# TorchScript

TorchScript serializes PyTorch models into programs that can be executed in non-Python processes. This is especially advantageous in production environments where Python may not be the most performant choice.

Transformers can export a model to TorchScript by:

1. creating dummy inputs to create a *trace* of the model to serialize to TorchScript
2. enabling the `torchscript` parameter in either `~PretrainedConfig.torchscript` for a randomly initialized model or from_pretrained() for a pretrained model

## Dummy inputs

The dummy inputs are used in the forward pass, and as the input values are propagated through each layer, PyTorch tracks the different operations executed on each tensor. The recorded operations are used to create the model trace. Once it is recorded, it is serialized into a TorchScript program.

```python
from transformers import BertModel, BertTokenizer, BertConfig
import torch

tokenizer = BertTokenizer.from_pretrained("google-bert/bert-base-uncased")
text = "[CLS] Who was Jim Henson ? [SEP] Jim Henson was a puppeteer [SEP]"
tokenized_text = tokenizer.tokenize(text)

masked_index = 8
tokenized_text[masked_index] = "[MASK]"
indexed_tokens = tokenizer.convert_tokens_to_ids(tokenized_text)
segments_ids = [0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1]

# creating a dummy input
tokens_tensor = torch.tensor([indexed_tokens])
```

```
segments_tensors = torch.tensor([segments_ids])
dummy_input = [tokens_tensor, segments_tensors]
```

The trace is created based on the provided inputs dimensions and it can only handle inputs with the same shape as the provided input during tracing. An input with a different size raises the error message shown below.

```
`The expanded size of the tensor (3) must match the existing size (7) at non-singleton
```

Try to create a trace with a dummy input size at least as large as the largest expected input during inference. Padding can help fill missing values for larger inputs. It may be slower though since a larger input size requires more calculations. Be mindful of the total number of operations performed on each input and track the model performance when exporting models with variable sequence lengths.

## Tied weights

Weights between the `Embedding` and `Decoding` layers are tied in Transformers and TorchScript can't export models with tied weights. Instantiating a model with `torchscript=True`, separates the `Embedding` and `Decoding` layers and they aren't trained any further because it would throw the two layers out of sync which can lead to unexpected results.

Models *without* a language model head don't have tied weights and can be safely exported without the `torchscript` parameter.

randomly initialized model pretrained model

```
config = BertConfig(
    vocab_size_or_config_json_file=32000,
    hidden_size=768,
    num_hidden_layers=12,
    num_attention_heads=12,
    intermediate_size=3072,
    torchscript=True,
)

model = BertModel(config)
model.eval()
```

## Export to TorchScript

Create the Torchscript program with torch.jit.trace, and save with torch.jit.save.

```
traced_model = torch.jit.trace(model, [tokens_tensor, segments_tensors])
torch.jit.save(traced_model, "traced_bert.pt")
```

Use torch.jit.load to load the traced model.

```
loaded_model = torch.jit.load("traced_bert.pt")
loaded_model.eval()

all_encoder_layers, pooled_output = loaded_model(*dummy_input)
```

To use the traced model for inference, use the `__call__` dunder method.

```
traced_model(tokens_tensor, segments_tensors)
```

## Deploy to AWS

TorchScript programs serialized from Transformers can be deployed on Amazon EC2 Inf1 instances. The instance is powered by AWS Inferentia chips, a custom hardware accelerator designed for deep learning inference workloads. AWS Neuron supports tracing Transformers models for deployment on Inf1 instances.

> AWS Neuron requires a Neuron SDK environment which is preconfigured on AWS DLAMI.

Instead of torch.jit.trace, use torch.neuron.trace to trace a model and optimize it for Inf1 instances.

```
import torch.neuron

torch.neuron.trace(model, [tokens_tensor, segments_tensors])
```

Refer to the AWS Neuron documentation for more information.

## Model architectures

BERT-based models - like DistilBERT or RoBERTa - run best on Inf1 instances for non-generative tasks such as extractive question answering, and sequence or token classification.

Text generation can be adapted to run on an Inf1 instance as shown in the Transformers MarianMT tutorial.

Refer to the Inference Samples/Tutorials (Inf1) guide for more information about which models can be converted out of the box to run on Inf1 instances.

&lt;/&gt; Update on GitHub

← ExecuTorch Text classification →