東京大学
情報理工学系研究科 創造情報学専攻
修士論文

Web ブラウザ上での高速な深層ニューラルネットワークの推論手法に関する研究

Acceleration Deep Neural Networks Executed on Web Browser

木倉 悠一郎
Yuichiro Kikura

指導教員　原田 達也 教授

2018 年 1 月

概要

TODO

Abstract

ToDo: ACMMM のコピーしただけ

Recently, deep neural network (DNN) is drawing a lot of attention because of its applications. However, it requires a lot of computational resources and tremendous processes in order to setup an execution environment based on hardware acceleration such as GPGPU. Therefore, providing DNN applications to end-users is very hard. To solve this problem, we have developed an installation-free web browser-based DNN execution framework, WebDNN. WebDNN optimizes the trained DNN model to compress model data and accelerate the execution. It executes the DNN model with novel JavaScript API to achieve zero-overhead execution. Empirical evaluations show that it achieves more than two-hundred times the unusual acceleration. WebDNN is an open source framework and you can download it from `https://github.com/mil-tokyo/webdnn`.

目次

第 1 章

序論

1.1　深層学習の台頭

近年，機械学習の様々な分野において深層ニューラルネットワーク（Deep Neural Network・DNN）が注目されている．DNN とは，階層が深いニューラルネットワーク（NN）の総称である．以前は学習が困難とされてきたが，情報端末の普及に伴う学習データ量の爆発的な増加，計算機性能の飛躍的な発達，学習理論の解明などに伴い，様々な分野で既存手法を凌駕する優れた成果を上げている．例えば，画像認識の分野では，DNN が人間を上回る認識精度を達成している (ToDo: 引用)．他にも，物体認識（ToDo: 引用），音声合成（ToDo: 引用），動画像生成（ToDo: 引用），自然言語処理（ToDo: 引用），医療診断（ToDo: 引用），など様々な分野で DNN は優れた成果を上げており，今後も幅広い分野での応用が期待されている．

しかしながら，DNN の実社会への応用には幾つかの問題が存在している．その 1 つが，DNN の実行には大量の計算資源を必要とすることである．一般に DNN モデルの学習には GPU などの汎用並列演算に特化したデバイスを多数使用し，数日から数週間程度の時間を要するため，一般家庭などのエンドユーザ端末上で学習を実行することは現状不可能である．学習済みモデルの推論実行には学習ほどの計算リソースは必要としないものの，エンドユーザー端末上での実行には数秒から数分の時間を要するモデルが多く，研究成果をすぐに実アプリケーションへと応用することは難しいのが現実である．特にエンドユーザーには，汎用並列計算に対応した GPU 計算環境を用意することは非常に難しく，手間がかかる．ToDo: あとで何とかする

特殊な計算環境を用意すること無く，実行環境のデバイス・オペレーティングシステム (OS) に高度にアプリケーションをチューニングすることで，エンドユーザーデバイスでも推論を高速に実行する手法も存在する．しかし，エンドユーザーデバイスには様々なデバイス・OS が存在し，その全てをそれぞれターゲットとして個別にチューニングすることは多大な費用を要し現実的ではない．

1.2　Web アプリケーションの普及

ToDo: 片仮名多すぎ

計算機性能の飛躍的な発達は深層学習の台頭だけではなく，Web 技術の発達をも大きく加速させた．ToDo: 日本語でおｋ．Web ブラウザは従来，インターネット上の文書を閲覧するためのアプリケーションであった．しかし，Web ブラウザの実行速度の上昇に伴い，文書の閲覧だけでなくメールの管理や地図の表示，ファイルの管理・編集などをこなす Web アプリケーションが登場した．これに伴い Web ブラウザが Web アプリケーション開発者へ提供する機能（API）も拡充され，現在では様々なアプリケーションが Web ブラウザから利用可能となっている．Web ブラウザは一般に様々なユーザーエンド端末にプリインストールされており，ユーザーは特別なセットアップ手順などを行うこと無く，簡単に Web アプリケーションへアクセスすることが可能である．また，開発者はこれまで様々なユーザーエンド端末・OS に合わせてアプリケーションを開発する必要があったが Web アプリケーションであれば，Web ブラウザという単一の実行環境のみをターゲットとすればよいため，開発コストを大幅に下げることが可能となる．

1.3　本研究の目的と貢献

本研究の目的は，Web ブラウザ上で DNN を高速実行するための手法の開発である．前述のように，今日の Web ブラウザは開発者・利用者双方にとって低コストでアプリケーションの開発・利用が可能なアプリケーション実行環境である．Web ブラウザ上で DNN を現実的な速度で実行することが可能になれば DNN の実社会への普及における問題点であった，「エンドユーザー環境への計算環境のセットアップ」という問題点が克服され，DNN を利用したアプリケーションが飛躍的に普及するものと期待される．本研究では，本来大量の計算資源が必要である DNN の推論を，Web ブラウザ上で効率的に実行するための枠組みを提案・実装し，Web ブラウザ上で DNN の推論が実用的速度で実行可能であることを実証する．

以降の本論文の構成は以下のとおりである。2 章では、現在までの DNN に関する研究とその問題点を、特に高速化およびウェブコンピューティングの観点から説明する。3 章では、本研究で提案するブラウザ上での DNN の高速実行のための枠組み「WebDNN」の概要を述べる。高速化のための具体的な手法の詳細については 4 章で述べ、5 章で実際の実行速度を既存手法と比較、考察する。

第 2 章

関連研究

2.1 深層ニューラルネットワーク (DNN)

DNN は単純な演算を層状に多数組み合わせて複雑な関数を表現する学習モデルである。従来、学習が困難であるとされてきたが近年の計算機性能の飛躍的向上やインターネットを通じた学習データの爆発的増加を背景に、様々なタスクで優れた成果を上げ、理論解析も進みつつある。DNN の各層は前の層からの入力を行列として受け取り、何らかの演算を適用した後、行列を出力する。演算の中にはパラメータを持つ物がある。例えば全結合層は入力行列にアフィン変換を施す層であり、アフィン変換の変換行列をパラメータとして持つ。目的の関数を表現するようモデルの各パラメータを適合させるプロセスは学習と呼ばれ、誤差逆伝搬法と呼ばれる手法が広く用いられている。これは、DNN の通常の演算（順伝搬）により得られた計算結果と正解との誤差の勾配を、Chain rule にもとづいて出力層から入力層へ逆伝搬させ、誤差を減少させる方向へ各層のパラメータを勾配降下法により更新する手法である ToDo: 引用。

DNN の応用先は様々だが、特に優れた成果を上げている分野の一つに、画像処理が挙げられる。ToDo: AlexNet らは、畳込み層 (Convolution) を用いたネットワーク構造（Convolutional Neural Network, CNN) で、既存モデルを大幅に上回る画像認識性能を達成した。また、文字列や音声など、可変長の系列データに対しても DNN モデルは適用されている。ToDo: 引用、誰？らは、系列データの入力を扱うことが出来る Recurrent Neural Network (RNN) を用いて云々江陵 ToDo: なんかすごいことができたよ。この他、ロボット制御 ToDo: 引用の山や、ゲーム AIToDo: 引用など、様々な分野で既存のモデルを上回る性能の DNN が提案されている。

しかしながら、いずれの場合もモデルの計算量は非常に大きく、計算時間短縮のために行列演算に適したメニーコアプロセッサを利用することが一般的となっている。その中でも特に広く用いられているデバイスが Graphic Processing Unit (GPU) である。GPU は元々、画像処理目的に開発されたプロセッサで、画像の各 pixel を並列に計算できるよう、大量のコアを備えている。これらのコアを汎用計算に用いる手法が General Purpose GPU (GPGPU) で、DNN には欠かせない技術となっている。ToDo: 引用、NVIDIA とかの何か？しかし、2017 年現在、GPGPU に広く用いられる GPU の価格は一般に数十万〜数百万円程度であり、これ

をエンドユーザー全員へ普及させることは現実的ではない。

DNN の普及に伴い、DNN を実装するプログラミングフレームワークも数多く登場している。このようなフレームワークは DNN モデルの構造をどのように定義するかによって大きく 2 種類に分類される。一方は、静的にネットワーク構造を宣言する define-and-run と呼ばれるもので、もう一方は入力データに応じて動的にネットワーク構造を決定する define-by-run と呼ばれるものである ToDo: 引用 chainer。define-and-run の場合、ネットワーク構造が実行前に決まっているため計算速度向上のための最適化が行いやすいという利点がある。一方、define-by-run の場合、入力データに依存した複雑な処理を行いやすいという長所がある。特に RNN の場合、時系列長に応じてネットワーク内で実際に行われる計算の内容が変化するため、define-and-run では実装が非常に複雑になる。いずれのフレームワークでも、GPU を使った並列計算に標準対応しており、開発者の負担を低減している。一般的に GPGPU による並列計算は難易度が高く、フレームワークのサポートを受けない開発は現実的ではない。

2.2　Web ブラウザコンピューティングの紹介

Web ブラウザは様々なプラットホーム上で動作するアプリケーション実行環境であり、これを科学計算へ用いようという研究が幾つか存在する。ToDo: Miura, Hidaka らは WebCL と呼ばれるブラウザ上から GPU を GPGPU 用途で利用するための API を用いてブラウザ上で DNN の訓練を行う手法を提案した。しかし、WebCL に標準対応したブラウザは存在せず、ユーザーエンドデバイスでの実行は難しい。ToDo: Keras.js の引用は、画像処理用の GPU API である WebGL を用いて GPGPU を行い、ブラウザ上で GPU を用いた DNN の実行を実現した。しかし、WebGL は画像処理用にデザインされた API であり、GPGPU 用途で利用するにはオーバーヘッドが大きいため、実行速度は十分ではなかった。このように、ブラウザ上で DNN を実行する試みはいくつか存在したものの、主に実行速度の遅さという問題点を本質的に解決したものは存在していない。

2.3　DNN モデル実行の高速化

DNN モデルによる推論は計算負荷が高い処理であり、高速化のための手法が多く研究されている。DNN モデルの高速化手法は大きく 3 種類に分類することができる。

一つ目の方法は、DNN モデル内部の潜在変数の数を減らし計算量を減らす手法である。pruning は、特徴量のうち出力への寄与率が低いものを取り除くことで計算量を削減する手法である。 ToDo: ICLR2017 NEC america ToDo: ICLR2017 distilation は、学習済みモデルと同様の出力をする、よりパラメータ数の少ない別のモデルを学習し直すことでパラメータ数を削減する手法である。 ToDo: 引用しかし、pruning では寄与率の閾値、distilation ではパラメータ数の少ないモデルの構造など、パラメータをどこまで減らすのかという設定を手動で行う必要があり、精度を保ちつつ高速化を行うには、人手による十分なチューニングが不可欠である。

二つ目の方法は、行列の低ランク近似を用いる手法である。DNN モデルを構成する演算の大部分は行列操作、特に行列積によって表現されるが、この行列積を、ランクの小さい複数の行列積に分解することで計算量を減らす事ができる。ToDo: MobileNet や ToDo: SqueezeNet では,Convolution を空間方向とチャンネル方向の 2 種類の Convolution の重ね合わせに近似しており、これも低ランク行列分解の一種と言える。これらの手法ではパラメータ調節や低ランク近似後の再学習など、モデルのチューニングが非常に重要であり、十分な精度を保ったままモデルの高速化を達成することは非常に難しい。また、この手法は大きな行列積が存在する演算毎に適用する手法であり、複数の演算の情報を含めた、より包括的な高速化ができないという問題点が存在する。

三つ目の方法は、DNN モデル内で扱う数値を量子化し、低精度で演算を行うことで高速化・省電力化する手法である。この手法では、DNN モデル内部での演算精度は低下するが、DNN モデルが扱う様々なタスクにおいて、モデル内部の演算精度はモデルの最終的な出力精度にあまり影響しないということが知られている ToDo: 引用。ToDo: HashNet らは、識別モデルにおいて、実データの分布に合わせた量子化を行うことで、識別精度を悪化させること無く高速化を達成した。しかしながら、低精度小数演算の性能は、CPU が特別な対応をしていない場合、必ずしも高速・高効率とは限らず、様々なデバイスでの汎用的な利用は難しい。特に、様々なデバイス上での実行を前提とするウェブアプリケーションには不適切である。ToDo: XNORNet らは、0 または 1 の 1bit のみで値を表現し、大幅な高速化とパラメータサイズの圧縮を行った。しかし、この手法では精度が大幅に低下することが報告されている。

このように、既存の高速化手法はそれぞれ問題を抱えており、ブラウザ上での DNN の実行が遅いという問題を解決するには至っていない。

第 3 章

システムの概要

本研究では，DNN モデルをブラウザ上で高速実行するための手法をまとめたフレームワーク「WebDNN」を提案する．本章では、WebDNN を構成する各モジュールと概念について説明する．

3.1 WebDNN

WebDNN による DNN モデル高速実行の流れを ToDo: 図???に示す．WebDNN は大きく分けて 2 つのモジュールから構成される。Graph Transpiler は学習済みモデルをブラウザ上での高速実行用に最適化する。Descriptor Runner は実際にクライアントデバイスのブラウザ上で DNN モデルを GPU を用いて実行する。

3.1.1 計算グラフ

DNN モデルは、単純な演算を多数合成することにより複雑な関数を表現している。このときの各演算と入出力変数の関係を表現したものが計算グラフである。WebDNN では、演算および変数をノード、演算・変数間の入出力の関係をエッジに持つ計算グラフとして DNN モデルを扱う。また、計算グラフは内部に閉路を持たない有向非巡回グラフとする。例として ToDo: 簡単な演算を図示を示す。

Operator

WebDNN の計算グラフに含まれるノードのうち、演算を表すノードを Operator という。Operator には、その演算の性質を表す Attribute が予め設定されており、この情報は最適化の際に使用される。例えば 2 つの行列のアダマール積（要素積）を表す `ElementwiseMul` という Operator は次のような情報を持つ.

- `Commutative`：この演算は交換法則を満たす
- `Associative`：この演算は結合法則を満たす

DNN モデルに使用される演算の種類は限られており、全ての演算について手作業で Attribute を設定することが可能である。

Variable

WebDNN の計算グラフに含まれるノードのうち、変数を表すノードを Variable という。各 Variable は多次元行列であり、行列の各次元には、意味論に基づく名前が割り当てられている。例えば画像データを表す 3 次元行列 ToDo: 図 には、画像の垂直方向に相当する次元に `H`, 画像の水平方向を表す次元に `W`, カラーチャンネルを表す次元に `C` という名前が割り当てられている。このような多次元行列は、計算機のメモリ上では 1 次元配列として扱われる。この時、データがメモリ上にどのように配置されるかは軸の順序 (Order) として表現される ToDo: 図。学習済み DNN モデルを表す計算グラフに含まれる Variable ノードには、実行前に値が確定している定数と入力データを元に実行時に計算されるため、値が未定である変数とが混在している。

3.2 Graph Transpiler

Graph Transpiler は、DNN モデルの学習用フレームワークを用いて構築された DNN モデルを最適化し、ブラウザ上での実行に必要なファイルを生成するモジュールであり、大きく 3 つのレイヤーから構成される。ToDo: 図

Graph Transpiler による最適化はウェブアプリケーションへのアクセスより前に一度だけ行われる。そのため、この最適化に要する時間は問題にならない。

3.2.1 Frontend

DNN モデルの学習用フレームワークは数多く存在し、学習済み DNN モデルのデータ形式も異なっている。そのため、各フレームワーク毎に最適化手法を直接実装した場合、実装量が膨大になる、拡張性が劣るなどの問題が生じる。そこで、学習済みモデルを一旦計算グラフへ変換し、計算グラフを最適化することを考える。このように、計算グラフという学習用フレームワーク非依存な内部表現へ変換することで、学習用フレームワークに依らない最適化が可能となる。

なお、実行前にモデルを最適化する性質上、 define-by-run 形式のフレームワークで構築されたモデルには対応することができない。そこで、このようなモデルにはダミーデータを用いて計算グラフの構造を予め固定したうえで最適化を行う。例として、代表的な define-by-run 形式の DNN モデル学習用フレームワークである ChainerToDo: 引用で学習したモデルを変換するコードを示す。

ソースコード 3.1. ダミーデータによる Chainer の学習済みモデル構造の固定

```
1 import numpy as np
2 import chainer
3 from webdnn.frontend.chainer import ChainerConverter
```

```
4
5 dummy_x = chainer.Varaible(np.zeros(*input_shape)) # ダミーの入力データを作成する
6 dummy_y = predict(dummy_x) # モデルを実行し、計算グラフを構築する
7
8 # 構築した計算グラフを WebDNN の内部表現へ変換する
9 graph = ChainerConverter().convert([dummy_x], [dummy_y])
```

3.2.2　Optimization

Frontend により生成された計算グラフは高速実行のために最適化される。WebDNN には予め様々な最適化ルール（OptimizeRule）が定義されている。これは計算グラフの特定の部分構造を別の構造に変換するものである。

3.2.3　Backend

最適化された計算グラフを元に、ブラウザ上での実行に必要な数種類のデータを生成する。まず、全 Variable をひとつながりのバッファとして確保し、各 Variable をどの位置に割り当てるのかを決定する。各 Variable の依存関係, およびサイズは確定しているため不要になった Variable のバッファを再利用することでメモリ消費量を削減することが可能である ToDo: 図。

次に、各 Operator の処理を実装したソースコードを生成する。このとき、Operator の種類毎に予め定義されたソースコードをコピーする場合と、最適化の結果に応じてソースコードを動的に生成する場合がある。

最後に、学習済みパラメータを、Variable の割り当てに沿って結合し、一つのバイナリデータを生成する。このデータは ToDo: 引用の手法を用いて自動的に圧縮される。

3.3　Descripter Runner

- Web ブラウザで GPU を利用する方法は様々
- 開発者が，全ての GPU API を個別にサポートするのは大変な労力
- 各 GPU API を `Backend` という概念に抽象化し，単一のインターフェースから，各実行環境に応じた最適な API を利用可能に
- 4 種類の backend を用意

3.3.1　WebGPU

WebGPU とは．自分でゼロから実装したことも述べる．

3.3.2 WebGL

WebGL とは.

3.3.3 WebAssembly

WebAssembly とは.

第 4 章

最適化

ToDo: 最適化の方法をひたすら列挙するのは構成的にどうなのか?

実行速度高速化・メモリ消費量削減の観点から最適化

- 計算グラフ中の特定の部分構造に対する変換規則を予め大量に用意する
- 部分構造パターンとしては，演算の種類，および属性が利用可能
- 深層学習は単純な演算を積み上げていくことで複雑な演算を表現するため，演算の種類は多くはなく，全てについて属性を定義することが可能

4.1 定数畳み込み

定数部分は畳み込む

4.2 演算順序の入れ替えによる式の整理

演算順序を入れ替え，定数を集約することにより定数畳み込みの効果を大きくする．

浮動小数点による計算では計算誤差が生じるため，演算順序を入れ替えることは計算結果に若干の悪影響を与える．しかし一般に深層学習モデルでは各演算の精度は最終的な出力に大きな影響を与えないことが知られているため，これは問題にならない．

4.3 計算カーネルの結合

メモリ上のデータの読み書きは非常に大きなコストであり，メモリアクセス回数を減らすことは速度向上に大きな影響が有る．複数の計算カーネルを 1 つにまとめることで，一度メモリからレジスタへ読み出したデータを，再利用することができ，メモリアクセス回数を減らすことができる．

4.4　データアラインメントの整理

ToDo: 説明が意味不明すぎる

各演算には，並列化可能な変数の方向を示す属性 Tensorwise が定義されている．これらを元に，連続する複数の演算で，なるべくデータのマージが必要の内容にデータアラインメントを整理する

4.5　演算の分割

デバイスの制約により，サイズの大きな変数は一度に扱えない場合があり，処理を細かく分割する必要がある．Tensorwise 属性を見ることにより，処理を分割できる．

4.6　メモリ確保の最適化

WebDNN では静的な計算グラフのみを扱うため，使用する変数のサイズは全てコンパイル時に確定している．そのため，メモリ確保計画を予め立てることができ，メモリ使用量を削減できる．

また，処理の中には入出力変数を 1 つにまとめられる処理 (Inplace) が存在する．この場合，さらにメモリ使用量を削減できる．

第 5 章

実験

5.1　最適化ルールの有無による実行速度の比較

ToDo: 各種最適化の有無による実行速度の比較

5.1.1　実験概要

5.1.2　結果と考察

5.2　他フレームワークとの実行速度の比較

ToDo: 同種のフレームワークとの比較

5.2.1　実験概要

5.2.2　結果と考察

第 6 章

結論と展望

- Web ブラウザ上での深層ニューラルネットワーク高速実行手法を開発した
- ToDo: 今後の展望

参考文献

謝辞

TODO

付録 A

ソースコード

```
int main () {
  ...
  ...
}
```