DNN アクセラレータ用高精度 NN モデルの開発手法に関する検討

1180393　吉 本　大 輔　【 コンピュータ構成学研究室 】

1　はじめに

近年、大規模な深層ニューラルネットワーク DNN(Deep Neural Network) を低速な IoT 機器でも実行する要求が高まっている．そのため，膨大な計算量を許容できる，高速かつ低電力な DNN 専用アクセラレータが各所で開発されている．しかし，いずれもハードウェアコストを削減するために，固定小数点演算あるいは低精度浮動小数点数演算回路しか搭載していないため，所望の認識精度を得るために様々な方法が検討されている．例えば，Minerva[1] は，中間データの精度を最適化してアクセラレータ回路をアルゴリズム水準から協調設計する方法論である．一方で，Intel 社や NVIDIA 社などから多くの DNN アクセラレータが既に商用化されており，これらを効果的に活用する方法論はあまり公開されていない．

そこで本研究では，Intel 社 NCS (Movidius Neural Compute Stick)[2] を対象に，DNN モデルの学習，学習後モデルの作成，実運用の各フェーズでの工夫によって，アクセラレータによる DNN の精度向上が可能かどうか検証した．

2　python 環境と HW 環境の比較検討

本研究で用いた NCS は，内部データ形式に 12bit 浮動小数点数を用いた VLIW プロセッサを複数搭載しており，USB 接続で運用できる DNN 専用ハードウェアである．図 1 は，DNN のオフライン学習から NCS の実運用までの全体的なデータの流れである．よって，NCS を Raspberry Pi などの安価で低速な IoT 機器に USB 接続して，事前に高性能 PC 上で tensorflow 等を用いて学習した DNN モデルをダウンロードすれば，大規模な DNN による高度な処理を実現できる．しかし，回路コストと演算精度のトレードオフの影響から，学習した環境上でテストした認識精度を得られない恐れがある．そこで，モデル構築時に用いる float 型の違いが認識精度に影響を与えるかを，手書き文字認識用データセット MNIST/EMNIST と通常の CNN を用いてこれらの問題を検証した．

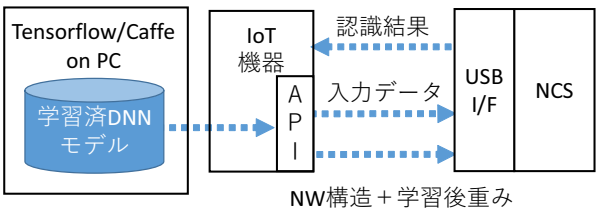

図 1　DNN アクセラレータを活用する際のデータの流れ

表 1　NCS による MNIST/EMNIST の正答率 [%]

learning set		MNIST		EMIST
test set		MNIST	EMNIST	EMINST
python		98.35	17.49	97.55
NCS	SDK	98.30	16.64	93.33
	proposed	98.34	16.66	93.76

具体的には，まず，tensorflow での学習時には float32 を用いた．NCS SDK 内の inception サンプルでは，NCS 用 DNN モデルのコンパイル時に float32 が用いられている．本研究では，この時に float16 を用いた場合の認識精度についても調査した．これは，学習段階では float32 を用いた影響で，DNN モデルのコンパイル時に桁落ちによる劣化が発生する可能性があるためである．言い換えれば，モデル変換時に，実際に NCS を動作させるプログラム内での演算精度 12bit に近づけたほうが認識精度が向上する可能性があると考えたからである．

3　検証評価

検証評価では，python 上の tensorflow 環境，提案手法を適用した NCS 環境，適用しない NCS 環境について，MNIST/EMNIST のテストデータセットの正答率を比較した．本評価は，ubuntu16.04 上で，python2.7.12，tensorflow1.4.1，NCS SDK1.09 を用いて実施した．用いた MNIST 用 DNN は，入力層 28 × 28，畳み込み層 5 × 5 × 16，全結合層 10 とした．学習回数は，MNIST 学習セットの場合 4000 回，EMNIST 学習セットの場合 8000 回とした．各学習セットとテストデータセットを組み合わせた場合の認識精度を表 1 に示す．結果から，DNN モデルのコンパイル時に float16 を用いると僅かながら python 環境との差を縮めることができた．

参考文献

[1] B. Reagen, et al., “Minerva: Enabling Low-Power, Highly-Accurate Deep Neural Network Accelerators”, ISCA ’16, pp. 267–278, 2016.

[2] “Intel Movidius Neural Compute Stick AI Programming,” https://developer.movidius.com/, accessed in Dec. 25, 2017.