In [ ]:
```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
```

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

In [ ]:
```
!pwd
```

```
/content
```

In [ ]:
```
!cd /content/drive/MyDrive/GradAssessment && ls
```

```
'10362-Article Text-13890-1-2-20201228.pdf'   layers.py        Model.ipynb
test.txt         valid.txt
 GA_Writeup.pdf                               Model_ex.ipynb   README.md
train.txt
```

In [ ]:
```
!pip install torch
!pip install transformers
```

```
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-pack
ages (2.1.0+cu118)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-p
ackages (from torch) (3.13.1)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.
10/dist-packages (from torch) (4.5.0)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-pack
ages (from torch) (1.12)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-p
ackages (from torch) (3.2.1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-pac
kages (from torch) (3.1.2)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-pac
kages (from torch) (2023.6.0)
Requirement already satisfied: triton==2.1.0 in /usr/local/lib/python3.10/d
ist-packages (from torch) (2.1.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.1
0/dist-packages (from jinja2->torch) (2.1.3)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/di
st-packages (from sympy->torch) (1.3.0)
Requirement already satisfied: transformers in /usr/local/lib/python3.10/di
st-packages (4.35.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-p
ackages (from transformers) (3.13.1)
Requirement already satisfied: huggingface-hub<1.0,>=0.16.4 in /usr/local/l
ib/python3.10/dist-packages (from transformers) (0.19.4)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dis
t-packages (from transformers) (1.23.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.1
0/dist-packages (from transformers) (23.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dis
t-packages (from transformers) (6.0.1)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.
10/dist-packages (from transformers) (2023.6.3)
```

```
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from transformers) (2.31.0)
Requirement already satisfied: tokenizers<0.19,>=0.14 in /usr/local/lib/python3.10/dist-packages (from transformers) (0.15.0)
Requirement already satisfied: safetensors>=0.3.1 in /usr/local/lib/python3.10/dist-packages (from transformers) (0.4.1)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/dist-packages (from transformers) (4.66.1)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.16.4->transformers) (2023.6.0)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.16.4->transformers) (4.5.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (2023.11.17)
```

In [ ]:
```
pip install torch --upgrade
```

```
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (2.1.0+cu118)
Collecting torch
  Downloading torch-2.1.1-cp310-cp310-manylinux1_x86_64.whl (670.2 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 670.2/670.2 MB 1.6 MB/s eta 0:00:00
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch) (3.13.1)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packages (from torch) (4.5.0)
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) (3.2.1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch) (3.1.2)
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)
  Downloading nvidia_cuda_nvrtc_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (23.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 23.7/23.7 MB 73.4 MB/s eta 0:00:00
Collecting nvidia-cuda-runtime-cu12==12.1.105 (from torch)
  Downloading nvidia_cuda_runtime_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (823 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 823.6/823.6 kB 61.9 MB/s eta 0:00:00
Collecting nvidia-cuda-cupti-cu12==12.1.105 (from torch)
```

```
  Downloading nvidia_cuda_cupti_cu12-12.1.105-py3-none-manylinux1_x86_64.wh
l (14.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 14.1/14.1 MB 92.3 MB/s eta 0:
00:00
Collecting nvidia-cudnn-cu12==8.9.2.26 (from torch)
  Downloading nvidia_cudnn_cu12-8.9.2.26-py3-none-manylinux1_x86_64.whl (73
1.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 731.7/731.7 MB 1.9 MB/s eta 0
:00:00
Collecting nvidia-cublas-cu12==12.1.3.1 (from torch)
  Downloading nvidia_cublas_cu12-12.1.3.1-py3-none-manylinux1_x86_64.whl (4
10.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 410.6/410.6 MB 2.8 MB/s eta 0
:00:00
Collecting nvidia-cufft-cu12==11.0.2.54 (from torch)
  Downloading nvidia_cufft_cu12-11.0.2.54-py3-none-manylinux1_x86_64.whl (1
21.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 121.6/121.6 MB 14.5 MB/s eta
0:00:00
Collecting nvidia-curand-cu12==10.3.2.106 (from torch)
  Downloading nvidia_curand_cu12-10.3.2.106-py3-none-manylinux1_x86_64.whl
(56.5 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.5/56.5 MB 31.3 MB/s eta 0:
00:00
Collecting nvidia-cusolver-cu12==11.4.5.107 (from torch)
  Downloading nvidia_cusolver_cu12-11.4.5.107-py3-none-manylinux1_x86_64.wh
l (124.2 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 124.2/124.2 MB 12.2 MB/s eta
0:00:00
Collecting nvidia-cusparse-cu12==12.1.0.106 (from torch)
  Downloading nvidia_cusparse_cu12-12.1.0.106-py3-none-manylinux1_x86_64.wh
l (196.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 196.0/196.0 MB 5.2 MB/s eta 0
:00:00
Collecting nvidia-nccl-cu12==2.18.1 (from torch)
  Downloading nvidia_nccl_cu12-2.18.1-py3-none-manylinux1_x86_64.whl (209.8
MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 209.8/209.8 MB 5.1 MB/s eta 0
:00:00
Collecting nvidia-nvtx-cu12==12.1.105 (from torch)
  Downloading nvidia_nvtx_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (99
kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 99.1/99.1 kB 13.6 MB/s eta 0:
00:00
Requirement already satisfied: triton==2.1.0 in /usr/local/lib/python3.10/d
ist-packages (from torch) (2.1.0)
Collecting nvidia-nvjitlink-cu12 (from nvidia-cusolver-cu12==11.4.5.107->to
rch)
  Downloading nvidia_nvjitlink_cu12-12.3.101-py3-none-manylinux1_x86_64.whl
(20.5 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 20.5/20.5 MB 90.3 MB/s eta 0:
00:00
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10
```

```
/dist-packages (from jinja2->torch) (2.1.3)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/di
st-packages (from sympy->torch) (1.3.0)
Installing collected packages: nvidia-nvtx-cu12, nvidia-nvjitlink-cu12, nvi
dia-nccl-cu12, nvidia-curand-cu12, nvidia-cufft-cu12, nvidia-cuda-runtime-c
u12, nvidia-cuda-nvrtc-cu12, nvidia-cuda-cupti-cu12, nvidia-cublas-cu12, nv
idia-cusparse-cu12, nvidia-cudnn-cu12, nvidia-cusolver-cu12, torch
  Attempting uninstall: torch
    Found existing installation: torch 2.1.0+cu118
    Uninstalling torch-2.1.0+cu118:
      Successfully uninstalled torch-2.1.0+cu118
ERROR: pip's dependency resolver does not currently take into account all t
he packages that are installed. This behaviour is the source of the followi
ng dependency conflicts.
torchaudio 2.1.0+cu118 requires torch==2.1.0, but you have torch 2.1.1 whic
h is incompatible.
torchdata 0.7.0 requires torch==2.1.0, but you have torch 2.1.1 which is in
compatible.
torchtext 0.16.0 requires torch==2.1.0, but you have torch 2.1.1 which is i
ncompatible.
torchvision 0.16.0+cu118 requires torch==2.1.0, but you have torch 2.1.1 wh
ich is incompatible.
Successfully installed nvidia-cublas-cu12-12.1.3.1 nvidia-cuda-cupti-cu12-1
2.1.105 nvidia-cuda-nvrtc-cu12-12.1.105 nvidia-cuda-runtime-cu12-12.1.105 n
vidia-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 nv
idia-nccl-cu12-2.18.1 nvidia-nvjitlink-cu12-12.3.101 nvidia-nvtx-cu12-12.1.
105 torch-2.1.1
```

In [ ]:
```python
import torch
from torch import nn
import torch.nn.functional as F
```

Deprecated Functions

In [ ]:

```python
# new way to tokenize
def build_char_vocab(corpus):
    char_set = set()
    for sentence in corpus:
        char_set.update(sentence)
    # <pad> and <unk>
    char_vocab = {'<pad>': 0, '<unk>': 1}
    char_vocab.update({char: idx + 2 for idx, char in enumerate(sorted(char_
    return char_vocab

corpus = ["Hello world", "This is an example sentence"]
char_vocab = build_char_vocab(corpus)
print(char_vocab)

def build_word_vocab(corpus):
    word_set = set()
    for sentence in corpus:
        word_set.update(sentence.split())
    # adding <pad> and <unk>
    word_vocab = {'<pad>': 0, '<unk>': 1} # gonna assume it doesn't need to
    word_vocab.update({word: idx + 2 for idx, word in enumerate(word_set)})
    return word_vocab

corpus = ["Hello world", "This is an example sentence"]
word_vocab = build_word_vocab(corpus)
print(word_vocab)
```

```
{'<pad>': 0, '<unk>': 1, ' ': 2, 'H': 3, 'T': 4, 'a': 5, 'c': 6, 'd': 7, '
e': 8, 'h': 9, 'i': 10, 'l': 11, 'm': 12, 'n': 13, 'o': 14, 'p': 15, 'r': 1
6, 's': 17, 't': 18, 'w': 19, 'x': 20}
{'<pad>': 0, '<unk>': 1, 'world': 2, 'an': 3, 'sentence': 4, 'is': 5, 'Hell
o': 6, 'This': 7, 'example': 8}
```

In [ ]:

```
# def one_hot_encode(index, vocab_size):
#     one_hot = torch.zeros(vocab_size)
#     one_hot[index] = 1
#     return one_hot

# def preprocess_corpus(corpus, char_vocab, max_length):
#     vocab_size = len(char_vocab)
#     processed_corpus = []
#     for sentence in corpus:
#         sentence_indices = [char_vocab.get(char, char_vocab['<unk>']) for
#         padded_indices = sentence_indices + [char_vocab['<pad>']] * (max_l
#         # one_hot_sentence = [one_hot_encode(index, vocab_size) for index
#         processed_corpus.append(torch.stack(one_hot_sentence))
#     return torch.stack(processed_corpus)

def preprocess_corpus(corpus, char_vocab, max_length=30):
        processed_corpus = []
    for sentence in corpus:
        words = sentence.split()  # Splitting the sentence into words
        for word in words:
            word_indices = [char_vocab.get(char, char_vocab['<unk>']) for ch
            # Pad each word to max_length
            padded_indices = word_indices + [char_vocab['<pad>']] * (max_ler
            processed_corpus.append(padded_indices[:max_length])
    return processed_corpus

corpus = ["Hello world", "This is an example sentence"]

processed_corpus = preprocess_corpus(corpus, char_vocab)

input_tensor = torch.tensor(processed_corpus)
print(input_tensor)
```

```
tensor([[ 9,  8, 11, 11, 14,  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],
        [19, 14, 16, 11,  7,  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],
        [18,  9, 10, 17,  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],
        [10, 17,  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],
        [ 5, 13,  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],
        [ 8, 20,  5, 12, 15, 11,  8,  0,  0,  0,  0,  0,  0,  0,  0,  0,
0,  0,
          0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0],
        [17,  8, 13, 18,  8, 13,  6,  8,  0,  0,  0,  0,  0,  0,  0,  0,
0,  0,
          0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0]])
```

Active Code

In [ ]:

```python
import torch
from torch import nn
import torch.nn.functional as F

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", device)

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

class HighwayBlock(nn.Module):
    def __init__(self, input_dim, output_dim):
        super().__init__()
        self.project = nn.Linear(input_dim, output_dim)
        self.transform = nn.Linear(input_dim, output_dim)
        self.trans_bias = nn.Parameter(torch.tensor(-2.0))

    def forward(self, x):
        proj_output = torch.relu(self.project(x))
        trans_output = torch.sigmoid(self.transform(x) + self.trans_bias)
        return trans_output * proj_output + (1 - trans_output) * x

class HighwayNetwork(nn.Module):
    def __init__(self, input_size, output_size, num_layers):
        super().__init__()
        self.num_layers = num_layers
```

```
        self.layers = []
        for i in range(num_layers):
            layer_size = input_size if i == 0 else output_size
            self.layers.append(HighwayBlock(layer_size, output_size))
            self.add_module(f'highway_block_{i}', self.layers[-1])

    def forward(self, x):
        for layer in self.layers:
            x = layer(x)
        return x

class ConvolutionBlock(nn.Module):
    def __init__(self, channels, kernel, features):
        super().__init__()
        self.conv_layer = nn.Conv2d(channels, features, kernel)

    def forward(self, x, size_reduce):
        conv_output = torch.tanh(self.conv_layer(x))
        pooled_output = F.max_pool2d(conv_output, kernel_size=[1, size_reduc
        return pooled_output.squeeze(3).squeeze(2)

class ConvolutionNetwork(nn.Module):
    def __init__(self, channel_size, kernel_sizes, feature_sizes):
        super().__init__()
        self.conv_blocks = nn.ModuleList()

        # applies the filters of differenent widths over input
        for i, (k_size, f_size) in enumerate(zip(kernel_sizes, feature_sizes
            self.conv_blocks.append(ConvolutionBlock(channel_size, (1, k_siz

    def forward(self, x):
        # squeezes output to accomodate for batches
        x = x.unsqueeze(2).transpose(1, 3)
        conv_outputs = [block(x, x.size(3) - k_size + 1) for block, k_size i
        return torch.cat(conv_outputs, 1)

class RNN(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers, dropout_rate):
        super().__init__()
        self.rnn_layers = nn.LSTM(input_size, hidden_size, num_layers=num_la
        self.hidden_size = hidden_size
        self.num_layers = num_layers

    # use this instead of hidden state init outside
    def init_hidden_state(self, batch_size):
        weight = next(self.parameters()).data
        return (torch.zeros(self.num_layers, batch_size, self.hidden_size),
                torch.zeros(self.num_layers, batch_size, self.hidden_size))

    def forward(self, x, hidden):
        output, hidden = self.rnn_layers(x, hidden)
        return output, hidden
```

```
class CharacterToWordModel(nn.Module):
    def __init__(self, char_vocab_size, char_embed_dim, word_vocab_size,
                 conv_out_size, hidden_dim, kernel_sizes, features, num_high
                 num_rnn_layers, dropout):
        super().__init__()
        self.char_vocab_size = char_vocab_size
        self.char_embed_dim = char_embed_dim
        self.word_vocab_size = word_vocab_size
        self.conv_out_size = conv_out_size
        self.hidden_dim = hidden_dim
        self.dropout_rate = dropout

        self.char_embedding = nn.Embedding(char_vocab_size, char_embed_dim,
        # print("charvocab", char_vocab_size, "char_embed_dim", char_embed_d
        self.conv_net = ConvolutionNetwork(char_embed_dim, kernel_sizes, fea
        self.highway_net = HighwayNetwork(conv_out_size, conv_out_size, num_
        self.rnn_net = RNN(conv_out_size, hidden_dim, num_rnn_layers, dropou
        self.output_layer = nn.Linear(hidden_dim, word_vocab_size)
        self.dropout = nn.Dropout(dropout)

        self.initw()

    def initw(self):
        rng = 0.1
        self.char_embedding.weight.data.uniform_(-rng, rng)
        self.output_layer.bias.data.fill_(0)
        self.output_layer.weight.data.uniform_(-rng, rng)

    def forward(self, input_chars, hidden_state):
        # print("input shape", input_chars.shape)
        emb = self.char_embedding(input_chars)
        # print("embedding shape", emb.shape)
        conv_output = self.conv_net(emb)
        highway_output = self.highway_net(conv_output)
        rnn_output, hidden_state = self.rnn_net(highway_output, hidden_state
        rnn_output = self.dropout(rnn_output)
        final_output = self.output_layer(rnn_output.view(-1, self.hidden_dim
        top_word_indices = torch.argmax(final_output, dim=-1)
        return final_output, hidden_state
```

```
Using device: cuda
```

In [ ]:
```
import os
path = '/content/drive/MyDrive/GradAssessment'
os.chdir(path)
print("Current Directory:", os.getcwd())

def load_dataset(file_path):
    with open(file_path, 'r') as file:
        data = file.readlines()
    return data

train_data = load_dataset('train.txt')
valid_data = load_dataset('valid.txt')
test_data = load_dataset('test.txt')

def preprocess_data(data):
    corpus = []
    for sentence in data:
        words = sentence.strip().split()
        for i in range(len(words) - 1):
            predictor = words[i]
            target = words[i + 1]
            corpus.append((predictor, target))
    return corpus
testing = [
    "Hello world",
    "This is an example sentence",
    "To be or not to be, that is the question",
    "I think, therefore I am",]

test = preprocess_data(testing)
print(len(test))
print(testing)
```

```
Current Directory: /content/drive/MyDrive/GradAssessment
18
['Hello world', 'This is an example sentence', 'To be or not to be, that is
the question', 'I think, therefore I am']
```

In [ ]:
```
pip install tqdm
```

```
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packa
ges (4.66.1)
```

In [ ]:
```
import torch
from torch.utils.data import DataLoader, TensorDataset

# def data_to_tensors(corpus, char_vocab, word_vocab):
#     predictors = []
#     targets = []
#     for predictor, target in corpus:
#         predictor_indices = [char_vocab.get(char, char_vocab['<unk>']) for
#         # print(predictors)
#         target_index = word_vocab.get(target, word_vocab['<unk>'])
#         predictors.append(predictor_indices)
#         targets.append(target_index)
#     print(len(predictors))
#     print(len(targets))
#     return torch.tensor(predictors), torch.tensor(targets)

# train_predictors, train_targets = data_to_tensors(train_corpus, char_vocab
# valid_predictors, valid_targets = data_to_tensors(valid_corpus, char_vocab
# test_predictors, test_targets = data_to_tensors(test_corpus, char_vocab, w

def data_to_tensors(corpus, char_vocab, word_vocab, max_sequence_length):
    predictors = []
    targets = []

    for predictor, target in corpus:
        predictor_indices = [char_vocab.get(char, char_vocab['<unk>']) for c
        target_index = word_vocab.get(target, word_vocab['<unk>'])
        if (target_index == 9999):
            print("index 9999: ", target)
        if (target_index == 10000):
            print("index 10000: ", target)
        # Pad the predictor sequence
        if len(predictor_indices) < max_sequence_length:
            predictor_indices += [char_vocab['<pad>']] * (max_sequence_lengt
        else:
            predictor_indices = predictor_indices[:max_sequence_length]

        predictors.append(predictor_indices)
        targets.append(target_index)

    return torch.tensor(predictors), torch.tensor(targets)

# train_predictors, train_targets = data_to_tensors(train_corpus, char_vocab
# valid_predictors, valid_targets = data_to_tensors(valid_corpus, char_vocab
# test_predictors, test_targets = data_to_tensors(test_corpus, char_vocab, w

# batch_size = 20
# train_dataset = TensorDataset(train_predictors, train_targets)
# train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=Tr
```

In [ ]:
```
import os
os.environ['CUDA_LAUNCH_BLOCKING'] = "1"
```

```python
def build_char_vocab(corpus):
    char_set = set()
    for predictor, target in corpus:
        char_set.update(predictor)
        char_set.update(target)
    char_vocab = {'<pad>': 0, '<unk>': 1}
    char_vocab.update({char: idx + 2 for idx, char in enumerate(sorted(char_
    return char_vocab

def build_word_vocab(corpus):
    word_set = set()
    for predictor, target in corpus:
        word_set.update([predictor, target])
    print("Total unique words not presets:", len(word_set))
    word_vocab = {'<pad>': 0}
    word_vocab.update({word: idx + 1 for idx, word in enumerate(sorted(word_
    print("Total unique words including <pad> and <unk>:", len(word_set) + 1
    return word_vocab

corpus = [
    "Hello world",
    "This is an example sentence",
    "To be or not to be, that is the question",
    "I think, therefore I am",
    "A journey of a thousand miles begins with a single step",
    "All that glitters is not gold",
    "Ask not what your country can do for you, ask what you can do for your
    "I have a dream",
    "Elementary, my dear Watson",
    "Houston, we have a problem",
    "Just keep swimming",
    "May the Force be with you",
    "Once upon a time in a land far, far away",
    "Winter is coming",
    "Keep calm and carry on",
    "Why so serious?",
    "There's no place like home",
    # "The cake is a lie",
    # "To infinity and beyond",
    # "Elementary, my dear Watson",
    # "It's a trap!",
    # "Life is like a box of chocolates",
    # "The pen is mightier than the sword",
    # "Knowledge is power",
    # "With great power comes great responsibility",
    # "The only thing we have to fear is fear itself",
    # "I have a dream",
    # "That's one small step for man, one giant leap for mankind",
    # "In the beginning, the universe was created",
    # "I'm just a simple man trying to make my way in the universe",
    # "Do or do not, there is no try",
    # "To boldly go where no one has gone before",
    # "A long time ago in a galaxy far, far away",
```

```
    # "Et tu, Brute?",
    # "You can't handle the truth!",
    # "I'm the king of the world!",
    # "They may take our lives, but they'll never take our freedom!",
    # "Frankly, my dear, I don't give a damn",
    # "You talking to me?",
    # "Here's looking at you, kid",
    # "I love the smell of napalm in the morning",
    # "Say hello to my little friend",
    # "Houston, we have a problem",
    # "I'm gonna make him an offer he can't refuse",
    # "Keep your friends close, but your enemies closer",
    # "I feel the need—the need for speed",
    # "Carpe diem. Seize the day, boys",
    # "Elementary, my dear Watson",
    # "Life moves pretty fast. If you don't stop and look around once in a w
    # "Nobody puts Baby in a corner"
]

train_corpus = preprocess_data(train_data)

char_vocab = build_char_vocab(train_corpus)
word_vocab = build_word_vocab(train_corpus)

char_vocab_size = len(char_vocab)
char_embed_dim = 50
word_vocab_size = len(word_vocab)
print("word vocab size:", word_vocab_size)
conv_out_size = 256
hidden_dim = 512
'''
from figure 1: Note that in the above
example we have twelve filters—three filters of width two
(blue), four filters of width three (yellow), and five filters
of width four (red). Just added one more possibility (5)
'''
kernel_sizes = [2, 3, 4, 5]
features = [64, 64, 64, 64]
num_highway_layers = 2
num_rnn_layers = 2
dropout = 0.1

model = CharacterToWordModel(char_vocab_size, char_embed_dim, word_vocab_siz
                             conv_out_size, hidden_dim, kernel_sizes, featur
                             num_highway_layers, num_rnn_layers, dropout)

for name, param in model.named_parameters():
    print(name, param.dtype)

model.to(device)
max_sequence_length = 50
train_predictors, train_targets = data_to_tensors(train_corpus, char_vocab,
```

```
batch_size = 256
train_dataset = TensorDataset(train_predictors, train_targets)
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True

import torch.optim as optim

from tqdm import tqdm

num_epochs = 100
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

for epoch in range(num_epochs):
    model.train()
    total_loss = 0.0
    total_correct = 0
    total_samples = 0

    progress_bar = tqdm(train_loader, desc=f'Epoch {epoch+1}/{num_epochs}')

    for inputs, targets in progress_bar:
        optimizer.zero_grad()

        hidden_state = None
        inputs = inputs.to(device)
        targets = targets.to(device)
        predictions, _ = model(inputs, hidden_state)

        if targets.max() >= word_vocab_size:
            print("Invalid target index found:", targets.max())
            print("Inputs:", inputs)
            print("Inputs Shape:", inputs.shape)
            print("Targets:", targets)
            print("Targets Shape:", targets)
            break

        loss = criterion(predictions.view(-1, word_vocab_size), targets.view
        loss.backward()
        optimizer.step()

        total_loss += loss.item()

        _, predicted = torch.max(predictions.data, -1)
        total_correct += (predicted.view(-1) == targets.view(-1)).sum().item
        total_samples += targets.numel()

        avg_loss = total_loss / total_samples
        accuracy = total_correct / total_samples * 100
        progress_bar.set_postfix(loss=avg_loss, accuracy=f'{accuracy:.2f}%')

    print(f"Epoch {epoch+1} completed. Loss: {avg_loss:.4f}, Accuracy: {accu
```

```
Total unique words not presets: 9998
Total unique words including <pad> and <unk>: 9999
word vocab size: 9999
char_embedding.weight torch.float32
conv_net.conv_blocks.0.conv_layer.weight torch.float32
conv_net.conv_blocks.0.conv_layer.bias torch.float32
conv_net.conv_blocks.1.conv_layer.weight torch.float32
conv_net.conv_blocks.1.conv_layer.bias torch.float32
conv_net.conv_blocks.2.conv_layer.weight torch.float32
conv_net.conv_blocks.2.conv_layer.bias torch.float32
conv_net.conv_blocks.3.conv_layer.weight torch.float32
conv_net.conv_blocks.3.conv_layer.bias torch.float32
highway_net.highway_block_0.trans_bias torch.float32
highway_net.highway_block_0.project.weight torch.float32
highway_net.highway_block_0.project.bias torch.float32
highway_net.highway_block_0.transform.weight torch.float32
highway_net.highway_block_0.transform.bias torch.float32
highway_net.highway_block_1.trans_bias torch.float32
highway_net.highway_block_1.project.weight torch.float32
highway_net.highway_block_1.project.bias torch.float32
highway_net.highway_block_1.transform.weight torch.float32
highway_net.highway_block_1.transform.bias torch.float32
rnn_net.rnn_layers.weight_ih_l0 torch.float32
rnn_net.rnn_layers.weight_hh_l0 torch.float32
rnn_net.rnn_layers.bias_ih_l0 torch.float32
rnn_net.rnn_layers.bias_hh_l0 torch.float32
rnn_net.rnn_layers.weight_ih_l1 torch.float32
rnn_net.rnn_layers.weight_hh_l1 torch.float32
rnn_net.rnn_layers.bias_ih_l1 torch.float32
rnn_net.rnn_layers.bias_hh_l1 torch.float32
output_layer.weight torch.float32
output_layer.bias torch.float32
Epoch 1/100: 100%|██████████| 3303/3303 [03:05<00:00, 17.83it/s, accuracy=1
1.28%, loss=0.0242]
Epoch 1 completed. Loss: 0.0242, Accuracy: 11.28%
Epoch 2/100: 100%|██████████| 3303/3303 [03:04<00:00, 17.88it/s, accuracy=1
4.27%, loss=0.0226]
Epoch 2 completed. Loss: 0.0226, Accuracy: 14.27%
Epoch 3/100: 100%|██████████| 3303/3303 [03:05<00:00, 17.78it/s, accuracy=1
5.23%, loss=0.0219]
Epoch 3 completed. Loss: 0.0219, Accuracy: 15.23%
Epoch 4/100: 100%|██████████| 3303/3303 [03:04<00:00, 17.91it/s, accuracy=1
5.91%, loss=0.0214]
Epoch 4 completed. Loss: 0.0214, Accuracy: 15.91%
Epoch 5/100: 100%|██████████| 3303/3303 [03:05<00:00, 17.84it/s, accuracy=1
6.34%, loss=0.021]
Epoch 5 completed. Loss: 0.0210, Accuracy: 16.34%
Epoch 6/100: 100%|██████████| 3303/3303 [03:05<00:00, 17.83it/s, accuracy=1
6.68%, loss=0.0207]
Epoch 6 completed. Loss: 0.0207, Accuracy: 16.68%
```

```
Epoch 7/100: 100%|██████████| 3303/3303 [03:05<00:00, 17.82it/s, accuracy=16.96%, loss=0.0204]
Epoch 7 completed. Loss: 0.0204, Accuracy: 16.96%
Epoch 8/100: 100%|██████████| 3303/3303 [03:04<00:00, 17.86it/s, accuracy=17.22%, loss=0.0202]
Epoch 8 completed. Loss: 0.0202, Accuracy: 17.22%
Epoch 9/100: 100%|██████████| 3303/3303 [03:05<00:00, 17.78it/s, accuracy=17.41%, loss=0.02]
Epoch 9 completed. Loss: 0.0200, Accuracy: 17.41%
Epoch 10/100: 100%|██████████| 3303/3303 [03:04<00:00, 17.90it/s, accuracy=17.58%, loss=0.0198]
Epoch 10 completed. Loss: 0.0198, Accuracy: 17.58%
Epoch 11/100: 100%|██████████| 3303/3303 [03:05<00:00, 17.79it/s, accuracy=17.71%, loss=0.0196]
Epoch 11 completed. Loss: 0.0196, Accuracy: 17.71%
Epoch 12/100: 100%|██████████| 3303/3303 [03:04<00:00, 17.87it/s, accuracy=17.82%, loss=0.0195]
Epoch 12 completed. Loss: 0.0195, Accuracy: 17.82%
Epoch 13/100: 100%|██████████| 3303/3303 [03:05<00:00, 17.81it/s, accuracy=17.94%, loss=0.0194]
Epoch 13 completed. Loss: 0.0194, Accuracy: 17.94%
Epoch 14/100: 100%|██████████| 3303/3303 [03:05<00:00, 17.80it/s, accuracy=18.01%, loss=0.0193]
Epoch 14 completed. Loss: 0.0193, Accuracy: 18.01%
Epoch 15/100: 100%|██████████| 3303/3303 [03:04<00:00, 17.94it/s, accuracy=18.08%, loss=0.0192]
Epoch 15 completed. Loss: 0.0192, Accuracy: 18.08%
Epoch 16/100: 100%|██████████| 3303/3303 [03:05<00:00, 17.79it/s, accuracy=18.14%, loss=0.0191]
Epoch 16 completed. Loss: 0.0191, Accuracy: 18.14%
Epoch 17/100: 100%|██████████| 3303/3303 [03:04<00:00, 17.85it/s, accuracy=18.19%, loss=0.019]
Epoch 17 completed. Loss: 0.0190, Accuracy: 18.19%
Epoch 18/100: 100%|██████████| 3303/3303 [03:04<00:00, 17.86it/s, accuracy=18.24%, loss=0.019]
Epoch 18 completed. Loss: 0.0190, Accuracy: 18.24%
Epoch 19/100: 100%|██████████| 3303/3303 [03:05<00:00, 17.81it/s, accuracy=18.27%, loss=0.0189]
Epoch 19 completed. Loss: 0.0189, Accuracy: 18.27%
Epoch 20/100: 100%|██████████| 3303/3303 [03:05<00:00, 17.84it/s, accuracy=18.29%, loss=0.0189]
Epoch 20 completed. Loss: 0.0189, Accuracy: 18.29%
Epoch 21/100: 100%|██████████| 3303/3303 [03:04<00:00, 17.88it/s, accuracy=18.32%, loss=0.0188]
Epoch 21 completed. Loss: 0.0188, Accuracy: 18.32%
Epoch 22/100: 100%|██████████| 3303/3303 [03:05<00:00, 17.82it/s, accuracy=18.35%, loss=0.0188]
Epoch 22 completed. Loss: 0.0188, Accuracy: 18.35%
```

```
Epoch 23/100: 100%|██████████| 3303/3303 [03:05<00:00, 17.84it/s, accuracy=
18.38%, loss=0.0187]
Epoch 23 completed. Loss: 0.0187, Accuracy: 18.38%
Epoch 24/100: 100%|██████████| 3303/3303 [03:06<00:00, 17.71it/s, accuracy=
18.39%, loss=0.0187]
Epoch 24 completed. Loss: 0.0187, Accuracy: 18.39%
Epoch 25/100: 100%|██████████| 3303/3303 [03:05<00:00, 17.85it/s, accuracy=
18.43%, loss=0.0187]
Epoch 25 completed. Loss: 0.0187, Accuracy: 18.43%
Epoch 26/100: 100%|██████████| 3303/3303 [03:04<00:00, 17.95it/s, accuracy=
18.46%, loss=0.0186]
Epoch 26 completed. Loss: 0.0186, Accuracy: 18.46%
Epoch 27/100: 100%|██████████| 3303/3303 [03:05<00:00, 17.80it/s, accuracy=
18.46%, loss=0.0186]
Epoch 27 completed. Loss: 0.0186, Accuracy: 18.46%
Epoch 28/100: 100%|██████████| 3303/3303 [03:05<00:00, 17.78it/s, accuracy=
18.47%, loss=0.0186]
Epoch 28 completed. Loss: 0.0186, Accuracy: 18.47%
Epoch 29/100: 100%|██████████| 3303/3303 [03:05<00:00, 17.81it/s, accuracy=
18.49%, loss=0.0186]
Epoch 29 completed. Loss: 0.0186, Accuracy: 18.49%
Epoch 30/100: 100%|██████████| 3303/3303 [03:05<00:00, 17.76it/s, accuracy=
18.48%, loss=0.0185]
Epoch 30 completed. Loss: 0.0185, Accuracy: 18.48%
Epoch 31/100: 100%|██████████| 3303/3303 [03:05<00:00, 17.84it/s, accuracy=
18.52%, loss=0.0185]
Epoch 31 completed. Loss: 0.0185, Accuracy: 18.52%
Epoch 32/100: 100%|██████████| 3303/3303 [03:04<00:00, 17.88it/s, accuracy=
18.49%, loss=0.0185]
Epoch 32 completed. Loss: 0.0185, Accuracy: 18.49%
Epoch 33/100: 100%|██████████| 3303/3303 [03:06<00:00, 17.74it/s, accuracy=
18.52%, loss=0.0185]
Epoch 33 completed. Loss: 0.0185, Accuracy: 18.52%
Epoch 34/100: 100%|██████████| 3303/3303 [03:06<00:00, 17.74it/s, accuracy=
18.54%, loss=0.0185]
Epoch 34 completed. Loss: 0.0185, Accuracy: 18.54%
Epoch 35/100: 100%|██████████| 3303/3303 [03:05<00:00, 17.78it/s, accuracy=
18.55%, loss=0.0185]
Epoch 35 completed. Loss: 0.0185, Accuracy: 18.55%
Epoch 36/100: 100%|██████████| 3303/3303 [03:04<00:00, 17.86it/s, accuracy=
18.54%, loss=0.0184]
Epoch 36 completed. Loss: 0.0184, Accuracy: 18.54%
Epoch 37/100: 100%|██████████| 3303/3303 [03:04<00:00, 17.94it/s, accuracy=
18.56%, loss=0.0184]
Epoch 37 completed. Loss: 0.0184, Accuracy: 18.56%
Epoch 38/100: 100%|██████████| 3303/3303 [03:05<00:00, 17.80it/s, accuracy=
18.56%, loss=0.0184]
Epoch 38 completed. Loss: 0.0184, Accuracy: 18.56%
```

```
Epoch 39/100: 100%|██████████| 3303/3303 [03:06<00:00, 17.76it/s, accuracy=18.60%, loss=0.0184]
Epoch 39 completed. Loss: 0.0184, Accuracy: 18.60%
Epoch 40/100: 100%|██████████| 3303/3303 [03:05<00:00, 17.76it/s, accuracy=18.57%, loss=0.0184]
Epoch 40 completed. Loss: 0.0184, Accuracy: 18.57%
Epoch 41/100: 100%|██████████| 3303/3303 [03:06<00:00, 17.74it/s, accuracy=18.57%, loss=0.0184]
Epoch 41 completed. Loss: 0.0184, Accuracy: 18.57%
Epoch 42/100: 100%|██████████| 3303/3303 [03:04<00:00, 17.87it/s, accuracy=18.60%, loss=0.0184]
Epoch 42 completed. Loss: 0.0184, Accuracy: 18.60%
Epoch 43/100: 100%|██████████| 3303/3303 [03:05<00:00, 17.85it/s, accuracy=18.59%, loss=0.0184]
Epoch 43 completed. Loss: 0.0184, Accuracy: 18.59%
Epoch 44/100: 100%|██████████| 3303/3303 [03:05<00:00, 17.82it/s, accuracy=18.59%, loss=0.0183]
Epoch 44 completed. Loss: 0.0183, Accuracy: 18.59%
Epoch 45/100: 100%|██████████| 3303/3303 [03:05<00:00, 17.84it/s, accuracy=18.61%, loss=0.0183]
Epoch 45 completed. Loss: 0.0183, Accuracy: 18.61%
Epoch 46/100: 100%|██████████| 3303/3303 [03:05<00:00, 17.80it/s, accuracy=18.62%, loss=0.0183]
Epoch 46 completed. Loss: 0.0183, Accuracy: 18.62%
Epoch 47/100: 100%|██████████| 3303/3303 [03:05<00:00, 17.78it/s, accuracy=18.62%, loss=0.0183]
Epoch 47 completed. Loss: 0.0183, Accuracy: 18.62%
Epoch 48/100: 100%|██████████| 3303/3303 [03:03<00:00, 18.00it/s, accuracy=18.60%, loss=0.0183]
Epoch 48 completed. Loss: 0.0183, Accuracy: 18.60%
Epoch 49/100: 100%|██████████| 3303/3303 [03:05<00:00, 17.78it/s, accuracy=18.65%, loss=0.0183]
Epoch 49 completed. Loss: 0.0183, Accuracy: 18.65%
Epoch 50/100: 100%|██████████| 3303/3303 [03:05<00:00, 17.80it/s, accuracy=18.60%, loss=0.0183]
Epoch 50 completed. Loss: 0.0183, Accuracy: 18.60%
Epoch 51/100: 100%|██████████| 3303/3303 [03:05<00:00, 17.80it/s, accuracy=18.61%, loss=0.0183]
Epoch 51 completed. Loss: 0.0183, Accuracy: 18.61%
Epoch 52/100: 100%|██████████| 3303/3303 [03:06<00:00, 17.74it/s, accuracy=18.65%, loss=0.0183]
Epoch 52 completed. Loss: 0.0183, Accuracy: 18.65%
Epoch 53/100: 100%|██████████| 3303/3303 [03:05<00:00, 17.79it/s, accuracy=18.63%, loss=0.0183]
Epoch 53 completed. Loss: 0.0183, Accuracy: 18.63%
Epoch 54/100: 100%|██████████| 3303/3303 [03:06<00:00, 17.72it/s, accuracy=18.63%, loss=0.0183]
Epoch 54 completed. Loss: 0.0183, Accuracy: 18.63%
```

```
Epoch 55/100: 100%|██████████| 3303/3303 [03:05<00:00, 17.81it/s, accuracy=18.67%, loss=0.0182]
Epoch 55 completed. Loss: 0.0182, Accuracy: 18.67%
Epoch 56/100: 100%|██████████| 3303/3303 [03:05<00:00, 17.81it/s, accuracy=18.63%, loss=0.0182]
Epoch 56 completed. Loss: 0.0182, Accuracy: 18.63%
Epoch 57/100: 100%|██████████| 3303/3303 [03:05<00:00, 17.82it/s, accuracy=18.66%, loss=0.0182]
Epoch 57 completed. Loss: 0.0182, Accuracy: 18.66%
Epoch 58/100: 100%|██████████| 3303/3303 [03:05<00:00, 17.80it/s, accuracy=18.64%, loss=0.0182]
Epoch 58 completed. Loss: 0.0182, Accuracy: 18.64%
Epoch 59/100: 100%|██████████| 3303/3303 [03:05<00:00, 17.85it/s, accuracy=18.65%, loss=0.0182]
Epoch 59 completed. Loss: 0.0182, Accuracy: 18.65%
Epoch 60/100: 100%|██████████| 3303/3303 [03:05<00:00, 17.80it/s, accuracy=18.62%, loss=0.0182]
Epoch 60 completed. Loss: 0.0182, Accuracy: 18.62%
Epoch 61/100: 100%|██████████| 3303/3303 [03:05<00:00, 17.82it/s, accuracy=18.66%, loss=0.0182]
Epoch 61 completed. Loss: 0.0182, Accuracy: 18.66%
Epoch 62/100: 100%|██████████| 3303/3303 [03:06<00:00, 17.75it/s, accuracy=18.67%, loss=0.0182]
Epoch 62 completed. Loss: 0.0182, Accuracy: 18.67%
Epoch 63/100: 100%|██████████| 3303/3303 [03:07<00:00, 17.58it/s, accuracy=18.69%, loss=0.0182]
Epoch 63 completed. Loss: 0.0182, Accuracy: 18.69%
Epoch 64/100: 100%|██████████| 3303/3303 [03:07<00:00, 17.62it/s, accuracy=18.66%, loss=0.0182]
Epoch 64 completed. Loss: 0.0182, Accuracy: 18.66%
Epoch 65/100: 100%|██████████| 3303/3303 [03:08<00:00, 17.48it/s, accuracy=18.66%, loss=0.0182]
Epoch 65 completed. Loss: 0.0182, Accuracy: 18.66%
Epoch 66/100: 100%|██████████| 3303/3303 [03:08<00:00, 17.51it/s, accuracy=18.66%, loss=0.0182]
Epoch 66 completed. Loss: 0.0182, Accuracy: 18.66%
Epoch 67/100: 100%|██████████| 3303/3303 [03:09<00:00, 17.40it/s, accuracy=18.64%, loss=0.0182]
Epoch 67 completed. Loss: 0.0182, Accuracy: 18.64%
Epoch 68/100: 100%|██████████| 3303/3303 [03:08<00:00, 17.49it/s, accuracy=18.67%, loss=0.0182]
Epoch 68 completed. Loss: 0.0182, Accuracy: 18.67%
Epoch 69/100: 100%|██████████| 3303/3303 [03:08<00:00, 17.53it/s, accuracy=18.65%, loss=0.0182]
Epoch 69 completed. Loss: 0.0182, Accuracy: 18.65%
Epoch 70/100: 100%|██████████| 3303/3303 [03:09<00:00, 17.47it/s, accuracy=18.67%, loss=0.0182]
Epoch 70 completed. Loss: 0.0182, Accuracy: 18.67%
```

```
Epoch 71/100: 100%|██████████| 3303/3303 [03:09<00:00, 17.48it/s, accuracy=18.68%, loss=0.0182]
Epoch 71 completed. Loss: 0.0182, Accuracy: 18.68%
Epoch 72/100: 100%|██████████| 3303/3303 [03:09<00:00, 17.47it/s, accuracy=18.66%, loss=0.0182]
Epoch 72 completed. Loss: 0.0182, Accuracy: 18.66%
Epoch 73/100: 100%|██████████| 3303/3303 [03:09<00:00, 17.46it/s, accuracy=18.67%, loss=0.0182]
Epoch 73 completed. Loss: 0.0182, Accuracy: 18.67%
Epoch 74/100: 100%|██████████| 3303/3303 [03:09<00:00, 17.42it/s, accuracy=18.70%, loss=0.0182]
Epoch 74 completed. Loss: 0.0182, Accuracy: 18.70%
Epoch 75/100: 100%|██████████| 3303/3303 [03:07<00:00, 17.58it/s, accuracy=18.68%, loss=0.0182]
Epoch 75 completed. Loss: 0.0182, Accuracy: 18.68%
Epoch 76/100: 100%|██████████| 3303/3303 [03:09<00:00, 17.42it/s, accuracy=18.70%, loss=0.0181]
Epoch 76 completed. Loss: 0.0181, Accuracy: 18.70%
Epoch 77/100: 100%|██████████| 3303/3303 [03:09<00:00, 17.40it/s, accuracy=18.68%, loss=0.0181]
Epoch 77 completed. Loss: 0.0181, Accuracy: 18.68%
Epoch 78/100: 100%|██████████| 3303/3303 [03:09<00:00, 17.47it/s, accuracy=18.67%, loss=0.0181]
Epoch 78 completed. Loss: 0.0181, Accuracy: 18.67%
Epoch 79/100: 100%|██████████| 3303/3303 [03:09<00:00, 17.46it/s, accuracy=18.64%, loss=0.0181]
Epoch 79 completed. Loss: 0.0181, Accuracy: 18.64%
Epoch 80/100: 100%|██████████| 3303/3303 [03:07<00:00, 17.62it/s, accuracy=18.69%, loss=0.0181]
Epoch 80 completed. Loss: 0.0181, Accuracy: 18.69%
Epoch 81/100: 100%|██████████| 3303/3303 [03:09<00:00, 17.41it/s, accuracy=18.67%, loss=0.0181]
Epoch 81 completed. Loss: 0.0181, Accuracy: 18.67%
Epoch 82/100: 100%|██████████| 3303/3303 [03:08<00:00, 17.51it/s, accuracy=18.70%, loss=0.0181]
Epoch 82 completed. Loss: 0.0181, Accuracy: 18.70%
Epoch 83/100: 100%|██████████| 3303/3303 [03:09<00:00, 17.46it/s, accuracy=18.66%, loss=0.0181]
Epoch 83 completed. Loss: 0.0181, Accuracy: 18.66%
Epoch 84/100: 100%|██████████| 3303/3303 [03:08<00:00, 17.53it/s, accuracy=18.70%, loss=0.0181]
Epoch 84 completed. Loss: 0.0181, Accuracy: 18.70%
Epoch 85/100: 100%|██████████| 3303/3303 [03:08<00:00, 17.55it/s, accuracy=18.66%, loss=0.0181]
Epoch 85 completed. Loss: 0.0181, Accuracy: 18.66%
Epoch 86/100: 100%|██████████| 3303/3303 [03:08<00:00, 17.50it/s, accuracy=18.67%, loss=0.0181]
Epoch 86 completed. Loss: 0.0181, Accuracy: 18.67%
```

```
Epoch 87/100: 100%|██████████| 3303/3303 [03:09<00:00, 17.44it/s, accuracy=
18.68%, loss=0.0181]
Epoch 87 completed. Loss: 0.0181, Accuracy: 18.68%
Epoch 88/100: 100%|██████████| 3303/3303 [03:08<00:00, 17.52it/s, accuracy=
18.71%, loss=0.0181]
Epoch 88 completed. Loss: 0.0181, Accuracy: 18.71%
Epoch 89/100: 100%|██████████| 3303/3303 [03:09<00:00, 17.45it/s, accuracy=
18.72%, loss=0.0181]
Epoch 89 completed. Loss: 0.0181, Accuracy: 18.72%
Epoch 90/100: 100%|██████████| 3303/3303 [03:08<00:00, 17.51it/s, accuracy=
18.72%, loss=0.0181]
Epoch 90 completed. Loss: 0.0181, Accuracy: 18.72%
Epoch 91/100: 100%|██████████| 3303/3303 [03:08<00:00, 17.56it/s, accuracy=
18.70%, loss=0.0181]
Epoch 91 completed. Loss: 0.0181, Accuracy: 18.70%
Epoch 92/100: 100%|██████████| 3303/3303 [03:09<00:00, 17.48it/s, accuracy=
18.68%, loss=0.0181]
Epoch 92 completed. Loss: 0.0181, Accuracy: 18.68%
Epoch 93/100: 100%|██████████| 3303/3303 [03:09<00:00, 17.47it/s, accuracy=
18.72%, loss=0.0181]
Epoch 93 completed. Loss: 0.0181, Accuracy: 18.72%
Epoch 94/100: 100%|██████████| 3303/3303 [03:09<00:00, 17.44it/s, accuracy=
18.69%, loss=0.0181]
Epoch 94 completed. Loss: 0.0181, Accuracy: 18.69%
Epoch 95/100: 100%|██████████| 3303/3303 [03:09<00:00, 17.47it/s, accuracy=
18.69%, loss=0.0181]
Epoch 95 completed. Loss: 0.0181, Accuracy: 18.69%
Epoch 96/100: 100%|██████████| 3303/3303 [03:08<00:00, 17.53it/s, accuracy=
18.69%, loss=0.0181]
Epoch 96 completed. Loss: 0.0181, Accuracy: 18.69%
Epoch 97/100: 100%|██████████| 3303/3303 [03:09<00:00, 17.44it/s, accuracy=
18.68%, loss=0.0181]
Epoch 97 completed. Loss: 0.0181, Accuracy: 18.68%
Epoch 98/100: 100%|██████████| 3303/3303 [03:09<00:00, 17.41it/s, accuracy=
18.71%, loss=0.0181]
Epoch 98 completed. Loss: 0.0181, Accuracy: 18.71%
Epoch 99/100: 100%|██████████| 3303/3303 [03:08<00:00, 17.49it/s, accuracy=
18.69%, loss=0.0181]
Epoch 99 completed. Loss: 0.0181, Accuracy: 18.69%
Epoch 100/100: 100%|██████████| 3303/3303 [03:06<00:00, 17.75it/s, accuracy
=18.69%, loss=0.0181]
Epoch 100 completed. Loss: 0.0181, Accuracy: 18.69%
```

In [ ]:

In [ ]:

```
test_predictors, test_targets = data_to_tensors(test_corpus, char_vocab, wor

batch_size = 20
train_dataset = TensorDataset(train_predictors, train_targets)
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True

# # validation and test datasets
# valid_dataset = TensorDataset(valid_predictors, valid_targets)
# valid_loader = DataLoader(valid_dataset, batch_size=batch_size, shuffle=Fa

test_dataset = TensorDataset(test_predictors, test_targets)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
```

In [ ]:

```
test_corpus = preprocess_data(train_data)

test_char_vocab = build_char_vocab(test_corpus)
test_word_vocab = build_word_vocab(test_corpus)

for name, param in model.named_parameters():
    print(name, param.dtype)

model.to(device)
max_sequence_length = 50
test_predictors, test_targets = data_to_tensors(test_corpus, test_char_vocab
batch_size = 256
test_dataset = TensorDataset(test_predictors, test_targets)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=True)

# import torch.optim as optim

# from tqdm import tqdm

criterion = nn.CrossEntropyLoss()

import torch
import math

def calculate_perplexity(model, data_loader, criterion):
    model.eval()  # eval mode
    total_loss = 0
    total_words = 0

    with torch.no_grad():
        for inputs, targets in data_loader:
            inputs = inputs.to(device)
            targets = targets.to(device)
            outputs, _ = model(inputs, None)
            loss = criterion(outputs.view(-1, word_vocab_size), targets.view
            total_loss += loss.item() * inputs.size(0)
            total_words += inputs.size(0)

    average_loss = total_loss / total_words
    perplexity = math.exp(average_loss)

    return perplexity

perplexity = calculate_perplexity(model, test_loader, criterion)
print("Perplexity:", perplexity)
```

```
Total unique words not presets: 9998
Total unique words including <pad> and <unk>: 9999
char_embedding.weight torch.float32
conv_net.conv_blocks.0.conv_layer.weight torch.float32
conv_net.conv_blocks.0.conv_layer.bias torch.float32
conv_net.conv_blocks.1.conv_layer.weight torch.float32
conv_net.conv_blocks.1.conv_layer.bias torch.float32
conv_net.conv_blocks.2.conv_layer.weight torch.float32
conv_net.conv_blocks.2.conv_layer.bias torch.float32
conv_net.conv_blocks.3.conv_layer.weight torch.float32
conv_net.conv_blocks.3.conv_layer.bias torch.float32
highway_net.highway_block_0.trans_bias torch.float32
highway_net.highway_block_0.project.weight torch.float32
highway_net.highway_block_0.project.bias torch.float32
highway_net.highway_block_0.transform.weight torch.float32
highway_net.highway_block_0.transform.bias torch.float32
highway_net.highway_block_1.trans_bias torch.float32
highway_net.highway_block_1.project.weight torch.float32
highway_net.highway_block_1.project.bias torch.float32
highway_net.highway_block_1.transform.weight torch.float32
highway_net.highway_block_1.transform.bias torch.float32
rnn_net.rnn_layers.weight_ih_l0 torch.float32
rnn_net.rnn_layers.weight_hh_l0 torch.float32
rnn_net.rnn_layers.bias_ih_l0 torch.float32
rnn_net.rnn_layers.bias_hh_l0 torch.float32
rnn_net.rnn_layers.weight_ih_l1 torch.float32
rnn_net.rnn_layers.weight_hh_l1 torch.float32
rnn_net.rnn_layers.bias_ih_l1 torch.float32
rnn_net.rnn_layers.bias_hh_l1 torch.float32
output_layer.weight torch.float32
output_layer.bias torch.float32
Perplexity: 88.7196591145479
```