FASEROH Task - 1 Generate Taylor Series Data

Pushpdeep Singh(pushpdeep30@gmail.com)

(I am using taylor series data around zero) (Also, many polynomials fucntions have taylor series equal to themselves so I ignore them so our model doesn't learn y=x function)(This dataset file is uploaded on my github repo -)

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount
```

```
cd drive/MyDrive
```

```
/content/drive/MyDrive
```

```
import functools
from random import choices
import time
import sympy as sp
x = sp.Symbol('x')
import numpy as np

# leaves and binary operators
leaves = ["x", "-5", "-4", "-3", "-2", "-1", "1", "2", "3", "4", "5"]
p_leaf = [2/3, 1/30, 1/30, 1/30, 1/30, 1/30, 1/30, 1/30, 1/30, 1/30, 1/30]
binary_operators = ["+", "-", "*", "/"]
unary_operators = ["exp", "sqrt", "sin", "cos", "tan", "sinh", "cosh", "tanh", "asin", "acos", "atan", "asinh", "acosh", "atan


@functools.lru_cache(128)
def unary_binary_subtrees(e, n):
    """ calculate D(e, n), helper for distribution_k_a """
    if e==0:
        return 0
    elif n==0:
        return 1
    else:
        return unary_binary_subtrees(e-1, n) + unary_binary_subtrees(e, n-1) + unary_binary_subtrees(e+1, n-1)


def distribution_k_a(e, n):
    """ distribution over position k(e, n) """
    population = []
    weights = []
    for k in range(e):
        population.append( (k, 1))
        e_n = unary_binary_subtrees(e, n)
        weights.append(unary_binary_subtrees(e-k, n-1)/e_n)
        population.append((k, 2))
        weights.append(unary_binary_subtrees(e-k+1, n-1)/e_n)
    return population, weights


class NodeBinary:
    operator = True
    binary = True
    operand = False
    def __init__(self, data, left=None, right=None):
        self.data = data
        self.left = left
        self.right = right


class NodeUnary:
    operator = True
    binary = False
    operand = False
    def __init__(self, data, middle=None):
        self.data = data
        self.middle = middle


class Leaf:
```

```
    operator = False
    binary = False
    operand = True
    def __init__(self, data):
        self.data = data


def random_binary_trees(n):
    """
    Generates a random binary tree
    see Appendix C:
        n = num nodes
        k = position
        a = arity (unary or binary operator)
        e = number of empty nodes
    """
    population, weights = distribution_k_a(1, n)
    k, a = choices(population, weights)[0]

    #  select root node
    if a == 1:
        operator = choices(unary_operators)[0]
        node = NodeUnary(operator)
        empty_node = [(node, 'middle')]
        e = 1
    else:
        operator = choices(binary_operators)[0]
        node = NodeBinary(operator)
        empty_node = [(node, 'left'), (node, 'right')]
        e = 2

    #  making the tree
    n = n - 1
    while n > 0:
        population, weights = distribution_k_a(e, n)
        k, a = choices(population, weights)[0]
        i = 0
        new_empty_node = []
        for ele in empty_node:
            if i < k:
                leave = choices(leaves, p_leaf)[0]
                value = Leaf(leave)
                setattr(ele[0], ele[1], value)
            elif i == k:
                if a == 1:
                    operator = choices(unary_operators)[0]
                    value = NodeUnary(operator)
                    setattr(ele[0], ele[1], value)
                    if ele[1] == 'left':
                        new_ele = ele[0].left
                    elif ele[1] == 'middle':
                        new_ele = ele[0].middle
                    else:
                        new_ele = ele[0].right
                    new_empty_node = [(new_ele, 'middle')]
                    e = e - k
                else:
                    operator = choices(binary_operators)[0]
                    value = NodeBinary(operator)
                    setattr(ele[0], ele[1], value)
                    if ele[1] == 'left':
                        new_ele = ele[0].left
                    elif ele[1] == 'middle':
                        new_ele = ele[0].middle
                    else:
                        new_ele = ele[0].right
                    new_empty_node = [(new_ele, 'left'), (new_ele, 'right')]
                    e = e - k + 1
            else:
                new_empty_node.append(ele)
            i += 1
        empty_node = new_empty_node
        n = n - 1

    if len(empty_node) != 0:
        for ele in empty_node:
            leave = choices(leaves, p_leaf)[0]
            value = Leaf(leave)
            setattr(ele[0], ele[1], value)

    return node
```

```
def traverse_unary_binary_prefix(root, seq=None, verbose=False):
    """ traverses a binary expression tree and generates prefix notation """
    if not seq:
        seq = []

    if verbose:
        print(root.data)

    seq.append(root.data)

    if root.binary:
        if root.left.operator:
            traverse_unary_binary_prefix(root.left, seq)
        else:
            if verbose:
                print(root.left.data)
            seq.append(root.left.data)

        if root.right.operator:
            traverse_unary_binary_prefix(root.right, seq)
        else:
            if verbose:
                print(root.right.data)
            seq.append(root.right.data)
    else:
        if root.middle.operator:
            traverse_unary_binary_prefix(root.middle, seq)
        else:
            if verbose:
                print(root.middle.data)
            seq.append(root.middle.data)

    return seq
```

```
import random

from sympy import *



OPERATORS = set(['+', '-', '*', '/', '(', ')', 'pow'])

UNARY_OPERATORS = set(["exp", "sqrt", "sin", "cos", "tan", "asin", "acos", "atan", "sinh", "cosh", "tanh", "asinh", "acosh", "

operator_class = ["<class 'sympy.core.add.Add'>", "<class 'sympy.core.mul.Mul'>", "<class 'sympy.core.power.Pow'>", "exp", "si

operator_dict = {"<class 'sympy.core.add.Add'>": "+", "<class 'sympy.core.mul.Mul'>": "*", "<class 'sympy.core.power.Pow'>": "

rational_number = ["<class 'sympy.core.numbers.Rational'>", "<class 'sympy.core.numbers.Half'>"]

exp1_number = ["<class 'sympy.core.numbers.Exp1'>"]

invalid_function = ["<class 'sympy.core.numbers.ImaginaryUnit'>", "<class 'sympy.core.numbers.ComplexInfinity'>", "<class 'sym

invalid_expression = ["zoo", "oo", "I", "-oo"]

#x = symbols('x')
def infix_to_prefix(expr, seq=None):
    if not seq:
        seq = []
    s1 = str(expr)
    s2 = str(expr.func)
    if s1 in invalid_expression or s2 in invalid_function:
        return False
    length = len(expr.args)
    if s2 in operator_class:
        op = operator_dict[s2]
        seq.append(op)
    elif s2 in rational_number:
        s1 = s1.split("/")
        seq.append("/")
        seq.append(s1[0])
        seq.append(s1[1])
    elif s2 in exp1_number:
        seq.append("exp")
        seq.append("1")
    else:
        seq.append(s1)

    for i in range(length):
        if i!=0 and i!=(length-1):
```

```
            seq.append(op)
        arg = expr.args[i]
        feedback = infix_to_prefix(arg, seq)
        if feedback==False:
            return False
    return seq

### PREFIX ===> INFIX ###
'''
Scan the formula reversely
1) When the token is an operand, push into stack
2) When the token is an operator, pop out 2 numbers from stack, merge them and push back to the stack
'''
def prefix_to_infix(formula):
    stack = []
    for ch in reversed(formula):
        if ch not in OPERATORS and ch not in UNARY_OPERATORS:
            stack.append(ch)
        else:
            if ch in OPERATORS:
                a = stack.pop()
                b = stack.pop()
                exp = "(" + " " + a + " " + ch + " " + b + " " + ")"
            else:
                a = stack.pop()
                exp = ch + " " + "(" + " " +  a + " " +")"
            stack.append(exp)
    return stack[-1].split()
```

```
import random
import time

from sympy import Symbol, diff, simplify
```

```
_FINISH = False
x = Symbol('x', real=True) # TODO


def simplify_timeout(expr):
    from multiprocessing import Process, Manager

    result = None
    def f(d, expr):
        from sympy import S; S.Half
        try:
            d['result'] = simplify(expr)
        except:
            d['result'] = "Error"

    with Manager() as manager:
        d = manager.dict()
        p = Process(target=f, args=(d, expr))
        p.start()
        p.join(timeout=3)
        if p.is_alive():
            p.terminate()
        else:
            result = d["result"]
    if result==None:
        return expr
    elif str(type(result))=="<class 'sympy.calculus.util.AccumulationBounds'>":
        return "Error"
    else:
        return result

def series_timeout(ff):
  from multiprocessing import Process, Manager
  expr = None
  def f(d, ff):
      from sympy import S; S.Half
      try:
          d['expr'] = ff.series(x, 0, 5).removeO()
      except:
          d['expr'] = "Error"

  with Manager() as manager:
      d = manager.dict()
```

```
      p = Process(target=f, args=(d, ff))
      p.start()
      p.join(timeout=3)
      if p.is_alive():
          p.terminate()
      else:
          expr = d["expr"]


  if expr==None:
      return expr

  else:
    return expr

def parse_expr_timeout(string):
    from multiprocessing import Process, Manager
    from sympy.parsing.sympy_parser import parse_expr

    expr = None
    def f(d, string):
        from sympy import S; S.Half
        try:
            d['expr'] = parse_expr(string, local_dict={'x': x}, evaluate=False)
        except:
            return None

    with Manager() as manager:
        d = manager.dict()
        p = Process(target=f, args=(d, string))
        p.start()
        p.join(timeout=3)
        if p.is_alive():
            p.terminate()
        else:
            expr = d["expr"]

    return expr
```

```
cd taylor/data_generation
```

```
/content/drive/MyDrive/taylor/data_generation
```

```
def generate_single_sequence():
    n = random.randint(1, 3)
    root = random_binary_trees(n)
    result = traverse_unary_binary_prefix(root)
    if 'x' not in result:
        return None

    result_infix = prefix_to_infix(result)
    result_expr = " ".join(result_infix)

    #  convert to sympy expression
    result_expr = parse_expr_timeout(result_expr)
    if result_expr is None:
        return None

    #REJECT IF POLYNOMIAL
    if result_expr.is_polynomial():
      return None

    #  simplification
    result_simp = simplify_timeout(result_expr)
    if result_simp == "Error":
        return None


    #  back to prefix
    result_simp_prefix = infix_to_prefix(result_simp)
    if not result_simp_prefix:
        return None

    #  generate target
    try:
        taylor_expn =  series_timeout(result_simp)
    except ValueError:
        return None
    if taylor_expn == "Error" or None:
      return None
```

```
    expression = simplify_timeout(taylor_expn)

    if expression == "Error":
        return None

    expression_prefix = infix_to_prefix(expression)
    if not expression_prefix:
        return None

    if expression_prefix == result_simp_prefix or len(expression_prefix)>50:
      return None

    expression_prefix = " ".join(expression_prefix)
    result_simp_prefix = " ".join(result_simp_prefix)
    return expression_prefix, result_simp_prefix


def generate_bwd(num):
    filename = "taylor_data_10000.txt"


    while True:
        if _FINISH:
            break
        sequence = []
        start = time.time()
        file = open(filename, "a")
        i = 1
        while i<=num:
            seq = generate_single_sequence()
            if seq is not None:
                expression_prefix, result_simp_prefix = seq
                sequence.append(result_simp_prefix + "\t" + expression_prefix + "\n")
                print((result_simp_prefix + "\t" + expression_prefix + "\n"))
                file.write(result_simp_prefix + "\t" + expression_prefix + "\n")
            else:
                continue
            i += 1

        end = time.time()
        print('process finished')

        file.close()
        return sequence
```

```
generate_bwd(10000)
```

```
    * cosh x exp 5  * / 1 24 * + 24 + pow x 4 * 12 pow x 2 exp 5

    cos + -1 pow x 2        + * pow x 2 sin 1 + * / -1 2 * pow x 4 cos 1 cos 1

    sinh x  + x * / 1 6 pow x 3

    * pow x 2 asinh x       pow x 3

    pow + 1 * -1 x / 1 2    + 1 + * / -5 128 pow x 4 + * / -1 2 x + * / -1 8 pow x 2 *

    + * -1 x cos + 5 x      + * -1 x + * -1 * x sin 5 + * / -1 2 * pow x 2 cos 5 + * /

    ---------------------------------------------------------------------------
    KeyboardInterrupt                         Traceback (most recent call last)
    <ipython-input-11-a539e182727c> in <cell line: 1>()
    ----> 1 generate_bwd(10000)

    ------------------------------------ 6 frames ------------------------------------
    /usr/lib/python3.9/selectors.py in select(self, timeout)
        414         ready = []
        415         try:
    --> 416             fd_event_list = self._selector.poll(timeout)
        417         except InterruptedError:
        418             return ready

    KeyboardInterrupt:

    SEARCH STACK OVERFLOW
```

Task-2 Processing Data for Model

```
cd -
```

```
/content/drive/MyDrive
```

```
import os
import io
import numpy as np
import re
import unicodedata
import urllib3
import shutil
import zipfile
import itertools
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import numpy as np
import pandas
import spacy
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
from tqdm import tqdm_notebook
import random
from collections import Counter
import torchtext
from torchtext.data import get_tokenizer
from tqdm import tqdm
import math
```

```
# Converts the unicode file to ascii
def unicode_to_ascii(s):
    return ''.join(c for c in unicodedata.normalize('NFD', s)
      if unicodedata.category(c) != 'Mn')

def preprocess_sentence(w):
    w = unicode_to_ascii(w.lower().strip())
    return w
```

```
# 1. Remove the accents
# 2. Clean the sentences
# 3. Return word pairs in the format: [equation, series]
def create_dataset(path, num_examples=None):
    lines = io.open(path, encoding='UTF-8').read().strip().split('\n')

    word_pairs = [[preprocess_sentence(w) for w in l.split('\t')]  for l in lines[:num_examples]]

    return zip(*word_pairs)
```

```
path_to_file = "./taylor/data_generation/taylor_cleaned_pre.txt"
```

```
eq, series = create_dataset(path_to_file)
print(eq[-1])
print(series[-1])
```

```
* / 1 3 * pow x -1 acos x
* pow x -1 + * / -1 3 x + * / -1 18 pow x 3 + * / -1 40 pow x 5 * / 1 6 pi
```

```
X_text, Y_text = create_dataset(path_to_file)
```

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

```
tokenizer = get_tokenizer(tokenizer = None, language = None)
```

```
class FunctionDataset(torch.utils.data.Dataset):

  def __init__(self, ops=3, max_seq_length=32):

    raw_input = []
    raw_output = []

    for i in range(len(X_text)):
      expr, tay = tokenizer(X_text[i]), tokenizer(Y_text[i])

      #discards expressions too long and nan values
      if(len(expr) + 2 <= max_seq_length and len(tay) + 2 <= max_seq_length):
```

```
        #insert start and end tokens
        expr.insert(0,'<SOS>')
        expr.append('<EOS>')

        tay.insert(0,'<SOS>')
        tay.append('<EOS>')

        raw_input.append(expr)
        raw_output.append(tay)
        print("expression", expr)
        print("taylor series around 0", tay)

    #generate vocab
    self.vocab = set()
    for expr, tay in zip(raw_input, raw_output):
      self.vocab |= set(expr) |(set(tay))

    #token -> idx
    self.token_to_idx = {value : index + 1 for index, value in enumerate(self.vocab)}

    #idx -> token
    self.idx_to_token = {index + 1 : value for index, value in enumerate(self.vocab)}

    self.input = []
    self.output = []
    for raw_expr, raw_tay in zip(raw_input, raw_output):
      expr = [self.token_to_idx[token] for token in raw_expr] + [0] * (max_seq_length - len(raw_expr))
      tay = [self.token_to_idx[token] for token in raw_tay] + [0] * (max_seq_length - len(raw_tay))

      self.input.append(torch.tensor(expr, dtype=torch.long, device=device))
      self.output.append(torch.tensor(tay, dtype=torch.long, device=device))

  def __len__(self):
    return len(self.input)

  def __getitem__(self, idx):
    return self.input[idx].to(device), self.output[idx].to(device)

  def get_alphabet(self):
    return self.vocab
```

```
d = FunctionDataset()
train_idx = list(range(0, int(9*len(d)/10)))
test_idx = list(range(int(9*len(d)/10), len(d)))
train_dataset = torch.utils.data.Subset(d, train_idx)
test_dataset = torch.utils.data.Subset(d, test_idx)
```

```
Streaming output truncated to the last 5000 lines.
expression ['<SOS>', '+', '2', '*', '/', '1', '5', 'sin', 'x', '<EOS>']
taylor series around 0 ['<SOS>', '+', '2', '+', '*', '/', '-1', '30', 'pow', 'x', '3', '*', '/', '1', '5', 'x', '<EOS>']
expression ['<SOS>', 'sinh', '*', '/', '2', '5', 'x', '<EOS>']
taylor series around 0 ['<SOS>', '*', '/', '2', '375', '*', 'x', '+', '75', '*', '2', 'pow', 'x', '2', '<EOS>']
expression ['<SOS>', '*', '-1', 'sinh', '*', 'x', 'pow', '+', '3', 'x', '-1', '<EOS>']
taylor series around 0 ['<SOS>', '*', '/', '1', '162', '*', 'x', '+', '-54', '+', '*', '-7', 'pow', 'x', '2', '+', '*', '
expression ['<SOS>', '+', 'x', '*', '4', 'sin', 'x', '<EOS>']
taylor series around 0 ['<SOS>', '*', '/', '1', '3', '*', 'x', '+', '15', '*', '-2', 'pow', 'x', '2', '<EOS>']
expression ['<SOS>', '*', 'tanh', '5', 'tanh', 'x', '<EOS>']
taylor series around 0 ['<SOS>', '*', '/', '-1', '3', '*', 'x', '*', '+', '-3', 'pow', 'x', '2', 'tanh', '5', '<EOS>']
expression ['<SOS>', '+', '4', '*', 'pow', 'x', '-1', 'tanh', 'x', '<EOS>']
taylor series around 0 ['<SOS>', '+', '5', '+', '*', '/', '-1', '3', 'pow', 'x', '2', '*', '/', '2', '15', 'pow', 'x', '4
expression ['<SOS>', 'tanh', '+', 'x', '*', '-1', 'tanh', 'x', '<EOS>']
taylor series around 0 ['<SOS>', '*', '/', '1', '3', 'pow', 'x', '3', '<EOS>']
expression ['<SOS>', 'pow', 'atanh', 'pow', 'x', '2', '/', '1', '2', '<EOS>']
taylor series around 0 ['<SOS>', 'x', '<EOS>']
expression ['<SOS>', 'cosh', '+', '5', 'pow', 'x', '2', '<EOS>']
taylor series around 0 ['<SOS>', '+', '*', 'pow', 'x', '2', 'sinh', '5', '+', '*', '/', '1', '2', '*', 'pow', 'x', '4', '
expression ['<SOS>', '*', '-1', 'asin', '+', 'x', '*', '-1', 'atan', 'x', '<EOS>']
taylor series around 0 ['<SOS>', '*', '/', '-1', '3', 'pow', 'x', '3', '<EOS>']
expression ['<SOS>', '*', '/', '-1', '2', '*', 'pow', 'x', '-1', 'asinh', 'x', '<EOS>']
taylor series around 0 ['<SOS>', '+', '/', '-1', '2', '+', '*', '/', '-3', '80', 'pow', 'x', '4', '*', '/', '1', '12', 'p
expression ['<SOS>', 'atanh', 'atanh', 'asinh', 'x', '<EOS>']
taylor series around 0 ['<SOS>', '+', 'x', '*', '/', '1', '2', 'pow', 'x', '3', '<EOS>']
expression ['<SOS>', '*', '5', 'cos', 'sin', 'x', '<EOS>']
taylor series around 0 ['<SOS>', '+', '5', '+', '*', '/', '-5', '2', 'pow', 'x', '2', '*', '/', '25', '24', 'pow', 'x', '
expression ['<SOS>', 'exp', 'asin', 'atan', 'x', '<EOS>']
taylor series around 0 ['<SOS>', '+', '1', '+', 'x', '+', '*', '/', '1', '2', 'pow', 'x', '2', '*', '/', '-1', '8', 'pow'
expression ['<SOS>', '*', '-1', 'atanh', '*', '4', 'cos', 'x', '<EOS>']
taylor series around 0 ['<SOS>', '+', '*', '-1', 'atanh', '4', '+', '*', '/', '-3', '50', 'pow', 'x', '4', '*', '/', '-2'
expression ['<SOS>', 'atanh', 'atan', 'atan', 'x', '<EOS>']
taylor series around 0 ['<SOS>', '+', 'x', '*', '/', '-1', '3', 'pow', 'x', '3', '<EOS>']
expression ['<SOS>', '*', '4', '*', 'x', 'atan', 'x', '<EOS>']
taylor series around 0 ['<SOS>', '*', '/', '4', '3', '*', 'pow', 'x', '2', '+', '3', '*', '-1', 'pow', 'x', '2', '<EOS>']
expression ['<SOS>', '*', '5', 'cosh', 'asin', 'x', '<EOS>']
taylor series around 0 ['<SOS>', '+', '5', '+', '*', '/', '5', '2', 'pow', 'x', '2', '*', '/', '25', '24', 'pow', 'x', '4
expression ['<SOS>', '*', '-3', 'pow', 'atan', 'pow', 'x', '2', '-1', '<EOS>']
```

```
taylor series around 0 ['<SOS>', '*', 'pow', 'x', '-2', '+', '-3', '*', '-1', 'pow', 'x', '4', '<EOS>']
expression ['<SOS>', 'atanh', 'sinh', 'x', '<EOS>']
taylor series around 0 ['<SOS>', '+', 'x', '*', '/', '1', '2', 'pow', 'x', '3', '<EOS>']
expression ['<SOS>', '+', '-4', 'asin', 'sinh', 'x', '<EOS>']
taylor series around 0 ['<SOS>', '+', '-4', '+', 'x', '*', '/', '1', '3', 'pow', 'x', '3', '<EOS>']
expression ['<SOS>', 'asinh', '*', '5', 'asinh', 'x', '<EOS>']
taylor series around 0 ['<SOS>', '+', '*', '5', 'x', '*', '/', '-65', '3', 'pow', 'x', '3', '<EOS>']
expression ['<SOS>', '*', 'pow', 'atanh', 'x', '-1', 'tanh', '4', '<EOS>']
taylor series around 0 ['<SOS>', '*', '/', '-1', '45', '*', 'pow', 'x', '-1', '*', '+', '-45', '+', '*', '4', 'pow', 'x',
expression ['<SOS>', '*', '-1', 'asin', '*', '5', 'atan', 'x', '<EOS>']
taylor series around 0 ['<SOS>', '+', '*', '-5', 'x', '*', '/', '-115', '6', 'pow', 'x', '3', '<EOS>']
expression ['<SOS>', '*', 'x', '+', '4', 'atan', 'x', '<EOS>']
taylor series around 0 ['<SOS>', '*', '/', '1', '3', '*', 'x', '+', '12', '+', '*', '-1', 'pow', 'x', '3', '*', '3', 'x',
expression ['<SOS>', 'cosh', '*', '4', 'x', '<EOS>']
taylor series around 0 ['<SOS>', '+', '1', '+', '*', '8', 'pow', 'x', '2', '*', '/', '32', '3', 'pow', 'x', '4', '<EOS>']
expression ['<SOS>', '+', '5', '*', 'x', 'pow', 'sinh', 'x', '-1', '<EOS>']
taylor series around 0 ['<SOS>', '+', '6', '+', '*', '/', '-1', '6', 'pow', 'x', '2', '*', '/', '7', '360', 'pow', 'x', '
expression ['<SOS>', '*', 'pow', 'x', '-1', '+', '4', 'cos', 'x', '<EOS>']
taylor series around 0 ['<SOS>', '+', '*', '5', 'pow', 'x', '-1', '+', '*', '/', '-1', '2', 'x', '*', '/', '1', '24', 'po
expression ['<SOS>', 'sinh', '+', 'x', '*', '-1', 'atanh', 'x', '<EOS>']
```

```
len(train_idx)
```

```
10065
```

```
len(test_idx)
```

```
1119
```

LSTM Model

```
#Defining Encoder and Decoder Class and then Model

class Encoder(nn.Module):
  def __init__(self, vocab_size, embedding_dim=512, num_layers=2, hidden_size=512, dropout=0.2):
    super(Encoder, self).__init__()
    self.embedding_dim = embedding_dim
    self.num_layers = num_layers
    self.hidden_size = hidden_size

    self.embedding = nn.Embedding(
        num_embeddings=vocab_size,
        embedding_dim=self.embedding_dim
    )
    self.lstm = nn.LSTM(
        input_size=self.embedding_dim,
        hidden_size=self.hidden_size,
        num_layers=self.num_layers,
        dropout=dropout,
    )
  """
  input shape (SEQUENCE_LENGTH, BATCH_SIZE)
  h,c shape (HIDDEN_SIZE)
  """
  def forward(self, x):
    embed = self.embedding(x)
    output, (h,c) = self.lstm(embed)
    return h, c

class Decoder(nn.Module):
  def __init__(self, vocab_size, embedding_dim=512, num_layers=2, hidden_size=512, dropout=0.2):
    super(Decoder, self).__init__()
    self.embedding_dim = embedding_dim
    self.num_layers = num_layers
    self.output_size = vocab_size
    self.hidden_size = hidden_size

    self.embedding = nn.Embedding(
        num_embeddings=vocab_size,
        embedding_dim=self.embedding_dim
    )
    self.lstm = nn.LSTM(
        input_size=self.embedding_dim,
        hidden_size=self.hidden_size,
        num_layers=self.num_layers,
        dropout=0.2,
    )
    self.out = nn.Linear(self.hidden_size, self.output_size)
    self.softmax = nn.LogSoftmax(dim=2)
    self.to(device)
  """
```

```
  input shape (BATCH_SIZE)
  output shape
  """
  def forward(self, input, h_0, c_0):
    embedded = self.embedding(input.unsqueeze(0))
    output, (h,c) = self.lstm(embedded, (h_0, c_0))
    output = self.out(output)
    output = self.softmax(output)
    return output.squeeze(0), h , c

class Model(nn.Module):
  def __init__(self, encoder, decoder):
    super(Model, self).__init__()
    self.encoder = encoder
    self.decoder = decoder
    self.to(device)
  """
  Input tensor of shape (SEQUENCE_LENGTH, BATCH_SIZE)
  Output tensor of shape (SEQUENCE_LENGTH, BATCH_SIZE, VOCAB_SIZE)

  if tgt is none use teacher forecasting
  """
  def forward(self, input, tgt=None):
    if len(input.shape) < 2:
        input = input.unsqueeze(1)
    batch_size = input.shape[1]
    h, c = enc(input)
    target = torch.zeros(batch_size, dtype=torch.long).to(device)
    if tgt is None:
      max_seq_length = input.shape[0]
      target[:] = d.token_to_idx['<SOS>']
    else:
      max_seq_length = tgt.shape[1]
      target[:] = tgt[:,0]
    outputs = torch.zeros(max_seq_length, batch_size, dec.output_size, dtype=torch.float).to(device)
    for i in range(max_seq_length):
        prediction, h, c = dec(target, h, c)
        outputs[i] = prediction
        if tgt is None:
          target = prediction.argmax(dim=1)
        else:
          target = tgt[:,i]
    return outputs
```

```
#Instantiate Model

enc = Encoder(len(d.get_alphabet()) + 1)
dec = Decoder(len(d.get_alphabet()) + 1)
m = Model(enc,dec).to(device)
```

```
#Train and Test Epoch

def train_epoch_LSTM(model, train_loader, optimizer, criterion, batch_size=256):
  model.train()
  total_loss = 0
  total_items = 0
  num_correct = 0
  for src, tgt in tqdm(train_loader):
    src = src.to(device)
    tgt = tgt.to(device)

    pred = model(src.squeeze().T,tgt=tgt[:,:-1])

    pred = pred.permute((1,2,0))
    tgt_out = tgt[:,1:]
    loss = criterion(pred, tgt_out)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    total_loss += loss.item()
    total_items += (tgt_out != 0).sum(dim=(0,1))

    num_correct += (torch.logical_and((pred.argmax(dim=1) == tgt_out), (tgt_out != 0))).sum(dim=(0,1))
  return total_loss, num_correct / total_items

def test_epoch_LSTM(model, test_loader, criterion, batch_size=256):
  model.eval()
  total_loss = 0
  total_items = 0
```

```
  num_correct = 0
  for src, tgt in tqdm(test_loader):
    src = src.to(device)
    tgt = tgt.to(device)

    pred = model(src.squeeze().T, tgt=tgt[:,:-1])
    pred = pred.permute((1,2,0))
    tgt_out = tgt[:,1:]

    loss = criterion(pred, tgt_out)

    total_loss += loss.item()
    total_items += (tgt_out != 0).sum(dim=(0,1))

    num_correct += (torch.logical_and((logits.argmax(dim=2) == tgt_out), (tgt_out != 0))).sum(dim=(0,1))
  return total_loss, num_correct / total_items
```

```
#Method to train LSTM Model
def train_LSTM(model, train_dataset,  test_dataset, batch_size=256, epochs=20):
  train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
  test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=batch_size, shuffle=True)
  criterion = nn.CrossEntropyLoss()


  optim = torch.optim.Adam(model.parameters(), lr=1e-3)
  for e in range(epochs):
    train_loss, train_acc = train_epoch_LSTM(model, train_loader, optim, criterion, batch_size=batch_size)
    test_loss, test_acc = train_epoch_LSTM(model, test_loader, optim, criterion, batch_size=batch_size)
    print(f'Epoch: {e + 1} Training Loss: {train_loss} Training Accuracy: {train_acc} Test Loss: {test_loss} Test Accuracy: {t
```

```
#Running training
train_LSTM(m, train_dataset, test_dataset, batch_size=256)
```

```
100%|██████████| 40/40 [00:05<00:00,  7.02it/s]
100%|██████████| 5/5 [00:00<00:00,  7.63it/s]
Epoch: 1 Training Loss: 79.44888865947723 Training Accuracy: 0.2158389538526535 Test Loss: 6.193756699562073 Test Accurac
100%|██████████| 40/40 [00:05<00:00,  7.04it/s]
100%|██████████| 5/5 [00:00<00:00,  7.71it/s]
Epoch: 2 Training Loss: 40.253355503082275 Training Accuracy: 0.5204954743385315 Test Loss: 4.453214883804321 Test Accura
100%|██████████| 40/40 [00:05<00:00,  6.92it/s]
100%|██████████| 5/5 [00:00<00:00,  7.50it/s]
Epoch: 3 Training Loss: 32.8906552195549 Training Accuracy: 0.5804269313812256 Test Loss: 3.918783187866211 Test Accuracy
100%|██████████| 40/40 [00:05<00:00,  6.81it/s]
100%|██████████| 5/5 [00:00<00:00,  7.32it/s]
Epoch: 4 Training Loss: 29.63807874917984 Training Accuracy: 0.6141420602798462 Test Loss: 3.545790135860443 Test Accurac
100%|██████████| 40/40 [00:05<00:00,  6.72it/s]
100%|██████████| 5/5 [00:00<00:00,  7.30it/s]
Epoch: 5 Training Loss: 26.25686478614807 Training Accuracy: 0.6641221046447754 Test Loss: 3.1763095259666443 Test Accura
100%|██████████| 40/40 [00:06<00:00,  6.63it/s]
100%|██████████| 5/5 [00:00<00:00,  7.15it/s]
Epoch: 6 Training Loss: 22.864578425884247 Training Accuracy: 0.7054709196090698 Test Loss: 2.723171830177307 Test Accura
100%|██████████| 40/40 [00:06<00:00,  6.57it/s]
100%|██████████| 5/5 [00:00<00:00,  7.00it/s]
Epoch: 7 Training Loss: 19.83882439136505 Training Accuracy: 0.7431525588035583 Test Loss: 2.3302100002765656 Test Accura
100%|██████████| 40/40 [00:06<00:00,  6.53it/s]
100%|██████████| 5/5 [00:00<00:00,  7.18it/s]
Epoch: 8 Training Loss: 17.453968107700348 Training Accuracy: 0.7710763216018677 Test Loss: 2.061834752559662 Test Accura
100%|██████████| 40/40 [00:06<00:00,  6.54it/s]
100%|██████████| 5/5 [00:00<00:00,  7.13it/s]
Epoch: 9 Training Loss: 15.570676654577255 Training Accuracy: 0.7931501865386963 Test Loss: 1.8753323554992676 Test Accur
100%|██████████| 40/40 [00:06<00:00,  6.60it/s]
100%|██████████| 5/5 [00:00<00:00,  7.20it/s]
Epoch: 10 Training Loss: 13.97776660323143 Training Accuracy: 0.8117914795875549 Test Loss: 1.687963455915451 Test Accura
100%|██████████| 40/40 [00:06<00:00,  6.65it/s]
100%|██████████| 5/5 [00:00<00:00,  7.30it/s]
Epoch: 11 Training Loss: 12.714886635541916 Training Accuracy: 0.8273229598999023 Test Loss: 1.5550552904605865 Test Accu
100%|██████████| 40/40 [00:05<00:00,  6.68it/s]
100%|██████████| 5/5 [00:00<00:00,  7.35it/s]
Epoch: 12 Training Loss: 11.601215869188309 Training Accuracy: 0.8419156670570374 Test Loss: 1.441448301076889 Test Accur
100%|██████████| 40/40 [00:05<00:00,  6.70it/s]
100%|██████████| 5/5 [00:00<00:00,  7.39it/s]
Epoch: 13 Training Loss: 10.676187172532082 Training Accuracy: 0.8552291989326477 Test Loss: 1.2904720306396484 Test Accu
100%|██████████| 40/40 [00:05<00:00,  6.72it/s]
100%|██████████| 5/5 [00:00<00:00,  7.29it/s]
Epoch: 14 Training Loss: 9.749161124229431 Training Accuracy: 0.866694450378418 Test Loss: 1.171163260936737 Test Accurac
100%|██████████| 40/40 [00:05<00:00,  6.75it/s]
100%|██████████| 5/5 [00:00<00:00,  7.32it/s]
Epoch: 15 Training Loss: 8.899881973862648 Training Accuracy: 0.8786056041717529 Test Loss: 1.077872559428215 Test Accura
100%|██████████| 40/40 [00:05<00:00,  6.74it/s]
100%|██████████| 5/5 [00:00<00:00,  7.34it/s]
Epoch: 16 Training Loss: 8.186477705836296 Training Accuracy: 0.8887330889701843 Test Loss: 0.9603532254695892 Test Accur
100%|██████████| 40/40 [00:05<00:00,  6.73it/s]
100%|██████████| 5/5 [00:00<00:00,  7.36it/s]
Epoch: 17 Training Loss: 7.484002277255058 Training Accuracy: 0.8990306854248047 Test Loss: 0.9212912917137146 Test Accur
100%|██████████| 40/40 [00:05<00:00,  6.69it/s]
100%|██████████| 5/5 [00:00<00:00,  7.38it/s]
```

```
Epoch: 18 Training Loss: 6.802447006106377 Training Accuracy: 0.9074389338493347 Test Loss: 0.8278125524520874 Test Accu
100%|██████████| 40/40 [00:05<00:00,  6.70it/s]
100%|██████████| 5/5 [00:00<00:00,  7.35it/s]
Epoch: 19 Training Loss: 6.1967692375183105 Training Accuracy: 0.9162402749061584 Test Loss: 0.7322542071342468 Test Accu
100%|██████████| 40/40 [00:06<00:00,  6.66it/s]
```

Transformer Model

```
#Defining Transformer Model

class PositionalEncoding(nn.Module):
    def __init__(self, emb_size: int, dropout, maxlen: int = 5000):
        super(PositionalEncoding, self).__init__()
        den = torch.exp(- torch.arange(0, emb_size, 2) * math.log(10000) / emb_size)
        pos = torch.arange(0, maxlen).reshape(maxlen, 1)
        pos_embedding = torch.zeros((maxlen, emb_size))
        pos_embedding[:, 0::2] = torch.sin(pos * den)
        pos_embedding[:, 1::2] = torch.cos(pos * den)
        pos_embedding = pos_embedding.unsqueeze(-2)

        self.dropout = nn.Dropout(dropout)
        self.register_buffer('pos_embedding', pos_embedding)

    def forward(self, token_embedding):
        return self.dropout(token_embedding +
                            self.pos_embedding[:token_embedding.size(0),:])

def generate_square_subsequent_mask(sz):
    mask = (torch.triu(torch.ones((sz, sz), device=device)) == 1).transpose(0, 1)
    mask = mask.float().masked_fill(mask == 0, float('-inf')).masked_fill(mask == 1, float(0.0))
    return mask

def create_mask(src, tgt):
  src_seq_len = src.shape[0]
  tgt_seq_len = tgt.shape[0]

  tgt_mask = generate_square_subsequent_mask(tgt_seq_len)
  src_mask = torch.zeros((src_seq_len, src_seq_len), device=device).type(torch.bool)

  src_padding_mask = (src == 0).transpose(0, 1)
  tgt_padding_mask = (tgt == 0).transpose(0, 1)
  return src_mask, tgt_mask, src_padding_mask, tgt_padding_mask

class TransformerModel(nn.Module):
    def __init__(self, num_encoder_layers, nhead, num_decoder_layers,
                 emb_size, src_vocab_size, tgt_vocab_size,
                 dim_feedforward:int = 512, dropout:float = 0.1):
        super(TransformerModel, self).__init__()
        encoder_layer = nn.TransformerEncoderLayer(d_model=emb_size, nhead=nhead,
                                                   dim_feedforward=dim_feedforward)
        self.transformer_encoder = nn.TransformerEncoder(encoder_layer, num_layers=num_encoder_layers)
        decoder_layer = nn.TransformerDecoderLayer(d_model=emb_size, nhead=nhead,
                                                   dim_feedforward=dim_feedforward)
        self.transformer_decoder = nn.TransformerDecoder(decoder_layer, num_layers=num_decoder_layers)

        self.generator = nn.Linear(emb_size, tgt_vocab_size)
        self.emb_size = emb_size
        self.src_tok_emb = self.embedding = nn.Embedding(src_vocab_size, emb_size)
        self.tgt_tok_emb = self.embedding = nn.Embedding(tgt_vocab_size, emb_size)
        self.positional_encoding = PositionalEncoding(emb_size, dropout=dropout)

    def forward(self, src, trg, src_mask,
                tgt_mask, src_padding_mask,
                tgt_padding_mask, memory_key_padding_mask):
        src_emb = self.positional_encoding(self.src_tok_emb(src)* math.sqrt(self.emb_size))
        tgt_emb = self.positional_encoding(self.tgt_tok_emb(trg)* math.sqrt(self.emb_size))
        memory = self.transformer_encoder(src_emb, src_mask, src_padding_mask)
        outs = self.transformer_decoder(tgt_emb, memory, tgt_mask, None,
                                        tgt_padding_mask, memory_key_padding_mask)
        return self.generator(outs)
```

```
#Instantiate model
model = TransformerModel(num_encoder_layers=6, nhead=8, num_decoder_layers=6,
                 emb_size=512, src_vocab_size=(len(d.get_alphabet()) + 1), tgt_vocab_size=(len(d.get_alphabet()) + 1),
                 dim_feedforward = 512, dropout = 0.2).to(device)
```

```
#Train and Test Epoch

def train_epoch_transformer(model, train_loader, optimizer, criterion, batch_size):
```

```
  model.train()
  total_loss = 0
  num_correct = 0
  total_items = 0
  for src, tgt in tqdm(train_loader):
      src = src.to(device).T
      tgt = tgt.to(device).T

      tgt_input = tgt[:-1, :]

      src_mask, tgt_mask, src_padding_mask, tgt_padding_mask = create_mask(src, tgt_input)

      logits = model(src, tgt_input, src_mask, tgt_mask,
                                src_padding_mask, tgt_padding_mask, src_padding_mask)

      optimizer.zero_grad()

      tgt_out = tgt[1:,:]
      loss = criterion(logits.reshape(-1, logits.shape[-1]), tgt_out.reshape(-1))

      loss.backward()
      optimizer.step()

      total_loss += loss.item()
      total_items += (tgt_out != 0).sum(dim=(0,1))

      num_correct += (torch.logical_and((logits.argmax(dim=2) == tgt_out), (tgt_out != 0))).sum(dim=(0,1))
  return total_loss / len(train_loader), num_correct / total_items


def test_epoch_transformer(model, test_loader, criterion, batch_size):
  model.eval()
  total_loss = 0
  num_correct = 0
  total_items = 0
  for src, tgt in tqdm(train_loader):
      src = src.to(device).T
      tgt = tgt.to(device).T

      tgt_input = tgt[:-1, :]

      src_mask, tgt_mask, src_padding_mask, tgt_padding_mask = create_mask(src, tgt_input)

      logits = model(src, tgt_input, src_mask, tgt_mask,
                                src_padding_mask, tgt_padding_mask, src_padding_mask)


      tgt_out = tgt[1:,:]
      loss = criterion(logits.reshape(-1, logits.shape[-1]), tgt_out.reshape(-1))


      total_loss += loss.item()
      total_items += (tgt_out != 0).sum(dim=(0,1))

      num_correct += (torch.logical_and((logits.argmax(dim=2) == tgt_out), (tgt_out != 0))).sum(dim=(0,1))
  return total_loss / len(train_loader), num_correct / total_items
```

```
#Defining Transformer Training Method

def train_transformer(model, train_dataset, test_dataset, batch_size=256, epochs=40):
  train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
  test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=batch_size, shuffle=True)
  criterion = nn.CrossEntropyLoss()

  optim = torch.optim.Adam(model.parameters(), lr=1e-4, betas=(0.9, 0.98), eps=1e-9)
  for e in range(epochs):
    train_loss, train_acc = train_epoch_transformer(model, train_loader, optim, criterion, batch_size=batch_size)
    test_loss, test_acc = train_epoch_transformer(model, test_loader, optim, criterion, batch_size=batch_size)
    print(f'Epoch: {e + 1} Training Loss: {train_loss} Training Accuracy: {train_acc} Test Loss: {test_loss} Test Accuracy: {t
```

```
#Running Training

train_transformer(model,train_dataset, test_dataset)
```

```
      0%|          | 0/40 [00:00<?, ?it/s]/usr/local/lib/python3.9/dist-packages/torch/nn/functional.py:4999: UserWarning: Su
      warnings.warn(
    /usr/local/lib/python3.9/dist-packages/torch/nn/functional.py:4999: UserWarning: Support for mismatched key_padding_mask
      warnings.warn(
    100%|██████████| 40/40 [00:18<00:00,  2.11it/s]
    100%|██████████| 5/5 [00:02<00:00,  2.37it/s]
    Epoch: 1 Training Loss: 2.272262266278267 Training Accuracy: 0.16307766735553741 Test Loss: 1.2473761320114136 Test Accur
    100%|██████████| 40/40 [00:18<00:00,  2.12it/s]
```

```
100%|██████████| 5/5 [00:02<00:00,  2.37it/s]
Epoch: 2 Training Loss: 1.017479281127453 Training Accuracy: 0.5379163026809692 Test Loss: 0.8975930333137512 Test Accura
100%|██████████| 40/40 [00:18<00:00,  2.12it/s]
100%|██████████| 5/5 [00:02<00:00,  2.36it/s]
Epoch: 3 Training Loss: 0.798090772330761 Training Accuracy: 0.6178738474845886 Test Loss: 0.7412149906158447 Test Accura
100%|██████████| 40/40 [00:18<00:00,  2.12it/s]
100%|██████████| 5/5 [00:02<00:00,  2.37it/s]
Epoch: 4 Training Loss: 0.6800898924469948 Training Accuracy: 0.6606778502464294 Test Loss: 0.6589356303215027 Test Accur
100%|██████████| 40/40 [00:18<00:00,  2.12it/s]
100%|██████████| 5/5 [00:02<00:00,  2.37it/s]
Epoch: 5 Training Loss: 0.6122173696756363 Training Accuracy: 0.6842361688613892 Test Loss: 0.598982059955597 Test Accura
100%|██████████| 40/40 [00:18<00:00,  2.12it/s]
100%|██████████| 5/5 [00:02<00:00,  2.36it/s]
Epoch: 6 Training Loss: 0.5576515465974807 Training Accuracy: 0.709431529045105 Test Loss: 0.5379390239715576 Test Accura
100%|██████████| 40/40 [00:18<00:00,  2.11it/s]
100%|██████████| 5/5 [00:02<00:00,  2.36it/s]
Epoch: 7 Training Loss: 0.5138253048062325 Training Accuracy: 0.7298389673233032 Test Loss: 0.5064972639083862 Test Accur
100%|██████████| 40/40 [00:18<00:00,  2.11it/s]
100%|██████████| 5/5 [00:02<00:00,  2.37it/s]
Epoch: 8 Training Loss: 0.4776188492774963 Training Accuracy: 0.7459455132484436 Test Loss: 0.4628971040248871 Test Accur
100%|██████████| 40/40 [00:18<00:00,  2.11it/s]
100%|██████████| 5/5 [00:02<00:00,  2.37it/s]
Epoch: 9 Training Loss: 0.4452914834022522 Training Accuracy: 0.7596287131309509 Test Loss: 0.4387588858604431 Test Accur
100%|██████████| 40/40 [00:18<00:00,  2.12it/s]
100%|██████████| 5/5 [00:02<00:00,  2.36it/s]
Epoch: 10 Training Loss: 0.41774471700191496 Training Accuracy: 0.7725139260292053 Test Loss: 0.4098490536212921 Test Acc
100%|██████████| 40/40 [00:18<00:00,  2.11it/s]
100%|██████████| 5/5 [00:02<00:00,  2.37it/s]
Epoch: 11 Training Loss: 0.3952220693230629 Training Accuracy: 0.784542441368103 Test Loss: 0.3943741977214813 Test Accur
100%|██████████| 40/40 [00:18<00:00,  2.11it/s]
100%|██████████| 5/5 [00:02<00:00,  2.37it/s]
Epoch: 12 Training Loss: 0.37633374631404876 Training Accuracy: 0.7930680513381958 Test Loss: 0.3759742558002472 Test Acc
100%|██████████| 40/40 [00:18<00:00,  2.11it/s]
100%|██████████| 5/5 [00:02<00:00,  2.36it/s]
Epoch: 13 Training Loss: 0.35654349997639656 Training Accuracy: 0.802051305770874 Test Loss: 0.35684517621994016 Test Acc
100%|██████████| 40/40 [00:18<00:00,  2.11it/s]
100%|██████████| 5/5 [00:02<00:00,  2.36it/s]
Epoch: 14 Training Loss: 0.3425788074731827 Training Accuracy: 0.8077487349510193 Test Loss: 0.3371453285217285 Test Accu
100%|██████████| 40/40 [00:18<00:00,  2.11it/s]
100%|██████████| 5/5 [00:02<00:00,  2.37it/s]
Epoch: 15 Training Loss: 0.32931220903992653 Training Accuracy: 0.8143615126609802 Test Loss: 0.3308455407619476 Test Acc
100%|██████████| 40/40 [00:18<00:00,  2.11it/s]
100%|██████████| 5/5 [00:02<00:00,  2.36it/s]
Epoch: 16 Training Loss: 0.3153288722038269 Training Accuracy: 0.8202349543571472 Test Loss: 0.3188141226768494 Test Accu
100%|██████████| 40/40 [00:18<00:00,  2.12it/s]
100%|██████████| 5/5 [00:02<00:00,  2.37it/s]
Epoch: 17 Training Loss: 0.30402581095695497 Training Accuracy: 0.8264780640602112 Test Loss: 0.30275996327400206 Test Ac
100%|██████████| 40/40 [00:18<00:00,  2.12it/s]
100%|██████████| 5/5 [00:02<00:00,  2.37it/s]
Epoch: 18 Training Loss: 0.29240358211100104 Training Accuracy: 0.8321520090103149 Test Loss: 0.2873951256275177 Test Acc
```