Open Lab 5

Transformers

CSCI 7850 - Deep Learning

Due: Nov. 9 @ 11:00pm

Assignment

Here are the details of what you need to do for this assignment:

- Create one python script (`translation-transformer.py`) that solves the ENG-POR problem using an Encoder-Decoder architecture. You will need to construct your network with the following properties:
  - Your code should utilize 10,000 sentences
  - Use length 100 random embeddings for your encodings (`torch.nn.Embedding()`)
  - You should utilize transformer blocks in your model (no recurrent layers)
  - Try to adjust the latent dimension size so that the number of trainable parameters is no higher than the LSTM model (used in OL4)
  - Utilize the 80/20 validation split rule to train your model for 200 epochs
  - Your script should print the validation accuracy at the end **without teacher forcing**
- Use your scripts to run your models - **perform 10 independent runs for each model**.
- Compile your data into a text file (`translation-results.txt`) that can be read in using `np.loadtxt` .
- Create one python script (`parity-transformer.py`) that solves the parity problem using an Encoder-Decoder architecture. You will need to construct your network with the following properties:
  - Your code should generate 1000 random bit strings (and their corresponding parity strings for targets) varying in length from 10 to 30 bits for training/validation data
  - Use length 20 random embeddings for your encodings (`torch.nn.Embedding()`)
  - You should utilize transformer blocks in your model (no recurrent layers)
  - Try to adjust the latent dimension size so that the number of trainable parameters is no higher than the LSTM model (used in OL4)
  - Utilize the 80/20 validation split rule to train your model for 1000 epochs
  - Your script should print the validation accuracy at the end **without teacher forcing**
- Use your scripts to run your models - **perform 10 independent runs for each model**.
- Compile your data into a text file (`parity-results.txt`) that can be read in using `np.loadtxt` .
- Create an iPython Notebook file named `OL5.ipynb` which reads in the compiled results to produce a boxplot comparing the performance of the problems/architectures from OL4 (SimpleRNN and LSTM) with the Transformer.

Submission

Create a zip archive which contains the following contents:

- translation-transformer.py
- translation-results.txt
- parity-transformer.py
- parity-results.txt
- OL5.ipynb

Upload your zip archive to the course assignment system by the deadline at the top of this document.

Data (Re)preparation

First, I will illustrate creating simple RNNs for this task - this is similar to what was explored in class. We will use random embeddings...

In [55]:
```
import numpy as np
import torch
import lightning.pytorch as pl
import torchmetrics
import torchvision
from torchinfo import summary
from torchview import draw_graph
from IPython.display import display
import sympy as sp
sp.init_printing(use_latex=True)
import pandas as pd
import matplotlib.pyplot as plt
```

In [56]:
```
if torch.cuda.is_available():
    print(torch.cuda.get_device_name())
    print(torch.cuda.get_device_properties("cuda"))
    print("Number of devices:",torch.cuda.device_count())
    device = ("cuda")
else:
    print("Only CPU is available...")
    device = ("cpu")
```

```
NVIDIA GeForce RTX 2080 Ti
_CudaDeviceProperties(name='NVIDIA GeForce RTX 2080 Ti', major=7, minor=5, total_memory=11011MB, multi_processor_count=6
8)
Number of devices: 1
```

ENG-POR data set

If cut off below...

https://raw.githubusercontent.com/luisroque/deep-learning-articles/main/data/eng-por.txt

In [57]:
```
url = "https://raw.githubusercontent.com/luisroque/deep-learning-articles/main/data/eng-por.txt"
```

In [58]:
```
import urllib
data = []
with urllib.request.urlopen(url) as raw_data:
    for line in raw_data:
        data.append(line.decode("utf-8").split('\t')[0:2])
data = np.array(data)
```

In [59]:
```
# Subset? - All of the data will take some time...
n_seq = data.shape[0]
n_seq = 10000 # Modify here as-needed (comment out for all!)
data = data[0:n_seq]
split_point = int(data.shape[0] * 0.8) # Keep 80/20 split
np.random.shuffle(data) # In-place modification
max_length = np.max([len(i) for i in data.flatten()]) + 2 # Add start/stop
max_length
```

Out[59]: 45

In [60]:
```
data[0]
```

Out[60]: array(["How's it going?", 'E aí?'], dtype='<U184')

In [61]:
```
i_to_c_eng = ['','<START>','<STOP>'] + list({char for word in data[:,0] for char in word})
c_to_i_eng = {i_to_c_eng[i]:i for i in range(len(i_to_c_eng))}
i_to_c_eng[1] = i_to_c_eng[2] = ''
```

In [62]:
```
i_to_c_por = ['','<START>','<STOP>'] + list({char for word in data[:,1] for char in word})
c_to_i_por = {i_to_c_por[i]:i for i in range(len(i_to_c_por))}
i_to_c_por[1] = i_to_c_por[2] = ''
```

In [63]:
```
def encode_seq(x,mapping,max_length=0):
    # String to integer
    return [mapping['<START>']] + \
           [mapping[i] for i in list(x)] + \
           [mapping['<STOP>']] + \
           [0]*(max_length-len(list(x))-2)

def decode_seq(x,mapping):
```

```
    # Integer-to-string
    try:
        idx = list(x).index(2) # Stop token?
    except:
        idx = len(list(x)) # No stop token found
    return ''.join([mapping[i] for i in list(x)[0:idx]])
```

In [64]:
```
data[0]
```

Out[64]:
```
array(["How's it going?", 'E aí?'], dtype='<U184')
```

In [65]:
```
data[0,0]
```

Out[65]:
```
"How's it going?"
```

In [66]:
```
temp = encode_seq(data[0,0],c_to_i_eng,max_length)
print(*temp)
```

```
1 71 48 13 45 41 70 29 39 70 43 48 29 38 43 49 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

In [67]:
```
decode_seq(temp,i_to_c_eng)
```

Out[67]:
```
"How's it going?"
```

In [68]:
```
data[0,1]
```

Out[68]:
```
'E aí?'
```

In [69]:
```
temp = encode_seq(data[0,1],c_to_i_por,max_length)
print(*temp)
```

```
1 58 65 34 86 36 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

In [70]:
```
decode_seq(temp,i_to_c_por)
```

Out[70]:
```
'E aí?'
```

In [71]:
```
X = np.vstack([encode_seq(x,c_to_i_eng,max_length) for x in data[:,0]])
Y = np.vstack([encode_seq(x,c_to_i_por,max_length) for x in data[:,1]])
```

In [72]:
```
enc_x_train = X[:split_point]
enc_x_val = X[split_point:]
enc_x_train
```

Out[72]:
```
array([[ 1, 71, 48, ...,  0,  0,  0],
       [ 1, 54, 64, ...,  0,  0,  0],
       [ 1,  4, 48, ...,  0,  0,  0],
       ...,
       [ 1, 71, 48, ...,  0,  0,  0],
       [ 1, 31, 29, ...,  0,  0,  0],
       [ 1, 32, 64, ...,  0,  0,  0]])
```

In [73]:
```
dec_x_train = Y[:,0:-1][:split_point]
dec_x_val = Y[:,0:-1][split_point:]
dec_x_train
```

Out[73]:
```
array([[ 1, 58, 65, ...,  0,  0,  0],
       [ 1, 84, 20, ...,  0,  0,  0],
       [ 1,  9, 20, ...,  0,  0,  0],
       ...,
       [ 1, 84, 20, ...,  0,  0,  0],
       [ 1, 76, 81, ...,  0,  0,  0],
       [ 1, 58, 66, ...,  0,  0,  0]])
```

In [74]:
```
dec_y_train = Y[:,1:][:split_point]
dec_y_val = Y[:,1:][split_point:]
dec_y_train
```

Out[74]:
```
array([[58, 65, 34, ...,  0,  0,  0],
       [84, 20, 12, ...,  0,  0,  0],
       [ 9, 20, 65, ...,  0,  0,  0],
       ...,
       [84, 20, 34, ...,  0,  0,  0],
       [76, 81, 35, ...,  0,  0,  0],
       [58, 66, 12, ...,  0,  0,  0]])
```

In [75]:
```
print(enc_x_train.shape)
print(dec_x_train.shape)
print(dec_y_train.shape)
```

```
(8000, 45)
(8000, 44)
(8000, 44)
```

In [76]:
```
print(enc_x_val.shape)
print(dec_x_val.shape)
print(dec_y_val.shape)
```

```
(2000, 45)
(2000, 44)
(2000, 44)
```

In [77]:
```
len(i_to_c_eng)
```

Out[77]: 73

In [78]:
```
len(i_to_c_por)
```

Out[78]: 87

In [79]:
```
enc_x_train.shape[1:]
```

Out[79]: (45,)

Transformer Encoder-Decoder

We have reviewed some of the core components of transformer models in class, but here are the main components used to build ENG-POR translation model using a Transformer instead of a recurrent network:

Encoder Component

In [80]:
```
class TransformerBlock(torch.nn.Module):
    def __init__(self,
                 latent_size = 64,
                 num_heads = 4,
                 dropout = 0.1,
                 **kwargs):
        super().__init__(**kwargs)
        self.layer_norm1 = torch.nn.LayerNorm(latent_size)
        self.layer_norm2 = torch.nn.LayerNorm(latent_size)
        self.dropout = torch.nn.Dropout(dropout)
        self.activation = torch.nn.GELU()
        self.linear = torch.nn.Linear(latent_size,
                                      latent_size)
        self.mha = torch.nn.MultiheadAttention(latent_size,
                                               num_heads,
                                               dropout=dropout,
                                               batch_first=True)
    def forward(self, x):
        y = x
        y = self.layer_norm1(y)
        y = self.mha(y,y,y)[0]
        x = y = x + y
        y = self.layer_norm2(y)
        y = self.linear(y)
        y = self.activation(y)
        return x + y
```

In [81]:
```
class EncoderNetwork(torch.nn.Module):
    def __init__(self,
                 num_tokens,
                 max_length,
                 latent_size = 64,
                 num_heads = 4,
                 n_layers = 8,
                 **kwargs):
        super().__init__(**kwargs)
        self.token_embedding = torch.nn.Embedding(num_tokens,
                                                  latent_size,
```

```
                                                padding_idx=0)
        self.position_embedding = torch.nn.Embedding(max_length+1,
                                                     latent_size,
                                                     padding_idx=0)
        self.transformer_blocks = torch.nn.Sequential(*[
            TransformerBlock(latent_size=latent_size,
                             num_heads=num_heads) for _ in range(n_layers)
        ])

    def forward(self, x):
        y = x
        y = self.token_embedding(y) + \
            self.position_embedding(torch.arange(1,y.shape[1]+1).long().to(device) * torch.sign(y).long().to(device))
        y = self.transformer_blocks(y)
        return y
```

In [82]:
```
enc_x_train[0:5].shape
```

Out[82]: $(5, 45)$

In [83]:
```
enc_net = EncoderNetwork(num_tokens=len(i_to_c_eng),
                         max_length=enc_x_train.shape[-1])
summary(enc_net,input_size=enc_x_train[0:5].shape,dtypes=[torch.long])
```

Out[83]:
```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
EncoderNetwork                           [5, 45, 64]               --
├─Embedding: 1-1                         [5, 45, 64]               4,672
├─Embedding: 1-2                         [5, 45, 64]               2,944
├─Sequential: 1-3                        [5, 45, 64]               --
│    └─TransformerBlock: 2-1             [5, 45, 64]               --
│    │    └─LayerNorm: 3-1               [5, 45, 64]               128
│    │    └─MultiheadAttention: 3-2      [5, 45, 64]               16,640
│    │    └─LayerNorm: 3-3               [5, 45, 64]               128
│    │    └─Linear: 3-4                  [5, 45, 64]               4,160
│    │    └─GELU: 3-5                    [5, 45, 64]               --
│    └─TransformerBlock: 2-2             [5, 45, 64]               --
│    │    └─LayerNorm: 3-6               [5, 45, 64]               128
│    │    └─MultiheadAttention: 3-7      [5, 45, 64]               16,640
│    │    └─LayerNorm: 3-8               [5, 45, 64]               128
│    │    └─Linear: 3-9                  [5, 45, 64]               4,160
│    │    └─GELU: 3-10                   [5, 45, 64]               --
│    └─TransformerBlock: 2-3             [5, 45, 64]               --
│    │    └─LayerNorm: 3-11              [5, 45, 64]               128
│    │    └─MultiheadAttention: 3-12     [5, 45, 64]               16,640
│    │    └─LayerNorm: 3-13              [5, 45, 64]               128
│    │    └─Linear: 3-14                 [5, 45, 64]               4,160
│    │    └─GELU: 3-15                   [5, 45, 64]               --
│    └─TransformerBlock: 2-4             [5, 45, 64]               --
│    │    └─LayerNorm: 3-16              [5, 45, 64]               128
│    │    └─MultiheadAttention: 3-17     [5, 45, 64]               16,640
│    │    └─LayerNorm: 3-18              [5, 45, 64]               128
│    │    └─Linear: 3-19                 [5, 45, 64]               4,160
│    │    └─GELU: 3-20                   [5, 45, 64]               --
│    └─TransformerBlock: 2-5             [5, 45, 64]               --
│    │    └─LayerNorm: 3-21              [5, 45, 64]               128
│    │    └─MultiheadAttention: 3-22     [5, 45, 64]               16,640
│    │    └─LayerNorm: 3-23              [5, 45, 64]               128
│    │    └─Linear: 3-24                 [5, 45, 64]               4,160
│    │    └─GELU: 3-25                   [5, 45, 64]               --
│    └─TransformerBlock: 2-6             [5, 45, 64]               --
│    │    └─LayerNorm: 3-26              [5, 45, 64]               128
│    │    └─MultiheadAttention: 3-27     [5, 45, 64]               16,640
│    │    └─LayerNorm: 3-28              [5, 45, 64]               128
│    │    └─Linear: 3-29                 [5, 45, 64]               4,160
│    │    └─GELU: 3-30                   [5, 45, 64]               --
│    └─TransformerBlock: 2-7             [5, 45, 64]               --
│    │    └─LayerNorm: 3-31              [5, 45, 64]               128
│    │    └─MultiheadAttention: 3-32     [5, 45, 64]               16,640
│    │    └─LayerNorm: 3-33              [5, 45, 64]               128
│    │    └─Linear: 3-34                 [5, 45, 64]               4,160
│    │    └─GELU: 3-35                   [5, 45, 64]               --
│    └─TransformerBlock: 2-8             [5, 45, 64]               --
│    │    └─LayerNorm: 3-36              [5, 45, 64]               128
│    │    └─MultiheadAttention: 3-37     [5, 45, 64]               16,640
│    │    └─LayerNorm: 3-38              [5, 45, 64]               128
│    │    └─Linear: 3-39                 [5, 45, 64]               4,160
│    │    └─GELU: 3-40                   [5, 45, 64]               --
==========================================================================================
Total params: 176,064
Trainable params: 176,064
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 0.21
==========================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 3.00
Params size (MB): 0.17
Estimated Total Size (MB): 3.17
==========================================================================================
```

In [ ]:
```
model_graph = draw_graph(enc_net, input_size=enc_x_train[0:5].shape, device=device,
                         hide_inner_tensors=True,hide_module_functions=True,
                         expand_nested=False, depth=3, dtypes=[torch.long])
model_graph.visual_graph
```

Decoder Component

In [85]:
```
class MaskedTransformerBlock(torch.nn.Module):
    def __init__(self,
```

```
                latent_size = 64,
                num_heads = 4,
                dropout = 0.1,
                **kwargs):
        super().__init__(**kwargs)
        self.layer_norm1 = torch.nn.LayerNorm(latent_size)
        self.layer_norm2 = torch.nn.LayerNorm(latent_size)
        self.layer_norm3 = torch.nn.LayerNorm(latent_size)
        self.dropout = torch.nn.Dropout(dropout)
        self.activation = torch.nn.GELU()
        self.linear = torch.nn.Linear(latent_size,
                                      latent_size)
        self.mha1 = torch.nn.MultiheadAttention(latent_size,
                                                num_heads,
                                                dropout=dropout,
                                                batch_first=True)
        self.mha2 = torch.nn.MultiheadAttention(latent_size,
                                                num_heads,
                                                dropout=dropout,
                                                batch_first=True)
    def make_causal_mask(self, sz: int):
        return torch.triu(torch.full((sz, sz), True), diagonal=1).to(device)
        # return torch.triu(torch.full((sz, sz), float('-inf')), diagonal=1)

    def forward(self, x):
        x_enc, x_dec = x
        x = y = x_dec
        y = self.layer_norm1(y)
        y = self.mha1(y,y,y,
                      attn_mask=self.make_causal_mask(y.shape[1]))[0]
        x = y = x + y
        y = self.layer_norm2(y)
        y = self.mha2(y,x_enc,x_enc)[0]
        x = y = x + y
        y = self.layer_norm3(y)
        y = self.linear(y)
        y = self.activation(y)
        return x_enc, x + y
```

In [86]:
```
class DecoderNetwork(torch.nn.Module):
    def __init__(self,
                num_tokens,
                max_length,
                latent_size = 64,
                num_heads = 4,
                n_layers = 8,
                **kwargs):
        super().__init__(**kwargs)
        self.token_embedding = torch.nn.Embedding(num_tokens,
                                                  latent_size,
                                                  padding_idx=0)
        self.position_embedding = torch.nn.Embedding(max_length+1,
                                                     latent_size,
                                                     padding_idx=0)
        self.transformer_blocks = torch.nn.Sequential(*[
            MaskedTransformerBlock(latent_size=latent_size,
                                   num_heads=num_heads) for _ in range(n_layers)
        ])
        self.output_layer = torch.nn.Linear(latent_size,
                                            num_tokens)

    def forward(self, x_enc, x_dec):
        y = x_dec
        y = self.token_embedding(y) + \
            self.position_embedding(torch.arange(1,y.shape[1]+1).long().to(device) * torch.sign(y).long().to(device))
        y = self.transformer_blocks((x_enc,y))[1]
        y = self.output_layer(y)
        return y
```

In [87]:
```
enc_x_train[0:5].shape
```

Out[87]: $(5, 45)$

In [88]:
```
dec_x_train[0:5].shape
```

Out[88]: $(5, 44)$

In [89]:
```
enc_net(torch.Tensor(enc_x_train[0:5]).long().to(device)).cpu().shape
```

Out[89]: torch.Size([5, 45, 64])

In [90]:
```
enc_x_train[0:5].shape[0:2]+(64,)
```

Out[90]: $(5, 45, 64)$

In [91]:
```
dec_net = DecoderNetwork(num_tokens=len(i_to_c_por),
                         max_length=dec_x_train.shape[-1])
summary(dec_net,input_size=[enc_x_train[0:5].shape[0:2]+(64,),
                            dec_x_train[0:5].shape], dtypes=[torch.float32, torch.long])
```

```
/opt/conda/lib/python3.11/site-packages/torch/nn/modules/activation.py:1160: UserWarning: Converting mask without torch.
bool dtype to bool; this will negatively affect performance. Prefer to use a boolean mask directly. (Triggered internall
y at ../aten/src/ATen/native/transformers/attention.cpp:150.)
  return torch._native_multi_head_attention(
```

Out[91]:

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
DecoderNetwork                           [5, 44, 87]               --
├─Embedding: 1-1                         [5, 44, 64]               5,568
├─Embedding: 1-2                         [5, 44, 64]               2,880
├─Sequential: 1-3                        [5, 45, 64]               --
│    └─MaskedTransformerBlock: 2-1       [5, 45, 64]               --
│    │    └─LayerNorm: 3-1               [5, 44, 64]               128
│    │    └─MultiheadAttention: 3-2      [5, 44, 64]               16,640
│    │    └─LayerNorm: 3-3               [5, 44, 64]               128
│    │    └─MultiheadAttention: 3-4      [5, 44, 64]               16,640
│    │    └─LayerNorm: 3-5               [5, 44, 64]               128
│    │    └─Linear: 3-6                  [5, 44, 64]               4,160
│    │    └─GELU: 3-7                    [5, 44, 64]               --
│    └─MaskedTransformerBlock: 2-2       [5, 45, 64]               --
│    │    └─LayerNorm: 3-8               [5, 44, 64]               128
│    │    └─MultiheadAttention: 3-9      [5, 44, 64]               16,640
│    │    └─LayerNorm: 3-10              [5, 44, 64]               128
│    │    └─MultiheadAttention: 3-11     [5, 44, 64]               16,640
│    │    └─LayerNorm: 3-12              [5, 44, 64]               128
│    │    └─Linear: 3-13                 [5, 44, 64]               4,160
│    │    └─GELU: 3-14                   [5, 44, 64]               --
│    └─MaskedTransformerBlock: 2-3       [5, 45, 64]               --
│    │    └─LayerNorm: 3-15              [5, 44, 64]               128
│    │    └─MultiheadAttention: 3-16     [5, 44, 64]               16,640
│    │    └─LayerNorm: 3-17              [5, 44, 64]               128
│    │    └─MultiheadAttention: 3-18     [5, 44, 64]               16,640
│    │    └─LayerNorm: 3-19              [5, 44, 64]               128
│    │    └─Linear: 3-20                 [5, 44, 64]               4,160
│    │    └─GELU: 3-21                   [5, 44, 64]               --
│    └─MaskedTransformerBlock: 2-4       [5, 45, 64]               --
│    │    └─LayerNorm: 3-22              [5, 44, 64]               128
│    │    └─MultiheadAttention: 3-23     [5, 44, 64]               16,640
│    │    └─LayerNorm: 3-24              [5, 44, 64]               128
│    │    └─MultiheadAttention: 3-25     [5, 44, 64]               16,640
│    │    └─LayerNorm: 3-26              [5, 44, 64]               128
│    │    └─Linear: 3-27                 [5, 44, 64]               4,160
│    │    └─GELU: 3-28                   [5, 44, 64]               --
│    └─MaskedTransformerBlock: 2-5       [5, 45, 64]               --
│    │    └─LayerNorm: 3-29              [5, 44, 64]               128
│    │    └─MultiheadAttention: 3-30     [5, 44, 64]               16,640
│    │    └─LayerNorm: 3-31              [5, 44, 64]               128
│    │    └─MultiheadAttention: 3-32     [5, 44, 64]               16,640
│    │    └─LayerNorm: 3-33              [5, 44, 64]               128
│    │    └─Linear: 3-34                 [5, 44, 64]               4,160
│    │    └─GELU: 3-35                   [5, 44, 64]               --
│    └─MaskedTransformerBlock: 2-6       [5, 45, 64]               --
│    │    └─LayerNorm: 3-36              [5, 44, 64]               128
│    │    └─MultiheadAttention: 3-37     [5, 44, 64]               16,640
│    │    └─LayerNorm: 3-38              [5, 44, 64]               128
│    │    └─MultiheadAttention: 3-39     [5, 44, 64]               16,640
│    │    └─LayerNorm: 3-40              [5, 44, 64]               128
│    │    └─Linear: 3-41                 [5, 44, 64]               4,160
│    │    └─GELU: 3-42                   [5, 44, 64]               --
│    └─MaskedTransformerBlock: 2-7       [5, 45, 64]               --
│    │    └─LayerNorm: 3-43              [5, 44, 64]               128
│    │    └─MultiheadAttention: 3-44     [5, 44, 64]               16,640
│    │    └─LayerNorm: 3-45              [5, 44, 64]               128
│    │    └─MultiheadAttention: 3-46     [5, 44, 64]               16,640
│    │    └─LayerNorm: 3-47              [5, 44, 64]               128
│    │    └─Linear: 3-48                 [5, 44, 64]               4,160
│    │    └─GELU: 3-49                   [5, 44, 64]               --
│    └─MaskedTransformerBlock: 2-8       [5, 45, 64]               --
│    │    └─LayerNorm: 3-50              [5, 44, 64]               128
│    │    └─MultiheadAttention: 3-51     [5, 44, 64]               16,640
│    │    └─LayerNorm: 3-52              [5, 44, 64]               128
│    │    └─MultiheadAttention: 3-53     [5, 44, 64]               16,640
│    │    └─LayerNorm: 3-54              [5, 44, 64]               128
│    │    └─Linear: 3-55                 [5, 44, 64]               4,160
│    │    └─GELU: 3-56                   [5, 44, 64]               --
├─Linear: 1-4                            [5, 44, 87]               5,655
==========================================================================================
Total params: 316,695
Trainable params: 316,695
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 0.25
```

```
==========================================================================================
Input size (MB): 0.06
Forward/backward pass size (MB): 3.98
Params size (MB): 0.20
Estimated Total Size (MB): 4.24
==========================================================================================
```

In [ ]:
```
model_graph = draw_graph(dec_net, input_size=[enc_x_train[0:5].shape[0:2]+(64,),
                                              dec_x_train[0:5].shape], device=device,
                         hide_inner_tensors=True,hide_module_functions=True,
                         expand_nested=False, depth=3, dtypes=[torch.float32, torch.long])
model_graph.visual_graph
```

Training Hooks

In [93]:
```
class EncDecLightningModule(pl.LightningModule):
    def __init__(self,
                 output_size,
                 **kwargs):
        super().__init__(**kwargs)
        self.mc_acc = torchmetrics.classification.Accuracy(task='multiclass',
                                                            num_classes=output_size,
                                                            ignore_index=0)
        self.cce_loss = torch.nn.CrossEntropyLoss(ignore_index=0)

    def predict(self, x):
        return torch.softmax(self(x),-1)

    def configure_optimizers(self):
        optimizer = torch.optim.Adam(self.parameters(), lr=0.001)
        return optimizer

    def training_step(self, train_batch, batch_idx):
        x_enc, x_dec, y_dec = train_batch
        y_pred = self(x_enc, x_dec)
        perm = (0,-1) + tuple(range(y_pred.ndim))[1:-1]
        acc = self.mc_acc(y_pred.permute(*perm),y_dec)
        loss = self.cce_loss(y_pred.permute(*perm),y_dec)
        self.log('train_acc', acc, on_step=False, on_epoch=True)
        self.log('train_loss', loss, on_step=False, on_epoch=True)
        return loss

    # Validate used for Teacher Forcing
    def validation_step(self, val_batch, batch_idx):
        x_enc, x_dec, y_dec = val_batch
        y_pred = self(x_enc, x_dec)
        perm = (0,-1) + tuple(range(y_pred.ndim))[1:-1]
        acc = self.mc_acc(y_pred.permute(*perm),y_dec)
        loss = self.cce_loss(y_pred.permute(*perm),y_dec)
        self.log('val_acc', acc, on_step=False, on_epoch=True)
        self.log('val_loss', loss, on_step=False, on_epoch=True)
        return loss

    # Test used for Non-Teacher Forcing
    def test_step(self, test_batch, batch_idx):
        x_enc, x_dec, y_dec = test_batch
        context = self.enc_net(x_enc)
        tokens = torch.zeros_like(x_dec).long()
        tokens[:,0] = 1
        for i in range(y_dec.shape[1]-1):
            tokens[:,i+1] = self.dec_net(context, tokens).argmax(-1)[:,i]
        y_pred = self(x_enc, tokens)
        perm = (0,-1) + tuple(range(y_pred.ndim))[1:-1]
        acc = self.mc_acc(y_pred.permute(*perm),y_dec)
        loss = self.cce_loss(y_pred.permute(*perm),y_dec)
        self.log('test_acc', acc, on_step=False, on_epoch=True)
        self.log('test_loss', loss, on_step=False, on_epoch=True)
        return loss
```

Encoder-Decoder Network

In [94]:
```
class EncDecNetwork(EncDecLightningModule):
    def __init__(self,
                 num_enc_tokens,
                 max_enc_length,
```

```
                num_dec_tokens,
                max_dec_length,
                latent_size = 64,
                num_heads = 4,
                n_layers = 8,
                **kwargs):
        super().__init__(output_size=num_dec_tokens,
                         **kwargs)
        self.enc_net = EncoderNetwork(num_enc_tokens,max_enc_length,latent_size,num_heads,n_layers)
        self.dec_net = DecoderNetwork(num_dec_tokens,max_dec_length,latent_size,num_heads,n_layers)

    def forward(self, x_enc, x_dec):
        return self.dec_net(self.enc_net(x_enc), x_dec)
```

In [95]:
```
enc_dec_net = EncDecNetwork(num_enc_tokens=len(i_to_c_eng),
                            max_enc_length=enc_x_train.shape[-1],
                            num_dec_tokens=len(i_to_c_por),
                            max_dec_length=dec_x_train.shape[-1],
                            latent_size=256,
                            n_layers=4)
summary(enc_dec_net,input_size=[enc_x_train[0:1].shape,
                                dec_x_train[0:1].shape],
        dtypes=[torch.long, torch.long])
```

```
/opt/conda/lib/python3.11/site-packages/torch/nn/modules/activation.py:1160: UserWarning: Converting mask without torch.
bool dtype to bool; this will negatively affect performance. Prefer to use a boolean mask directly. (Triggered internall
y at ../aten/src/ATen/native/transformers/attention.cpp:150.)
  return torch._native_multi_head_attention(
```

Out[95]:
```
==========================================================================================
Layer (type:depth-idx)                        Output Shape              Param #
==========================================================================================
EncDecNetwork                                 [1, 44, 87]               --
├─EncoderNetwork: 1-1                         [1, 45, 256]              --
│    └─Embedding: 2-1                         [1, 45, 256]              18,688
│    └─Embedding: 2-2                         [1, 45, 256]              11,776
│    └─Sequential: 2-3                        [1, 45, 256]              --
│    │    └─TransformerBlock: 3-1             [1, 45, 256]              329,984
│    │    └─TransformerBlock: 3-2             [1, 45, 256]              329,984
│    │    └─TransformerBlock: 3-3             [1, 45, 256]              329,984
│    │    └─TransformerBlock: 3-4             [1, 45, 256]              329,984
├─DecoderNetwork: 1-2                         [1, 44, 87]               --
│    └─Embedding: 2-4                         [1, 44, 256]              22,272
│    └─Embedding: 2-5                         [1, 44, 256]              11,520
│    └─Sequential: 2-6                        [1, 45, 256]              --
│    │    └─MaskedTransformerBlock: 3-5       [1, 45, 256]              593,664
│    │    └─MaskedTransformerBlock: 3-6       [1, 45, 256]              593,664
│    │    └─MaskedTransformerBlock: 3-7       [1, 45, 256]              593,664
│    │    └─MaskedTransformerBlock: 3-8       [1, 45, 256]              593,664
│    └─Linear: 2-7                            [1, 44, 87]               22,359
==========================================================================================
Total params: 3,781,207
Trainable params: 3,781,207
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 0.62
==========================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 2.94
Params size (MB): 2.49
Estimated Total Size (MB): 5.44
==========================================================================================
```

In [ ]:
```
model_graph = draw_graph(enc_dec_net,
                         input_size=[enc_x_train[0:1].shape,
                                     dec_x_train[0:1].shape],
                         device=device,
                         hide_inner_tensors=True,hide_module_functions=True,
                         expand_nested=False, depth=4, dtypes=[torch.long, torch.long])
model_graph.visual_graph
```

In [97]:
```
dec_y_train.shape
```

Out[97]: $(8000, 44)$

In [98]:
```
enc_dec_net(torch.Tensor(enc_x_train[0:1]).long().to(device),
            torch.Tensor(dec_x_train[0:1]).long().to(device)).cpu()
```

Out[98]:
```
tensor([[[-1.3927, -0.4417,  0.8434,  ...,  0.8372, -0.6745,  0.4935],
         [-1.8138, -0.8162, -0.5340,  ..., -0.0609,  0.1085, -0.1081],
         [-0.9278,  0.4802,  0.1831,  ..., -0.3730, -0.1435,  0.6456],
         ...,
         [-0.9879, -0.5244, -0.5175,  ..., -0.4498,  0.9318,  0.9619],
         [-1.0089, -0.8443, -0.6268,  ..., -0.3052,  0.8023,  0.9801],
         [-1.0601, -0.7317, -0.7081,  ..., -0.3290,  0.7242,  0.8368]]],
       grad_fn=<ToCopyBackward0>)
```

Training Time

In [99]:
```
batch_size = 128
xy_train = torch.utils.data.DataLoader(list(zip(torch.Tensor(enc_x_train).long(),
                                                torch.Tensor(dec_x_train).long(),
                                                torch.Tensor(dec_y_train).long())),
                                       shuffle=True, batch_size=batch_size,
                                       num_workers=8)
xy_val = torch.utils.data.DataLoader(list(zip(torch.Tensor(enc_x_val).long(),
                                              torch.Tensor(dec_x_val).long(),
                                              torch.Tensor(dec_y_val).long())),
                                     shuffle=False, batch_size=batch_size,
                                     num_workers=8)
```

```
/opt/conda/lib/python3.11/site-packages/torch/utils/data/dataloader.py:560: UserWarning: This DataLoader will create 8 w
orker processes in total. Our suggested max number of worker in current system is 2, which is smaller than what this Dat
aLoader is going to create. Please be aware that excessive worker creation might get DataLoader running slow or even fre
eze, lower the worker number to avoid potential slowness/freeze if necessary.
  warnings.warn(_create_warning_msg(
```

In [100…
```
logger = pl.loggers.CSVLogger("lightning_logs",
                              name="Open_Lab_5",
                              version="demo-0")
```

In [101…
```
trainer = pl.Trainer(logger=logger,
                     max_epochs=300,
                     enable_progress_bar=True,
                     log_every_n_steps=0,
                     enable_checkpointing=False,
                     callbacks=[pl.callbacks.TQDMProgressBar(refresh_rate=50)])
```

```
GPU available: True (cuda), used: True
TPU available: False, using: 0 TPU cores
IPU available: False, using: 0 IPUs
HPU available: False, using: 0 HPUs
```

In [102…
```
trainer.validate(enc_dec_net, xy_val)
```

```
/opt/conda/lib/python3.11/site-packages/lightning/fabric/loggers/csv_logs.py:195: UserWarning: Experiment logs directory
lightning_logs/Open_Lab_5/demo-0 exists and is not empty. Previous log files in this directory will be deleted when the
new ones are saved!
  rank_zero_warn(
LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
SLURM auto-requeueing enabled. Setting signal handlers.
Validation: 0it [00:00, ?it/s]
```

```
/opt/conda/lib/python3.11/site-packages/torch/nn/modules/activation.py:1160: UserWarning: Converting mask without torch.
bool dtype to bool; this will negatively affect performance. Prefer to use a boolean mask directly. (Triggered internall
y at ../aten/src/ATen/native/transformers/attention.cpp:150.)
  return torch._native_multi_head_attention(
```

Runningstage.validating metric	DataLoader 0
val_acc	0.0023976529482752085
val_loss	5.240775108337402

Out[102…
```
[{'val_acc': 0.0023976529482752085, 'val_loss': 5.240775108337402}]
```

In [103…
```
trainer.test(enc_dec_net, xy_val)
```

```
LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
SLURM auto-requeueing enabled. Setting signal handlers.
Testing: 0it [00:00, ?it/s]
```

Runningstage.testing metric	DataLoader 0
test_acc	0.004485361743718386
test_loss	5.212940692901611

Out[103…

```
[{'test_acc': 0.004485361743718386, 'test_loss': 5.212940692901611}]
```

In [104…

```
trainer.fit(enc_dec_net, xy_train, xy_val)
```

```
LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]

  | Name     | Type               | Params
------------------------------------------------
0 | mc_acc   | MulticlassAccuracy | 0
1 | cce_loss | CrossEntropyLoss   | 0
2 | enc_net  | EncoderNetwork     | 1.4 M
3 | dec_net  | DecoderNetwork     | 2.4 M
------------------------------------------------
3.8 M     Trainable params
0         Non-trainable params
3.8 M     Total params
15.125    Total estimated model params size (MB)
SLURM auto-requeueing enabled. Setting signal handlers.
Sanity Checking: 0it [00:00, ?it/s]
Training: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
/opt/conda/lib/python3.11/site-packages/lightning/pytorch/trainer/call.py:53: UserWarning: Detected KeyboardInterrupt, a
ttempting graceful shutdown...
  rank_zero_warn("Detected KeyboardInterrupt, attempting graceful shutdown...")
```

In [105…

```
results = pd.read_csv(logger.log_dir+"/metrics.csv")
results
```

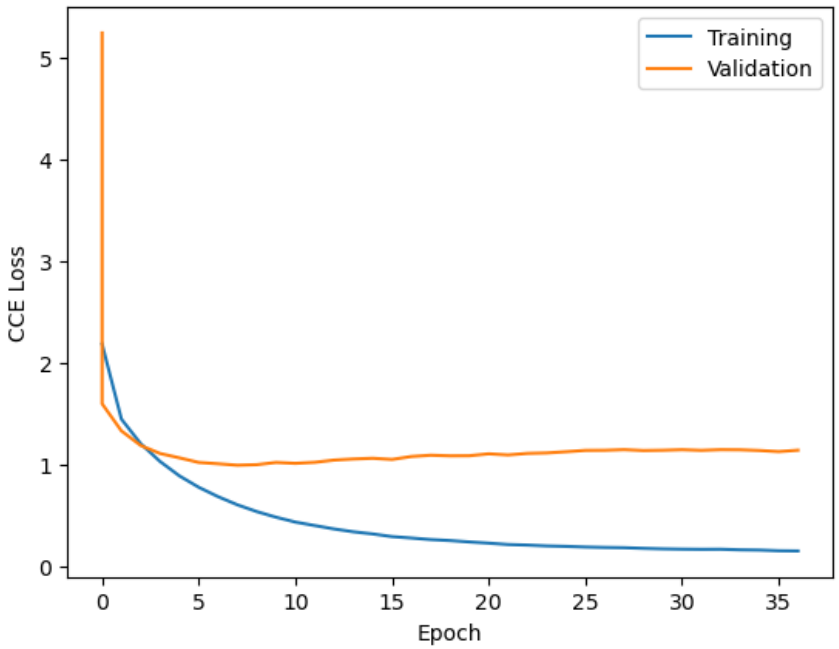
Out[105…

	val_acc	val_loss	epoch	step	test_acc	test_loss	train_acc	train_loss
0	0.002398	5.240775	0	0	NaN	NaN	NaN	NaN
1	NaN	NaN	0	0	0.004485	5.212941	NaN	NaN
2	0.528579	1.597087	0	62	NaN	NaN	NaN	NaN
3	NaN	NaN	0	62	NaN	NaN	0.401328	2.183989
4	0.599970	1.330301	1	125	NaN	NaN	NaN	NaN
…	…	…	…	…	…	…	…	…
71	NaN	NaN	34	2204	NaN	NaN	0.949558	0.159563
72	0.769081	1.126995	35	2267	NaN	NaN	NaN	NaN
73	NaN	NaN	35	2267	NaN	NaN	0.951770	0.152600
74	0.769768	1.140576	36	2330	NaN	NaN	NaN	NaN
75	NaN	NaN	36	2330	NaN	NaN	0.952653	0.150719

76 rows × 8 columns

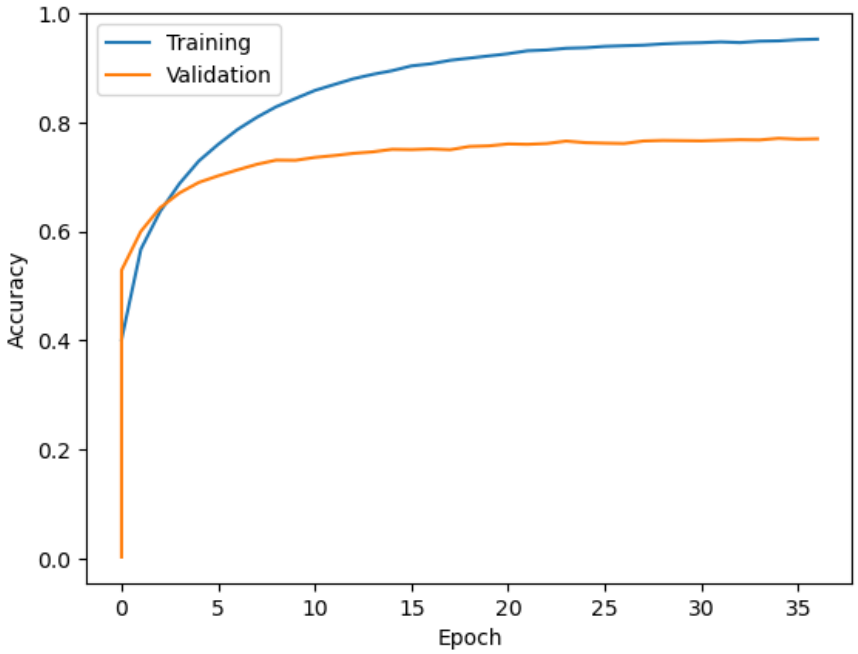
In [106…

```
plt.plot(results["epoch"][np.logical_not(np.isnan(results["train_loss"]))],
         results["train_loss"][np.logical_not(np.isnan(results["train_loss"]))],
         label="Training")
plt.plot(results["epoch"][np.logical_not(np.isnan(results["val_loss"]))],
         results["val_loss"][np.logical_not(np.isnan(results["val_loss"]))],
         label="Validation")
plt.legend()
plt.ylabel("CCE Loss")
plt.xlabel("Epoch")
plt.show()
```



In [107…

```
plt.plot(results["epoch"][np.logical_not(np.isnan(results["train_acc"]))],
         results["train_acc"][np.logical_not(np.isnan(results["train_acc"]))],
         label="Training")
plt.plot(results["epoch"][np.logical_not(np.isnan(results["val_acc"]))],
         results["val_acc"][np.logical_not(np.isnan(results["val_acc"]))],
         label="Validation")
plt.legend()
plt.ylabel("Accuracy")
plt.xlabel("Epoch")
plt.show()
```



Direct Validation of Results

Teacher Forcing

In [124…
```
# What should we see?
i = 0
print('Input:', enc_x_val[i])
print('Output:', dec_y_val[i])
```

```
Input: [ 1 59 70 72 29 62 40 70 39 18 42 29 38 41  5  2  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
Output: [74 64 81 30 81 65 13 30 12 79 33 69  2  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
```

In [125…
```
print('Input:', decode_seq(enc_x_val[i],i_to_c_eng))
print('Output:', decode_seq(dec_y_val[i],i_to_c_por))
```

```
Input: I like trains.
Output: Adoro trens.
```

In [128…
```
enc_dec_net.to(device)
print(None)
```

```
None
```

In [129…
```
result = enc_dec_net(torch.Tensor(enc_x_val[i:i+1]).long().to(device),
                     torch.Tensor(dec_x_val[i:i+1]).long().to(device)).cpu().detach().numpy()
result.argmax(-1)[0]
```

Out[129…
```
array([60, 32, 81, 30, 81, 65, 13, 30, 12, 53, 33, 34,  2, 69, 69, 33, 81,
       69, 69, 69, 33, 69, 69, 69, 69, 69, 69, 69, 69, 69, 69, 69, 69,
       69, 69, 69, 69, 30, 33, 69, 69, 69, 69])
```

In [130…
```
# Only if the above fails due to device management reasons...
# result = enc_dec_net(torch.Tensor(enc_x_val[i:i+1]).long().to(device),
#                      torch.Tensor(dec_x_val[i:i+1]).long().to(device)).cpu().detach().numpy()
# result.argmax(-1)[0]
```

In [131…
```
decode_seq(result.argmax(-1)[0],i_to_c_por)
```

Out[131…
```
'Gmoro treisa'
```

In [132…
```
trainer.validate(enc_dec_net, xy_val)
```

```
LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
SLURM auto-requeueing enabled. Setting signal handlers.
Validation: 0it [00:00, ?it/s]
```

Runningstage.validating metric	DataLoader 0
val_acc	0.7685006856918335
val_loss	1.1492973566055298

Out[132… [{'val_acc': 0.7685006856918335, 'val_loss': 1.1492973566055298}]

Non-Teacher Forcing

In [135…
```
# Get the gestalt context for the input sequence(s)
context = torch.Tensor(enc_x_val[i:i+1]).long()

# Prep a starting token...
token = torch.zeros((1,dec_y_val.shape[1])).long()
token[0,0] = 1
token
```

Out[135…
```
tensor([[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])
```

In [137…
```
# What do we get with just one pass?
enc_dec_net.to(device)
result = enc_dec_net(context.to(device),token.to(device))
result.cpu().detach().numpy().argmax(-1)[0]
```

Out[137…
```
array([60, 34, 81, 34, 12, 81, 81, 12, 12, 12, 12, 12, 69, 12, 12, 69, 12,
       12, 69, 12, 12, 69, 12, 69, 69, 69, 69, 12, 69, 69, 12, 69, 69, 69,
       69, 69, 69, 69, 69, 69, 69, 69, 69, 69])
```

In [139…
```
decode_seq(result.cpu().detach().numpy().argmax(-1)[0],i_to_c_por)
```

Out[139… 'Gaoaeooeeeee.ee.ee.ee.e....e..e.............'

In [140…
```
token[0,1] = result[0,0].argmax(-1)
token
```

Out[140…
```
tensor([[ 1, 60,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
          0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
          0,  0,  0,  0,  0,  0,  0,  0]])
```

In [145…
```
# Feed next token in...
enc_dec_net.to(device)
result = enc_dec_net(context.to(device),token.to(device))
result.cpu().detach().numpy().argmax(-1)[0]
```

Out[145…
```
array([58, 81, 69, 69, 69, 12, 12, 69, 12, 12, 69, 69, 69, 69, 69, 69, 69,
       69, 69, 12, 12, 69, 12, 69, 69, 69, 69, 69, 69, 69, 69, 69, 69, 69,
       69, 69, 69, 69, 69, 69, 69, 69, 69, 69])
```

In [147…
```
decode_seq(result.cpu().detach().numpy().argmax(-1)[0],i_to_c_por)
```

Out[147… 'Eo...ee.ee.........ee.e.....................'

Complete Sequence with Non-Teacher Forcing

In [150…
```
# Complete max_length cycles with the decoder
context = torch.Tensor(enc_x_val[i:i+1]).long()
token = torch.zeros((1,dec_y_val.shape[1])).long()
token[0,0] = 1

for x in range(dec_y_val.shape[1]-1):
    enc_dec_net.to(device)
    result = enc_dec_net(context.to(device),token.to(device)).argmax(-1)
    if result[0,x] == 2:
        break
    token[0,x+1] = result[0,x]
enc_dec_net.to(device)
result = enc_dec_net(context.to(device),token.to(device)).argmax(-1).cpu().detach().numpy()[0]
result
```

Out[150…
```
array([60, 81, 33, 13, 81, 65, 64, 12, 65, 13, 30, 12, 79, 33, 69,  2, 30,
       69, 30, 69, 30, 30, 30, 69, 30, 69, 30, 69, 30, 30, 30, 30, 12, 69,
       30, 30, 69, 69, 30, 30, 30, 69, 69, 69])
```

In [152…
```
decode_seq(result,i_to_c_por)
```

Out[152…
```
'Gosto de trens.'
```

In [153…
```
result.shape
```

Out[153… $(44,)$

In [154…
```
dec_y_val.shape
```

Out[154… $(2000,\ 44)$

Accuracy **without** teacher forcing...

In [155…
```
trainer.test(enc_dec_net, xy_val)
```

```
LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
SLURM auto-requeueing enabled. Setting signal handlers.
/opt/conda/lib/python3.11/site-packages/torch/utils/data/dataloader.py:560: UserWarning: This DataLoader will create 8 w
orker processes in total. Our suggested max number of worker in current system is 2, which is smaller than what this Dat
aLoader is going to create. Please be aware that excessive worker creation might get DataLoader running slow or even fre
eze, lower the worker number to avoid potential slowness/freeze if necessary.
  warnings.warn(_create_warning_msg(
Testing: 0it [00:00, ?it/s]
/opt/conda/lib/python3.11/site-packages/torch/nn/modules/activation.py:1160: UserWarning: Converting mask without torch.
bool dtype to bool; this will negatively affect performance. Prefer to use a boolean mask directly. (Triggered internall
y at ../aten/src/ATen/native/transformers/attention.cpp:150.)
  return torch._native_multi_head_attention(
```

Runningstage.testing metric	DataLoader 0
test_acc	0.35138800740242004
test_loss	8.84199333190918

Out[155…
```
[{'test_acc': 0.35138800740242004, 'test_loss': 8.84199333190918}]
```

Parity Problem Revisited...

Our problem will consist of the solution to the even/odd parity determination for a binary sequence. For example, if we have the binary sequence 1010111001, then we have an even number of ones and the sequence has even parity. For the sequence 1011111001, we have an odd number of ones and the sequence has odd parity. However, turning this into an iterative problem means we move from the left to the right and decide to map each digit to even (0) or odd (1), based on whether we have encountered an even or odd number of ones so far in the sequence:

Input:	**1**	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**0**	**1**
Output:	1	1	0	0	1	0	1	1	1	0

or for the second sequence:

Input:	**1**	**0**	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**1**
Output:	1	1	0	1	0	1	0	0	0	1

Boxplot Example

In [32]:
```
data = np.random.random(size=(50,3))
plt.boxplot(data,notch=True)
plt.ylabel('Accuracy (P)')
plt.xticks([1,2,3],['Simple','LSTM','Transformer'])
plt.show()
```