Open Lab 4

Recurrent Encoder-Decoder

CSCI 7850 - Deep Learning

Due: Oct. 26 @ 11:00pm

Assignment

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

1. Create one python script (`translation-simple.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 the top 10,000 sentences, shuffled (for monte-carlo sampling).
   - Use length 100 random embeddings for your encodings (`torch.nn.Embedding()`)
   - You should utilize simple recurrent layers in your model (`torch.nn.RNN()`)
   - Your recurrent layers should use the hyperbolic tangent activation function
   - 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**
2. Create one python script (`translation-lstm.py`) that solves the ENG-POR problem using an Encoder-Decoder architecture:
   - You should keep all architecture settings the same, **except** use LSTM recurrent layers (`torch.nn.LSTM()`)
3. Use your scripts to run your models - **perform 10 independent runs for each model**.
   - Compile your validation accuracy data into a single, two-column text file (`translation-results.txt`) that can be read in using `np.loadtxt` .
4. Create one python script (`parity-simple.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 simple recurrent layers in your model (`torch.nn.RNN()`)
   - Your recurrent layers should use the hyperbolic tangent activation function
   - 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**
5. Create one python script (`parity-lstm.py`) that solves the parity problem using an Encoder-Decoder architecture:
   - You should keep all architecture settings the same, **except** use LSTM recurrent layers (`torch.nn.LSTM()`)
6. Use your scripts to run your models - **perform 10 independent runs for each model**.
   - Compile your validation data into a text file (`parity-results.txt`) that can be read in using `np.loadtxt` .
7. Create an iPython Notebook file named `OL4.ipynb` which reads in the compiled results files that you created above to produce a boxplot comparing the performance of the problems/architectures.

Submission

Create a zip archive which contains the following contents:

- translation-simple.py
- translation-lstm.py
- translation-results.txt
- parity-simple.py
- parity-lstm.py
- parity-results.txt
- OL4.ipynb

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

Recurrent Encoder-Decoder

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 [1]:
```
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 [2]:
```
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, mul
ti_processor_count=68)
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 [3]:
```
url = "https://raw.githubusercontent.com/luisroque/deep-learning-articles/main/data/eng-por.txt"
```

In [4]:
```
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 [5]:
```
# Subset? - All of the data will take some time...
n_seq = data.shape[0]
n_seq = 100
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[5]:
```
23
```

In [6]:
```
data[0]
```

Out[6]:
```
array(['Listen.', 'Ouça-me!'], dtype='<U184')
```

In [7]:
```
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 [8]:
```
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 [9]:
```
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 [10]:
```
data[0]
```

Out[10]:
```
array(['Listen.', 'Ouça-me!'], dtype='<U184')
```

In [11]:
```
data[0,0]
```

Out[11]:
```
'Listen.'
```

In [12]:
```
temp = encode_seq(data[0,0],c_to_i_eng,max_length)
print(*temp)
```

```
1 19 20 36 12 35 40 32 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

In [13]:
```
decode_seq(temp,i_to_c_eng)
```

Out[13]:
```
'Listen.'
```

In [14]:
```
data[0,1]
```

Out[14]:
```
'Ouça-me!'
```

In [15]:
```
temp = encode_seq(data[0,1],c_to_i_por,max_length)
print(*temp)
```

```
1 46 11 20 27 37 17 45 29 2 0 0 0 0 0 0 0 0 0 0 0 0 0
```

In [16]:
```
decode_seq(temp,i_to_c_por)
```

Out[16]:
```
'Ouça-me!'
```

In [17]:
```
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 [18]:
```
enc_x_train = X[:split_point]
enc_x_val = X[split_point:]
enc_x_train
```

Out[18]:
```
array([[ 1, 19, 20, ...,  0,  0,  0],
       [ 1, 37, 41, ...,  0,  0,  0],
       [ 1, 30, 42, ...,  0,  0,  0],
       ...,
       [ 1, 27, 35, ...,  0,  0,  0],
       [ 1, 27, 42, ...,  0,  0,  0],
       [ 1, 34, 42, ...,  0,  0,  0]])
```

In [19]:
```
dec_x_train = Y[:,0:-1][:split_point]
dec_x_val = Y[:,0:-1][split_point:]
dec_x_train
```

Out[19]:
```
array([[ 1, 46, 11, ...,  0,  0,  0],
       [ 1, 19, 50, ...,  0,  0,  0],
       [ 1, 31, 52, ...,  0,  0,  0],
       ...,
       [ 1,  3, 51, ...,  0,  0,  0],
       [ 1,  3, 51, ...,  0,  0,  0],
       [ 1, 35, 27, ...,  0,  0,  0]])
```

In [20]:
```
dec_y_train = Y[:,1:][:split_point]
dec_y_val = Y[:,1:][split_point:]
dec_y_train
```

Out[20]:
```
array([[46, 11, 20, ...,  0,  0,  0],
       [19, 50, 13, ...,  0,  0,  0],
       [31, 52, 42, ...,  0,  0,  0],
       ...,
       [ 3, 51,  6, ...,  0,  0,  0],
       [ 3, 51, 36, ...,  0,  0,  0],
       [35, 27, 11, ...,  0,  0,  0]])
```

In [21]:
```
print(enc_x_train.shape)
print(dec_x_train.shape)
print(dec_y_train.shape)
```

```
(80, 23)
(80, 22)
(80, 22)
```

In [22]:
```
print(enc_x_val.shape)
print(dec_x_val.shape)
print(dec_y_val.shape)
```

```
(20, 23)
(20, 22)
(20, 22)
```

In [23]:
```
len(i_to_c_eng)
```

Out[23]: 44

In [24]:
```python
len(i_to_c_por)
```

Out[24]: 56

In [25]:
```python
enc_x_train.shape[1:]
```

Out[25]: $(23,)$

In [26]:
```python
class RecurrentResidual(torch.nn.Module):
    def __init__(self,
                 latent_size = 64,
                 bidirectional = False,
                 **kwargs):
        super().__init__(**kwargs)
        self.layer_norm = torch.nn.LayerNorm(latent_size)
        self.rnn_layer = torch.nn.RNN(latent_size,
                                      latent_size // 2 if bidirectional else latent_size,
                                      bidirectional=bidirectional,
                                      batch_first=True)
    def forward(self, x):
        return x + self.rnn_layer(self.layer_norm(x))[0]
```

Encoder Component

In [27]:
```python
class EncoderNetwork(torch.nn.Module):
    def __init__(self,
                 num_tokens,
                 latent_size = 64, # Use something divisible by 2
                 n_layers = 4,
                 **kwargs):
        super().__init__(**kwargs)
        self.embedding = torch.nn.Embedding(num_tokens,
                                            latent_size,
                                            padding_idx=0)
        self.dropout = torch.nn.Dropout1d(0.1) # Whole token dropped
        self.rnn_layers = torch.nn.Sequential(*[
            RecurrentResidual(latent_size,True) for _ in range(n_layers)
        ])

    def forward(self, x):
        y = x
        y = self.embedding(y)
        y = self.dropout(y)
        y = self.rnn_layers(y)[:,-1]
        return y
```

In [28]:
```python
enc_x_train[0:5].shape
```

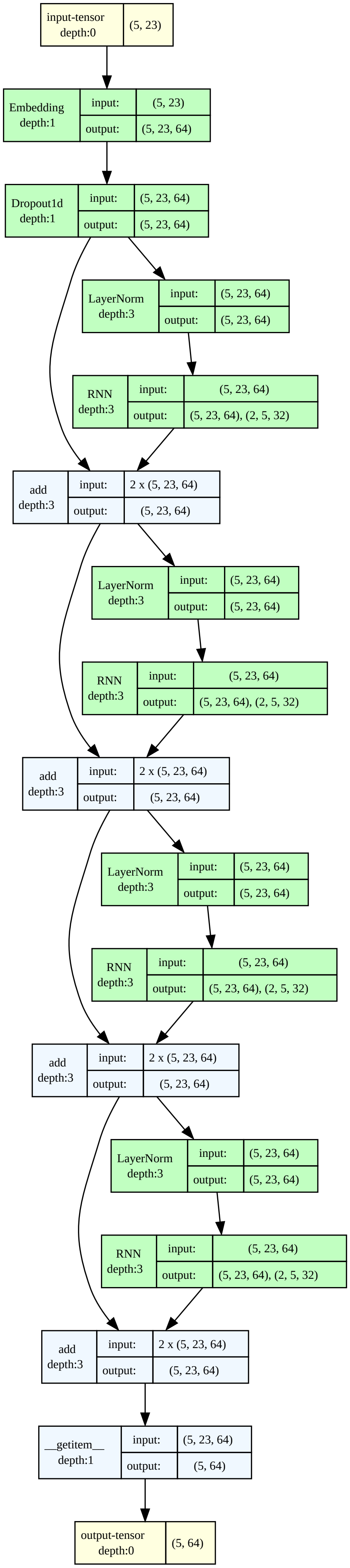
Out[28]: $(5,\ 23)$

In [29]:
```python
enc_net = EncoderNetwork(num_tokens=len(i_to_c_eng))

summary(enc_net,input_data=torch.Tensor(enc_x_train[0:5]).long())
```

Out[29]:
```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
EncoderNetwork                           [5, 64]                   --
├─Embedding: 1-1                         [5, 23, 64]               2,816
├─Dropout1d: 1-2                         [5, 23, 64]               --
├─Sequential: 1-3                        [5, 23, 64]               --
│    └─RecurrentResidual: 2-1            [5, 23, 64]               --
│    │    └─LayerNorm: 3-1               [5, 23, 64]               128
│    │    └─RNN: 3-2                     [5, 23, 64]               6,272
│    └─RecurrentResidual: 2-2            [5, 23, 64]               --
│    │    └─LayerNorm: 3-3               [5, 23, 64]               128
│    │    └─RNN: 3-4                     [5, 23, 64]               6,272
│    └─RecurrentResidual: 2-3            [5, 23, 64]               --
│    │    └─LayerNorm: 3-5               [5, 23, 64]               128
│    │    └─RNN: 3-6                     [5, 23, 64]               6,272
│    └─RecurrentResidual: 2-4            [5, 23, 64]               --
│    │    └─LayerNorm: 3-7               [5, 23, 64]               128
│    │    └─RNN: 3-8                     [5, 23, 64]               6,272
==========================================================================================
Total params: 28,416
Trainable params: 28,416
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 2.90
==========================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.53
Params size (MB): 0.11
Estimated Total Size (MB): 0.64
==========================================================================================
```

In [30]:
```python
model_graph = draw_graph(enc_net, input_data=torch.Tensor(enc_x_train[0:5]).long(), device=device,
                        hide_inner_tensors=True,hide_module_functions=True,
                        expand_nested=False, depth=3, dtypes=[torch.long])
model_graph.visual_graph
```

Out[30]:


input-tensor depth:0 (5, 23)
Embedding depth:1 input: (5, 23) output: (5, 23, 64)
Dropout1d depth:1 input: (5, 23, 64) output: (5, 23, 64)
LayerNorm depth:3 input: (5, 23, 64) output: (5, 23, 64)
RNN depth:3 input: (5, 23, 64) output: (5, 23, 64), (2, 5, 32)
add depth:3 input: 2 x (5, 23, 64) output: (5, 23, 64)
LayerNorm depth:3 input: (5, 23, 64) output: (5, 23, 64)
RNN depth:3 input: (5, 23, 64) output: (5, 23, 64), (2, 5, 32)
add depth:3 input: 2 x (5, 23, 64) output: (5, 23, 64)
LayerNorm depth:3 input: (5, 23, 64) output: (5, 23, 64)
RNN depth:3 input: (5, 23, 64) output: (5, 23, 64), (2, 5, 32)
add depth:3 input: 2 x (5, 23, 64) output: (5, 23, 64)
LayerNorm depth:3 input: (5, 23, 64) output: (5, 23, 64)
RNN depth:3 input: (5, 23, 64) output: (5, 23, 64), (2, 5, 32)
add depth:3 input: 2 x (5, 23, 64) output: (5, 23, 64)
__getitem__ depth:1 input: (5, 23, 64) output: (5, 64)
output-tensor depth:0 (5, 64)

Decoder Component

In [31]:
```
class DecoderNetwork(torch.nn.Module):
    def __init__(self,
                 num_tokens,
                 latent_size = 64, # Use something divisible by 2
                 n_layers = 4,
                 **kwargs):
        super().__init__(**kwargs)
        self.embedding = torch.nn.Embedding(num_tokens,
                                            latent_size,
                                            padding_idx=0)
        self.dropout = torch.nn.Dropout1d(0.1) # Whole token dropped
        self.linear = torch.nn.Linear(latent_size*2,
                                      latent_size)
        self.rnn_layers = torch.nn.Sequential(*[
            RecurrentResidual(latent_size,False) for _ in range(n_layers)
        ])
        self.output_layer = torch.nn.Linear(latent_size,
                                            num_tokens)

    def forward(self, x_enc, x_dec):
        y_enc = x_enc.unsqueeze(1).repeat(1,x_dec.shape[1],1)
        y_dec = self.embedding(x_dec)
        y_dec = self.dropout(y_dec)
        y = y_enc
        y = torch.concatenate([y_enc,y_dec],-1)
        y = self.linear(y)
        y = self.rnn_layers(y)
        y = self.output_layer(y)
        return y
```

In [32]:
```
enc_x_train[0:5].shape
```

Out[32]: $(5, 23)$

In [33]:
```
dec_x_train[0:5].shape
```

Out[33]: $(5, 22)$

In [34]:
```
# Passed through the encoder network - output tensor shape for decoder
enc_net(torch.Tensor(enc_x_train[0:5]).long().to(device)).shape
```

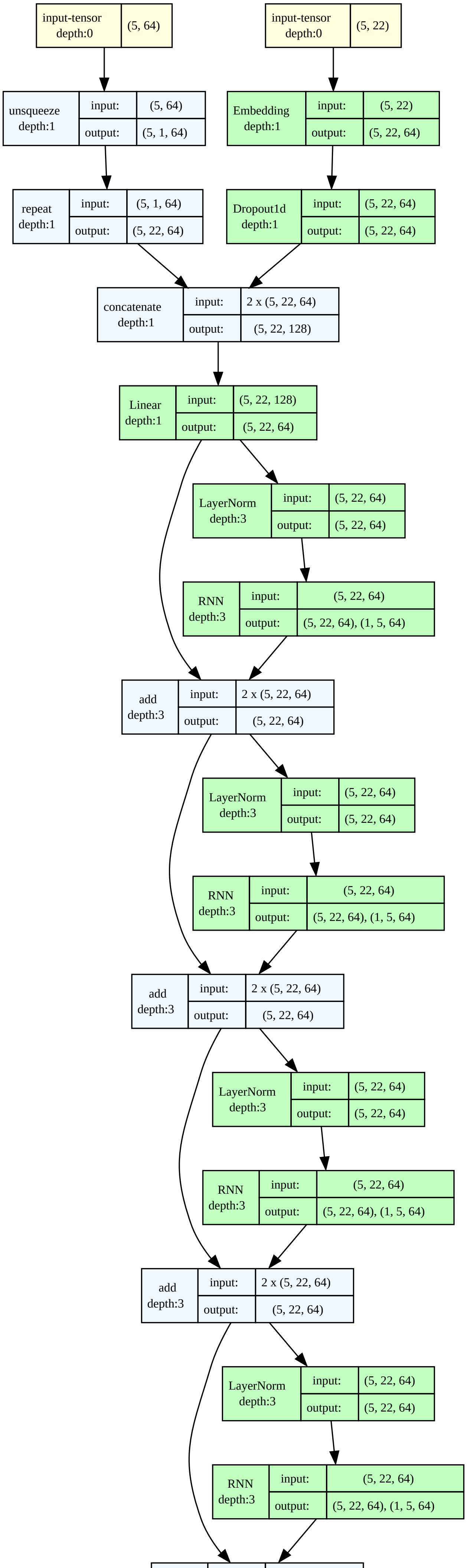
Out[34]:
```
torch.Size([5, 64])
```

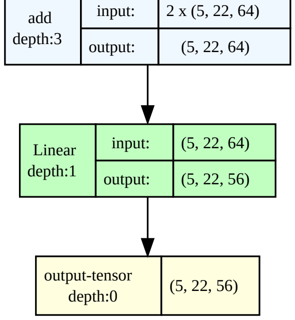
In [35]:
```
dec_net = DecoderNetwork(num_tokens=len(i_to_c_por))

summary(dec_net,input_data=[enc_net(torch.Tensor(enc_x_train[0:5]).long().to(device)).cpu(),
                            torch.Tensor(dec_x_train[0:5]).long()])
```

Out[35]:
```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
DecoderNetwork                           [5, 22, 56]               --
├─Embedding: 1-1                         [5, 22, 64]               3,584
├─Dropout1d: 1-2                         [5, 22, 64]               --
├─Linear: 1-3                            [5, 22, 64]               8,256
├─Sequential: 1-4                        [5, 22, 64]               --
│    └─RecurrentResidual: 2-1            [5, 22, 64]               --
│    │    └─LayerNorm: 3-1               [5, 22, 64]               128
│    │    └─RNN: 3-2                     [5, 22, 64]               8,320
│    └─RecurrentResidual: 2-2            [5, 22, 64]               --
│    │    └─LayerNorm: 3-3               [5, 22, 64]               128
│    │    └─RNN: 3-4                     [5, 22, 64]               8,320
│    └─RecurrentResidual: 2-3            [5, 22, 64]               --
│    │    └─LayerNorm: 3-5               [5, 22, 64]               128
│    │    └─RNN: 3-6                     [5, 22, 64]               8,320
│    └─RecurrentResidual: 2-4            [5, 22, 64]               --
│    │    └─LayerNorm: 3-7               [5, 22, 64]               128
│    │    └─RNN: 3-8                     [5, 22, 64]               8,320
├─Linear: 1-5                            [5, 22, 56]               3,640
==========================================================================================
Total params: 49,272
Trainable params: 49,272
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 3.74
==========================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.61
Params size (MB): 0.20
Estimated Total Size (MB): 0.81
==========================================================================================
```

In [36]:
```python
model_graph = draw_graph(dec_net, input_data=[enc_net(torch.Tensor(enc_x_train[0:5]).long().to(devic
                            torch.Tensor(dec_x_train[0:5]).long()], device=device,
                        hide_inner_tensors=True,hide_module_functions=True,
                        expand_nested=False, depth=3, dtypes=[torch.long])
model_graph.visual_graph
```

Out[36]:


input-tensor depth:0 (5, 64)
input-tensor depth:0 (5, 22)
unsqueeze depth:1 input: (5, 64) output: (5, 1, 64)
Embedding depth:1 input: (5, 22) output: (5, 22, 64)
repeat depth:1 input: (5, 1, 64) output: (5, 22, 64)
Dropout1d depth:1 input: (5, 22, 64) output: (5, 22, 64)
concatenate depth:1 input: 2 x (5, 22, 64) output: (5, 22, 128)
Linear depth:1 input: (5, 22, 128) output: (5, 22, 64)
LayerNorm depth:3 input: (5, 22, 64) output: (5, 22, 64)
RNN depth:3 input: (5, 22, 64) output: (5, 22, 64), (1, 5, 64)
add depth:3 input: 2 x (5, 22, 64) output: (5, 22, 64)
LayerNorm depth:3 input: (5, 22, 64) output: (5, 22, 64)
RNN depth:3 input: (5, 22, 64) output: (5, 22, 64), (1, 5, 64)
add depth:3 input: 2 x (5, 22, 64) output: (5, 22, 64)
LayerNorm depth:3 input: (5, 22, 64) output: (5, 22, 64)
RNN depth:3 input: (5, 22, 64) output: (5, 22, 64), (1, 5, 64)
add depth:3 input: 2 x (5, 22, 64) output: (5, 22, 64)
LayerNorm depth:3 input: (5, 22, 64) output: (5, 22, 64)
RNN depth:3 input: (5, 22, 64) output: (5, 22, 64), (1, 5, 64)



Training Hooks

In [37]:

```
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
```

This `test_step` function is customized for this lab - we need to feed in the predicted outputs one step at a time for non-teacher forcing, and this takes quite a bit more time to run compared to the standard validation procedure.

Encoder-Decoder Network

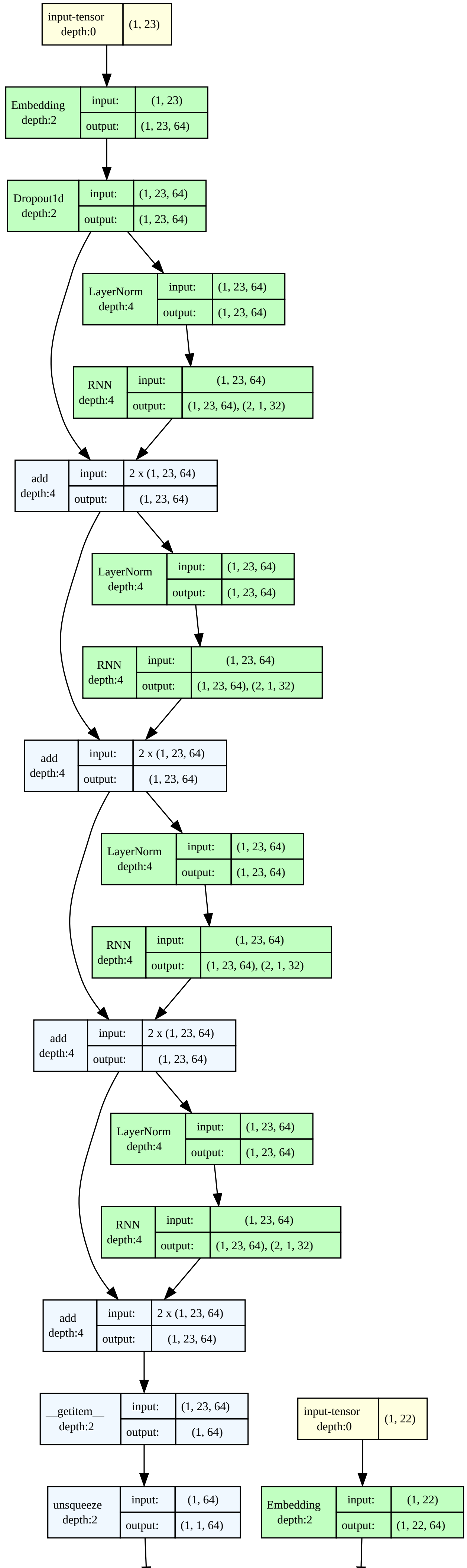
In [38]:
```
class EncDecNetwork(EncDecLightningModule):
    def __init__(self,
                 num_enc_tokens,
                 num_dec_tokens,
                 latent_size = 64, # Use something divisible by 2
                 n_layers = 4,
                 **kwargs):
        super().__init__(output_size=num_dec_tokens,
                         **kwargs)
        self.enc_net = EncoderNetwork(num_enc_tokens,latent_size,n_layers)
        self.dec_net = DecoderNetwork(num_dec_tokens,latent_size,n_layers)

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

In [39]:
```
enc_dec_net = EncDecNetwork(num_enc_tokens=len(i_to_c_eng),
                            num_dec_tokens=len(i_to_c_por))
summary(enc_dec_net,input_data=[torch.Tensor(enc_x_train[0:1]).long(),
                                torch.Tensor(dec_x_train[0:1]).long()])
```

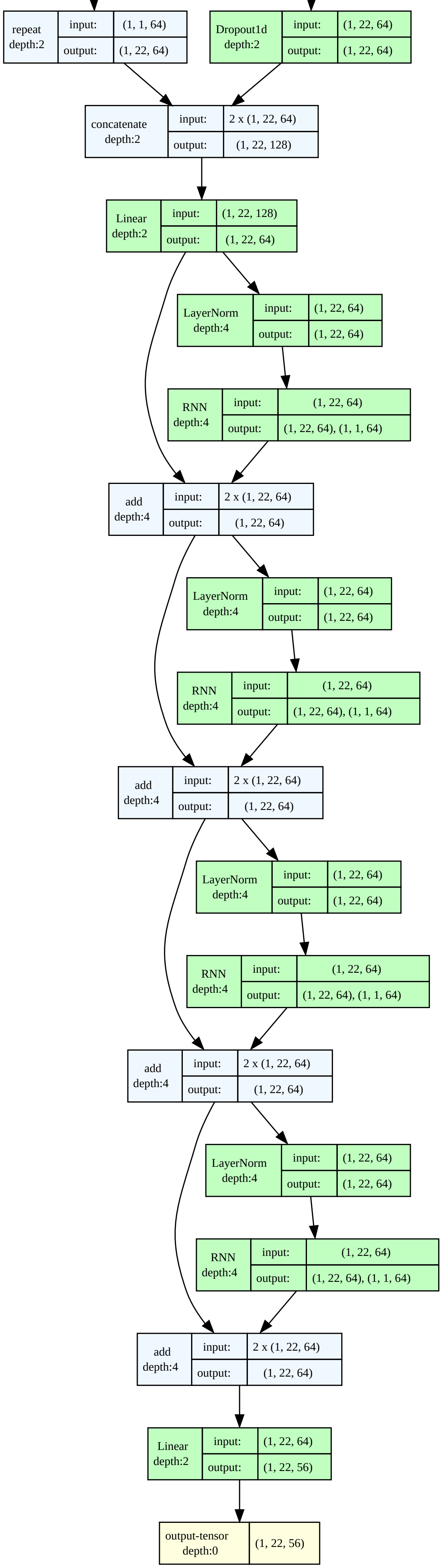
Out[39]:
```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
EncDecNetwork                            [1, 22, 56]               --
├─EncoderNetwork: 1-1                    [1, 64]                   --
│    └─Embedding: 2-1                    [1, 23, 64]               2,816
│    └─Dropout1d: 2-2                    [1, 23, 64]               --
│    └─Sequential: 2-3                   [1, 23, 64]               --
│    │    └─RecurrentResidual: 3-1       [1, 23, 64]               6,400
│    │    └─RecurrentResidual: 3-2       [1, 23, 64]               6,400
│    │    └─RecurrentResidual: 3-3       [1, 23, 64]               6,400
│    │    └─RecurrentResidual: 3-4       [1, 23, 64]               6,400
├─DecoderNetwork: 1-2                    [1, 22, 56]               --
│    └─Embedding: 2-4                    [1, 22, 64]               3,584
│    └─Dropout1d: 2-5                    [1, 22, 64]               --
│    └─Linear: 2-6                       [1, 22, 64]               8,256
│    └─Sequential: 2-7                   [1, 22, 64]               --
│    │    └─RecurrentResidual: 3-5       [1, 22, 64]               8,448
│    │    └─RecurrentResidual: 3-6       [1, 22, 64]               8,448
│    │    └─RecurrentResidual: 3-7       [1, 22, 64]               8,448
│    │    └─RecurrentResidual: 3-8       [1, 22, 64]               8,448
│    └─Linear: 2-8                       [1, 22, 56]               3,640
==========================================================================================
Total params: 77,688
Trainable params: 77,688
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 1.33
==========================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.23
Params size (MB): 0.31
Estimated Total Size (MB): 0.54
==========================================================================================
```

In [40]:
```
model_graph = draw_graph(enc_dec_net,
                         input_data=[torch.Tensor(enc_x_train[0:1]).long(),
                                     torch.Tensor(dec_x_train[0:1]).long()],
                         device=device,
                         hide_inner_tensors=True,hide_module_functions=True,
                         expand_nested=False, depth=4, dtypes=[torch.long])
model_graph.visual_graph
```

Out[40]:



input-tensor depth:0
(1, 23)
Embedding depth:2
input: (1, 23)
output: (1, 23, 64)
Dropout1d depth:2
input: (1, 23, 64)
output: (1, 23, 64)
LayerNorm depth:4
input: (1, 23, 64)
output: (1, 23, 64)
RNN depth:4
input: (1, 23, 64)
output: (1, 23, 64), (2, 1, 32)
add depth:4
input: 2 x (1, 23, 64)
output: (1, 23, 64)
LayerNorm depth:4
input: (1, 23, 64)
output: (1, 23, 64)
RNN depth:4
input: (1, 23, 64)
output: (1, 23, 64), (2, 1, 32)
add depth:4
input: 2 x (1, 23, 64)
output: (1, 23, 64)
LayerNorm depth:4
input: (1, 23, 64)
output: (1, 23, 64)
RNN depth:4
input: (1, 23, 64)
output: (1, 23, 64), (2, 1, 32)
add depth:4
input: 2 x (1, 23, 64)
output: (1, 23, 64)
LayerNorm depth:4
input: (1, 23, 64)
output: (1, 23, 64)
RNN depth:4
input: (1, 23, 64)
output: (1, 23, 64), (2, 1, 32)
add depth:4
input: 2 x (1, 23, 64)
output: (1, 23, 64)
__getitem__ depth:2
input: (1, 23, 64)
output: (1, 64)
input-tensor depth:0
(1, 22)
unsqueeze depth:2
input: (1, 64)
output: (1, 1, 64)
Embedding depth:2
input: (1, 22)
output: (1, 22, 64)


repeat
depth:2
input:
(1, 1, 64)
output:
(1, 22, 64)
Dropout1d
depth:2
input:
(1, 22, 64)
output:
(1, 22, 64)
concatenate
depth:2
input:
2 x (1, 22, 64)
output:
(1, 22, 128)
Linear
depth:2
input:
(1, 22, 128)
output:
(1, 22, 64)
LayerNorm
depth:4
input:
(1, 22, 64)
output:
(1, 22, 64)
RNN
depth:4
input:
(1, 22, 64)
output:
(1, 22, 64), (1, 1, 64)
add
depth:4
input:
2 x (1, 22, 64)
output:
(1, 22, 64)
LayerNorm
depth:4
input:
(1, 22, 64)
output:
(1, 22, 64)
RNN
depth:4
input:
(1, 22, 64)
output:
(1, 22, 64), (1, 1, 64)
add
depth:4
input:
2 x (1, 22, 64)
output:
(1, 22, 64)
LayerNorm
depth:4
input:
(1, 22, 64)
output:
(1, 22, 64)
RNN
depth:4
input:
(1, 22, 64)
output:
(1, 22, 64), (1, 1, 64)
add
depth:4
input:
2 x (1, 22, 64)
output:
(1, 22, 64)
LayerNorm
depth:4
input:
(1, 22, 64)
output:
(1, 22, 64)
RNN
depth:4
input:
(1, 22, 64)
output:
(1, 22, 64), (1, 1, 64)
add
depth:4
input:
2 x (1, 22, 64)
output:
(1, 22, 64)
Linear
depth:2
input:
(1, 22, 64)
output:
(1, 22, 56)
output-tensor
depth:0
(1, 22, 56)

Final training preparations...

In [41]:
```
batch_size = 20
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 DataLo
ader will create 8 worker processes in total. Our suggested max number of worker in current system i
s 2, which is smaller than what this DataLoader is going to create. Please be aware that excessive w
orker creation might get DataLoader running slow or even freeze, lower the worker number to avoid po
tential slowness/freeze if necessary.
  warnings.warn(_create_warning_msg(
```

In [42]:
```
logger = pl.loggers.CSVLogger("lightning_logs",
                              name="Open_Lab_4",
                              version="demo-0")
```

In [43]:
```
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 [44]:
```
trainer.validate(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 DataLo
ader will create 8 worker processes in total. Our suggested max number of worker in current system i
s 2, which is smaller than what this DataLoader is going to create. Please be aware that excessive w
orker creation might get DataLoader running slow or even freeze, lower the worker number to avoid po
tential slowness/freeze if necessary.
  warnings.warn(_create_warning_msg(
Validation: 0it [00:00, ?it/s]
```

Runningstage.validating metric	DataLoader 0
val_acc	0.0
val_loss	4.279879093170166

Out[44]:
```
[{'val_acc': 0.0, 'val_loss': 4.279879093170166}]
```

In [45]:
```
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	0.0059523810632526875
test_loss	4.23967981338501

Out[45]: [{'test_acc': 0.0059523810632526875, 'test_loss': 4.23967981338501}]

In [46]:
```
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     | 28.4 K
3 | dec_net  | DecoderNetwork     | 49.3 K
------------------------------------------------
77.7 K    Trainable params
0         Non-trainable params
77.7 K    Total params
0.311     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]
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]
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]
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]
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]
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]
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]
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]
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]
```

```
`Trainer.fit` stopped: `max_epochs=300` reached.
```

In [47]:

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

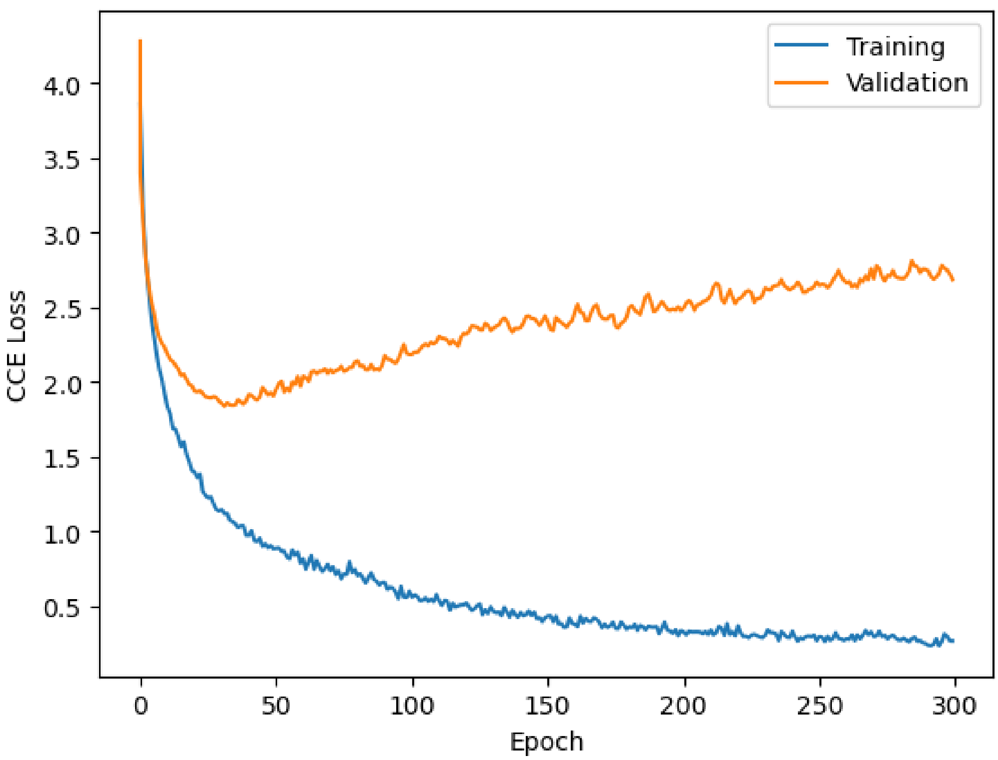
Out[47]:

	val_acc	val_loss	epoch	step	test_acc	test_loss	train_acc	train_loss
0	0.000000	4.279879	0	0	NaN	NaN	NaN	NaN
1	NaN	NaN	0	0	0.005952	4.23968	NaN	NaN
2	0.154762	3.394269	0	3	NaN	NaN	NaN	NaN
3	NaN	NaN	0	3	NaN	NaN	0.093578	3.859718
4	0.238095	3.073233	1	7	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...
597	NaN	NaN	297	1191	NaN	NaN	0.870890	0.294473
598	0.553571	2.719983	298	1195	NaN	NaN	NaN	NaN
599	NaN	NaN	298	1195	NaN	NaN	0.886950	0.266533
600	0.535714	2.684143	299	1199	NaN	NaN	NaN	NaN
601	NaN	NaN	299	1199	NaN	NaN	0.879583	0.265495

602 rows × 8 columns

In [48]:

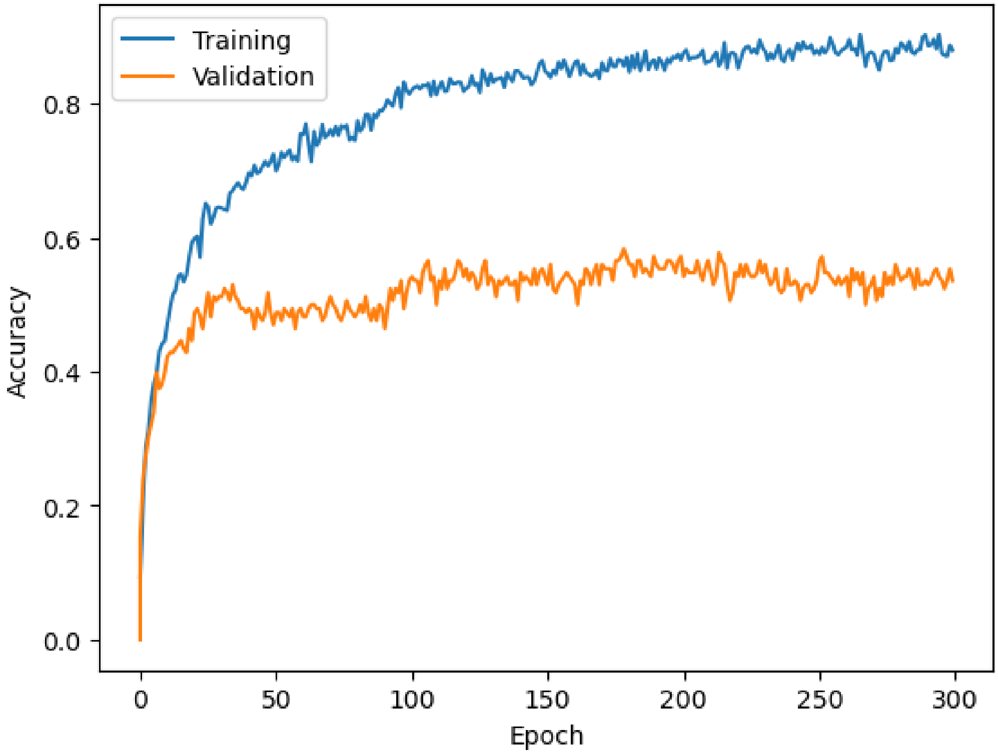
```
plt.plot(results["epoch"][np.logical_not(np.isnan(results["train_loss"]))],
         results["train_loss"][np.logical_not(np.isnan(results["train_loss"]))],
         label="Training")
plt.plot(results["epoch"][np.logical_not(np.isnan(results["val_loss"]))],
         results["val_loss"][np.logical_not(np.isnan(results["val_loss"]))],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



In [49]:

```
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 [50]:
```
# What should we see?
i = 0
print('Input:', enc_x_val[i])
print('Output:', dec_y_val[i])
```

```
Input: [ 1 34 21 20 12 32  2  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
Output: [ 9 47 32 45 18 45 29  2  0  0  0  0  0  0  0  0  0  0  0  0  0]
```

In [51]:
```
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: Wait.
Output: Espere!
```

In [52]:
```
result = enc_dec_net(torch.Tensor(enc_x_val[i:i+1]).long(),
                     torch.Tensor(dec_x_val[i:i+1]).long()).cpu().detach().numpy()
result.argmax(-1)[0]
```

Out[52]:
```
array([48, 47, 32, 45, 18, 45, 17, 42,  2, 18, 51, 17, 45, 45, 17,  2, 45,
       17,  2, 45, 45,  2])
```

In [53]:
```
# 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 [54]:
```
decode_seq(result.argmax(-1)[0],i_to_c_por)
```

Out[54]:
```
'Csperem.'
```

In [55]:
```
trainer.validate(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 DataLo
ader will create 8 worker processes in total. Our suggested max number of worker in current system i
s 2, which is smaller than what this DataLoader is going to create. Please be aware that excessive w
orker creation might get DataLoader running slow or even freeze, lower the worker number to avoid po
tential slowness/freeze if necessary.
  warnings.warn(_create_warning_msg(
Validation: 0it [00:00, ?it/s]
```

Runningstage.validating metric	DataLoader 0
val_acc	0.5357142686843872
val_loss	2.684142589569092

Out[55]:
```
[{'val_acc': 0.5357142686843872, 'val_loss': 2.684142589569092}]
```

Non-Teacher Forcing

In [74]:
```python
# Get the gestalt context for the input sequence(s)
context = enc_dec_net.enc_net(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[74]:
```
tensor([[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])
```

In [75]:
```python
# What do we get with just one pass?
result = enc_dec_net.dec_net(context,token)
result.cpu().detach().numpy().argmax(-1)[0]
```

Out[75]:
```
array([48, 51, 18, 45, 45, 29,  2,  2,  2,  2,  2,  2, 45, 45,  2,  2,  2,
        2,  2,  2,  2,  2])
```

In [76]:
```python
decode_seq(result.cpu().detach().numpy().argmax(-1)[0],i_to_c_por)
```

Out[76]:
```
'Coree!'
```

In [77]:
```python
token[0,1] = result[0,0].argmax(-1)
token
```

Out[77]:
```
tensor([[ 1, 48,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
          0,  0,  0,  0]])
```

In [78]:
```python
# Feed next token in...
result = enc_dec_net.dec_net(context,token)
result.cpu().detach().numpy().argmax(-1)[0]
```

Out[78]:
```
array([48, 51, 18, 18, 45, 29,  2,  2,  2, 45, 45,  2, 45, 45,  2,  2,  2,
        2,  2,  2,  2,  2])
```

In [79]:
```python
decode_seq(result.cpu().detach().numpy().argmax(-1)[0],i_to_c_por)
```

Out[79]:
```
'Corre!'
```

Complete Sequence with Non-Teacher Forcing

In [80]:
```python
# Complete max_length cycles with the decoder
context = enc_dec_net.enc_net(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):
    result = enc_dec_net.dec_net(context,token).argmax(-1)
    if result[0,x] == 2:
        break
    token[0,x+1] = result[0,x]
result = enc_dec_net.dec_net(context,token).argmax(-1).cpu().detach().numpy()[0]
result
```

Out[80]:
```
array([48, 51, 18, 18, 45, 29,  2,  2,  2, 18, 45, 18, 45, 17, 17, 45,  2,
        2,  2,  2,  2, 45])
```

In [81]:
```
decode_seq(result,i_to_c_por)
```

Out[81]:
```
'Corre!'
```

In [82]:
```
result.shape
```

Out[82]: $(22,)$

In [83]:
```
dec_y_val.shape
```

Out[83]: $(20, 22)$

Accuracy **without** teacher forcing...

In [84]:
```
results = 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 worker processes in total. Our suggested max number of worker in current system is 2, which is smaller than what this DataLoader is going to create. Please be aware that excessive worker creation might get DataLoader running slow or even freeze, lower the worker number to avoid potential slowness/freeze if necessary.
  warnings.warn(_create_warning_msg(
Testing: 0it [00:00, ?it/s]
```

Runningstage.testing metric	DataLoader 0
test_acc	0.4166666567325592
test_loss	5.355428695678711

In [85]:
```
print("Test Accuracy:",results[0]['test_acc'])
```

```
Test Accuracy: 0.4166666567325592
```

Parity Problem Revisited...

Our other 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

Note that we will utilize an Encoder-Decoder architecture for this lab, which isn't necessarily the best fit for this task since we are hoping for full parity bitstring reconstruction from the context representation. However, this will be an interesting benchmark which we will use in the next lab assignment, so we are getting prepared for that now.

Boxplot Example

In [34]:
```
data = np.random.random(size=(50,2))
plt.boxplot(data,notch=True)
plt.ylabel('Accuracy (P)')
plt.xticks([1,2],['Simple','LSTM'])
plt.show()
```