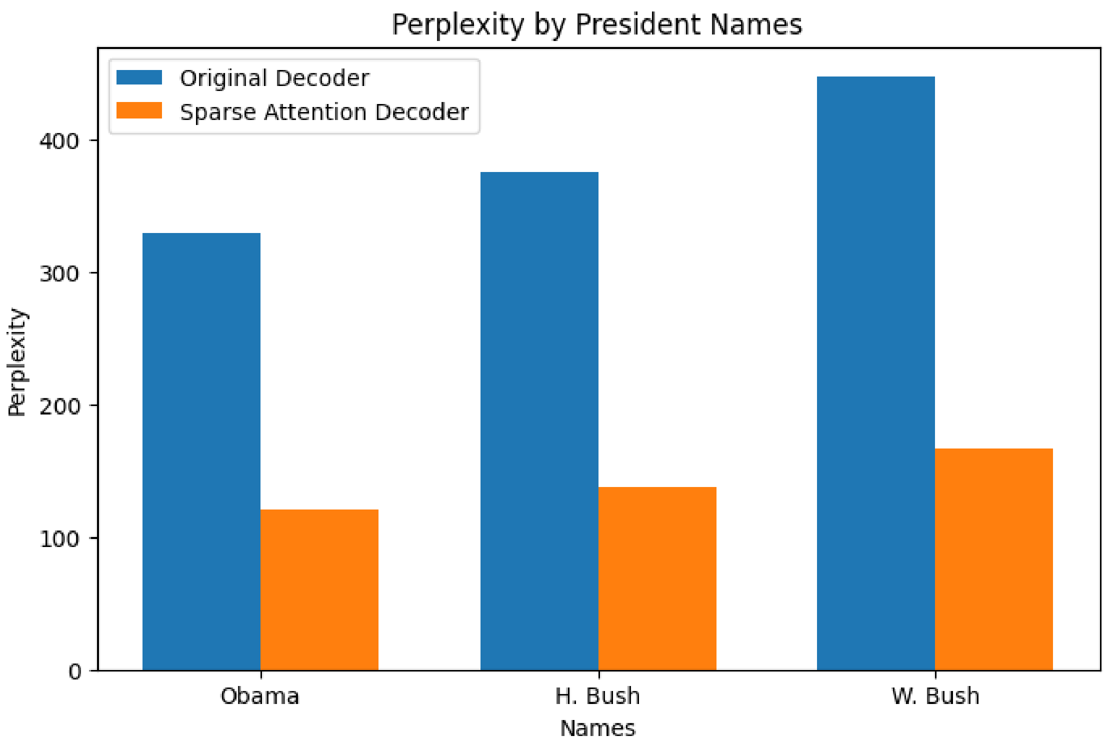
[2]:
```
# Data for the additional perplexity values for specific tokens
labels = ['Obama', 'H. Bush', 'W. Bush']
original_specific_perplexity = [329.2258, 376.1543, 447.5279]
sparse_specific_perplexity = [120.572, 137.098, 166.675]

x = range(len(labels))  # the label locations
width = 0.35  # the width of the bars

fig, ax = plt.subplots(figsize=(8, 5))
rects1 = ax.bar(x, original_specific_perplexity, width, label='Original␣
↪Decoder')
rects2 = ax.bar([p + width for p in x], sparse_specific_perplexity, width,␣
↪label='Sparse Attention Decoder')

# Adding some text for labels, title and custom x-axis tick labels, etc.
ax.set_xlabel('Names')
ax.set_ylabel('Perplexity')
ax.set_title('Perplexity by President Names')
ax.set_xticks([p + width / 2 for p in x])
ax.set_xticklabels(labels)
ax.legend()

# Displaying the plot
plt.show()
```



[4]:
```
import matplotlib.pyplot as plt

# Data from the tables
epochs = list(range(1, 16))
train_loss_original = [1.0799, 1.0524, 0.9871, 0.9203, 0.8102, 0.6834, 0.5679,␣
↪0.4473, 0.3482, 0.2920,
                       0.2406, 0.1963, 0.1514, 0.1411, 0.0985]
train_loss_sparse = [1.0787, 1.0542, 1.0026, 0.9114, 0.7922, 0.6751, 0.5311, 0.
↪3927, 0.3020, 0.2324,
                     0.1807, 0.1338, 0.1108, 0.0997, 0.0937]
test_accuracy_original = [33.33, 48.67, 44.00, 51.87, 62.00, 67.87, 74.27, 79.
↪07, 78.80, 82.53,
                          82.67, 84.40, 85.60, 83.47, 85.07]
test_accuracy_sparse = [33.33, 44.93, 49.87, 50.40, 63.20, 69.87, 77.33, 79.87,␣
↪80.80, 83.33,
                        84.40, 85.07, 85.33, 85.20, 85.33]

# Creating the plot
plt.figure(figsize=(12, 8))

# Plotting train loss
plt.subplot(2, 1, 1)
```

```
plt.plot(epochs, train_loss_original, label='Original Encoder - Train Loss',␣
  ↪marker='o')
plt.plot(epochs, train_loss_sparse, label='Encoder with Learning Rate Scheduler␣
  ↪- Train Loss', marker='x')
plt.title('Training Loss and Test Accuracy Over Epochs')
plt.ylabel('Train Loss')
plt.xlabel('Epoch')
plt.legend()

# Plotting test accuracy
plt.subplot(2, 1, 2)
plt.plot(epochs, test_accuracy_original, label='Original Encoder - Test␣
  ↪Accuracy', marker='o')
plt.plot(epochs, test_accuracy_sparse, label='Encoder with Learning Rate␣
  ↪Scheduler - Test Accuracy', marker='x')
plt.ylabel('Test Accuracy (%)')
plt.xlabel('Epoch')
plt.legend()

# Show the plots
plt.tight_layout()
plt.show()
```