低い数値精度でのディープラーニングのトレーニングと推論

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「Lower Numerical Precision Deep Learning Inference and Training」の日本語参考訳です。

はじめに

今日、ほとんどの商用ディープラーニング・アプリケーションは、トレーニングと推論ワークロードに 32 ビット浮動小数点精度を使用しています。ディープラーニングのトレーニングと推論は低い精度で行うことが可能であり、トレーニングに 16 ビットの乗数を使用し、推論には 8 ビットの乗数を使用することで、精度をほとんどまたは全く損なうことなくそれぞれの処理を実行できることを多くの研究者が実証しています。これらの低い数値精度を使用する (トレーニングでは 16 ビットの乗数を 32 ビットに累積し、推論では 8 ビットの乗数を 32 ビットに累積する) ことは、今後数年間で標準となるでしょう。

インテルは、インテル® Xeon® プロセッサーやその他の製品において、低い精度でトレーニングと推論を実現できるように積極的に取り組んでいます。低い精度を使用する利点は主に 2 つあります。1 つ目は、多くの操作はメモリー帯域幅依存であり、精度を下げることでキャッシュを効率良く利用し、帯域幅のボトルネックを軽減できます。これにより、メモリー階層でデータを高速に移動できるようになり、計算リソースを最大限に利用できます。2 つ目は、低い数値精度では乗数が必要とするシリコン面積と電力が少ないため、ハードウェアは 1 秒間により多くの操作を処理できるようになります。

この記事では、低い数値精度でのディープラーニングのトレーニングと推論の歴史を振り返り、その実現に向けたインテルの取り組みを紹介します。具体的には、第 1 世代インテル® Xeon® スケーラブル・プロセッサーで利用可能な命令と、インテル® Deep Learning Boost (インテル® DL Boost) と呼ばれる組込みアクセラレーターとして機能する第 2 世代インテル® Xeon® スケーラブル・プロセッサーで利用可能な命令について説明します。モデルの重みと活性化を量子化する方法と、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) (英語) で利用可能な低い数値精度の関数について述べます。最後に、ディープラーニング・フレームワークでこれらの低い数値精度の関数を利用し、異なる数値精度間の変換オーバーヘッドを軽減する方法を示します。各セクションは、どこからでも読み始められるように構成されています。インテル® Xeon® スケーラブル・プロセッサーを利用したディープラーニングのトレーニングと推論の商用例を含む詳細な説明は、こちらを参照してください。

ディープラーニングにおける低い精度の歴史

さまざまなモデルにおいて精度をほとんどまたは全く損なうことなく、ディープラーニングのトレーニングを 16 ビットの乗数で行い、推論を 8 ビットの乗数で行って、それぞれを高い精度に累積できることが研究者により実証されています。

Vanhoucke ほか (2011) (英語) は、CPU 上の音声認識タスクにおいて、活性化と重みを 8 ビットに量子化し、バイアスと最初の層の入力は完全な精度を維持しています。

Hwang ほか (2014) (英語) は、パフォーマンスをほとんど損なうことなく、フィード・フォワード・プロパゲーションで量子化された重み -1、0、1 を使用して単純なネットワークをトレーニングし、MNIST と TIMIT データセットを使用してバックワード・プロパゲーションで高い精度の重みを更新しています。

Courbariaux ほか (2015) (英語) は、MNIST、CIFAR-10、および SVHN データセットで低い数値精度の乗数と高い数値精度のアキュムレーターを使用してモデルをトレーニングし、高い精度の重みを更新しています。また、将来の取り組みとして、動的固定点 (テンソルまたは高次元配列に対して 1 つの共有指数を有する) と Gupta ほか (2015) (英語) の確率的丸めを組み合わせることを提案しています。

Miyashita ほか (2016) (英語) は、重みと活性化を基数 2 の対数表現でエンコードしています (重み/活性化は非一様分布であるため)。また、5 ビットの重みと 4 ビットの活性化で CIFAR-10 を トレーニングして、パフォーマンスの低下を最小限に抑えています。

Rastegari ほか (2016) (英語) は、2 進重みで AlexNet を トレーニングし (最初と最後の層を除く)、完全な精度の重みを更新して、2.9% の Top-1 精度低下を達成しています。そして検証結果を基に、小さなチャネル数やフィルターサイズの全結合層や畳み込み層での 2 進化は行わないように推奨しています。

インテルラボの Mellempudi ほか (2017) (英語) は、畳み込み層で 4 ビットの重みと 8 ビットの活性化で ResNet-101 をトレーニングし、完全な精度で更新して、2% の Top-1 精度低下を達成しています。

Micikevicius ほか (2017) (英語) は、AlexNet、VGG-D、 GoogLeNet、ResNet-50、Faster R-CNN*、Multibox SSD、DeepSpeech2、Sequence-to-Sequence、bigLSTM、および DCGAN において、精度をほとんどまたは全く損なうことなく、16 ビット浮動小数点の乗数と完全な精度のアキュムレーターでトレーニングし、完全な精度の重みで更新しています (一部のモデルは、完全な精度の結果と一致させるため勾配スケーリングが必要です)。

Baidu* の研究者たち (2017) (英語) は、1 ビットの符号、4 ビットの整数部、3 ビットの小数部から成る 8 ビットの固定精度を使用しています。

Sze ほか (2017) (英語) は、低い数値精度を使用して (ただし最初と最後の層には完全な精度を使用)、結果の精度をほとんどまたは全く損なうことなく、さまざまな量子化手法 (論文の表 3 を参照) を使用しています。

Das ほか (2018) (英語) は、16 ビット整数の乗数と 32 ビット のアキュムレーターを使用して ResNet-50、GoogLeNet、VGG-16、および AlexNet をトレーニングしています。

Google* の研究者たち (2018) (英語) は、トレーニング・ワークロード向けに Google* の Tensorflow* Processing Unit (TPU) でネイティブにサポートされおり、将来のインテル® Xeon® プロセッサーとインテル® Nervana™ ニューラル・ネットワーク・プロセッサーで利用可能な bfloat16 数値形式について説明しています。

最も一般的な 16 ビット数値形式は、16 ビット IEEE 浮動小数点数 (*fp16*)、bfloat16 (*bf16*)、16 ビット整数 (*int16*) であり、最も一般的な 8 ビット数値形式は、8 ビット整数 (*int8*) と 8 ビット Microsoft* 浮動小数点数 (*ms-fp8*) です。図 1 は、これらの形式の相違点を示しています。

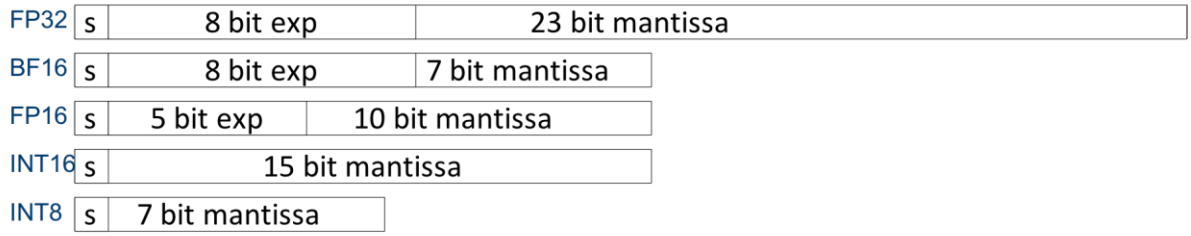

図 1. さまざまな数値形式。*s* は符号ビットを示します。FP32 と BF16 のダイナミックレンジは同じですが、FP32 は仮数部が大きいため高い精度を提供できます。

インテル® AVX-512 とインテル® DL Boost 機能を搭載したインテル® Xeon® スケーラブル・プロセッサーで低い数値精度の利用

インテル® Xeon® スケーラブル・プロセッサーでは、512 ビットの FMA (Fused Multiply Add) 命令を含むインテル® アドバンスト・ベクトル・エクステンション 512 (インテル® AVX-512) 命令セットを利用できます。これらの命令は、低い精度の乗算と高い精度の累算の利用を可能にします。2 つの 8 ビット値を乗算して、結果を 32 ビットに累算するには、3 つの命令が必要であり、8 ビット・ベクトルの 1 つは符号なし int8 (*u8*) 形式、もう 1 つは符号付き int8 (*s8*) 形式で、累算は符号付き int32 (*s32*) 形式でなければなりません。これにより、3 倍の命令で 4 倍の入力を処理できます (つまり、計算処理能力が 33.33% 向上します)。低い数値精度の操作はメモリー使用量が少なく、周波数が高いため、さらなるパフォーマンス・ゲインが得られる可能性があります。詳細は、図 2 を参照してください [1]。

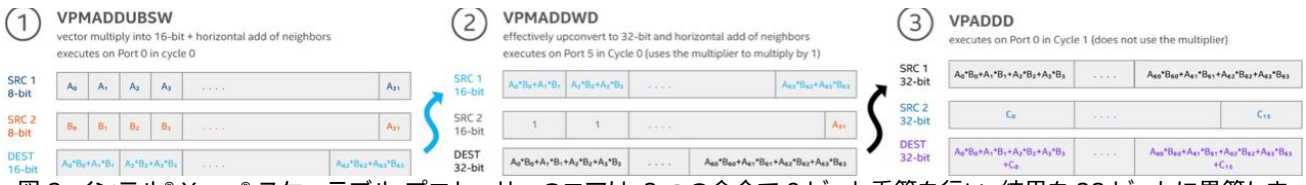

図 2. インテル® Xeon® スケーラブル・プロセッサーのコアは、3 つの命令で 8 ビット乗算を行い、結果を 32 ビットに累算します。VPMADDUBSW 命令 *u8×s8→s16* で乗算を行い、VPMADDWD 命令 *broadcast1 s16→s32* で 32 ビットに変換して、VPADDD 命令 *s32→s32* で結果をアキュムレーターに加算します。これにより、*fp32* と比較して、3 倍の命令数で 4 倍の入力を処理できます (計算処理能力が 33.33% 向上し、メモリー使用量が ¼ に減ります)。低い数値精度の操作はメモリー使用量が少なく、周波数が高いため、さらなるスピードアップをもたらします。画像の出典: Israel Hirsh。

潜在的な問題は、VPMADDUBSW 命令 *u8×s8→s16* で発生する可能性があるオーバーフローに対する未定義の動作です (図 2 を参照)。これは、2 つの入力値が最大値に近い場合に問題となります [2]。入力値の精度を 1 ビット下げることでこの問題を軽減できます。Facebook* で使用されている手法 (英語) は行列乗算を 2 つの行列乗算に分割して、1 つは 8 ビット乗算と 16 ビット累算を使用して (オーバーフローを回避するため) 小さな値を処理し、もう 1 つは完全な精度でスパースな大きな値を処理します。

インテルは、DL パフォーマンスをさらに向上するため、第 2 世代インテル® Xeon® スケーラブル・プロセッサーでインテル® Deep Learning Boost (インテル® DL Boost) と呼ばれる新しいインテル® AVX-512 命令を追加しています。AVX512_VNNI (Vector Neural Network Instruction: ベクトル・ニューラル・ネットワーク命令) と呼ばれる最初のインテル® DL Boost テクノロジーは、第 2 世代インテル® Xeon® スケーラブル・プロセッサー (開発コード名 Cascade Lake) およびその他の将来のマイクロアーキテクチャーで利用できます (表 1-1 を参照 (英語))。AVX512_VNNI には、図 3 に示す 8 ビット乗算と 32 ビット累算の FMA 命令 (*u8×s8→s32*) が含まれています。s32 に累算することでオーバーフローのリスクを排除します。理論上のピーク計算ゲインは、fp32 操作と比較して 4x *int8* 操作です。実際には、メモリー帯域幅のボトルネックにより、ゲインは小さくなります。

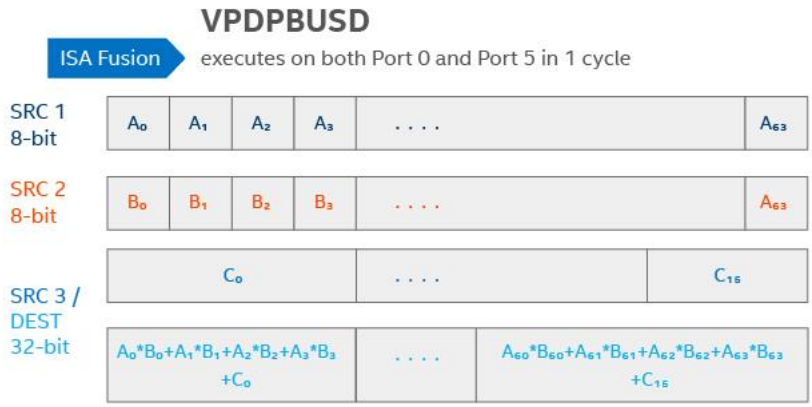

図 3. インテル® DL Boost の AVX512_VNNI VPDPBUSD 命令は、8 ビット乗算と 32 ビット累算を 1 命令 *u8×s8→s32* で行い、fp32 操作と比較して 4x int8 操作という理論上のピーク計算ゲインをもたらします。画像の出典: Israel Hirsh。

インテル® コンパイラーの AVX512_VNNI 命令サポートは今後提供される予定です。GCC 8 (英語) 開発コードと LLVM/Clang 6.0 (英語) コンパイラーは、すでに AVX512_VNNI 命令をサポートしています。また、X86 Encoder Decoder (XED) (英語) と インテル® Software Developer Emulator (インテル®SDE) (英語) の 2017 年 10 月のアップデートも AVX512_VNNI 命令をサポートしています。

インテル® MKL-DNN ライブラリーの低い精度のプリミティブ

インテル® MKL-DNN ライブラリーには、内積、畳み込み、正規化線形関数 (ReLU)、バッチ正規化 (BN) などさまざまなモデルで使用される一般的なディープラーニング関数とプリミティブに加えて、テンソルや高次元配列のレイアウト操作に必要な関数が含まれています。インテル® MKL-DNN は、インテル® AVX-512、インテル® AVX-2、インテル® ストリーミング SIMD 拡張命令 4.2 (インテル® SSE4.2) 対応のインテル® プロセッサー向けに最適化されています。これらの関数は、トレーニングと推論ワークロードに *fp32* を使用します。

最近、畳み込み、ReLU、融合畳み込み + ReLU、プーリング層で 8 ビット精度の推論ワークロードをサポートするため新しい関数が追加されました。8 ビットの LSTM (Long Short-Term Memory)、ほかの融合命令、Winograd 畳み込み向け関数は今後対応が予定されています。AVX512_VNNI 命令を使用する 16 ビット乗算のインテル® MKL-DNN サポートは、命令が利用可能になった後に対応予定です。

インテル® MKL-DNN は、次の理由から、8 ビット精度の局所反応正規化 (LRN)、ソフトマックス、バッチ正規化 (BN) 層の実装を提供していません (*fp32* 精度でのみ提供しています)。最近のモデルは LRN を使用しません。古いモデルは代わりにバッチ正規化を使用するように変更できます。ソフトマックス関数と BN は、8 ビットでは精度を維持できないため、完全な精度を必要とします。さらに、BN 推論層とそれに続く畳み込みは、「フレームワーク」セクションの低い数値精度の有効化で説明したように、重みの値をスケーリングし、バイアスを変更することで前の層に吸収できるため不要です。

インテル® MKL-DNN は、*u8* 形式の活性化 (または入力) 値、*s8* 形式の重み、*s32* 形式のバイアス (バイアスは *fp32* で保持可能で、計算全体に占める割合はごくわずか) で 8 ビットの畳み込み操作を実装します。図 4 は、8 ビット乗算を *s32* に累算する推論操作のプロセスを示します。

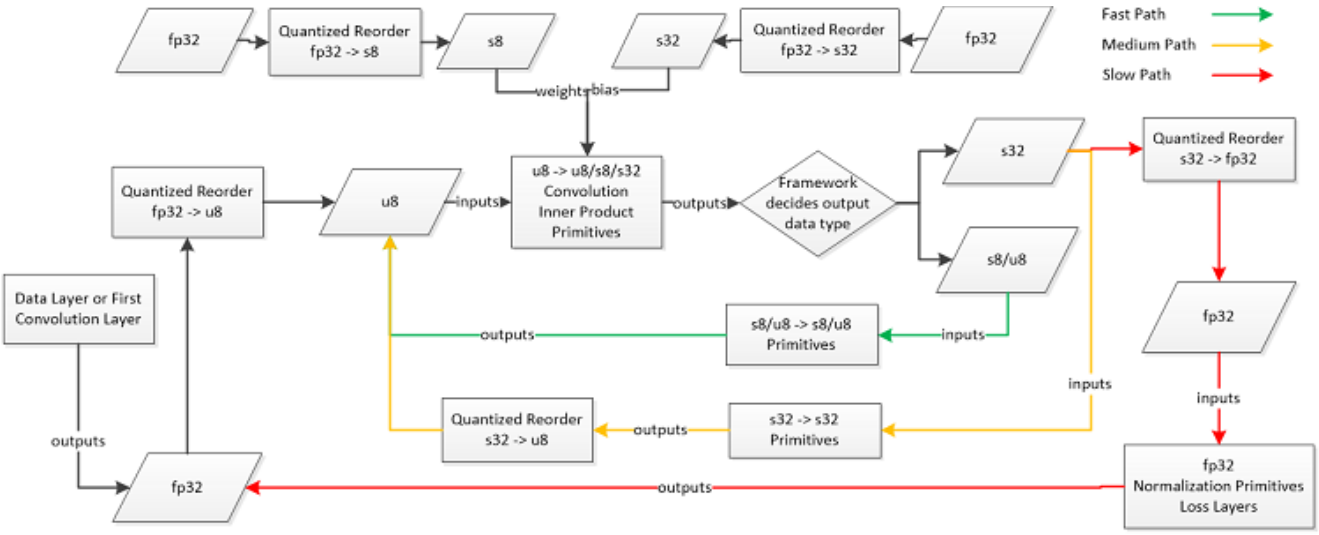

図 4. データ層または最初の畳み込み層の活性化は、次の畳み込み層の入力として *u8* 形式に量子化されます。重みは *s8* に量子化され、バイアスは *s32* にフォーマットされ、*s32* 畳み込み累積値に加算されます。フレームワークは、次の層のパラメーターに応じて、畳み込み出力に *s8*、*u8*、または *s32* 形式を選択します。画像の出典: Jiong Gong。

負でない値と重みを持つ活性化の 8 ビットの量子化

インテル® MKL-DNN は現在、ReLU 活性化関数の後の活性化は負ではないと想定しています。負の値を持つ活性化の量子化方法は後述します。インテル® MKL-DNN は、次のように、特定のテンソルやテンソルの各チャネル (どちらを選択するかはフレームワーク開発者によります) の値を量子化します。

$R_{\{a,w\}} = \max(abs(T_{\{a,w\}}))$。ここで、$T_{\{a,w\}}$ は重み *w* あるいは活性化またはモデル入力 *a* に対応するテンソルです。

$Q_a = {}^{255}/_{Ra}$ は、負ではない値を持つ活性化の量子化係数で、$Q_w = {}^{127}/_{Rw}$ は重みの量子化係数です。量子化された活性化、重み、およびバイアスは次のとおりです。

$$\boldsymbol{a}_{u8} = \|Q_a \boldsymbol{a}_{f32}\| \in [0,255]$$

$$\boldsymbol{W}_{s8} = \|Q_w \boldsymbol{W}_{f32}\| \in [-127,127]$$

$$\boldsymbol{b}_{s32} = \|Q_a Q_w \boldsymbol{b}_{f32}\| \in [-2^{31}, 2^{31}-1]$$

関数 $\|\cdot\|$ は最も近い整数に丸めます。*s8* 形式は -128 をサポートしますが、量子化された *s8* 形式の重みの最小値は -127 を使用することに注意してください。

8 ビット乗算と 32 ビット累算を使用するアフィン変換の結果は次のとおりです。

$$\boldsymbol{x}_{s32} = \boldsymbol{W}_{s8}\boldsymbol{a}_{u8} + \boldsymbol{b}_{s32} \approx Q_a Q_w\left(\boldsymbol{W}_{f32}\boldsymbol{a}_{f32} + \boldsymbol{b}_{f32}\right) = Q_a Q_w \boldsymbol{x}_{f32}$$

近似は、式が丸め操作を無視しているためで、

$$\boldsymbol{x}_{f32} = \boldsymbol{W}_{f32}\boldsymbol{a}_{f32} + \boldsymbol{b}_{f32} \approx \frac{1}{Q_a Q_w}\boldsymbol{x}_{s32} = D\boldsymbol{x}_{s32}$$

は *f32* 形式のアフィン変換で、$D = {}^{1}/_{Qa\ Qw}$ は逆量子化係数です。

u8 および *s8* 形式への量子化では、ゼロ値は丸めなしで特定の値にマップされます。ゼロが最も一般的な値の 1 つであることを考えると、量子化エラーを軽減し統計精度を改善する正確なマップを使用したほうが良いでしょう。

上記の量子化係数は、インテル® Xeon® スケーラブル・プロセッサーでは fp32 形式にすることができます。ただし、一部のアーキテクチャー (例: FPGA) は除算をサポートしておらず、シフトを使用します。そのようなアーキテクチャーでは、スカラーは最も近い 2 の累乗に丸められ、スケーリングはビット・シフトで行われます。統計精度の低下は最小 (通常 1% 未満) に抑えられます。

効率良い 8 ビット の乗算

図 5 は、***A*** × ***W*** について 8 ビット乗算を効率良く実行する方法を示します。インテル® MKL-DNN は、活性化テンソルに *N H W C* データレイアウトを使用します。ここで、*N* はバッチサイズ、*H* は高さ、*W* は幅、*C* はチャネルの数です。そして、重みテンソルには、($^{O}/_{16}$)K ($^{C}/_{4}$)*T16o4c* データレイアウトを使用します。ここで、*O* はカーネルまたは出力チャネルの数、*C* は入力チャネルの数、*K* は高さ、*T* は幅です。

灰色で表示されたテンソル ***A*** の最初の 32 ビット (4 つの *int8* 値) は、512 ビット・レジスターを埋めるため 16 回ブロードキャストされます。インテル® MKL-DNN は、重みを量子化した後、テンソル ***W*** のデータレイアウトを変更します。テンソル ***W*** のデータレイアウトは、16 列のグループによって ***W′*** に再配置され、各列はメモリーで連続して読み取られる 32 ビット (4 つの int8 値) を保持し、列 1 の最初の 4 つの値はレジスターの最初の 32 ビット (赤)、次の 4x1 はレジスターの次の 32 ビット (オレンジ)、... (緑) というように格納されます。最初のブロックの下にある第 2、3、4 ブロック (黄) も同じパターンで再配置されます。そして、次のブロックのセット (青) が続きます。実際には、テンソル ***W*** は通常、4x1 のスキャッター値ではなく 1x4 の連続するメモリー値にアクセスするため、メモリーレイアウトを再配置する前に転置されます。このデータレイアウトの変更は通常一度行われ、すべての推論反復で再利用できるように格納されます。

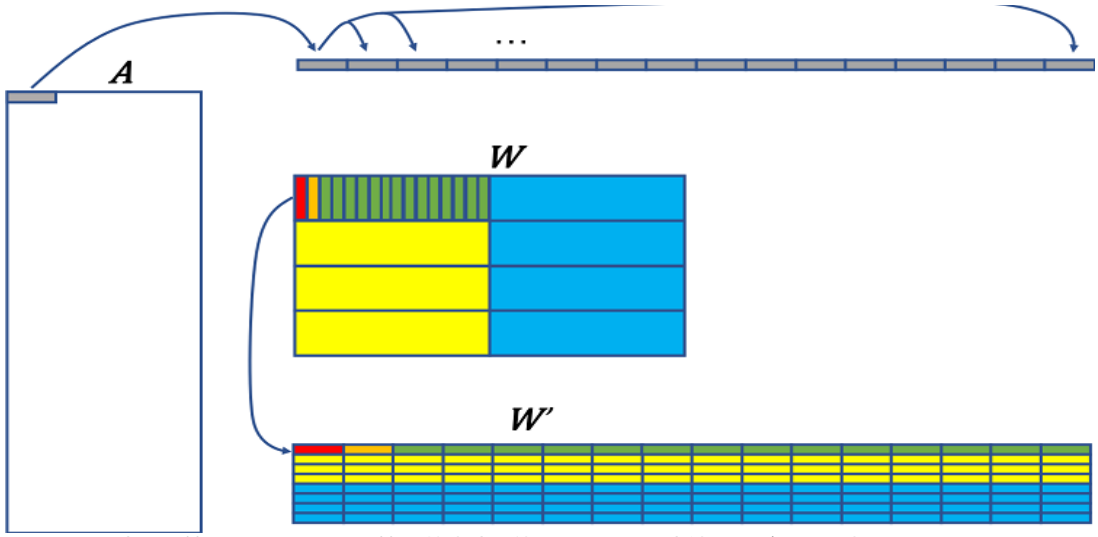

図 5. ***A×W*** の積を計算するため *int8* 乗算を効率良く使用するには、連続するビットを読み取るためテンソル ***W*** のデータレイアウト変換が必要です。***A*** の 32 ビットのグループは、テンソル ***W′*** の 512 ビットのグループと乗算される 512 ビット・レジスターを埋めるため 16 回ブロードキャストされます。

A の最初の 4 つの *int8* 値 (16 回コピーされる) は、***W′*** の 64 個の *int8* 値 (512 ビット) と乗算され、累算されます。***A*** の次の 4 つの値は、***W′*** の次の 64 個の *int8* 値と乗算される別のレジスターに 16 回ブロードキャストされます。これを ***A*** の最初の行が読み取られ、結果が集計されるまで繰り返し行います。(8 ビット FMA の 3 つの命令がすべて完了した時点の) 出力は、最初の 16 個の出力値です (*s32* では 512 ビットが必要です)。次に、***A*** の最初の行は、***W′*** の次の値と乗算され、出力の次の 16 個の値を生成します。

インテル® Xeon® スケーラブル・プロセッサーには最大 32 個の 512 ビット・レジスターがあります。2 つの FMA ユニットを備えたプロセッサーで 512 ビット・レジスター・ポート・スキームを実行すると、[3] ポート 0 の

FMA のレイテンシーは 4 サイクルで、ポート 5 の FMA のレイテンシーは 6 サイクルです。*int8* でディープラーニング・ワークロードに使用される命令はバイパスをサポートし、ポート 0 と 5 のレイテンシーはどちらも 5 サイクルです。実際には、これらのレイテンシーを隠蔽するため、W' の複数の行が複数のレジスターにロードされます。

トレーニング用の 16 ビット関数

インテル® MKL-DNN の AVX512_VNNI 命令を使用する 16 ビット乗算サポートは、将来この命令が利用可能になった場合に備えて設計されています。それにもかかわらず、研究者はすでに AVX512_4VNNI (QVNNI とも呼ばれ、一部のインテル® Xeon Phi™ プロセッサーで利用可能) の AVX512_4VNNI VP4DPWSSD 命令 (前述の AVX512_VNNI VPDPWSSD 命令に似ている) を利用して、16 ビット乗算と 32 ビット累算を使用してさまざまな CNN モデルのトレーニングを行っています (英語)。

そして、ハイパーパラメーターを変更せずに、*fp32* モデルと同じ反復回数で、ResNet-50、GoogLeNet-v1、VGG-16、AlexNet の *fp32* 統計パフォーマンスに匹敵する結果を達成しています。活性化、重み、勾配の格納に *s16* を使用し、各反復の後に *s16* に量子化される重みの更新のため *fp32* 重みのマスターコピーを保持しています。また、テンソル乗算により量子化/逆量子化係数の管理を容易にする 2 の累乗の量子化係数を使用しています。

フレームワークで低い数値精度を有効にする

一般的なフレームワークは、ユーザーがすべての関数定義を記述することなく、モデルを定義できるようにします。さまざまな関数の実装に関する詳細は、フレームワーク・ユーザーから見えないようにすることができます。これらの実装は、フレームワーク開発者によって行われます。このセクションでは、低い数値精度を有効にするため、フレームワーク・レベルで必要な変更について説明します。

重みの量子化は、推論を開始する前に行われます。活性化を効率良く量子化するには、量子化係数を事前に計算する必要があります。活性化の量子化係数は、通常、前述のように範囲を見つけるため検証データセットをサンプリングして、事前計算されます。範囲外のテスト・データセットの値は、その範囲に飽和されます。負の活性化値の場合、飽和前の範囲は *s8 = – 128* 値を使用するため、$-{}^{128R_{a'}}/_{127}$ にできます。ここで、$R_{a'}$ はこれらの活性化の最大絶対値です。そして、これらのスカラーがファイルに書き込まれます。

負の値を持つ活性化または入力の 8 ビット量子化

負の値を持つ活性化または入力の量子化は、次のように、フレームワーク・レベルで実装できます。
$Q_{a'} = {}^{127}/_{Ra'}$ は負の値を持つ活性化の量子化係数です。*s8* 量子化形式は $\boldsymbol{a}_{s8} = //Q_{a'}\ \boldsymbol{a}_{f32}\ // \in [-128, 127]$ です。ここで、関数 $||\cdot||$ は最も近い整数に丸めます。しかし、AVX512_VNNI VPMADDUBSW 命令または AVX512_VNNI VPDPBUSD 命令 (どちらも「インテル® Xeon® スケーラブル・プロセッサーで低い数値精度の利用」セクションで説明) を利用するため、活性化は *u8* 形式でなければなりません。そのため、$\boldsymbol{a}_{s8}$ のすべての値は、負ではなくなるように $K = 128$ でシフトされます。

$$\boldsymbol{a}_{u8} = \boldsymbol{a}_{s8} + K\mathbf{1} \in [0, 255]$$

ここで、$\mathbf{1}$ はすべての 1 のベクトルで、バイアス $\boldsymbol{b}_{f32}$ は次のように変更されます。

$$\boldsymbol{b}'_{f32} = \boldsymbol{b}_{f32} - \frac{K}{Q'_a}\boldsymbol{W}_{f32}\mathbf{1}$$

重みと変更したバイアスの量子化方法は、以前と同じです。

$$\boldsymbol{W}_{s8} = \|Q_w \boldsymbol{W}_{f32}\| \in [-128,127]$$

$$\boldsymbol{b'}_{s32} = \|Q_{a'} Q_w \boldsymbol{b'}_{f32}\| \in [-2^{31}, 2^{31}-1]$$

8 ビット乗算と 32 ビット累算を使用するアフィン変換は、次のようになります。

$$\boldsymbol{x}_{s32} = \boldsymbol{W}_{s8}\boldsymbol{a}_{u8} + \boldsymbol{b'}_{s32} \approx Q_w \boldsymbol{W}_{f32}\left(Q_{a'}\boldsymbol{a}_{f32} + K\mathbf{1}\right) + Q_w Q_{a'}\left(\boldsymbol{b}_{f32} - \frac{K}{Q'_a}\boldsymbol{W}_{f32}\mathbf{1}\right) =$$

$$Q_{a'}Q_w\left(\boldsymbol{W}_{f32}\boldsymbol{a}_{f32} + \boldsymbol{b}_{f32}\right) = Q_{a'}Q_w\boldsymbol{x}_{f32}$$

ここで、

$$\boldsymbol{x}_{f32} = \boldsymbol{W}_{f32}\boldsymbol{a}_{f32} + \boldsymbol{b}_{f32} \approx \frac{1}{Q_{a'}Q_w}\boldsymbol{x}_{s32} = D\boldsymbol{x}_{s32}$$

$D = {}^{1}/_{Qa'\ Qw}$ は逆量子化係数です。

入力信号はすでに *u8* 形式で (例: RGB イメージ)、平均信号を減算する前処理ステップが必要な場合、上記の式を使用できます。ここで、K は平均、$\boldsymbol{a}_{u8}$ は入力信号 (前処理されていない)、$Q_{a'} = 1$ です。

研究者は最初の畳み込み層を *fp32* 形式で保持し、ほかの畳み込み層を *int8* 形式にすることがよくあります (「ディープラーニングにおける低い精度の歴史」セクションを参照)。これらの量子化手法を利用することで、統計精度を大幅に低下させることなく、*int8* ですべての畳み込み層を使用できることが分かっています。

要約すると、負の値を持つ活性化を使用するには、活性化を *s8* 形式に量子化してから、*K=128* でシフトして *u8* 形式にします。追加の変更はバイアスの変更のみです。

$$\boldsymbol{b'}_{f32'} = \boldsymbol{b}_{f32} - {}^{K}/_{Q'a}\,\boldsymbol{W}_{f32}\,\boldsymbol{1}$$

畳み込み層の場合、積 $\boldsymbol{W}_{f32}\,\boldsymbol{1}$ は $\boldsymbol{b}_{f32}$ と共有される次元を除くすべての次元について、$\boldsymbol{W}_{f32}$ のすべての値の合計と等しくなるように一般化されます。詳細は付録 A を参照してください。

融合量子化

融合量子化は、次のように逆量子化と量子化を組み合わせることでパフォーマンスを向上するため、*fp32* への変換が不要です。層 *l+1* の活性化は次のとおりです。

$$\boldsymbol{a}_{f32}^{(l+1)} = g\left(\boldsymbol{x}_{f32}^{(l)}\right) = g\left(D^{(l)}\boldsymbol{x}_{s32}^{(l)}\right)$$

ここで、$g(\cdot)$ は非線形活性化関数です。ReLU 活性化関数であると仮定すると、次のように *u8* 形式で活性化を表現できます。

$$\boldsymbol{a}_{u8}^{(l+1)} = \left\|Q_a^{(l+1)}\boldsymbol{a}_{f32}^{(l+1)}\right\| = \left\|Q_a^{(l+1)}D^{(l)}\max\left(0, \boldsymbol{x}_{s32}^{(l)}\right)\right\|$$

ここで、積 $Q_a^{(l+1)}D^{(l)}$ は、*fp32* 表現を計算することなく、*u8* 形式で次の層の量子化された活性化を計算できるようにします。

$g(\cdot)$ が (次の式に示すように) ReLU 関数で、$Q \geq 0$ の場合、次の特性を持ちます。

$$Qg\left(D^{(l)}\boldsymbol{x}_{s32}^{(l)} + D^{(h)}\boldsymbol{x}_{s32}^{(h)}\right) = g\left(QD^{(l)}\boldsymbol{x}_{s32}^{(l)} + QD^{(h)}\boldsymbol{x}_{s32}^{(h)}\right)$$

この特性は、スキップ接続分岐がさまざまな活性化に依存する可能性がある、ResNet などのスキップ接続を持つモデルに役立ちます。例えば、Caffe の deploy.prototxt (英語) にある ResNet-50 の作者の命名法を使用すると (図 6 を参照)、層 *res2b_branch2a* (式では *2b2a* に省略して表記) の量子化された入力活性化は、次のようになります。

$$\begin{aligned}\boldsymbol{a}_{u8}^{(2b2a)} &= Q_a^{(2b2a)} g\left(D^{(2a1)}\boldsymbol{x}_{s32}^{(2a1)} + D^{(2a2c)}\boldsymbol{x}_{s32}^{(2a2c)}\right) \\ &= g\left(Q_a^{(2b2a)}D^{(2a1)}\boldsymbol{x}_{s32}^{(2a1)} + Q_a^{(2b2a)}D^{(2a2c)}\boldsymbol{x}_{s32}^{(2a2c)}\right)\end{aligned}$$

ここで、$au8^{(2b2a)} \in [0,127]$ ([0,255] ではない)。ReLU 関数の前に積があり、$Qa^{(2b2a)} = {}^{127}/_{Ra}{}^{(2b2a)}$ は量子化係数であるため、$Qa^{(2b2a)}\, D^{(2a1)}\boldsymbol{xs32}^{(2a1)} \in [-128, 127]$ は *s8* 形式です。この手順に従うと、付録 B に示すように、活性化 $a^{(2c2a)}{}_{u8}$ は $\boldsymbol{xs32}^{(2a1)}$、$\boldsymbol{xs32}^{(2a2c)}$、および $\boldsymbol{xs32}^{(2b2c)}$ に依存します。同様に、活性化 $\boldsymbol{au8}^{(3ca)}$ は $\boldsymbol{xs32}^{(2a1)}$、$\boldsymbol{xs32}^{(2a2c)}$、$\boldsymbol{xs32}^{(2b2c)}$、および $\boldsymbol{xs32}^{(2c2c)}$ に依存します。

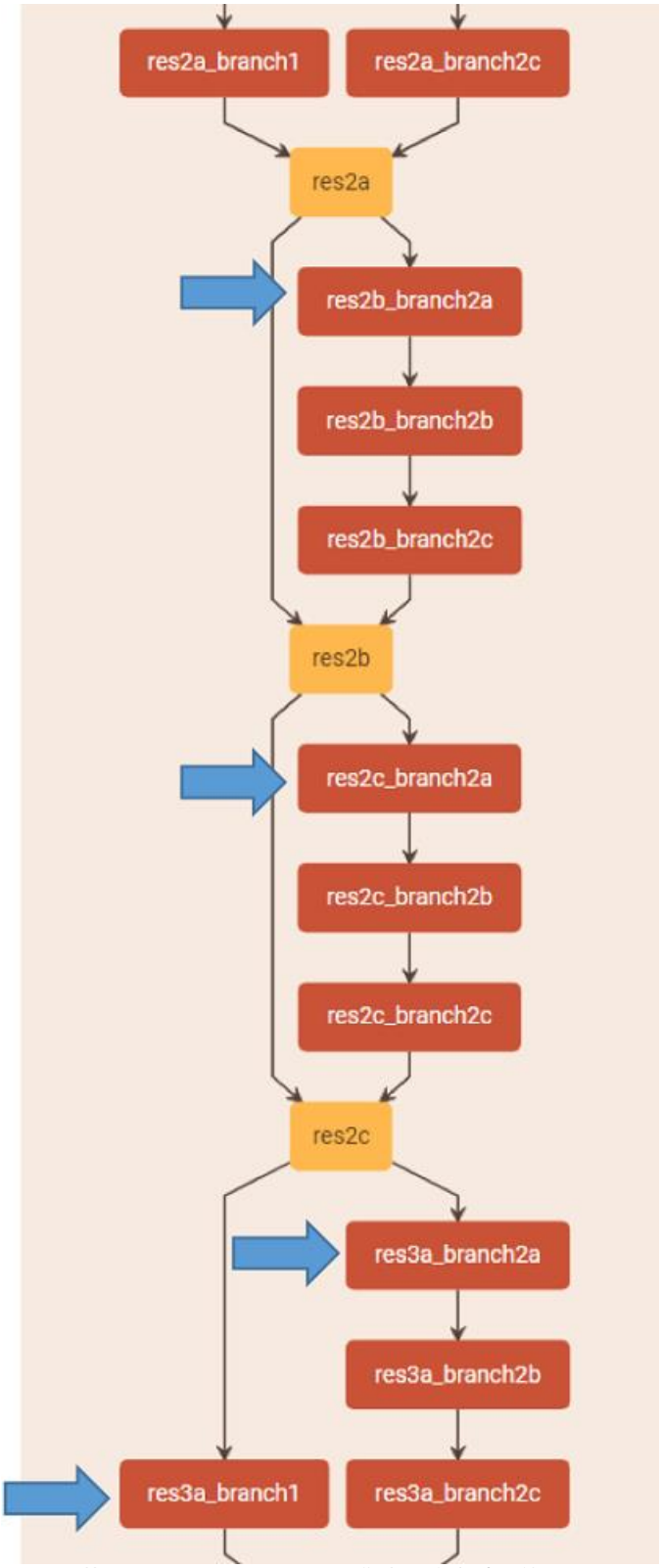

図 6. Caffe の deploy.prototxt (英語) にある ResNet-50 の作者の命名法を使用した ResNet-50 の残差ブロックの第 2 グループ (と第 3 グループの最初の分岐) の概略図。青い矢印でマークされた層は、2 つ以上の活性化に依存します。画像の出典: Barukh Ziv、Etay Meiri、Eden Segal。

バッチ正規化

バッチ正規化 (BN) 推論層は、重みの値をスケーリングしバイアスを変更することで前の層に吸収できるため不要です。この手法は推論にのみ適用可能で、低い数値精度に固有のものではありません。インテル® MKL-DNN の代わりに、フレームワーク・レベルで実装することもできます。BN は通常、アフィン変換 ***x = Wa + b*** の後、かつ活性化関数の前に適用されます (詳細は、オリジナルの BN 論文 (英語) を参照)。BN は、***x*** をゼロ平均および単位ノルムに正規化して、正規化したベクトルをトレーニング中に学習したパラメーター γ と β でそれぞれスケールおよびシフトします。トレーニング反復中、***x*** はミニバッチ統計を使用して正規化されます。例えば、***x*** の平均 *E* と分散 *V* は、トレーニングセット全体の統計、またはトレーニング中に計算されたこれらの統計の移動平均などを使用して事前計算されます。推論中、BN 出力 ***y*** は次のように求められます。

$$\boldsymbol{y} = BN(\boldsymbol{x}) = \gamma\frac{\boldsymbol{x} - E\mathbf{1}}{V} + \beta\mathbf{1} = \gamma\frac{\boldsymbol{Wa} + \boldsymbol{b} - E\mathbf{1}}{V} + \beta\mathbf{1} = \frac{\gamma}{V}\boldsymbol{Wa} + \frac{\gamma}{V}\boldsymbol{b} + \left(\beta - \frac{\gamma E}{V}\right)\mathbf{1} = \boldsymbol{W'a} + \boldsymbol{b'}$$

ここで、**W’** = $\gamma/_V$ **W** および **b’** = $\gamma/_V$ **b** + (β − $\gamma E/_V$)**1**。つまり、推論中、先行する畳み込み層または全結合層で重みとバイアスを調整して BN 層を省略することができます。

フレームワーク

インテルは、インテル® Optimization for Caffe (英語) で 8 ビットの推論を実現しました。インテルの DL 推論エンジン (英語)、Apache* MXNet (英語)、TensorFlow* (英語) の最適化は、2018 年 Q2 にリリースされました。この記事の執筆時点では、これらの 8 ビットの最適化はすべて CNN モデルに制限されています。RNN モデルとその他のフレームワークについても今後対応予定です。

インテル® Optimization for Caffe では、図 7 に示すように事前計算済みスカラーを含むように model.prototxt ファイルが変更されています。この記事の執筆時点では、インテル® Optimization for Caffe は 2 の累乗または通常の *fp32* 値の量子化係数を提供しており、テンソルごとまたはチャネルごとに 1 つの量子化係数を使用できます。これらの量子化係数は、インテル® Optimization for Caffe に組込まれているサンプリング・ツールを使用して計算されます。

```
layer {
  name: "conv2"  type: "Convolution"
  …
  quantization_param {
    precision: DYNAMIC_FIXED_POINT
    bw_layer_in: 8    // input bit-width
    bw_layer_out: 8   // output bit-width
    bw_params: 8      // weights bit-width
    fl_layer_in: 0    // input fraction length
    fl_layer_out: -2  // output fraction length
    fl_params: 8      // weights fraction length
  }
}
```

図 7. model.prototxt ファイルに量子化係数を追加。画像の出典 Haihao Shen。

インテルのディープラーニング推論エンジン (英語) は、インテル® ディープラーニング・デプロイメント・ツールキット (英語) とインテル® コンピューター・ビジョン SDK に含まれています。Linux* および Windows* で利用可能で、最初は Caffe、MXNet、および TensorFlow* フレームワークでトレーニングされたモデルをサポートします。推論エンジンは、さまざまなハードウェア・バックエンド向けに、統一された API を提供することで、DL ソリューションのデプロイメントを容易にします。インテル® AVX2 およびインテル® AVX-512 対応

インテル® Xeon® プロセッサー、Intel Atom® プロセッサー、インテル® HD グラフィックス、インテル® Arria® 10 (インテル® A10) ディスクリート・カードなど、ハードウェアに応じてさまざまな数値精度で推論を行うことができます。インテル® Xeon® スケーラブル・プロセッサーでは 8 ビットの推論をサポートします。

TensorFlow* は 8 ビットの推論とさまざまな量子化手法 (英語) をサポートしています。スケールを動的に計算したり、トレーニングやキャリブレーション中に統計を収集して、量子化係数を割り当てることができます。これらのスカラーを含む TensorFlow* グラフは、ファイルに出力されます。スカラーを持つグラフは、推論中に量子化され、実行されます。TensorFlow* は 2 つの量子化手法をサポートします。1 つは、インテル® MKL-DNN と似ており、加法に関する逆元として最小値と最大値を設定します。もう 1 つは、最小値と最大値に任意の値を使用し、オフセットとスケールを必要とします (インテル® MKL-DNN ではサポートされていません)。詳細は、Pete Warden 氏のブログ (英語) を参照してください。ただし、大分前に執筆されたブログであるため、TensorFlow* で量子化するすべての方法が含まれているわけではないことに注意してください。

また、TensorFlow* では、低い数値精度で再トレーニングまたは微調整を行うことができます。微調整は統計パフォーマンスを向上します。*fp32* でトレーニングしたモデルの場合、重みを量子化した後、量子化した重みでモデルを微調整して、各トレーニング反復の後に重みを再量子化します。

GemmLowP (英語) は、TensorFlow* Lite で採用されている Google* ライブラリーです。u8 乗算を使用します: $f32 = D \times (u8 - K)$。ここで、*K* は *fp32 = 0* にマップする *u8* 値、*D > 0* は逆量子化係数です。

この記事の執筆時点では、Apache* MXNet ブランチは 8 ビットをサポートしていません。ただし、MXNet の主なコントリビューターの 1 つのブランチ (英語) は 8 ビットの推論をサポートしています。このブランチでは、2 つの量子化手法を利用できます。1 つは、最小値が 0 にマップされ、最大値が 255 にマップされます (ゼロは正確な値にマップされないことに注意してください)。もう 1 つ (英語) は、最大絶対値が -127 または 127 のいずれかにマップされます (インテル® MKL-DNN と同様にゼロはゼロにマップされます)。この記事の手法との主な違いは、この MXNet ブランチのスカラーは事前計算されません。実際の推論ステップ中に計算されるため、低い数値精度の利点が損なわれます。このブランチでは、活性化のスカラーは入力からのスカラーと重みからのスカラーを乗算して計算されます: 活性化スカラー = 入力スカラー * 重みスカラー。ここで、入力 = 入力スカラー * 量子化された入力、重み = 重みスカラー * 量子化された重み、活性化 = 活性化スカラー * 量子化された活性化。入力、重み、活性化、およびスカラーは *fp32* 形式、量子化された入力と量子化された重みは *int8* 形式、量子化された活性化は *int32* 形式です (詳細 (英語))。活性化の最小値と最大値が保持される一方で、*fp32* 層に遭遇したときにのみ値が逆量子化されます (例: ソフトマックス)。

TensorRT は、インテル® MKL-DNN と同様に *s8* 形式に量子化 (英語) し、さらに量子化分布と参照分布の間の KL 発散を最小限に抑えることでより狭い範囲を見つけます。

TPU チームは、*int8* 乗算を使用する TPU は LSTM モデルを含むさまざまなモデルで使用されていると主張 (英語) しています。ソフトウェア・スタックは、TensorFlow* グラフからの API 呼び出しを TPU 命令に変換 (英語) します。

Caffe2 のドキュメントでは、「量子化計算などの将来に対する柔軟性」があると述べて (英語) いますが、この記事の執筆時点では量子化の計画は明らかにされていません。

PyTorch* にはさまざまな量子化オプションを提供するブランチ (英語) がありますが、どれが優れいているかについての議論はありません。

Microsoft* は、インテル® Stratix® 10 FPGA 上で実行するカスタムの 8 ビット浮動小数点形式 (*ms-fp8*) を使用する Project Brainwave (英語) を発表しました。この形式、量子化手法、フレームワーク実装の詳細は、この記事の執筆時点ではまだ公開されていません。Project Brainwave は、CNTK と TensorFlow* をサポートしており、主要フレームワークでトレーニングされたモデルをグラフベースの内部中間表現に変換することで、今後サポートを拡大していく予定です。

モデルとグラフの最適化

モデルの最適化により推論パフォーマンスをさらに向上できます。例えば、ResNet では、図 8 に示すように、最終結果を変更することなくストライド操作を先行する層へ移動し、操作数を減らすことができます。この変更は、8 ビットと 32 ビットの両方に適用されます。

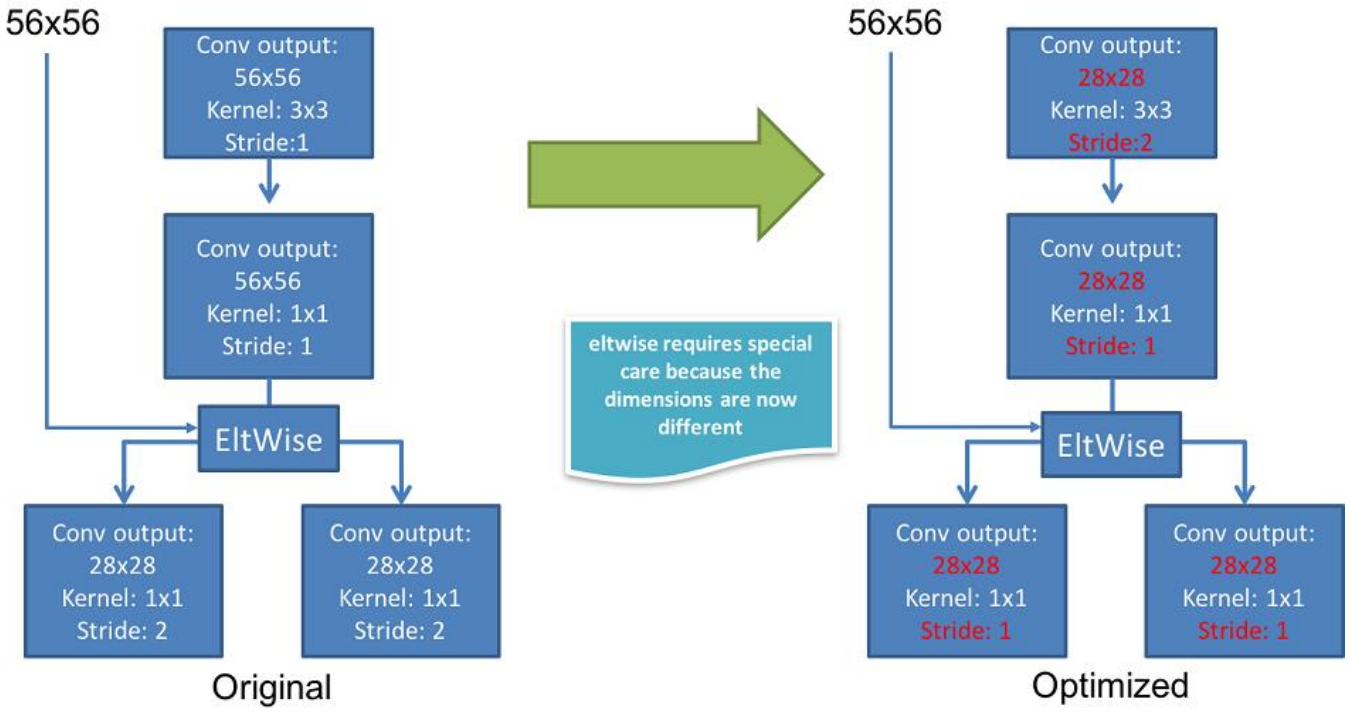

図 8. 左のブロックのストライド 2 の層は推論中の先行する層へ移動できます。これにより、結果を変更することなく操作数を減らせます。出典: Eden Segal および Etay Meiri。

まとめ

低い数値精度の推論とトレーニングは、統計精度をほとんどまたは全く損なうことなく、計算パフォーマンスを向上できます。インテルは、第 1 世代インテル® Xeon® スケーラブル・プロセッサー上で 8 ビット精度の推論を可能にします。また、ハードウェアおよびソフトウェア対応コンパイラー、インテル® MKL-DNN ライブラリー、および主要ディープラーニング・フレームワークにおいて、将来のマイクロプロセッサーで 8 ビット精度の推論と 16 ビット精度のトレーニングを実現します。

謝辞

有用な意見を提供してくれたフレームワーク最適化チームのリーダー達、Israel Hirsh、Alex Heinecke、Vadim Pirogov、Frank Zhang、Rinat Rappoport、Barak Hurwitz、Dipankar Das、Dheevatsa Mudigere、Naveen Mellempudi、Dhiraj Kalamkar、Bob Valentine、AG Ramesh、Nagib Hakim をはじめとするインテル® Xeon® プロセッサーのアーキテクトと研究者の皆さん、並びに素晴らしいレビューを行ってくれた R. Chase Adams、Nikhil Murthy、Banu Nagasundaram、Todd Wilson、Alexis Crowell、Emily Hudson に謝意を表します。

著者紹介

Andres Rodriguez 博士: データセンター・グループ (DCG) および AI 製品グループ (AIPG) 所属のシニア主席エンジニアで、インテルの顧客向けに AI ソリューションを設計し、インテルにおける AI 製品の技術リーダーとして活躍しています。AI 分野で 13 年の経験があり、マシンラーニングの研究でカーネギーメロン大学から博士号を取得しています。マシンラーニングに関して、雑誌、会議、書籍の章で 20 を超える査読論文を発表しています。

Eden Segal: Pre-Enabling チームのソフトウェア開発者として、インテル® プロセッサー上でのピーク・アルゴリズム・パフォーマンスを見つけるためディープラーニング・アルゴリズムの最適化に取り組んでいます。ハードウェア (ライブラリーを介して) からディープラーニング・フレームワークに至る、インテルのディープラーニング・スタック全体のパフォーマンスを向上にこの知識が活かされています。

Etay Meiri: Pre-Enabling チームのソフトウェア開発者として、インテル® プロセッサー上でのピーク・アルゴリズム・パフォーマンスを見つけるためディープラーニング・アルゴリズムの最適化に取り組んでいます。ハードウェア (ライブラリーを介して) からディープラーニング・フレームワークに至る、インテルのディープラーニング・スタック全体のパフォーマンスを向上にこの知識が活かされています。

Evarist Fomenko: 応用数学の修士号を持つ、インテル® MKL とインテル® MKL-DNN のソフトウェア開発エンジニアで、ライブラリー関数の設計と最適化に取り組むとともに、社内外のチームの統合を支援しています。インテルでハードウェアの最適化において 5 年の経験があります。

Young Jin Kim 博士: AI 製品グループ (AIPG) のシニア・マシンラーニング・エンジニアで、最先端の手法を利用してインテルのハードウェア・アーキテクチャー向けにディープラーニング・ソフトウェア・フレームワークの開発および最適化を行っています。AI において 10 年以上の経験があります。ディープラーニングとハイパフォーマンス・コンピューティングに関する研究でジョージア工科大学から博士号を取得しています。ジャーナルや会議における査読出版物が 10 以上あります。

Haihao Shen: コンピューター・サイエンスの修士号を持つ、ソフトウェア & サービスグループ (SSG) のマシンラーニング & トランスレーション・チームのディープラーニング・エンジニアです。低い精度の推論とモデル最適化を含む、インテル® Optimization for Caffe の開発を主導しています。インテルでソフトウェアの最適化と検証において 6 年の経験があります。上海交通大学を卒業しています。

Barukh Ziv 博士: SSG Pre-Enabling グループのシニア・ソフトウェア・エンジニアで、将来の世代のインテル® Xeon® プロセッサー向けに DL アプリケーションの効率良い実装を設計しています。DL 最適化において 2 年の経験があります。カウナス工科大学でテクニカルサイエンスの博士号を取得しています。ジャーナルや会議における査読出版物が 5 以上あります。

付録 A: 負の値を持つ活性化または入力の量子化の詳細

同じ式 (「負の値を持つ活性化または入力の 8 ビット量子化」セクションを参照) が畳み込み層に一般化されることを皆さんに示すため、各テンソルエントリーのインデックスを使用して畳み込み出力を求める手順を説明します。$\boldsymbol{W}_{f32} \in \mathbb{R}^{O\times C\times K\times T}$ は、O 個のカーネルまたは出力チャネルと C 個の入力チャネルを持つ、高さ K、幅 T の重みテンソルです。変更済みのバイアスは次のように表すことができます。

$$b'_{f32}[o_i] = b_{f32}[o_i] - \frac{K}{Q'_a}\sum_{c_i}\sum_{\kappa_i}\sum_{\tau_i} W_{f32}[o_i, c_i, \kappa_i, \tau_i] = b_{f32}[o_i] - \frac{K}{Q'_a}\bar{W}_{f32}[o_i]$$

ここで、

$$\bar{W}_{f32}[o_i] = \sum_{c_i}\sum_{\kappa_i}\sum_{\tau_i} W_{f32}[o_i, c_i, \kappa_i, \tau_i]$$

また、o_i、c_i、κ_i、および τ_i はそれぞれカーネルまたは出力チャネル、入力チャネル、カーネルの高さ、カーネルの幅のインデックスです。畳み込み出力は、次のように表すことができます。バッチサイズは 1 (バッチインデックスを省略して説明を簡潔にするため)、活性化はすでに *fp32* 形式でゼロパディング (または *u8* 形式で *K=128* を使用して同等のパディング) が適用されており、畳み込みストライドは 1 であると仮定します。

$$\begin{aligned}
x_{s32}[o_i, h_i, w_i] &= b'_{s32}[o_i] + \sum_{c_i}\sum_{\kappa_i}\sum_{\tau_i} a_{u8}[c_i, h_i+\kappa_i, w_i+\tau_i] W_{s8}[o_i, c_i, \kappa_i, \tau_i] \\
&\approx Q_{a'}Q_w b'_{f32}[o_i] + \sum_{c_i}\sum_{\kappa_i}\sum_{\tau_i}\left(Q_{a'}a_{f32}[c_i, h_i+\kappa_i, w_i+\tau_i] + K\right)Q_w W_{f32}[o_i, c_i, \kappa_i, \tau_i] \\
&= Q_{a'}Q_w\left(b_{f32}[o_i] - \frac{K}{Q'_a}\bar{W}_{f32}[o_i]\right) + \sum_{c_i}\sum_{\kappa_i}\sum_{\tau_i} Q_w Q_{a'} a_{f32}[c_i, h_i+\kappa_i, w_i+\tau_i] W_{f32}[o_i, c_i, \kappa_i, \tau_i] \\
&\qquad + \sum_{c_i}\sum_{\kappa_i}\sum_{\tau_i} K Q_w W_{s8}[o_i, c_i, \kappa_i, \tau_i] \\
&= Q_{a'}Q_w b_{f32}[o_i] - K Q_w \bar{W}_{f32}[o_i] + \sum_{c_i}\sum_{\kappa_i}\sum_{\tau_i} Q_w Q_{a'} a_{f32}[c_i, h_i+\kappa_i, w_i+\tau_i] W_{f32}[o_i, c_i, \kappa_i, \tau_i] \\
&\qquad + K Q_w \bar{W}_{f32}[o_i] \\
&= Q_{a'}Q_w\left(b_{f32}[o_i] + \sum_{c_i}\sum_{\kappa_i}\sum_{\tau_i} a_{f32}[c_i, h_i+\kappa_i, w_i+\tau_i] W_{f32}[o_i, c_i, \kappa_i, \tau_i]\right) \\
&= Q_{a'}Q_w x_{f32}[o_i, h_i, w_i]
\end{aligned}$$

付録 B: スキップ接続による融合量子化の詳細

図 6 の青い矢印でマークされた層への活性化入力は次のとおりです。以下の式では、層 *res2b_branch2a* は *2b2a* とし、ほかの層についても同様に省略して記載しています。

$$\begin{aligned}\boldsymbol{a}_{u8}^{(2b2a)} &= Q_a^{(2b2a)}\boldsymbol{a}_{f32}^{(2b2a)}\\ &\approx Q_a^{(2b2a)}g\left(D^{(2a1)}\boldsymbol{s}_{32}^{(2a1)}+D^{(2a2c)}\boldsymbol{s}_{32}^{(2a2c)}\right)\\ &= g\left(Q_a^{(2b2a)}D^{(2a1)}\boldsymbol{s}_{32}^{(2a1)}+Q_a^{(2b2a)}D^{(2a2c)}\boldsymbol{s}_{32}^{(2a2c)}\right)\end{aligned}$$

$$\begin{aligned}\boldsymbol{a}_{u8}^{(2c2a)} &= Q_a^{(2c2a)}\boldsymbol{a}_{f32}^{(2c2a)}\\ &\approx Q_a^{(2c2a)}g\left(\boldsymbol{a}_{f32}^{(2b2a)}+D^{(2b2c)}\boldsymbol{s}_{32}^{(2b2c)}\right)\\ &\approx Q_a^{(2c2a)}g\left(g\left(D^{(2a1)}\boldsymbol{s}_{32}^{(2a1)}+D^{(2a2c)}\boldsymbol{s}_{32}^{(2a2c)}\right)+D^{(2b2c)}\boldsymbol{s}_{32}^{(2b2c)}\right)\\ &= g\left(g\left(Q_a^{(2c2a)}D^{(2a1)}\boldsymbol{s}_{32}^{(2a1)}+Q_a^{(2c2a)}D^{(2a2c)}\boldsymbol{s}_{32}^{(2a2c)}\right)+Q_a^{(2c2a)}D^{(2b2c)}\boldsymbol{s}_{32}^{(2b2c)}\right)\end{aligned}$$

$$\begin{aligned}\boldsymbol{a}_{u8}^{(3a2a)} &= Q_a^{(3a2a)}\boldsymbol{a}_{f32}^{(3a2a)}\\ &\approx Q_a^{(3a2a)}g\left(\boldsymbol{a}_{f32}^{(2c2a)}+D^{(2c2c)}\boldsymbol{s}_{32}^{(2c2c)}\right)\\ &\approx Q_a^{(3a2a)}g\left(g\left(g\left(D^{(2a1)}\boldsymbol{s}_{32}^{(2a1)}+D^{(2a2c)}\boldsymbol{s}_{32}^{(2a2c)}\right)+D^{(2b2c)}\boldsymbol{s}_{32}^{(2b2c)}\right)+D^{(2c2c)}\boldsymbol{s}_{32}^{(2c2c)}\right)\\ &= g\left(g\left(g\left(Q_a^{(3a2a)}D^{(2a1)}\boldsymbol{s}_{32}^{(2a1)}+Q_a^{(3a2a)}D^{(2a2c)}\boldsymbol{s}_{32}^{(2a2c)}\right)+Q_a^{(3a2a)}D^{(2b2c)}\boldsymbol{s}_{32}^{(2b2c)}\right)\right.\\ &\qquad\left.+Q_a^{(3a2a)}D^{(2c2c)}\boldsymbol{s}_{32}^{(2c2c)}\right)\end{aligned}$$

$$\begin{aligned}\boldsymbol{a}_{u8}^{(3a1)} &= Q_a^{(3a1)}\boldsymbol{a}_{f32}^{(3a1)}\\ &\approx Q_a^{(3a1)}g\left(\boldsymbol{a}_{f32}^{(2c2a)}+D^{(2c2c)}\boldsymbol{s}_{32}^{(2c2c)}\right)\\ &\approx Q_a^{(3a1)}g\left(g\left(g\left(D^{(2a1)}\boldsymbol{s}_{32}^{(2a1)}+D^{(2a2c)}\boldsymbol{s}_{32}^{(2a2c)}\right)+D^{(2b2c)}\boldsymbol{s}_{32}^{(2b2c)}\right)+D^{(2c2c)}\boldsymbol{s}_{32}^{(2c2c)}\right)\\ &= g\left(g\left(g\left(Q_a^{(3a1)}D^{(2a1)}\boldsymbol{s}_{32}^{(2a1)}+Q_a^{(3a1)}D^{(2a2c)}\boldsymbol{s}_{32}^{(2a2c)}\right)+Q_a^{(3a1)}D^{(2b2c)}\boldsymbol{s}_{32}^{(2b2c)}\right)\right.\\ &\qquad\left.+Q_a^{(3a1)}D^{(2c2c)}\boldsymbol{s}_{32}^{(2c2c)}\right)\end{aligned}$$

1. 次のように計算できます: インテル® AVX-512 の周波数 * コア数 * コアごとの FMA 数 * FMA ごとに 2 操作 * SIMD ベクトル長 / 数値形式のビット数 / 命令数。インテル® Xeon® Platinum プロセッサー、インテル® Xeon® Gold 6xxx/5122 プロセッサーでは、コアごとにポート 0 と 5 にある 2 つの 512 ビット FMA ユニットを利用して並列に処理できます。ほかのインテル® Xeon® スケーラブル・プロセッサー SKU では、コアごとにポート 0 に 1 つの FMA ユニットがあります。インテル® AVX-512 命令では、*fp32*、*int16*、および *int8* FMA はそれぞれ 1、2、および 3 命令を必要とします。インテル® Xeon® Platinum 8180 は、ソケットごとに 28 コア、コアごとに 2 つの FMA があります。ソケットごとの *fp32* OPS の概算は、インテル® AVX-512 周波数 1.99GHz * 28 コア * コアごとに 2 つの FMA ユニット * FMA ごとに 2 つの OPS * 512 ビット/32 ビット/1 命令 = 3.570 *fp32* TOPS です。ソケットごとの *int8* OPS の概算は、インテル® AVX-512 周波数 2.17GHz * 28 コア * コアごとに 2 つの FMA ユニット * FMA ごとに 2 つの OPS * 512 ビット/8 ビット/3 命令 = 5.185 *int8* TOPS です。各 SKU のインテル® AVX-512 の周波数は、こちら (英語) で確認できます (これらの周波数は *fp64* 操作に対応したものです。数値精度が低いほど周波数は高くなり、上記の fp32 と int8 TOPS の計算に使用されたものは推定値です)。インテル® AVX-512 の最大ターボ周波数は、高 OPS ワークロードを実行中は、完全に維持されないことがあります。
2. 実際には、ReLU 活性化関数により処理されたアクティベーションの場合、これらの *u8* 値は最大値よりも最小値に近くなることが多いです。

3. インテル® Xeon® Platinum プロセッサー、インテル® Xeon® Gold 6xxx/5122 プロセッサーでは、コアごとに 2 つの 512 ビット FMA ユニット (並列に処理) を利用できます。その他のインテル® Xeon® スケーラブル・プロセッサーでは、コアごとに 1 つの FMA ユニットを利用できます。

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。