High Performance Machine Learning Assignment 3

Name: Jayraj M. Pamnani
NYU ID: jmp10051

Wandb Project: https://wandb.ai/jmp10051-new-york-university/HPML_HW3_Jayraj?nw=nwuserjmp10051

Problem 1: Chatbot Seq-2-Seq Model

Q1.1: Make a copy of the notebook of the tutorial, follow the instructions to train and evaluate the chatbot model in your local Google Colab environment.

Solution: Answered in the chatbot.ipynb file.

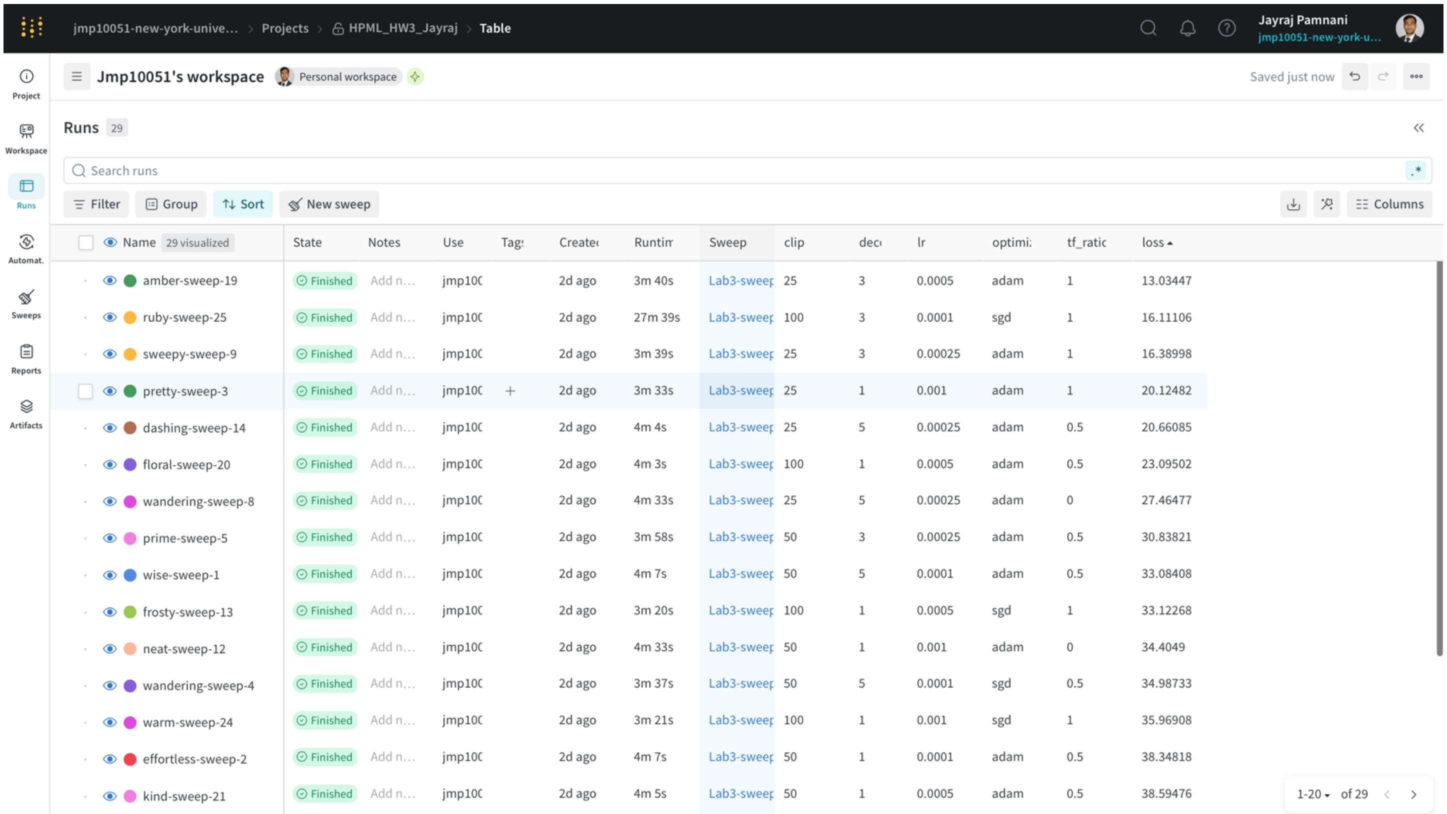
Q1.2: Learn how to use Weights and Biases (W&B) to run a hyperparameter sweep and instrument the notebook to use the Weights and Biases integration to help you run some hyperparameter sweeps in the next steps. Watch the video tutorial provided in the references section.

Q1.3 & 1.4:

Solution:

Hyperparameter Sweeps:

Total Runs: 25 (in the screenshot it shows 29 because the initial 4 runs were a test)



Name 29 visualized	State	Notes	Use	Tag:	Create	Runtir	Sweep	clip	deci	lr	optimi:	tf_ratic	loss
amber-sweep-19	Finished	Add n...	jmp10C		2d ago	3m 40s	Lab3-sweer	25	3	0.0005	adam	1	13.03447
ruby-sweep-25	Finished	Add n...	jmp10C		2d ago	27m 39s	Lab3-sweer	100	3	0.0001	sgd	1	16.11106
sweepy-sweep-9	Finished	Add n...	jmp10C		2d ago	3m 39s	Lab3-sweer	25	3	0.00025	adam	1	16.38998
pretty-sweep-3	Finished	Add n...	jmp10C	+	2d ago	3m 33s	Lab3-sweer	25	1	0.001	adam	1	20.12482
dashing-sweep-14	Finished	Add n...	jmp10C		2d ago	4m 4s	Lab3-sweer	25	5	0.00025	adam	0.5	20.66085
floral-sweep-20	Finished	Add n...	jmp10C		2d ago	4m 3s	Lab3-sweer	100	1	0.0005	adam	0.5	23.09502
wandering-sweep-8	Finished	Add n...	jmp10C		2d ago	4m 33s	Lab3-sweer	25	5	0.00025	adam	0	27.46477
prime-sweep-5	Finished	Add n...	jmp10C		2d ago	3m 58s	Lab3-sweer	50	3	0.00025	adam	0.5	30.83821
wise-sweep-1	Finished	Add n...	jmp10C		2d ago	4m 7s	Lab3-sweer	50	5	0.0001	adam	0.5	33.08408
frosty-sweep-13	Finished	Add n...	jmp10C		2d ago	3m 20s	Lab3-sweer	100	1	0.0005	sgd	1	33.12268
neat-sweep-12	Finished	Add n...	jmp10C		2d ago	4m 33s	Lab3-sweer	50	1	0.001	adam	0	34.4049
wandering-sweep-4	Finished	Add n...	jmp10C		2d ago	3m 37s	Lab3-sweer	50	5	0.0001	sgd	0.5	34.98733
warm-sweep-24	Finished	Add n...	jmp10C		2d ago	3m 21s	Lab3-sweer	100	1	0.001	sgd	1	35.96908
effortless-sweep-2	Finished	Add n...	jmp10C		2d ago	4m 7s	Lab3-sweer	50	1	0.0001	adam	0.5	38.34818
kind-sweep-21	Finished	Add n...	jmp10C		2d ago	4m 5s	Lab3-sweer	50	1	0.0005	adam	0.5	38.59476

Q1.5:

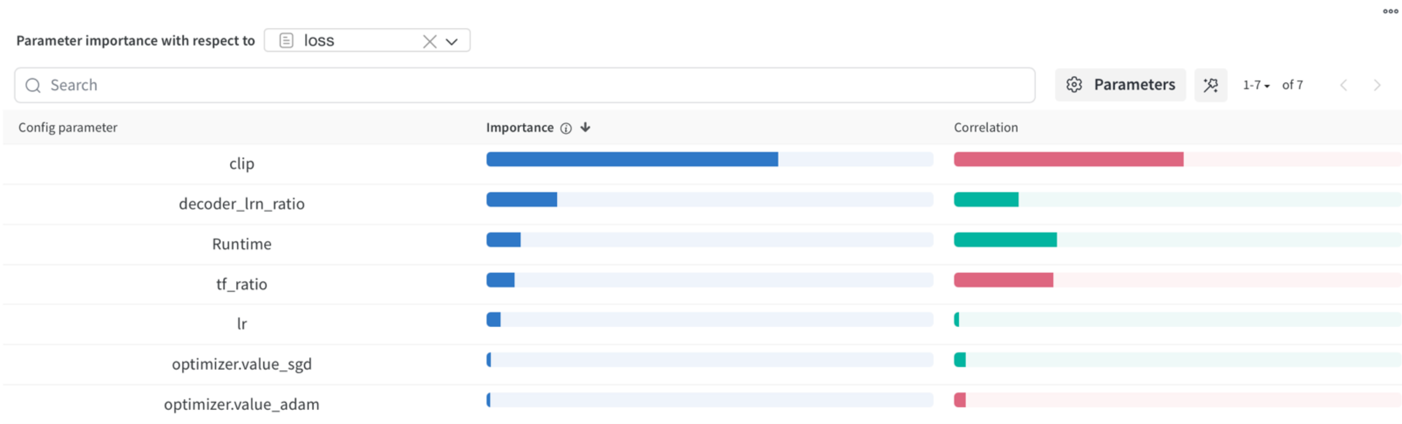
Solution:

From above results, I observed that the run with the following parameters has least loss of 13.03447.

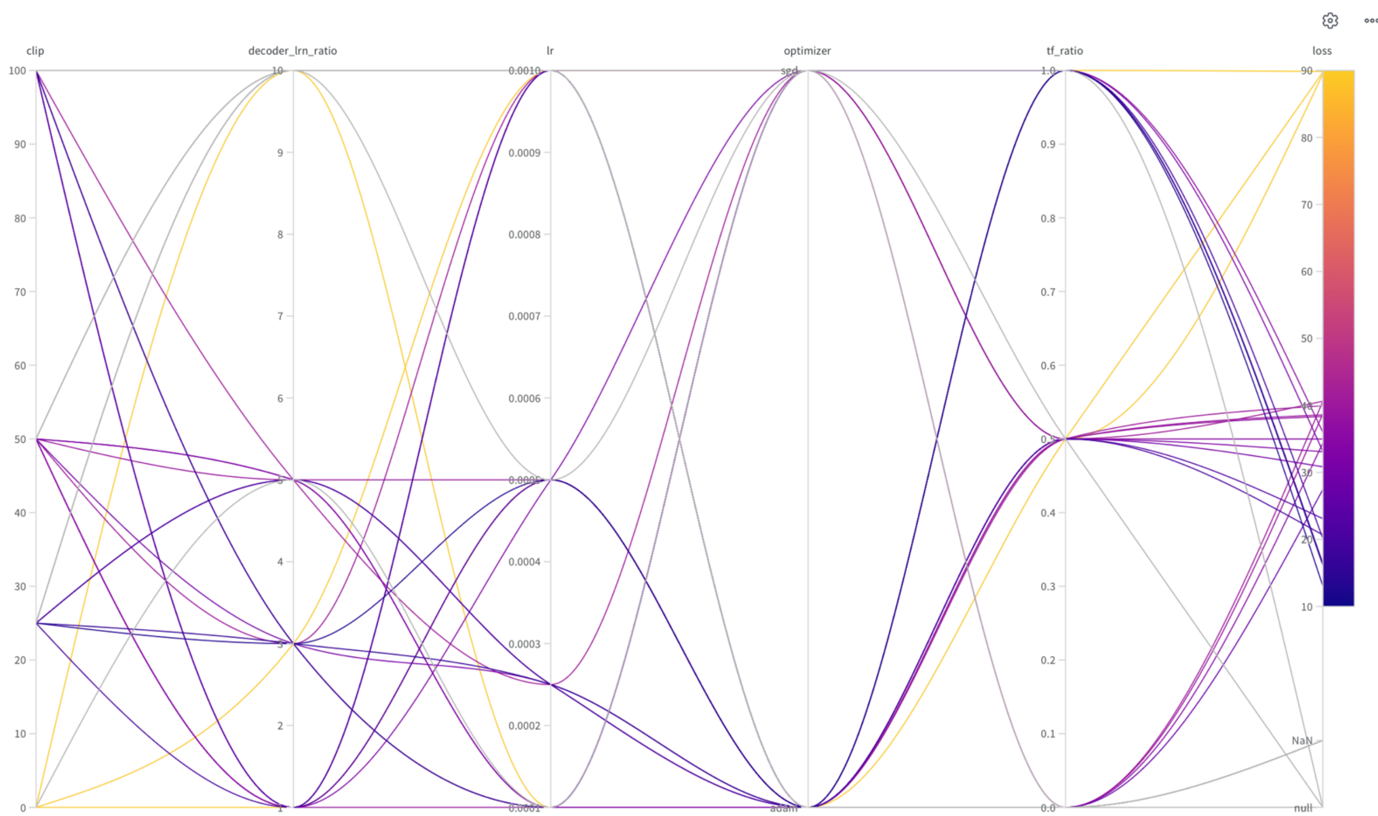
Hyperparameters value of above model:

1. Optimizer: adam
2. Learning Rate: 0.0005
3. Clip: 25.0
4. Teacher Forcing Ratio: 1
5. Decoder Learning Ratio: 3

Feature Importance:

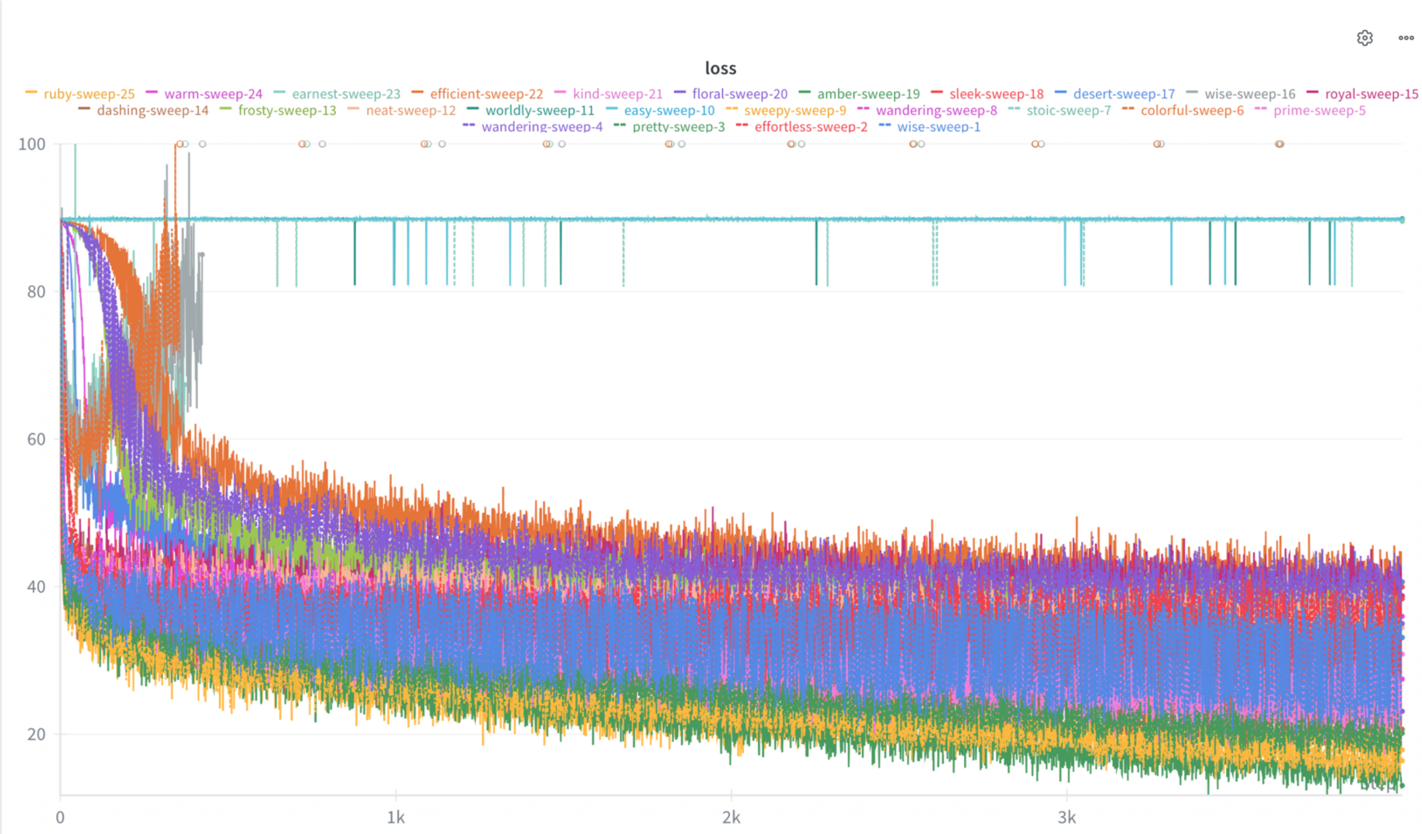


From the panel above, we can see that the clip parameter has a significant impact on the loss. A higher clip value results in lower loss, as indicated by its negative correlation. Additionally, both ADAM and SGD optimizers show a strong correlation with loss, but ADAM has a positive correlation, making it the preferable choice. The importance metric for clip is considerably higher than other hyperparameters, suggesting that if the clip value is too low, the loss will remain high regardless of other parameter values, as evident from the diagram.



Variation of loss with iterations

There were 4000 iterations for each run!



Q1.6:

Solution:

Measurement of time and memory Consumption of model's operators

This is my profiler.txt file:

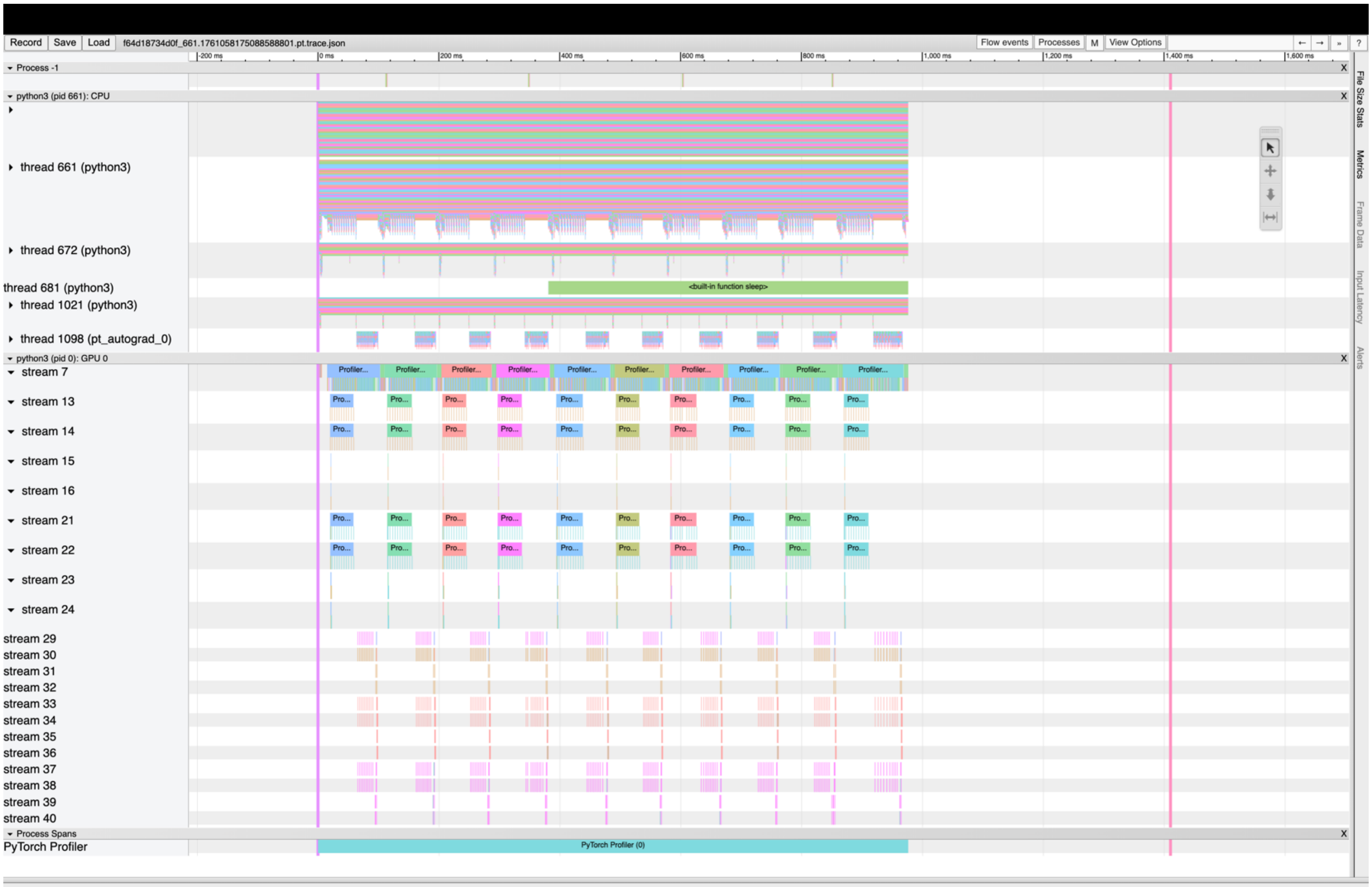
Name	Self CPU	CPU total	CPU time avg	CUDA total	CUDA time avg	CPU Mem	Self CPU Mem	CUDA Mem	Self CUDA Mem
aten::empty	523.000us	523.000us	6.226us	0.000us	0.000us	24 b	24 b	335.39 Mb	335.39 Mb
aten::embedding	4.636ms	6.319ms	574.455us	56.000us	5.091us	0 b	0 b	24.00 Kb	0 b
aten::reshape	27.000us	41.000us	3.727us	0.000us	0.000us	0 b	0 b	0 b	0 b
aten::view	31.000us	31.000us	1.292us	0.000us	0.000us	0 b	0 b	0 b	0 b
aten::index_select	1.028ms	1.631ms	148.273us	56.000us	5.091us	0 b	0 b	24.00 Kb	0 b
aten::resize_	99.000us	99.000us	9.000us	0.000us	0.000us	0 b	0 b	24.00 Kb	24.00 Kb
cudaLaunchKernel	33.995ms	33.995ms	157.384us	209.000us	0.968us	0 b	0 b	0 b	0 b
ous namespace)::indexSelectS...	0.000us	0.000us	0.000us	56.000us	5.091us	0 b	0 b	0 b	0 b
aten::to	3.000us	3.000us	3.000us	0.000us	0.000us	0 b	0 b	0 b	0 b
aten::_pack_padded_sequence	51.000us	964.000us	964.000us	3.000us	3.000us	16 b	0 b	4.00 Kb	0 b
aten::slice	201.000us	210.000us	16.154us	0.000us	0.000us	0 b	0 b	0 b	0 b
aten::as_strided	69.000us	69.000us	0.476us	0.000us	0.000us	0 b	0 b	0 b	0 b
aten::cat	1.278ms	1.735ms	55.968us	133.000us	4.290us	0 b	0 b	54.00 Kb	54.00 Kb
aten::narrow	7.000us	16.000us	16.000us	0.000us	0.000us	0 b	0 b	0 b	0 b
cudaMemcpyAsync	49.000us	49.000us	24.500us	0.000us	0.000us	0 b	0 b	0 b	0 b
Memcpy DtoD (Device -> Device)	0.000us	0.000us	0.000us	6.000us	3.000us	0 b	0 b	0 b	0 b
aten::select	101.000us	113.000us	5.136us	0.000us	0.000us	0 b	0 b	0 b	0 b
aten::item	8.000us	10.000us	5.000us	0.000us	0.000us	0 b	0 b	0 b	0 b
aten::_local_scalar_dense	4.000us	4.000us	2.000us	0.000us	0.000us	0 b	0 b	0 b	0 b
aten::zeros	27.000us	1.613ms	537.667us	2.000us	0.667us	0 b	0 b	8.00 Kb	0 b

Q1.7:

Solution:

PyTorch Profiler Tracing

The file is named as trace.json.



Q1.8:

Solution: Answered in the chatbot.ipynb file.

Problem 2: TorchScript Seq-2-Seq Model

Q2.1. Explain the differences between tracing and scripting and how they are used in TorchScript?

Solution:

Tracing: In tracing, the function takes both the model and sample input data to record computations and construct a graph-based function. However, it only captures the computations performed for the given input, making it incapable of handling data-dependent control flow.

Scripting: In scripting, only the model is required, without the need for sample data. This approach converts the model code into TorchScript, a subset of Python that includes all control flows.

For models without control flow dependencies, torch.jit.trace() can be used without modifying the model, as it directly converts it into TorchScript. However, for scripting, the model code may need adjustments to conform to TorchScript syntax before using torch.jit.script(). When using tracing, it is necessary to set the model's device and dropout layers to test mode before tracing, as the traced model does not inherently handle these operations. In contrast, scripting allows setting the device and dropout layers to test mode just before inference, similar to how it's done in eager mode.

Q2.2: Explain the changes needed in the chatbot model to allow for scripting.

Solution:

In the given model, there are three sub-modules: Encoder, Decoder, and GreedySearchDecoder. The third sub-module, GreedySearchDecoder, requires scripting due to its input-based control flow.

Changes required in GreedySearchDecoder:

1. Passing decoder_n_layers as a Constructor Argument: Earlier, it was using fetching this value from decoder but since we are using traced version of decoder we will not be able to access that value anymore and hence we need to pass this to its constructor for it to use.

   ```
   #Modified GreedySearchDecoder for scripting the module

   class GreedySearchDecoderScript(torch.jit.ScriptModule):
       def __init__(self, encoder, decoder, decoder_n_layers):
   ```

2. Explicit Type Annotations for Forward Method Arguments: By default, TorchScript assume all parameters of function as tensor. Hence, in case we need to pass any argument with different type like int in our case, we need to specify the type in python function.

3. Adding More Attributes to Handle Global Variables: Earlier, we were accessing various variable available in global scope of our python environment, but TorchScript version do not have access to those variables, and we need to save value of those variables from global scope to attributes of the model class. (_SOS_token, _device, _decoder_n_layers)

   ```
   self._device = device
   self._SOS_token = SOS_token
   self._decoder_n_layers = decoder_n_layers
   ```

Q2.3: Convert the model that you trained in the previous exercise to Torchscript.

Solution: Answered in the Jupyter Notebook.

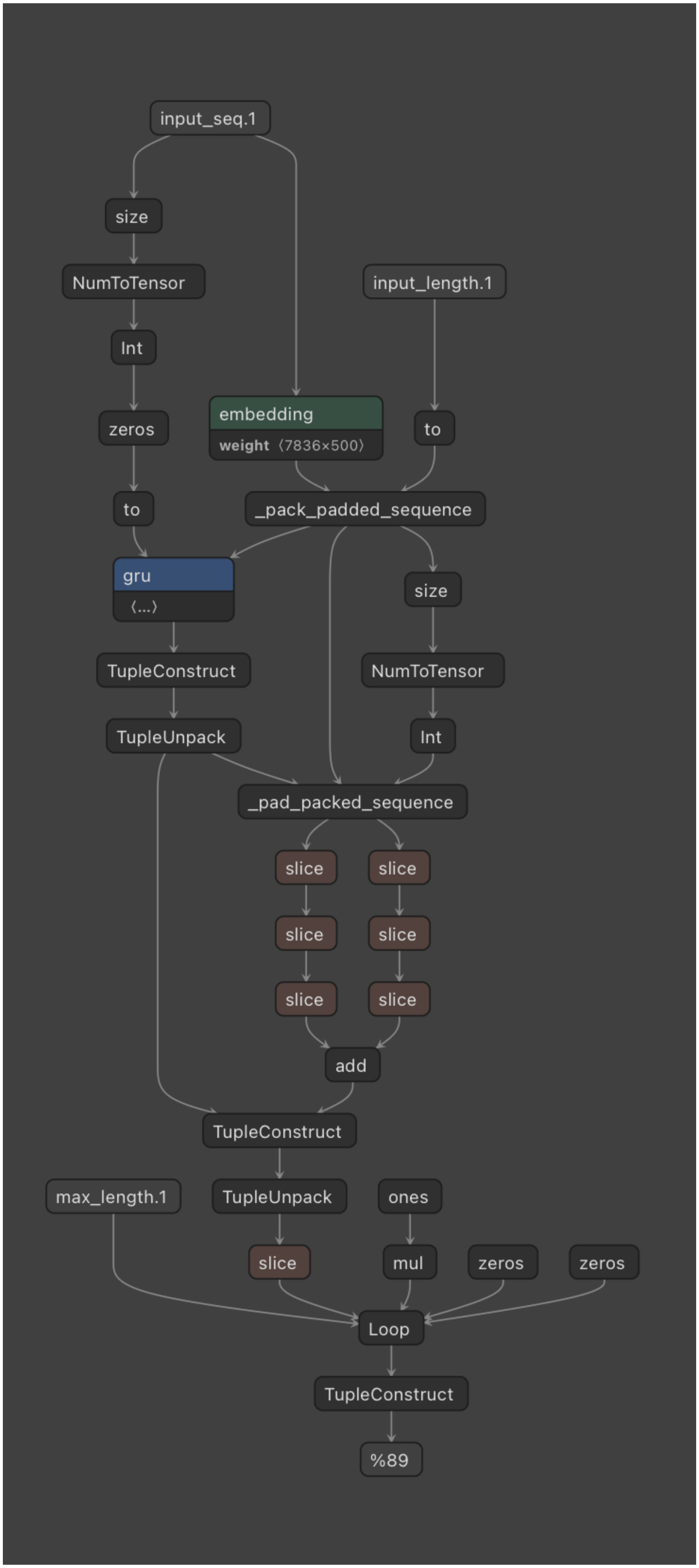
Q2.4: Print graph of the converted model.

Solution:

```
graph(%self.1 : __torch__.GreedySearchDecoderScript,
      %input_seq.1 : Tensor,
      %input_length.1 : Tensor,
      %max_length.1 : int):
  %62 : bool = prim::Constant[value=0]()
  %49 : bool = prim::Constant[value=1]() # /tmp/ipython-input-2626984651.py:26:8
  %115 : Device = prim::Constant[value="cuda"]()
  %15 : NoneType = prim::Constant() # :0:0
  %14 : int = prim::Constant[value=0]() # /tmp/ipython-input-2626984651.py:23:34
  %16 : int = prim::Constant[value=2]() # /tmp/ipython-input-2626984651.py:19:41
  %19 : int = prim::Constant[value=1]() # /tmp/ipython-input-2626984651.py:21:35
  %21 : int = prim::Constant[value=4]() # /tmp/ipython-input-2626984651.py:21:68
```

```
  %encoder.1 : __torch__.EncoderRNN = prim::GetAttr[name="encoder"](%self.1)
  %9 : (Tensor, Tensor) = prim::CallMethod[name="forward"](%encoder.1, %input_seq.1, 
%input_length.1) # /tmp/ipython-input-2626984651.py:17:42
  %encoder_outputs.1 : Tensor, %encoder_hidden.1 : Tensor = prim::TupleUnpack(%9)
  %decoder_hidden.1 : Tensor = aten::slice(%encoder_hidden.1, %14, %15, %16, %19) # 
/tmp/ipython-input-2626984651.py:19:25
  %20 : int[] = prim::ListConstruct(%19, %19)
  %26 : Tensor = aten::ones(%20, %21, %15, %115, %15) # /tmp/ipython-input-
2626984651.py:21:24
  %decoder_input.1 : Tensor = aten::mul(%26, %19) # /tmp/ipython-input-
2626984651.py:21:24
  %29 : int[] = prim::ListConstruct(%14)
  %all_tokens.1 : Tensor = aten::zeros(%29, %21, %15, %115, %15) # /tmp/ipython-
input-2626984651.py:23:21
  %35 : int[] = prim::ListConstruct(%14)
  %all_scores.1 : Tensor = aten::zeros(%35, %15, %15, %115, %15) # /tmp/ipython-
input-2626984651.py:24:21
  %all_tokens0 : Tensor, %all_scores0 : Tensor, %decoder_hidden0 : Tensor, 
%decoder_input0 : Tensor = prim::Loop(%max_length.1, %49, %all_tokens.1, 
%all_scores.1, %decoder_hidden.1, %decoder_input.1) # /tmp/ipython-input-
2626984651.py:26:8
    block0(%50 : int, %all_tokens0.7 : Tensor, %all_scores0.7 : Tensor, 
%decoder_hidden0.5 : Tensor, %decoder_input0.5 : Tensor):
      %decoder.1 : __torch__.LuongAttnDecoderRNN = 
prim::GetAttr[name="decoder"](%self.1)
      %57 : (Tensor, Tensor) = prim::CallMethod[name="forward"](%decoder.1, 
%decoder_input0.5, %decoder_hidden0.5, %encoder_outputs.1) # /tmp/ipython-input-
2626984651.py:28:45
      %decoder_output.1 : Tensor, %decoder_hidden1.1 : Tensor = 
prim::TupleUnpack(%57)
      %decoder_scores.1 : Tensor, %decoder_input1.1 : Tensor = 
aten::max(%decoder_output.1, %19, %62) # /tmp/ipython-input-2626984651.py:30:44
      %68 : Tensor[] = prim::ListConstruct(%all_tokens0.7, %decoder_input1.1)
      %all_tokens1.1 : Tensor = aten::cat(%68, %14) # /tmp/ipython-input-
2626984651.py:32:25
      %73 : Tensor[] = prim::ListConstruct(%all_scores0.7, %decoder_scores.1)
      %all_scores1.1 : Tensor = aten::cat(%73, %14) # /tmp/ipython-input-
2626984651.py:33:25
      %decoder_input2.1 : Tensor = aten::unsqueeze(%decoder_input1.1, %14) # 
/tmp/ipython-input-2626984651.py:35:28
      -> (%49, %all_tokens1.1, %all_scores1.1, %decoder_hidden1.1, %decoder_input2.1)
  %89 : (Tensor, Tensor) = prim::TupleConstruct(%all_tokens0, %all_scores0)
  return (%89)
```

The following graph has been made using netron.app website.


input_seq.1
size
NumToTensor
input_length.1
Int
embedding
weight ⟨7836×500⟩
zeros
to
to
_pack_padded_sequence
gru
⟨...⟩
size
TupleConstruct
NumToTensor
TupleUnpack
Int
_pad_packed_sequence
slice
slice
slice
slice
slice
slice
add
TupleConstruct
max_length.1
TupleUnpack
ones
slice
mul
zeros
zeros
Loop
TupleConstruct
%89

Q2.5: Evaluate the Torchscript model.

Solution:

Code:

```
import torch

# Example inputs for the TorchScript model
input_seq = torch.randint(0, 5000, (10, 1)).cuda()
input_length = torch.tensor([10]).cpu()
max_length = 20

# Load the TorchScript model
torchscript_model = torch.jit.load("/content/gpu.pt")

# Ensure the model is on the correct device
torchscript_model = torchscript_model.cuda()

# Run inference
with torch.no_grad():
    tokens, scores = torchscript_model(input_seq, input_length, max_length)

print("Generated tokens:", tokens)
print("Token scores:", scores)
```

Output:

```
/usr/local/lib/python3.12/dist-packages/torch/nn/modules/module.py:1784: UserWarning: RNN module weights are not part of single contiguous chunk of memory. This means they need to be compacted at every call, possibly greatly increasing memory usage. To compact weights again call flatten_parameters(). (Triggered internally at /pytorch/aten/src/ATen/native/cudnn/RNN.cpp:1479.)
  return forward_call(*args, **kwargs)
Generated tokens: tensor([6456, 3957, 3957, 7558, 7558, 2856,  265, 3950, 3950, 7635, 2171, 3957,
        3957, 3957, 7558, 7558, 2153,  265,  265, 2466], device='cuda:0')
Token scores: tensor([0.0002, 0.0002, 0.0002, 0.0002, 0.0002, 0.0002, 0.0002, 0.0002, 0.0002,
        0.0002, 0.0002, 0.0002, 0.0002, 0.0002, 0.0002, 0.0002, 0.0002, 0.0002,
        0.0002, 0.0002], device='cuda:0')
```

Q2.6: Comparing Latency (displayed as latency table)

Solution:

	Latency on CPU (ms)	Latency on GPU (ms)
Pytorch	323.401267	10.454443
Torchscript	35.967107	12.759383
SpeedUp	8.991584	0.819353

Q2.7: Save and serialize it for use in a non-Python deployment environment.
Solution: Answered in the Jupyter Notebook.

Q2.8: Show how to use the model in a non-Python environment. For example in a C++ program.

Solution: Answered in the attached Readme.md file.