MERITインターンシップ報告書

2023年10月4日
理学系研究科・物理学専攻・小形研究室・博士1年
MERIT-WINGS11期生
開田 亮佑

実施概要

受け入れ先　株式会社Preferred Networks
実施期間　2023年8月10日～2023年9月27日
テーマ　深層学習モデルを社会実装するためのフレームワーク・ライブラリ開発

背景

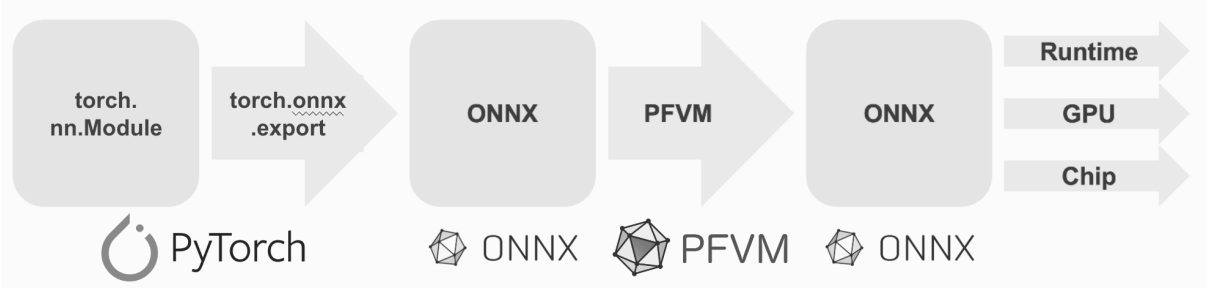

図*1.* 典型的な深層学習の推論ワークロード

図1は、典型的な深層学習の推論ワークロードの概要を模式的に表したものです。ワークロードでは、PyTorchなどのフレームワークで構築されたモデルをランタイムやチップ上で実行しますが、その前に、ONNXといった統一的なフォーマットに変換し、最適化する処理がよく行われます。
PFNではPFVMと呼ばれる深層学習の計算グラフ向けの最適化コンパイラ・ランタイムを内製しており、その入出力にはONNXと呼ばれるモデルフォーマットが用いられています。
DL Ecosystemチームでは、こうした最適化フレームワークの開発と活用を手がけており、ONNXのエクスポートにかかるコストを削減するために、より柔軟なONNXのエクスポータが求められていました。

既存のエクスポータの問題点

既存のONNXエクスポータとして、PyTorchで開発されているtorch.onnx.exportが挙げられます。しかし、torch.onnx.exportには以下の問題点がありました。

1. 深層学習モデルの計算グラフを取得するために、モデルを実際に実行する必要がある。多くの場合エクスポートはCPUで実行する必要があり、計算グラフの取得に時間がかかる
2. エクスポータの開発にpybind11経由で使えるtorch.jitのC++ APIが多用されている。そのため、拡張やデバッグが容易ではない

これらの課題を解決するために、新しいエクスポータの開発では以下の工夫を行いました。

1. PyTorchの新しいグラフ取得メソッド（torch.fx.symbolic_trace/torch.compile）を活用する
2. Pythonで実装されたPyTorchやONNXのエコシステム（特にtorch.fx.GraphとONNX Script）を活用する

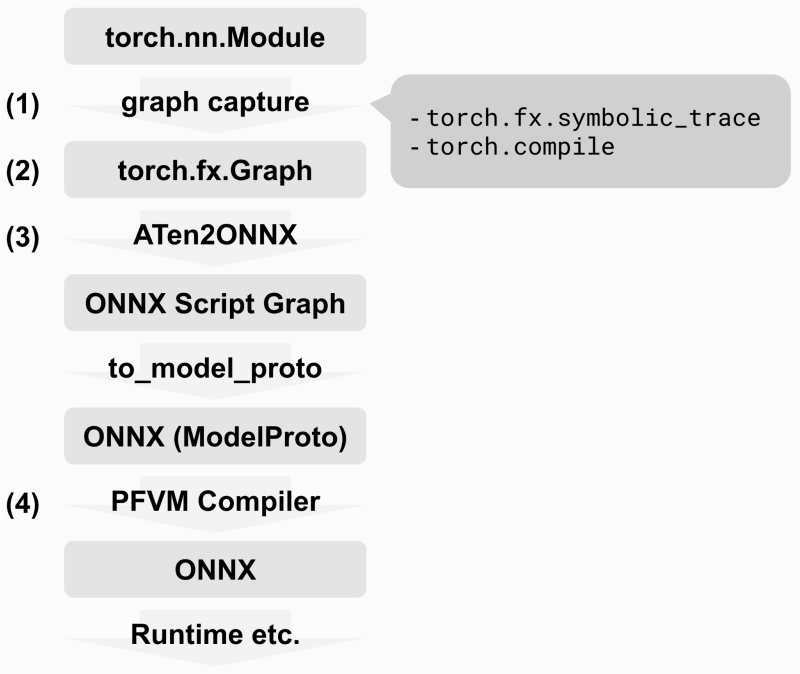

図2. エクスポータの内部で行われる処理

図2は、エクスポータの内部で行われる処理を模式的に表したものです。PyTorchで構築したtorch.nn.Moduleから計算グラフをtorch.fx.Graphとして取得します。その後ATenの関数（後述）をONNXのオペレータに変換する処理を経て、ONNXのモデルが構築されます。構築されたONNXのモデルは、PFVMといったフレームワークで最適化されたのち、ランタイムまたは実機上で実行されます。この流れのうち(1)から(4)の点について開発を行いました。

開発したエクスポータの評価

1. エクスポート可能なモデル

新たに開発したエクスポータを用いることで、MNISTやtorchvisionのResNetの推論用および学習用のONNXモデルを出力することができました。また、比較的実践的なモデルとして、Googleが開発した言語モデルであるT5の出力（推論のみ）も達成しました。ただし、T5の出力はHugging Faceがサポートするカスタムfx.Tracerを用いて初めて達成することができました。symbolic_traceを用いてより広範囲のモデルを出力可能とするには、ONNXのエクスポータだけではなく、計算グラフの取得にも修正を加える必要があると考えられます。

2. エクスポートに要する時間

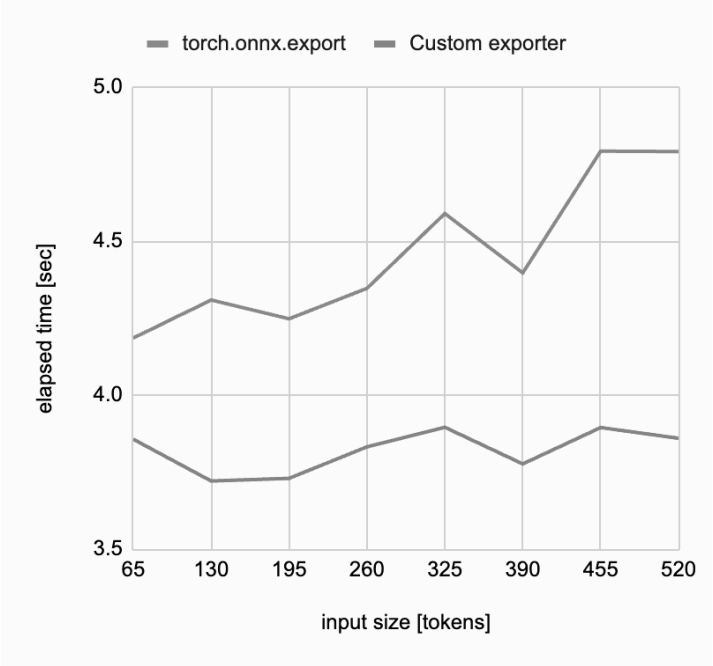

図*3.* モデルの出力にかかる時間のサンプル入力サイズ依存性。*(*青色*)*既存の*ONNX*エクスポータである*torch.onnx.export*の場合。*(*赤色*)*新たに開発したエクスポータの場合。このプロファイルは*Google*が開発した言語モデルである*T5のHugging faceのtransformers実装*を対象に取得されたものである。サンプル入力はエクスポータに渡す入力例を表し、入力サイズはトークナイズされたテキストのトークン数を表す

図3は、言語モデルT5を用いて、サンプル入力サイズに対しモデルの出力にかかる時間をプロットしたグラフです。既存のONNXエクスポータであるtorch.onnx.exportでは、計算グラフの取得にモデルの実行を要するため、エクスポートにかかる時間が入力サイズに応じて増大しています。一方、新たに開発したエクスポータでは、モデルを直接実行しないため、入力サイズに依存しない時間でモデルを出力でき、エクスポートにかかる時間も、既存の実装に比べ1割強程度短縮されていることがわかります。

3. エクスポートに要する時間における、各メソッドが占める割合

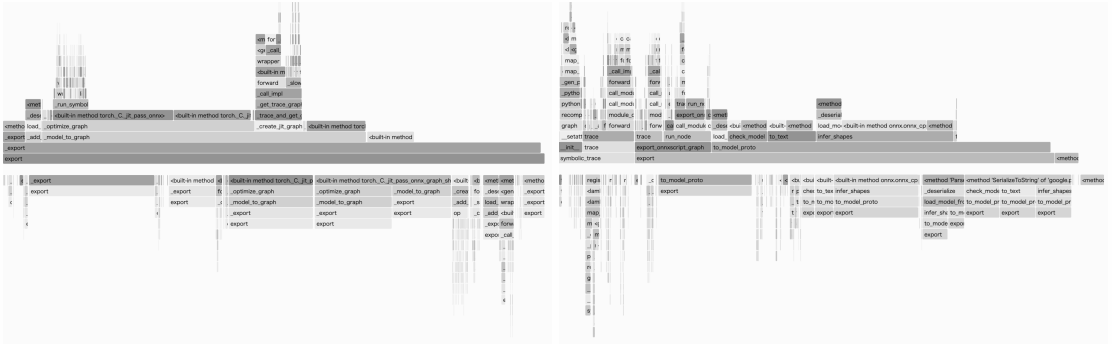

図*4.* エクスポートに要する時間における、各メソッドが占める割合。*(*左*)*既存の*ONNX*エクスポータである*torch.onnx.export*の場合。*(*右*)*新たに開発したエクスポータの場合。

図4は、再びT5を用いて、ONNXのエクスポートにおけるフレームグラフを取得したものです。既存のエクスポータであるtorch.onnx.export、および新たに開発したエクスポータの両方で、計算グラフの取得に要する時間は全体の時間の2割程度に留まっています。これは、計算グラフの取得だけでなく、テンソルの形状推論といった、モデルのノード数に比例する処理も律速するためだと考えられます（ただし、より大きなモデルでは事情は変わるかもしれません）。したがって、エクスポート全体を高速化するには、計算グラフの取得だけでなく、これらの処理も高速化する必要があると結論できます。

謝辞

本インターンシップに際しお世話になりました株式会社Preferred Networksの皆さま、特にお忙しい中親身に議論・相談いただいたDL Ecosystemチームの皆さまに心よりお礼申し上げます。
最後に、本インターンシップへの参加をご快諾・ご支援いただいきました指導教員の小形正男教授、副指導教員の求幸年教授、並びにMERIT-WINGSの皆さまに感謝申し上げます。