3-2　自動並列化深層学習ミドルウェア RaNNC

3-2　RaNNC: Automatic Parallelization Middleware for Deep Learning

田仲 正弘
TANAKA Masahiro

深層学習において、ニューラルネットを大規模化をすることで、劇的に学習性能を向上できることが知られている。大規模ニューラルネットの学習には、多数の GPU を用いた並列処理が必須となるが、既存のフレームワークを用いる場合、ニューラルネット定義を並列化のために大幅に書き換える必要があり、専門家にとっても容易ではない。また、著名な既存フレームワークは、Transformer と呼ばれる種類のニューラルネットにしか適用できない。こうした課題を解決するため、NICT データ駆動知能システム研究センター（DIRECT）では、自動並列化深層学習ミドルウェア RaNNC を開発してきた。RaNNC は、計算時間や必要メモリ量を自動的に分析し、ニューラルネットを自動的に分割して多数の GPU に配置し、並列で学習を行う。また、既存フレームワークと異なり、適用できるニューラルネットの種類に制限がない点でも優れている。

In deep learning, scaling up of neural networks improves the quality of learning results. Training of a large-scale neural network needs to be parallelized using many GPUs. Existing software frameworks for training large-scale neural networks require users to rewrite the definition of a neural network significantly for parallelization. The rewriting is hard for even experts in parallel processing and often takes several months. In addition, some well-known frameworks can train only a specific type of neural networks called Transformer. To solve the issues, the Data-driven Intelligent System Research Center (DIRECT), NICT, has been developing an automatic parallelization middleware, RaNNC. RaNNC automatically partitions a neural network by analyzing the computation time and memory usage and computes it on multiple GPUs in parallel. Unlike the existing frameworks, RaNNC can also train any type of neural networks.

1　はじめに

近年の人工知能は、深層学習の発展によって目覚ましい進歩を遂げている。深層学習では、ニューラルネットと呼ばれるモデルを用いるが、近年の深層学習の研究で、ニューラルネットを大規模化する、すなわちニューラルネットの持つ学習パラメータの数を増加することによって、劇的に学習性能が向上することが広く知られるようになった[1]。例えば、2018 年に発表され、言語処理分野でのブレークスルーとなった BERT [2] は、3.4 億個のパラメータを持ち、発表当時としては最大規模のニューラルネットとされた。しかし、そのわずか 2 年後には、BERT の約 500 倍である 1,750 億個ものパラメータを持つ GPT-3 [3]が発表されている。

ニューラルネットの学習には膨大な計算が必要となるため、GPU（Graphics Processing Unit）を初めとする、計算を高速化させるための装置が広く用いられる。ニューラルネットの学習パラメータ数を増加すると、必要な計算量が飛躍的に増加するため、大規模ニューラルネットの学習を現実的な時間で完了させるには、GPU 等の計算装置を多数用いて並列に計算をすることが必須となる。

従来から、大規模ニューラルネット学習を目的として、Megatron-LM [4]（NVIDIA）や MeshTensorFlow [5]（Google）といった深層学習のための並列計算を行うソフトウェアフレームワークが開発されてきた。しかし、これらのフレームワークを用いる場合には、専門的な知識を持つ技術者が、ニューラルネット定義のプログラムを書き換えて、学習パラメータやその計算を、どのように分割し、どの GPU に割り当てるかを指定する必要がある。最終的な学習速度は、この分割や割り当てによって大きく変わってくるが、こうしたプログラムの書き換えには大きな手間がかかる上に、どの

ように書き換えると高速に学習できるかは、対象のニューラルネットや計算機環境、各種の設定によって異なる。そのため、たとえ高度な知識を持つ専門家であっても、最終的に効率的な計算を行える分割や割り当てを決定するまでには、多くの試行錯誤が必要となる。また、Megatron-LM 等の著名な既存ソフトウェアは、GPT-3 や BERT など、Transformer [6] と呼ばれる特定のニューラルネットの種類にしか適用できないという制限がある。

著者らが開発した自動並列化深層学習ミドルウェア RaNNC（Rapid Neural Network Connector）は、ニューラルネットの分割や、GPU への計算の割り当てをほぼ自動化することで、大規模ニューラルネットの学習を劇的に簡単化するものである。RaNNC は、深層学習ソフトウェアのデファクトスタンダードである PyTorch [7] を用いて記述されたニューラルネット定義を受け取り、そこに定義された計算の種類、GPU の処理速度、利用可能なメモリ量などを考慮して、自動的に試行錯誤を繰り返しながら、学習パラメータや計算処理を分割し、複数の GPU への割り当てを決定する。また、上述の既存ソフトウェアと異なり、適用できるニューラルネットの種類に、基本的に制限がない。

本章では、RaNNC 開発の背景となった、大規模ニューラルネット学習の課題について概観した後、RaNNC の特徴や内部の機構について説明する。また、学習できるニューラルネットの規模や計算速度について、既存ソフトウェアとの比較実験の結果を示す。

2　大規模ニューラルネット学習の課題

前述のように、大規模ニューラルネットの学習には膨大な計算が必要となるため、多数の GPU を使った並列処理が行われる。こうした並列化には、一般的にデータ並列とモデル並列の 2 種類がある（図 1）。

PyTorch を含む多くの深層学習ソフトウェアは、学習データを分割して、複数の GPU に割り当てるデータ並列のみをサポートしている。データ並列では、各々の GPU のメモリに、ニューラルネットの学習パラメータ全体をコピーし、それぞれの GPU では異なる入力（訓練例）を与えて計算したあと、その結果を集約する。データ並列の利点は、ニューラルネットの種類を問わずに適用可能であることである。一方で、極めて多くの学習パラメータを持つニューラルネットは、全ての学習パラメータが 1 台の GPU のメモリに格納できないため、学習できない。

モデル並列では、ニューラルネットを分割し、分割で得られたより小さなニューラルネット（部分ニューラルネットと呼ぶ）を各々の GPU に割り当てる。各 GPU は、分割された部分ニューラルネットに関係する学習パラメータのみをメモリに記憶すれば良いため、学習パラメータの多い巨大ニューラルネットの学習も可能になる。その反面、モデル並列では、ニューラルネットの構造や、計算機の特性を考慮して、個別にチューニングが必要となる。具体的には、モデル並列のために決定するべき要素として、以下が挙げられる。

(S1) ニューラルネットをどのように複数の部分ニューラルネットに分割するか

(S2) 得られた部分ニューラルネットを、何台の GPU にコピーするか

（データ並列の併用）

(S3) 各部分ニューラルネット（コピーを含む）をどの GPU に配置するか

これらの決定は相互に依存しており、一見すると各 GPU の分担が不均等で、低い処理速度しか得られなさ

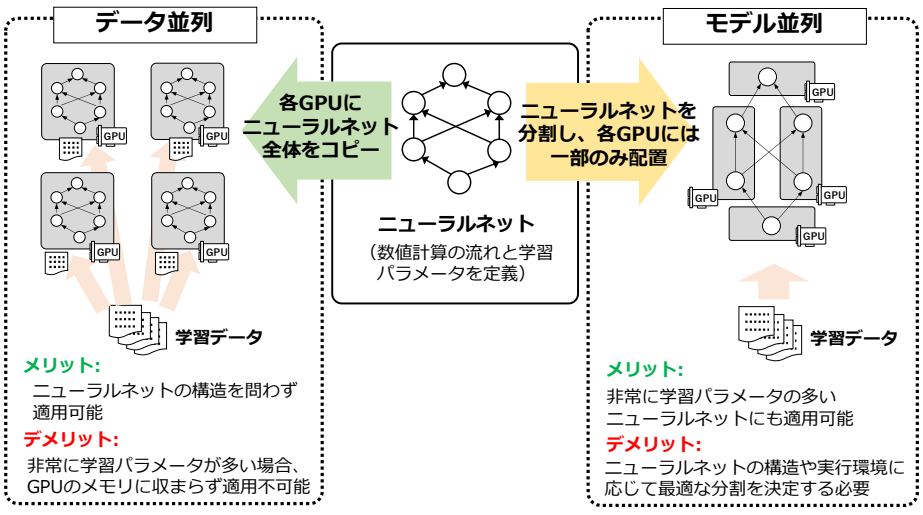


図 1　深層学習の並列化方式

表 1　既存フレームワークとの比較

	自動化			適用できるニューラルネットの種類
	(S1) 分割方法決定	(S2) コピー数決定	(S3) GPU 配置決定	
Megatron-LM（NVIDIA）	×	×	×	Transformer に限定
Mesh-TensorFlow（Google）	×	×	×	Transformer に限定
GPipe（Google）	×	×	×	制限無し
PipeDream-2BW（スタンフォード大、Microsoft）	×	○	×	制限無し
RaNNC（著者らが開発）	○	○	○	**制限無し**

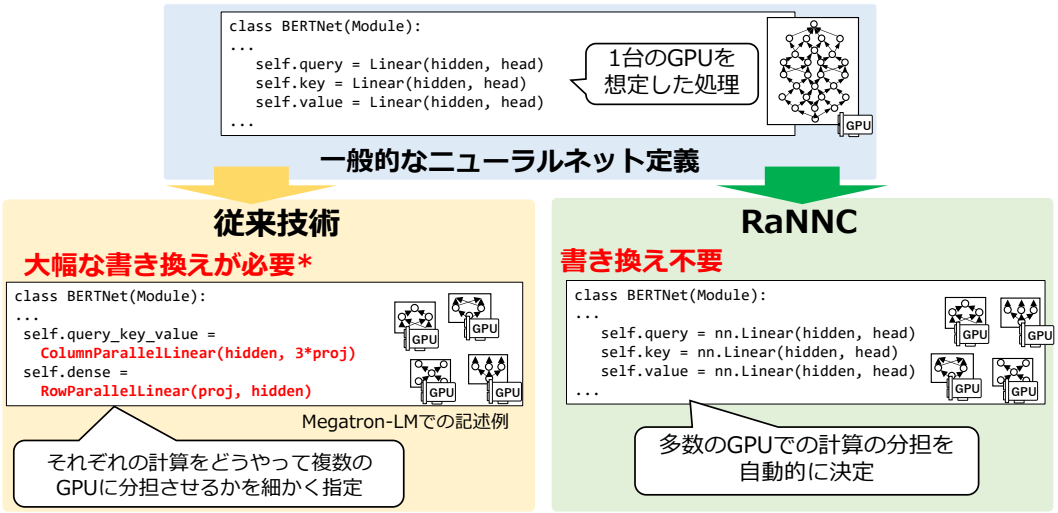


図 2　RaNNC の概要

そうな分割方法が、コピー数や GPU 配置によっては、実際には高速な処理速度を達成することもある。モデル並列を用いる大規模ニューラルネット学習のための著名な既存フレームワークでは、これらの決定の多くは自動化されておらず（表 1）、高い処理速度を得るためには、試行錯誤を通じて良い組合せを探しながらチューニングするほかない。そのため、高度な知識を持つ専門の技術者であっても、多数の GPU を用いた並列処理で、十分な処理速度を得られるまでには、相当の作業コストを要する。

なお、表 1 に示した従来フレームワークの PipeDream-2BW [8] は、分割可能な箇所を人間が指定すると、限られた組合せ（せいぜい数十パターン程度）から、分割数やコピー数を自動で決定する機能を持つが、可能な分割数等はあらかじめ人間が指定しておく必要がある。

3　RaNNC の概要

前述の課題を解決し、大規模ニューラルネット学習を容易にするために、DIRECT と東京大学が共同で開発したのが、自動並列化深層学習ミドルウェア RaNNC（Rapid Neural Network Connector）である。RaNNC は現在、GitHub においてオープンソースで公開されている*1。ライセンスは MIT ライセンスとしており、商用目的を含め，無償で利用できる。

*1　https://github.com/nict-wisdom/rannc

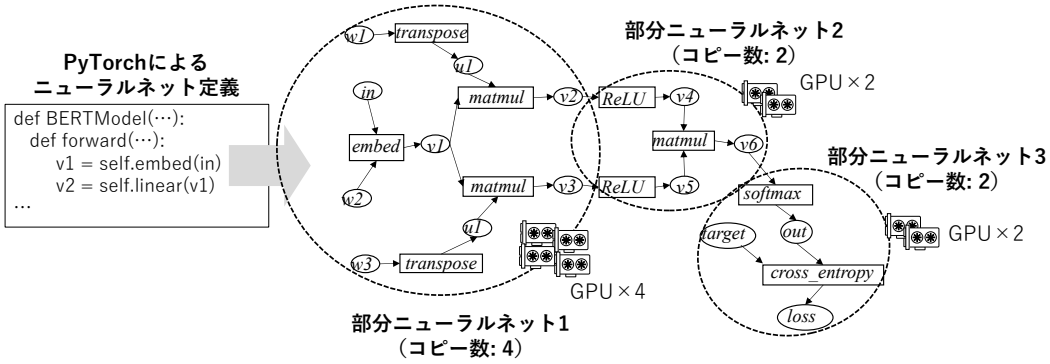


図 3　ニューラルネットの分割例

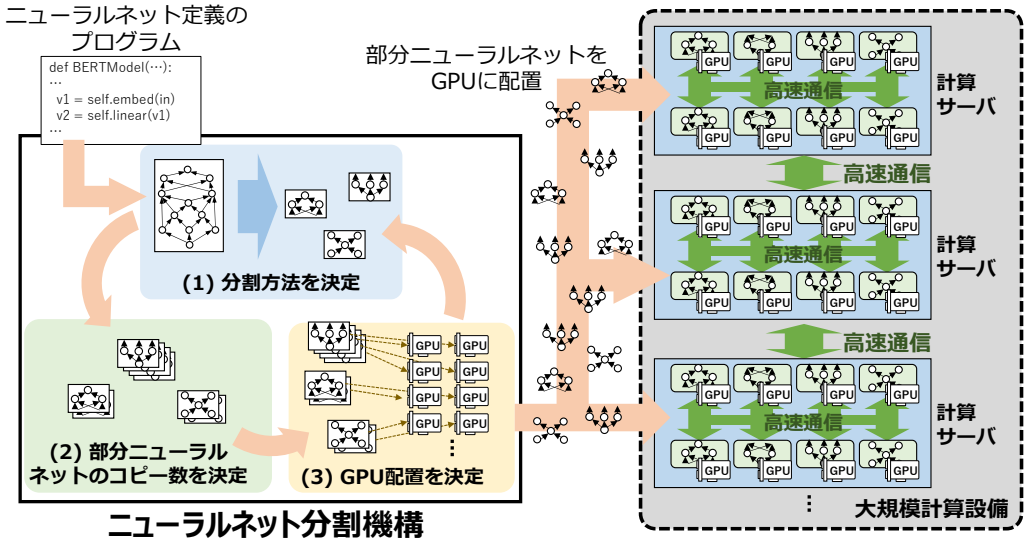


図 4　RaNNC による自動並列化

RaNNC の基本的なアイデアは、並列処理分野のトップレベルの国際会議 IPDPS 2021（IEEE International Parallel and Distributed Processing Symposium）で発表された [9]。また、PyTorch の開発を主導する Meta（旧 Facebook）が主催する PyTorch Annual Hackathon 2021（110 か国から 1,947 人が参加、応募 65 件）で First Place（第一位、PyTorch Developer Tools & Libraries 部門）を、産経新聞社主催「第 35 回独創性を拓く 先端技術大賞」において、社会人部門の最優秀賞である経済産業大臣賞を受賞するなど、高い評価を得ている。

RaNNC は、代表的な深層学習フレームワークである PyTorch のために記述されたニューラルネット定義を与えると、自動的にニューラルネットの分割を決定し、モデル並列とデータ並列を併用した並列分散学習を実現する。そのため、既存のものも含め、並列化を意識せずに 1 台の GPU を想定して書かれたニューラルネット定義のプログラムをそのまま使用できる（図 2）。有望なニューラルネットについて、構造はそのままに、隠れ層サイズやレイヤ数等といった学習パラメータに影響する設定のみを変更して性能向上を図ることは多いが、RaNNC を使用していれば、GPU 1 台またはデータ並列のみでは動作しない規模まで学習パラメータを増やしても、GPU メモリに収まるように、自動でモデル並列のためにニューラルネットを分割し、学習することが可能である。

また先に述べたとおり、広く使用されている既存フレームワークである Megatron-LM や Mesh-TensorFlow は、Transformer と呼ばれる特定のニューラルネットの種類にしか適用できないのに対し、RaNNC は基本的に適用できるニューラルネットの種類に制約がない。例えば、データ駆動知能システム研究センターでは、畳み込みニューラルネットと呼ばれるニューラルネットと BERT を組み合わせた BERTAC [10] を発表しているが、そのようなニューラルネットのパラメータ数を、GPU のメモリに収まらない規模にまで増加させて学習する場合には、Megatron-LM 等は利用できず、RaNNC を用いる必要がある。

図 3 は、RaNNC によるニューラルネット分割の例を示している。RaNNC の内部では、ニューラルネッ

トは、データと計算をノードとし、それらの依存関係をエッジによって示す計算グラフとして表現される（楕円のノードがデータ、矩形のノードが計算を示している）。図では、計算グラフとして表現されたニューラルネットを、3つの部分ニューラルネットに分割しているが、この分割は前述のモデル並列のための決定のうち、(S1) に相当する。図中のグラフは、それ以外にも様々な分割が可能であるが、可能な分割パターンの数は、ニューラルネットの計算グラフに含まれるノードの数が大きくなるに従って、指数関数的に大きくなる。実際には、大規模ニューラルネットに対応する計算グラフは、数千・数万のノードを持つため、単純に全ての可能な分割を数え上げ、それらを比較して優れたものを選択することは、実質的に不可能である。

また多数の GPU が利用できる場合、データ並列を併用するため、それぞれの部分ニューラルネットを複数の GPU に複製して配置する。これにより、モデル並列を用いるよりも実行効率を改善できる。そのため、どの部分ニューラルネットをいくつ複製するかも同時に決定する必要がある。これは、前述の決定の (S2)）に相当する。図3の例では、合計で8台の GPU が利用可能であるという前提の元、合計8になるように、3つの部分ニューラルネットに、それぞれコピーの数を設定している。

また、一般に1台のサーバは複数台の GPU を備えており（～8台程度）、多数の GPU が必要な場合、複数のサーバを使用するのが普通である。そこで、それぞれの部分ニューラルネットを、その複製も含めて、どのサーバのどの GPU に配置するかを決定することで、実際にニューラルネット学習の並列処理が可能となる（(S3) に相当）。

ニューラルネットの分割方法・コピー数・GPU 配置の決定の組合せは膨大な数に上るため、RaNNC では、まず有望な組合せ（数千～数万パターン程度）に絞り込んだ上で、それぞれの組合せについて、学習時の計算を部分的に実行しながら自動的に試行錯誤し、高速な学習速度を達成できるものを選択する（図4）。

計算グラフの分割を決める基準は、第一に、部分グラフの学習パラメータ及びその計算のために必要なメモリサイズが、GPU のメモリに収まることであるが、そのほかに、部分グラフ同士の通信時間を極力小さくすることが重要となる。これは、ある部分ニューラルネットの計算結果を、後続の部分ニューラルネットに送る際に、長い通信時間がかかると、GPU の処理に空き時間（アイドル時間）が生じ、実行効率が低下するためである。GPU は相互に通信可能であるが、通常は同一サーバ内の GPU 間の通信は、異なるサーバの GPU 間の通信より大幅に高速である。そこで RaNNC は、できるだけ部分ニューラルネット間で通信されるデータ量が少なくなるように分割箇所を決めると共に、相互に多くのデータを通信する部分ニューラルネットの組合せを、できるだけ同一サーバ内の GPU に配置するようにする。

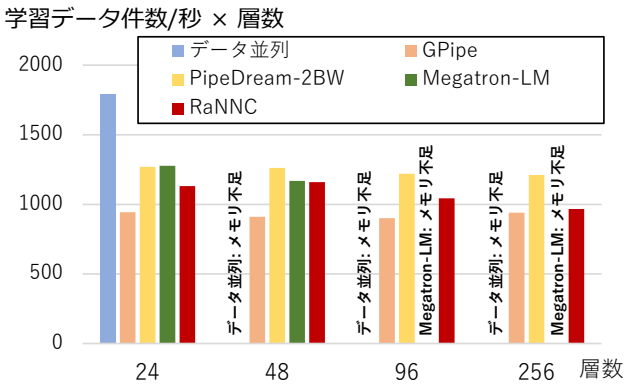
また、RaNNC は表1の GPipe, PipeDream-2BW 等でも利用されるパイプライン並列と呼ばれる方式を採用しているが、パイプライン並列では、各部分ニューラルネットの処理時間が均等なほど、アイドル時間が減少し、実行効率が改善する。部分ニューラルネットの処理時間は、そこに含まれる計算の内容だけではなく、データ並列の併用において使用するコピーの数によって決定される。そのため、部分ニューラルネットの処理時間を均等にするには、部分ニューラルネットへの分割と、使用するコピーの数を同時に考慮して、それらの組合せを決定しなければならない。単純な手法では、そうした組合せは無数にあり、単純に数え上げて優れたものを選ぶことはできない。そこで RaNNC では、動的計画法によるアルゴリズムを導入し、効率的に有望な組合せのみを探索している。

上記では、説明を簡単にするため、RaNNC によるニューラルネット分割のアルゴリズムは、概略のみを説明した。技術的な詳細については、著者らの論文 [9] を参照されたい。

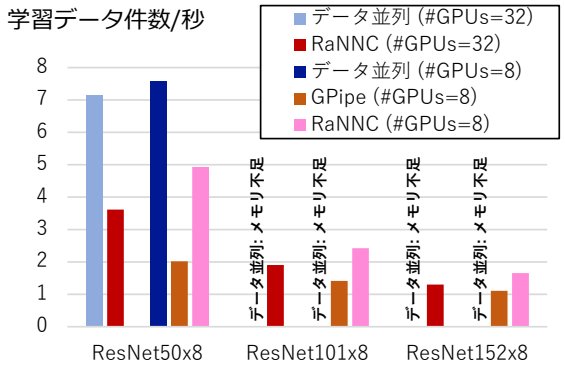
4　評価実験

RaNNC の速度を評価するため、モデル並列による学習を行う関連フレームワークとの比較実験を行った。比較対象として、表1に示した Megatron-LM、GPipe [11]、PipeDream-2BW を用いた。実験には、NVIDIA V100 を8台備えたサーバを最大4台（100 Gbps の InfiniBand で相互接続）を用いた。モデル並列を用いるフレームワークとして、表1の Mesh-TensorFlow が知られているが、バックエンドに用いるフレームワークとして TensorFlow [12] を用いており、PyTorch を用いる他のフレームワークとは厳密な比較が難しいため、ここでは比較対象から除外した。

隠れ層サイズを 2048（原論文の2倍）とした BERT について、レイヤ数を変化させながら、事前学習の実行速度を調査した結果を図5 (a) に示す。GPU を32台用い、GPU ごとのバッチサイズは8とした。レイヤ数が256の場合で、学習パラメータ数は129億となる。なお、GPipe のオリジナルの実装では TensorFlow を使用しており、またデータ並列とモデル並列の併用が不可能なため、RaNNC との比較が困難であった。そこで、PipeDream-2BW の著者らによって開発された、PyTorch を用い、かつデータ並列とモデル並列の併用



(a) 大規模BERT



(b) 大規模ResNet

図5　関連ソフトウェアとの速度比較

が可能な実装[*2]を用いた。

この実験では、Megatron-LMと比較して、約5倍の大きなネットワークまで学習できることが確認できた[*3]。また比較可能な規模のネットワークでは、ほぼ同等の速度となった。PipeDream-2BWは、RaNNCよりも幾分高速であることが確認されたが、高速化の目的で勾配の集約のための通信を非同期化しているため、parameter staleness [13]と呼ばれる問題により、学習精度低下の可能性があるという難点がある。

また、画像分類等に用いられるResNet [14]について、原論文で示された50レイヤ・101レイヤ・152レイヤのそれぞれの設定について、畳み込みレイヤのフィルタサイズをオリジナルの8倍に拡大し、学習速度を比較した。もっともパラメータが多い152レイヤの設定では、37億パラメータとなる。

前述のとおり、Megatron-LMはTransformer系のニューラルネットにしか適用できないため、本実験には使用していない。またBERTの学習における評価に用いたPipeDream-2BWやGPipeの実装は、BERTに特化されており、ResNetで動作させることができなかった。そのためこの比較では、PyTorchを用いたGPipeの実装であるtorchgpipe[*4]を用いた。torchgpipeはモデル並列のみに対応し、データ並列と併用できず、また複数サーバのGPUを利用できない。そのため、サーバ1台(GPU8台)を用いたモデル並列で実行した。

図5(b)に学習速度の比較を示す。サーバ4台(GPU32台)とサーバ1台(GPU8台)の2通りを実行し、GPU1台あたりのバッチサイズは16とした。図に示すように、データ並列では学習不可能な規模のネットワークにおいて、GPipeより顕著に高速に動作することが確認できた。

5　おわりに

深層学習のニューラルネットの大規模化に止まる兆しはなく、GPT-3以後も、NAVERが学習した韓国語の言語モデルHyperCLOVA[*5](2,040億パラメータ)、MicrosoftとNVIDIAによるMT-NLG [15](5,300億パラメータ)、GoogleのPaLM [16](5,400億パラメータ)など、各組織から相次いで大規模ニューラルネット学習が発表されている。また、GPT-NeoX [17]、BLOOM[*6]、YaLM[*7]など、学習済みの巨大言語モデルを、誰もが利用できるようにオープン化する試みも進んでいる。多数のGPUを持たない深層学習ユーザにとって、こうした規模のニューラルネットを初めから学習するのは困難であるが、こうした学習済みのニューラルネットに対して追加学習を行うことは十分可能であり、その際には、ユーザの計算機環境やニューラルネット構造の特徴に応じて自動的に学習を並列化できるRaNNCは有用であり、今後の利用範囲はますます広がっていくと思われる。

RaNNCは2021年3月の一般公開後も、継続的な開発により、安定性や堅牢性の向上、より巨大なニューラルネットを学習するための機能強化などを実現してきた。その結果、計算資源の制約から前述のMT-NLGやPaLMの規模には及ばないものの、本稿の執筆時点で、RaNNCを用いて約2,000億パラメータを超えるBERT(原論文の$\mathrm{BERT}_{\mathrm{Large}}$の約600倍)の学習が可能なことを確認できている。しかし、パラメータの規模が大きくなるとともに、学習したニューラルネットの

＊2　https://github.com/msr-fiddle/pipedream/tree/pipedream_2bw
＊3　Megatron-LMは、パイプライン並列のための機能を有効にすることで、より大きなネットワークを学習できる可能性があるが、本実験を実施した2020年8月時点では当該機能が実装されていなかったため、利用していない。また、パイプライン並列と同時に使用されるgradient checkpointingと呼ばれる機能により、メモリ使用量は減少するが、速度は低下する。
＊4　https://github.com/kakaobrain/torchgpipe
＊5　https://naver-ai-now.kr/
＊6　https://bigscience.huggingface.co/blog/bloom
＊7　https://github.com/yandex/YaLM-100B

利用に必要な計算資源の規模とそのためのコストも大きくなり、技術移転等を通じた活用を阻むことになる。そこで、現在は、DIRECT で独自に構築した高品質な日本語コーパスを用いて、200 億パラメータの BERT の学習を進めている。学習は現在も実行中であるが、従来から使用していた 4 億パラメータの BERT と比較してより高い精度が得られており、学習された 200 億パラメータの BERT を、NICT でこれまで開発してきた大規模 Web 情報分析システム WISDOM X[*8] や、高齢者介護支援用マルチモーダル音声対話システム MICSUS[*9] 等のシステムに組み込むことで、性能の向上が期待される。

＊8　https://www.wisdom-nict.jp/
＊9　https://www.youtube.com/watch?v=gCUrC3f9-Go

【参考文献】

1 J. Kaplan, S. McCandlish, et al., “Scaling laws for neural language models,” CoRR, vol.abs/2001.08361, 2020.
2 J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding,” Proceedings of the 17th Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT), pp.4171–4186, 2019.
3 T. B. Brown, B. Mann, N. Ryder, et al., “Language models are few-shot learners,” CoRR, vol.abs/2005.14165, 2020.
4 M. Shoeybi, M. Patwary, R. Puri, et al., “Megatron-LM: Training multi-billion parameter language models using model parallelism,” CoRR, vol.abs/1909.08053, 2019.
5 N. Shazeer, Y. Cheng, N. Parmar, et al., “Mesh-TensorFlow: deep learning for supercomputers,” Proceedings of the 32nd Conference on Neural Information Processing Systems (NeurIPS 2018), pp.10435–10444, 2018.
6 A. Vaswani, N. Shazeer, N. Parmar, et al., “Attention is all you need,” Proceedings of the 31st Conference on Neural Information Processing Systems (NeurIPS 2017), pp.5998–6008, 2017.
7 A. Paszke, S. Gross, F. Massa, et al., “PyTorch: An imperative style, high-performance deep learning library,” Proceedings of the 33rd Conference on Neural Information Processing Systems (NeurIPS 2019), pp.8024–8035, 2019.
8 D. Narayanan, A. Phanishayee, K. Shi, X. Chen, and M. Zaharia, “Memory-efficient pipeline-parallel DNN training,” CoRR, vol.abs/2006.09503, 2020.
9 M. Tanaka, K. Taura, T. Hanawa, and K. Torisawa, “Automatic graph partitioning for very large-scale deep learning,” Proceedings of the 35th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2021), pp.1004–1013, 2021.
10 J.-H. Oh, R. Iida, J. Kloetzer, and K. Torisawa, “BERTAC: Enhancing transformer-based language models with adversarially pretrained convolutional neural networks,” Proceedings of the the Joint Conference of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (ACL-IJCNLP 2021), pp.2103–2115, 2021.
11 Y. Huang, Y. Cheng, A. Bapnao, et al., “GPipe: Efficient training of giant neural networks using pipeline parallelism,” CoRR, vol.abs/1811.06965, 2018.
12 M. Abadi, A. Agarwal, P. Barham, et al., “TensorFlow: Large-scale machine learning on heterogeneous distributed systems.” CoRR, vol.abs/1603.04467, 2016.
13 Q. Ho, J. Cipar, H. Cui, J. K. Kim, et al., “More effective distributed ML via a stale synchronous parallel parameter server,” Proceedings of the 27th International Conference on Neural Information Processing Systems (NeurIPS 2018), pp.1223–1231, 2013.
14 K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2016), pp.770–778, 2016.
15 S. Smith, M. Patwary, B. Norick, et al., “Using DeepSpeed and Megatron to train Megatron-Turing NLG 530B, a large-scale generative language model,” CoRR, vol.abs/2201.11990, 2022.
16 A. Chowdhery, S. Narang, J. Devlin, et al., “PaLM: Scaling language modeling with Pathways,” CoRR, vol.abs/2204.02311, 2022.
17 S. Black, S. Biderman, E. Hallahan, et al., “GPT-NeoX-20B: An open-source autoregressive language model,” Proceedings of the ACL Workshop on Challenges and Perspectives in Creating Large Language Models, 2022.

田仲 正弘（たなか まさひろ）
ユニバーサルコミュニケーション研究所
データ駆動知能システム研究センター
主任研究員
博士（情報学）
大規模並列計算

【受賞歴】
2022 年　産経新聞社 第 35 回 独創性を拓く先端技術大賞
　　経済産業大臣賞（社会人部門最優秀賞）
2021 年　First Place at PyTorch Annual Hackathon 2021（PyTorch Developer Tools & Libraries category）
2016 年　公益財団法人通信文化協会 第 61 回 前島密賞