最近のインテル® アーキテクチャーにおける TensorFlow* の最適化

この記事は、インテル® デベロッパー・ゾーンに公開されている「TensorFlow* Optimizations on Modern Intel® Architecture」の日本語参考訳です。

インテル コーポレーション: Elmoustapha Ould-Ahmed-Vall、Mahmoud Abuzaina、Md Faijul Amin、Jayaram Bobba、Roman S Dubtsov、Evarist M Fomenko、Mukesh Gangadhar、Niranjan Hasabnis、Jing Huang、Deepthi Karkada、Young Jin Kim、Srihari Makineni、Dmitri Mishura、Karthik Raman、AG Ramesh、Vivek V Rane、Michael Riera、Dmitry Sergeev、Vamsi Sripathi、Bhavani Subramanian、Lakshay Tokas、Antonio C Valles

Google*: Andy Davis、Toby Boyd、Megan Kacholia、Rasmus Larsen、Rajat Monga、Thiru Palanisamy、Vijay Vasudevan、Yao Zhang (敬称略)

TensorFlow* は最先端のディープラーニングおよびマシンラーニング・フレームワークで、インテルと Google* のエンジニアがインテルのハードウェアからパフォーマンスを最大限に引き出すために重要な役割を果たしました。この記事は、人口知能 (AI) コミュニティーにインテル® Xeon® プロセッサーおよびインテル® Xeon Phi™ プロセッサー・ベースのプラットフォームにおける TensorFlow* の最適化を紹介するものです。これらの最適化はインテルと Google* のエンジニアのコラボレーションによる成果であり、2017 年に開催されたインテル AI Day で、インテルの Diane Bryant および Google* の Diane Greene 氏により発表されました。

この記事では、この最適化の間に直面したさまざまなパフォーマンスの課題と、課題を解決するために採用したソリューションについて説明します。また、一般的なニューラル・ネットワーク・モデルのサンプルにおけるパフォーマンスの向上についてもレポートします。これらの最適化は、桁違いのパフォーマンスをもたらします。例えば、この記事の計測では、インテル® Xeon Phi™ プロセッサー 7250 ベースのシステムで、訓練が最大 70 倍、推論が最大 85 倍と、パフォーマンスが大幅に向上しました。インテル® Xeon® プロセッサー E5 v4 (開発コード名 Broadwell) およびインテル® Xeon Phi™ プロセッサー 7250 ベースのプラットフォームは、インテルの次世代製品の基礎となるものです。特に、ユーザーは、インテル® Xeon® スケーラブル・プロセッサー上でのパフォーマンスの向上を期待しています。

最近の CPU におけるディープラーニング・モデルのパフォーマンスの最適化には、ハイパフォーマンス・コンピューティング (HPC) におけるパフォーマンス重視のアプリケーションの最適化と同様に、多くの課題があります。

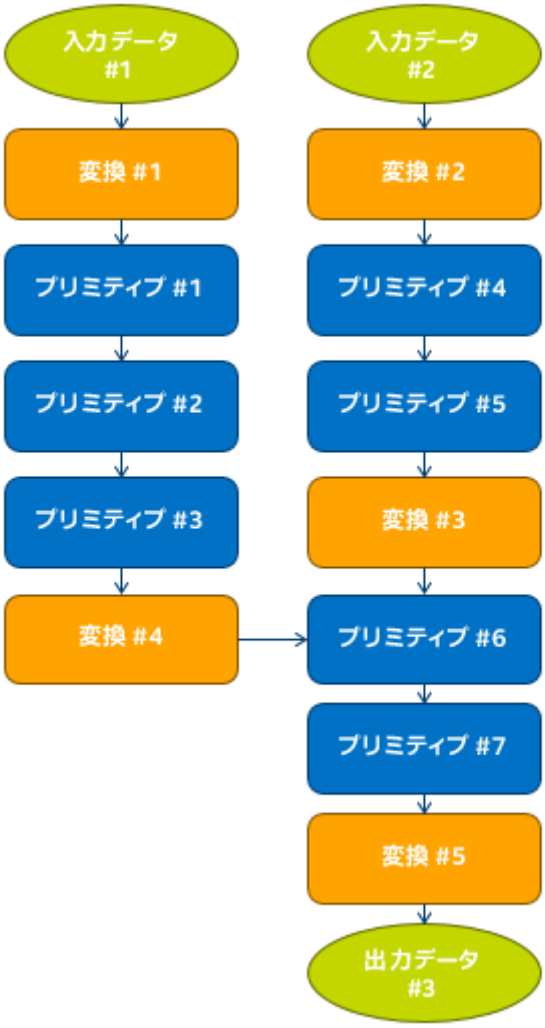
1. コードのリファクタリングには、最新のベクトル命令を利用する必要があります。畳み込み、行列乗算、バッチ正規化などのすべての主要なディープラーニング・プリミティブは、最新の SIMD 命令 (インテル® Xeon® プロセッサーではインテル® アドバンスト・ベクトル・エクステンション 2 (インテル® AVX2)、インテル® Xeon Phi™ プロセッサーではインテル® AVX-512) 向けにベクトル化します。
2. パフォーマンスを最大限に引き出すには、利用可能なすべてのコアを効率的に利用することに特に注意を払います。つまり、複数の層にわたる並列処理だけでなく、1 つの層や操作内の並列処理にも注目します。
3. 可能な限り、実行ユニットが必要なときにデータが利用できるようにします。このため、空間的および一時的局所性を向上するプリフェッチ、キャッシュ・ブロッキング手法とデータ形式をバランスよく使用します。

これらの要件を満たすため、インテルは、一般的なビルディング・ブロックを効率的に実装できるように、異なるディープラーニング・フレームワークの内部で使用できる、多くの最適化されたディープラーニング・プリミティブを開発しました。これらのビルディング・ブロックは、行列乗算や畳み込みのほかに次の演算を含みます。

- 直接バッチ畳み込み
- 内積
- プーリング: 最大、最小、平均
- 正規化: チャネル全体にわたる局所反応正規化 (LRN)、バッチ正規化
- 活性化: 正規化線形関数 (ReLU)
- データ操作: 多次元転置 (変換)、分割、連結、合計、スケール

これらのディープ・ニューラル・ネットワーク向けインテル® マス・カーネル・ライブラリー (インテル® MKL-DNN) の最適化されたプリミティブの詳細は、インテル® MKL の DNN プリミティブを参照してください。

TensorFlow* では、これらの演算で可能な限りインテル® MKL-DNN のプリミティブが利用されるように、インテルにより最適化されたバージョンの演算を実装しました。これはインテル® アーキテクチャーでスケーラブルなパフォーマンスを実現するために必要なステップですが、一方で、その他の多くの最適化も実装しなければなりませんでした。特に、パフォーマンス上の理由から、インテル® MKL は TensorFlow* のデフォルトのレイアウトと異なるレイアウトを使用しています。そのため、レイアウト変換のオーバーヘッドを最小限に抑える必要がありました。また、データ・サイエンティストやほかの TensorFlow* ユーザーが、既存のニューラル・ネットワーク・モデルを変更することなく、これらの最適化を活用できるようにする必要もありました。



グラフの最適化

次のようなグラフの最適化を行いました。

1. CPU で実行するときにデフォルトの TensorFlow* の演算をインテルにより最適化されたバージョンと置換する。ユーザーは、ニューラル・ネットワーク・モデルを変更することなく、既存の Python* プログラムを実行するだけでパフォーマンスの向上を認識できます。
2. 不要でコストのかかるデータレイアウト変換を排除する。
3. CPU のキャッシュを効率的に再利用するように複数の演算を融合する。
4. 高速なバックプロパゲーションが可能になるように中間ステートを制御する。

これらのグラフの最適化により、TensorFlow* プログラマーに追加の負担をかけることなく、優れたパフォーマンスを得ることができます。データレイアウト最適化は重要なパフォーマンス最適化です。ネイティブ TensorFlow* データ形式は、CPU 上の特定のテンソル演算では最も効率的なデータレイアウトではありません。その場合、TensorFlow* のネイティブ形式から内部形式へのデータレイアウト変換操作を追加して、CPU

上で演算を行い、変換操作の出力を TensorFlow* 形式に戻します。これらの変換により発生するパフォーマンス・オーバーヘッドは最小限に抑える必要があります。このデータレイアウト最適化は、インテル® MKL の最適化された演算を使用して完全に実行できるサブグラフを識別し、そのサブグラフの操作内で変換を行いません。自動的に挿入される変換ノードは、サブグラフの境界のデータレイアウト変換を処理します。もう 1 つの重要な最適化は、1 つのインテル® MKL 演算として効率的に実行できる複数の演算を自動的に融合するフュージョン・パスです。

その他の最適化

さまざまなディープラーニング・モデルで最高の CPU パフォーマンスが得られるように、TensorFlow* フレームワークの多くのコンポーネントを微調整しました。TensorFlow* の既存のプール・アロケーターを使用して、カスタム・プール・アロケーターを作成しました。このカスタム・プール・アロケーターは、TensorFlow* とインテル® MKL の両方で同じメモリープールを (インテル® MKL の imalloc 機能を使用して) 共有し、操作が完了するまでメモリーをオペレーティング・システムに返さないため、コストのかかるページミスやページクリアを回避できます。さらに、複数のスレッド・ライブラリー (TensorFlow* で使用する Pthreads* およびインテル® MKL で使用する OpenMP*) が共存でき、ライブラリー間で CPU リソースの競合が発生しないように、これらのライブラリーを注意深くチューニングしました。

パフォーマンス・テスト

上記で説明した最適化を行うことで、インテル® Xeon® プラットフォームおよびインテル® Xeon Phi™ プラットフォームの両方でパフォーマンスが大幅に向上しました。3 つの一般的な ConvNet ベンチマーク (英語) でテストした、最も一般的な手法のベースラインの値と最適化されたパフォーマンスの値を以下に示します。

1. インテル® Xeon® プロセッサー (開発コード名 Broadwell) およびインテル® Xeon Phi™ プロセッサー (開発コード名 Knights Landing) のパフォーマンスを引き出すには、次のパラメーターが重要です。使用するニューラル・ネットワーク・モデルおよびプラットフォームに合わせて、これらのパラメーターをチューニングすることを推奨します。ここでは、インテル® Xeon® プロセッサーおよびインテル® Xeon Phi™ プロセッサーの両方で最高のベンチマーク結果が得られるように、これらのパラメーターを注意深くチューニングしました。
    1. データ形式: パフォーマンスを最大限に引き出すため、特定のニューラル・ネットワーク・モデルでは NCHW 形式を指定することを推奨します。TensorFlow* のデフォルトの NHWC 形式は CPU では最も効率的なデータレイアウトではないため、多少の変換のオーバーヘッドが発生します。
    2. inter-op/intra-op: 各モデルおよび CPU プラットフォームで最適な設定になるように、TensorFlow* の intra-op/inter-op パラメーターをテストすることを推奨します。これらの設定は、複数の層にわたる並列処理だけでなく、1 つの層内の並列処理にも影響します。

3. バッチサイズ: バッチサイズも、すべてのコアを利用するために利用可能な並列処理 (およびワーキングセットのサイズとメモリーのパフォーマンス) に影響を与える重要なパラメーターです。
4. OMP_NUM_THREADS: パフォーマンスを最大限に引き出すには、利用可能なすべてのコアを効率的に利用する必要があります。インテル® Xeon Phi™ プロセッサーはハイパースレッディングのレベル (1 から 4) を制御するため、この設定は特に重要です。
5. 行列乗算の転置: 一部の行列サイズでは、2 つ目の入力行列 b を転置すると Matmul 層のパフォーマンス (キャッシュの再利用) が向上します。これは下記の 3 つのモデルで使用されているすべての Matmul 演算に当てはまります。ほかの行列サイズでは、この設定をテストすることを推奨します。
6. KMP_BLOCKTIME: 並列領域の実行を完了した後に各スレッドが待機する時間 (ミリ秒) を、さまざまな設定でテストすることを推奨します

インテル® Xeon® プロセッサー (開発コード名 - 2 ソケット - 22 コア) の設定例

Benchmark	Data Format	Inter_op	Intra_op	KMP_BLOCKTIME	Batch size
ConvNet-AlexnetNet	NCHW	1	44	30	2048
ConvNet-Googlenet V1	NCHW	2	44	1	256
ConvNet-VGG	NCHW	1	44	1	128

インテル® Xeon Phi™ プロセッサー (開発コード名 Knights Landing - 68 コア) の設定例

Benchmark	Data Format	Inter_op	Intra_op	KMP_BLOCKTIME	OMP_NUM_THREADS	Batch size
ConvNet-AlexnetNet	NCHW	1	136	30	136	2048
ConvNet-Googlenet V1	NCHW	2 training 1 inference	68	Infinite	68	256
ConvNet-VGG	NCHW	1	136	1	136	128

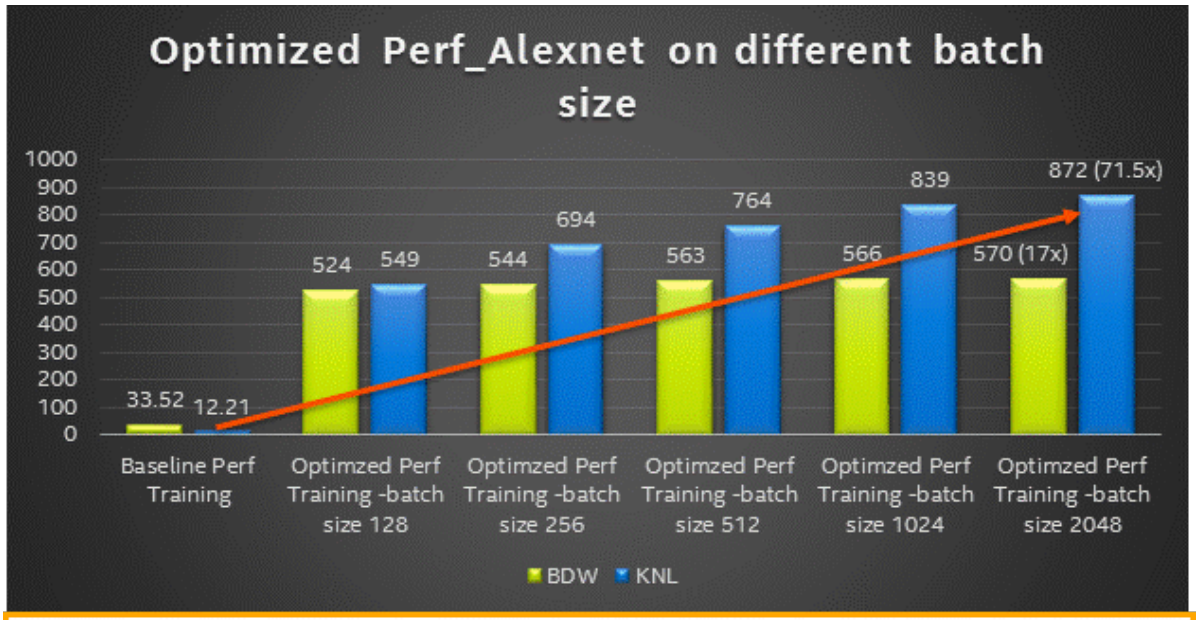
1. インテル® Xeon® プロセッサー (開発コード名 - 2 ソケット - 22 コア) のパフォーマンス結果

Benchmark	Metric	Batch Size	Baseline Performance Training	Baseline Performance Inference	Optimized Performance Training	Optimized Performance Inference	Speedup Training	Speedup Inference
ConvNet-Alexnet	Images / sec	128	33.52	84.2	524	1696	**15.6x**	**20.2x**
ConvNet-GoogleNet v1	Images / sec	128	16.87	49.9	112.3	439.7	**6.7x**	**8.8x**
ConvNet-VGG	Images / sec	64	8.2	30.7	47.1	151.1	**5.7x**	**4.9x**

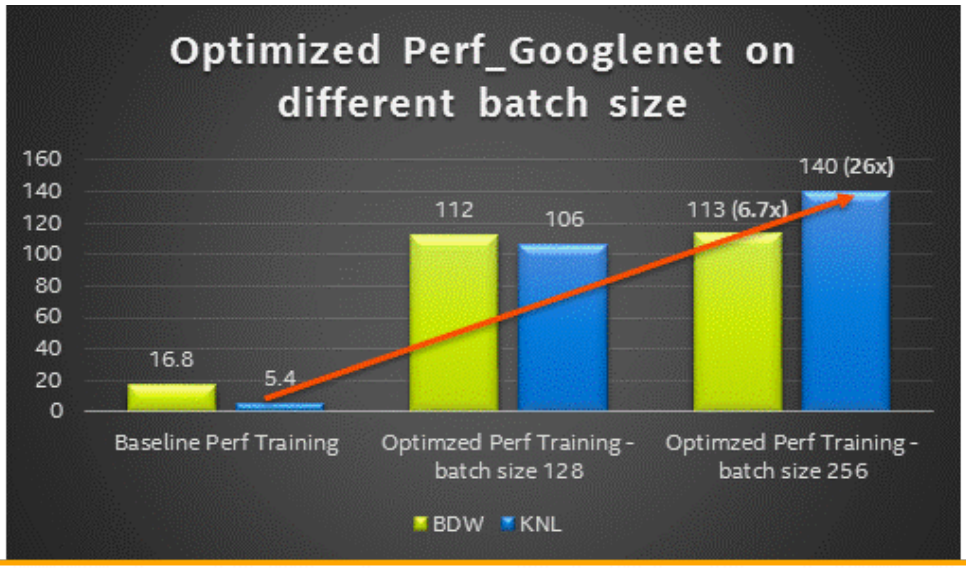
2. インテル® Xeon Phi™ プロセッサー (開発コード名 Knights Landing - 68 コア) のパフォーマンス結果

Benchmark	Metric	Batch Size	Baseline Performance Training	Baseline Performance Inference	Optimized Performance Training	Optimized Performance Inference	Speedup Training	Speedup Inference
ConvNet-Alexnet	Images / sec	128	12.21	31.3	549	2698.3	**45x**	**86.2x**
ConvNet-GoogleNet v1	Images / sec	128	5.43	10.9	106	576.6	**19.5x**	**53x**
ConvNet-VGG	Images / sec	64	1.59	24.6	69.4	251	**43.6x**	**10.2x**

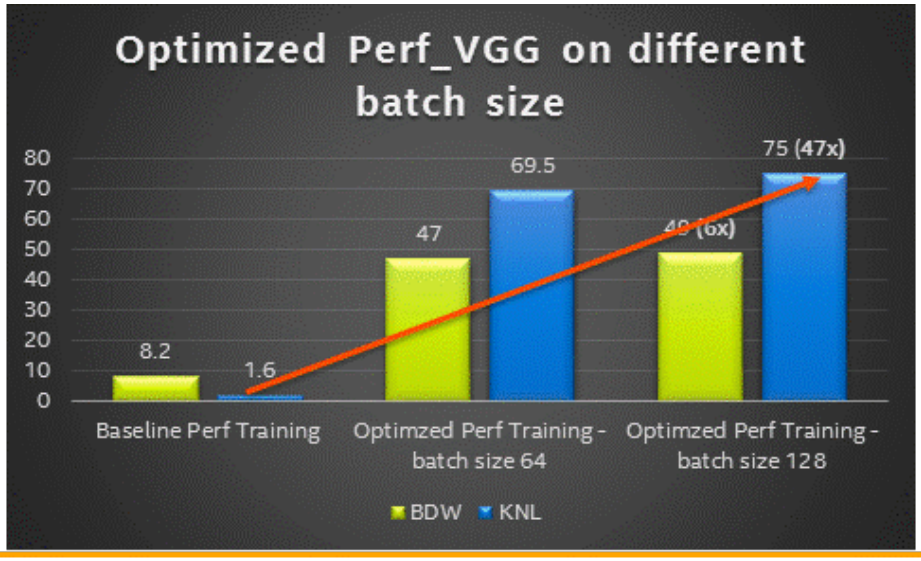
3. インテル® Xeon® プロセッサー (開発コード名 Broadwell) およびインテル® Xeon Phi™ プロセッサー (開発コード名 Knights Landing) の異なるバッチサイズのパフォーマンス結果 - 訓練



72x Speedup From New Optimizations – available through Google's TensorFlow Git



26x Speedup From New Optimizations – available through Google's TensorFlow Git



47x Speedup From New Optimizations – available through Google's TensorFlow Git

最適化された TensorFlow* のインストール

インテル® Optimization for TensorFlow* インストール・ガイド (英語) の説明に従って pip や conda を使用して事前にビルドされたバイナリーパッケージをインストールするか、下記の説明に従ってソースからビルドします。

1. TensorFlow* ソース・ディレクトリーから "./configure" を実行します。インテル® MKL を使用するオプションを選択した場合、tensorflow/third_party/mkl/mklml に最新のマシンラーニング向けインテル® MKL が自動的にダウンロードされます。
2. 次のコマンドを実行して、最適化された TensorFlow* ビルドのインストールに使用する pip パッケージを作成します。
    - GCC コンパイラーの特定のバージョンを指すように PATH を変更します。
      export PATH=/PATH/gcc/bin:$PATH
    - 新しい GLIBC を指すように LD_LIBRARY_PATH を変更します。
      export LD_LIBRARY_PATH=/PATH/gcc/lib64:$LD_LIBRARY_PATH
    - インテル® Xeon® プロセッサーおよびインテル® Xeon Phi™ プロセッサー向けにビルドします。
      bazel build --config=mkl --copt="-DEIGEN_USE_VML" -c opt
      //tensorflow/tools/pip_package:
      build_pip_package
3. 最適化された TensorFlow* wheel をインストールします。
    1. bazel-bin/tensorflow/tools/pip_package/build_pip_package ~/path_to_save_wheel
       pip install --upgrade --user ~/path_to_save_wheel /wheel_name.whl

システム構成

BDW4	KNL11
Architecture: x86_64	Architecture: x86_64
CPU op-mode(s): 32-bit, 64-bit	CPU op-mode(s): 32-bit, 64-bit
Byte Order: Little Endian	Byte Order: Little Endian
CPU(s): 44	CPU(s): 272
On-line CPU(s) list: 0-43	On-line CPU(s) list: 0-271
Thread(s) per core: 1	Thread(s) per core: 4
Core(s) per socket: 22	Core(s) per socket: 68
Socket(s): 2	Socket(s): 1
NUMA node(s): 2	NUMA node(s): 2
Vendor ID: GenuineIntel	Vendor ID: GenuineIntel
CPU family: 6	CPU family: 6
Model: 79	Model: 87
Model name: Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20GHz	Model name: Intel(R) Xeon Phi(TM) CPU 7250 @000000 1.40GHz
Stepping: 1	Stepping: 1
CPU MHz: 2426.273	CPU MHz: 1400
BogoMIPS: 4397.87	BogoMIPS: 2793.45
Virtualization: VT-x	L1d cache: 32K
L1d cache: 32K	L1i cache: 32K
L1i cache: 32K	L2 cache: 1024K
L2 cache: 256K	NUMA node0 CPU(s): 0-271
L3 cache: 56320K	
NUMA node0 CPU(s): 0-21	
NUMA node1 CPU(s): 22-43	

AI にもたらすもの

TensorFlow* の最適化は、この広く利用可能で広く適用されたフレームワークを使用して作成されたディープラーニング・アプリケーションをインテル® プロセッサー上でより高速に実行して、柔軟性、アクセシビリティー、スケールを向上できることを意味します。例えば、インテル® Xeon Phi™ プロセッサーは、コア数およびノード数でほぼ線形にスケーリングするように設計されており、マシンラーニング・モデルの訓練時間を大幅に短縮できます。TensorFlow* の将来のパフォーマンスも、インテル® プロセッサーのパフォーマンスの向上とともにスケーリングされ、さらに大きく、より困難な AI ワークロードを制御できるようになります。

インテルと Google* のコラボレーションによる TensorFlow* の最適化は、開発者やデータ・サイエンティストが AI をより簡単に利用できるように、また、AI アプリケーションを (エッジからクラウドまで) あらゆる種類のデバイスで実行できるようにするために取り組んでいるプロジェクトの一部です。インテルは、これが、ビジネス、科学、エンジニアリング、医療、社会の差し迫った問題を解決する、次世代の AI アルゴリズムおよびモデルを作成するための鍵であると考えています。

このコラボレーションにより、主要なインテル® Xeon® プロセッサーおよびインテル® Xeon Phi™ プロセッサー・ベースのプラットフォームにおけるパフォーマンスは大幅に向上しました。これらの情報は、Google* の TensorFlow* GitHub* リポジトリー (英語) から入手できます。我々は、AI コミュニティーにこれらの最適化のテストを依頼しており、テストのフィードバックを楽しみにしています。

コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。