Aman Chopra
ac5140

Written Questions:

1. Tracing and scripting are two techniques for converting a dynamic computation graph that is used by PyTorch's eager execution mode into a static computation graph that can be optimized and executed more efficiently. Tracing involves running a model with sample inputs and recording the operations that occur during the inference. These operations are used to create a static computation graph. In other words, we are "tracing" the execution of the model with sample inputs to capture its behavior. Tracing is good for models that don't have any complex control flow or dynamic behavior. For example, if a PyTorch tensor was passed into a pandas dataframe and some computations were performed, tracing wouldn't be able to capture this behavior. Scripting, on the other hand, is much more flexible and is the preferred choice for more complex models that have dynamic behavior. Here, we write a python script that specifies the model's operations using TorchScript's subset of the Python language. It allows for explicitly defining how the model should be converted into a static graph. To use tracing in Pytorch, we define the model as usual and then create a trace by calling **torch.jit.trace** and passing in the model and sample input data; this returns a traced model that is a TorchScript module that we can use for inference or serialization. To use scripting, we define the model as usual and then annotate the **forward** function of the model with the **@torch.jit.script** decorator or call **torch.jit.script** and pass in the model; similarly, this returns a TorchScript module that we can use for inference or serialization.

2. There are no changes needed to the encoder and decoder models, since their **forward** functions don't contain any data-dependent control flows. For these models, we simply use tracing to convert them to TorchScript modules. For the Greedy Search Decoder, there is more dynamic behavior with the need to iteratively run forward passes through the decoder. Because of this behavior, we use scripting to convert the model to a TorchScript module. In the GreedySearchDecoder class, we make the following changes:
    a. Add the **decoder_n_layers** as a parameter to the constructor. This is needed because the encoder and decoder models that are passed into the constructor won't be **nn.Module** instances after they are converted to TorchScript. Instead they will be **TracedModule** instances, which means the decoder won't be able to access **n_layers** by calling **decoder.n_layers.**
    b. Add constants for the new attributes that can't be accessed through global variables outside the scope of this function, since the models are no longer Python objects. Making this constant list allows using these attributes as literals when building the graph after a **forward** call. We add the line **__constants__ = ['device', '_SOS_token', '_decoder_n_layers']**.
    c. Add types to the parameters of the **forward** method. TorchScript function parameters are Tensor types by default. Since, we are adding a parameter for **max_length**, we need to specify that it is an **int**, so it's not assumed to be a

Tensor. Likewise, even though the other parameters **input_seq** and **input_length** are Tensors, it is good practice to explicitly define their types in the method signature.

d. When scripting, we can't initialize tensors literally using **torch.LongTensor([[SOS_token]])** like we did in the originally GreedySearchDecoder class. Instead, we need to use a torch function like **torch.ones** to initialize the tensor, by multiplying it by the SOS_token stored in the **self._SOS_token** constant.

Before tracing the encoder and decoder models, we need to transfer them to the device using **.to** and set the dropout layers to test using **.eval**. After this, we can wrap the models like doing **traced_encoder = torch.jit.trace(encoder, (test_seq, test_seq_length))** to convert the model to TorchScript. For the decoder, we use the same as above. For the GreedySearchDecoder, we use **scripted_searcher = torch.jit.script(GreedySearchDecoder(traced_encoder, traced_decoder, decoder.n_layers))**.

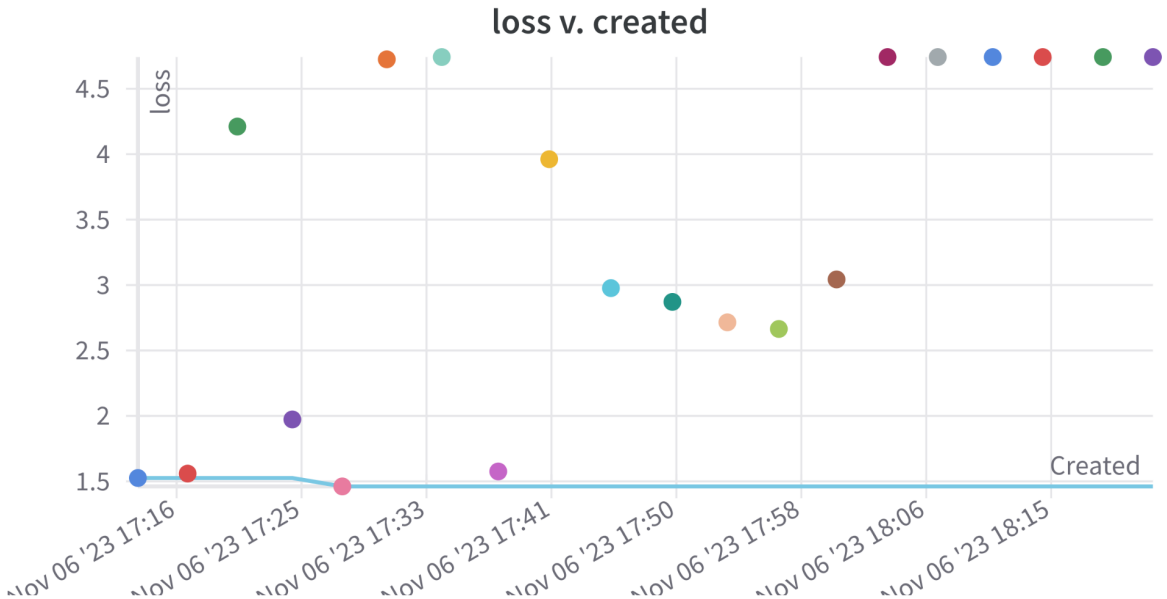
3.

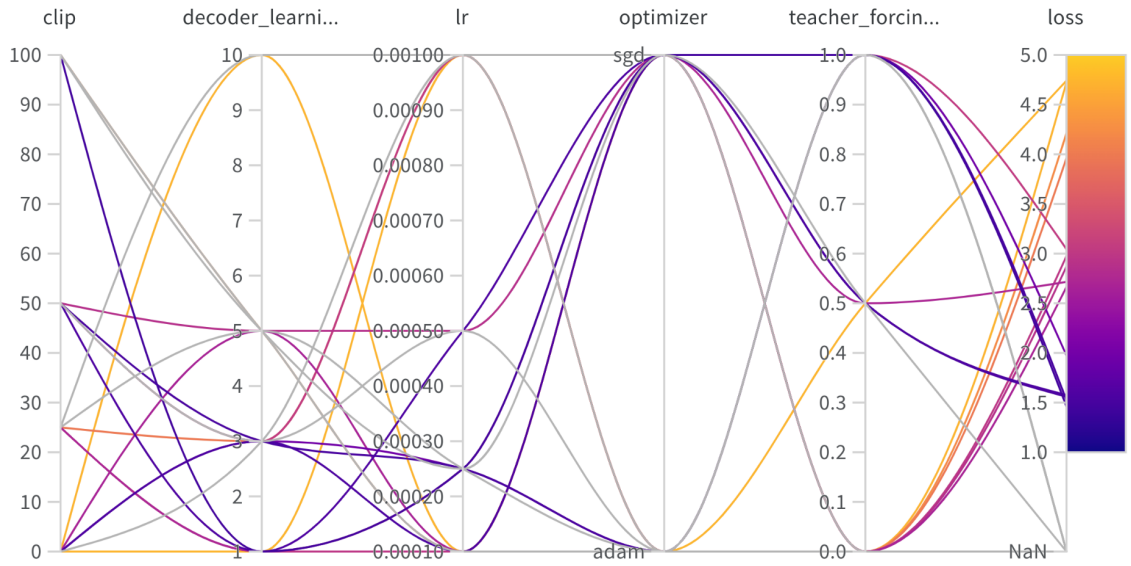
	Latency on CPU (ms)	Latency on GPU (ms)
PyTorch	41.021882220020416	45.37607797994497
TorchScript	41.78662137998799	38.386942050046855

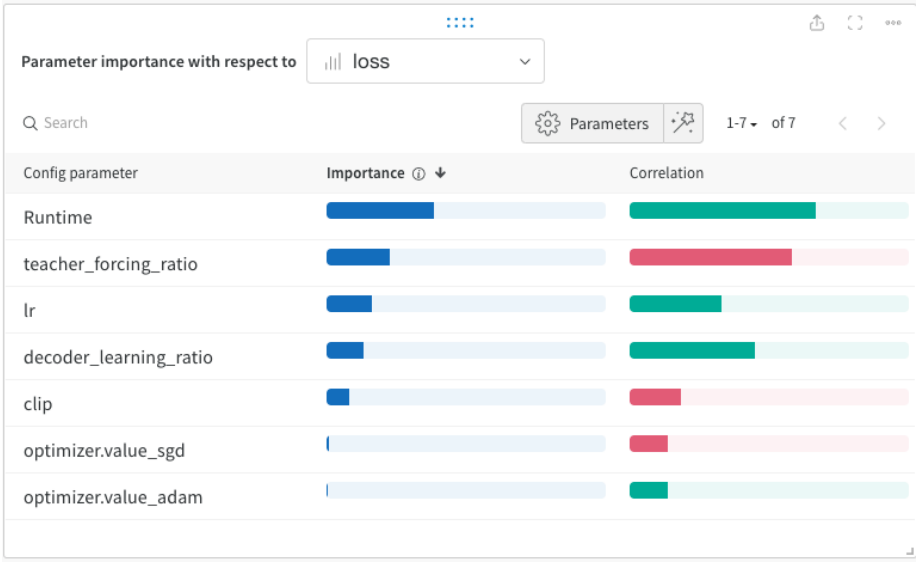
The TorchScripted model seems to perform even worse on CPU. However, on GPU, it offers a ~15.4% speedup.

W&B Project URL:
https://wandb.ai/amanichopra/chatbot-sweep?workspace=user-amanichopra

W&B Charts:


loss v. created
loss
4.5
4
3.5
3
2.5
2
1.5
Created
Nov 06 '23 17:16
Nov 06 '23 17:25
Nov 06 '23 17:33
Nov 06 '23 17:41
Nov 06 '23 17:50
Nov 06 '23 17:58
Nov 06 '23 18:06
Nov 06 '23 18:15


clip
decoder_learni...
lr
optimizer
teacher_forcin...
loss
sgd
adam
NaN



Chrome Tracing: