車載制御ECUへのAI適用技術開発

Technology Development of AI Application for In-vehicle Control ECU

横山 夏軌
Natsuki YOKOYAMA

森山 裕
Yuu MORIYAMA

要旨

近年のAI技術の発展は目覚ましく、一般家庭向けにもAIプロダクトが普及しつつあるなど、もはや一過性のブームではないといわれている。

このような状況の中、当社でも車載制御ECU（Electric Control Unit）にAIを搭載する技術開発を推進してきた。しかし、現時点での車載マイコンはAI実装を想定されておらず、その実現に向けては多くの課題が存在する。

通常、AIはGPU（Graphics Processing Unit）などの並列演算環境に実装されることを前提に研究が進められており、普及するAI開発ライブラリ類もこの方針を踏襲している。したがって、開発したAIは処理負荷やアーキテクチャのギャップにより、そのままでは車載マイコンへ実装することができない。

そこで、AIの構造がネットワークモデルであること、および、それが車載コード生成で利用されるブロック線図と相互変換可能であることに着目し、また、自動車制御開発で実績のあるMBD（Model Based Development）プロセスと融合できるAI設計とマイコン実装を繋ぐ手法を開発した。

Abstract

Development of the AI technology in recent years is remarkable, and it is said that the AI products are no longer just a passing fad as they are becoming increasingly common even for general households.

We, DENSO TEN Limited, have also promoted the technology development for installing AI on an in-vehicle control ECU (Electric Control Unit) under such circumstances. However, AI is not supposed to be installed on an in-vehicle microcomputer at this time, and there are a number of issue toward the realization.

Usually, AI is being researched on the assumption that it will be installed on a parallel computing environment such as a GPU (Graphics Processing Unit), and also the AI development libraries which are widely used follow this policy.

Accordingly, the developed AI cannot be installed on the in-vehicle microcomputer as it is due to the processing load and architecture gap.

Therefore, we forcused on the fact that the AI structure was a network model, and that it could be interconverted with the block diagram used for in-vehicle code generation. In addition, a method was developed to connect microcomputer implementation and the AI design capable of being combined with MBD (Model Based Development) process, which had a proven track record in automotive control development.

1. はじめに

ダートマス会議にて「人工知能（AI）」という言葉が提唱された1956年からおよそ60年が経過し、より実用的で現実的な技術として世界的にAI活用普及が進みつつある。2010年ごろより始まったこの第3次AIブームでは数々の実用的なシステムが実現しており、出口を見失い終息した過去2度の

AIブームとは異なって一過性のブームで終わらないと見られている。

各産業分野での成功事例も揃いつつあり、例えば単純なマニュアル化や定式化が困難な専門的知見をAIに代替させる技術では人間の専門家と同等以上の性能が達成されている。その応用分野は医療や機械工業、電子機器や建築など多岐に渡るが、自動車制御の分野にも複雑で専門的知見を要するという点で成功事例との共通点が多く、大いにAI活用による効果が期待できるものと考えられる。

このような状況の中、当社でも車載制御ECUにAIを搭載する技術開発を推進してきた。しかし、現時点での車載マイコンはAI実装を想定されておらず、その実現に向けては多くの課題が存在する。本稿では、車載マイコンへのAI実装と自動車制御開発で実績のあるMBD（Model Based Development）プロセスへの融合について論じる。

2. 車載制御ECUへのAI実装課題

前述のとおり、車載制御ECUに搭載される車載マイコンはAI実装を想定されていない。車載マイコンは厳しい温度・消費電力などの条件下で、極めて高い信頼性が要求される一方、AI開発環境として主流となっているGPU（Graphics Processing Unit）と比較すると十分な処理性能は有していない。また、並列処理に特化したGPUとはアーキテクチャも大きく異なる。多くのAI開発環境はGPUに最適化されているため、作成されたAIは処理性能やアーキテクチャのギャップが大きく、車載マイコンへの搭載が難しい。

ところで、AIには、与えられた入力に対して望ましい出力を得られるようにする「学習」と学習結果に基づき入力に対して出力を返す「推論」の二つのプロセスが存在する。現在、車載マイコンへの搭載を検討しているAIは「推論」を行えるものであり、極めて高い処理能力を要する「学習」は潤沢なリソースをもつGPUサーバ環境などで実施することに合理性がある。

GPUサーバ環境向けには多くのライブラリが流通しており、一定程度の知見があればAI開発を実施することができる。しかし、通常その成果物はGPUとの連携を前提としたPythonコードであり、そのまま車載マイコン環境へ展開することはできない。車載マイコンへ実装するには、何らかの方法でコンパイラの存在するC言語に変換する必要がある。PythonコードをC言語に変換する手法も幾つか提案がされているが、車載マイコンのアーキテクチャに対応したPythonコードを準備する必要があるため、問題の解決にはならない。

そこで、MBDプロセスで実績のあるブロック線図を応用することを提案する。MBDでは、Simulink® などのツールを用いてブロック線図ベースでの制御設計を行う。このブロック線図はPC上でのシミュレーションが可能で、かつ、車載マイコンに実装可能なC言語に変換する機能を持つ。よって、AIをブロック線図で表現することができるかに課題を絞ることができる。

一方、処理性能のギャップについては本質的な解決は難しい。仕様上、GPUと車載マイコンには処理性能に大きな差異があり、これは両者に求められる役割が異なるためである。そのため、GPUサーバ環境上で開発するAIは推論時に車載マイコンで実行可能な規模に制限する必要がある。とはいえ、AIが厳密にどの程度の処理性能を必要とするのかは実装する前に判別困難であるので、処理負荷を事前見積できるかが焦点となる。

3. ネットワークモデルとブロック線図の相互変換

前章で述べたAIをブロック線図で表現する手法について考える。本稿で論じるAIを深層学習とすると、その構成はネットワークモデルとなる。ネットワークモデルは重み付き有向グラフである。また、一般的なブロック線図はブロックの値を重みとして捉えれば重み付き有向グラフと捉えられる。即ち、ネットワークモデルのブロック線図への変

換は可能である（**図 1**）。かつ、この変換では情報の喪失が生じないため可逆であり、ブロック線図からネットワークモデルへの逆変換もできる。

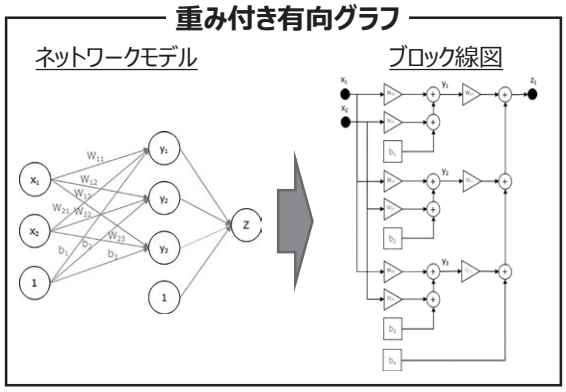

図 1　ネットワークモデルのブロック線図への変換

例えば次のようなネットワークモデルを考える（**図 2**）。x は入力ノード、y は中間ノード、z は出力ノードとし、重みを w、バイアスを b で表す。c は値として 1 を持つ定数とする。

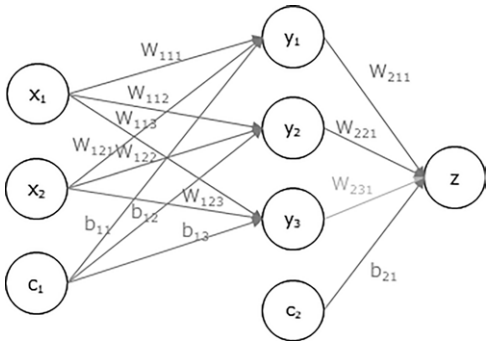

図 2　ネットワークモデルの例

これを重み付き有向グラフとして捉えると、隣接行列は次のように表現できる（**表 1**）。

表 1　ネットワークモデルの隣接行列

	x_1	x_2	c_1	y_1	y_2	y_3	c_2	z
x_1	0	0	0	w_{111}	w_{112}	w_{113}	0	0
x_2	0	0	0	w_{121}	w_{122}	w_{123}	0	0
c_1	0	0	0	b_{11}	b_{12}	b_{13}	0	0
y_1	0	0	0	0	0	0	0	w_{211}
y_2	0	0	0	0	0	0	0	w_{221}
y_3	0	0	0	0	0	0	0	w_{231}
c_2	0	0	0	0	0	0	0	b_{21}
z	0	0	0	0	0	0	0	0

隣接行列からグラフの行列式を作成すると次のように表現できる（式 (1), 式 (2)）。

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} w_{111} & w_{121} & b_{11} \\ w_{112} & w_{122} & b_{12} \\ w_{113} & w_{123} & b_{13} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ c_1 \end{bmatrix} \quad \cdots(1)$$

$$z = \begin{bmatrix} w_{211} & w_{221} & w_{231} & b_{21} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ c_2 \end{bmatrix} \quad \cdots(2)$$

ここで、x, y, z, w, b をそれぞれ X, Y, Z, W, B のようにベクトルで表現し、c=1 を代入すると、次のように変形できる（式 (3), 式 (4)）。

$$Y = \begin{bmatrix} W_1 & B_1 \end{bmatrix} \begin{bmatrix} X \\ 1 \end{bmatrix} = W_1 X + B_1 \quad \cdots(3)$$

$$Z = \begin{bmatrix} W_2 & B_2 \end{bmatrix} \begin{bmatrix} Y \\ 1 \end{bmatrix} = W_2 Y + B_2 \quad \cdots(4)$$

上式をブロック線図で表現すると次のとおりとなり、ネットワークモデルをブロック線図に変換ができる（**図 3**）。

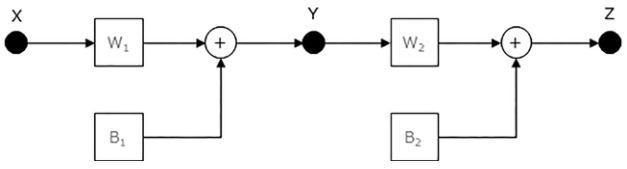

図 3　変換されたブロック線図

また、ブロック線図は階層化によって制御構造を抽象化する機能を持つ。通常、制御設計者はAIの内部構造に関心をもたないため、AIを複雑な機能を持つブロックとして扱えばネットワークモデルを隠蔽することができる。対して、AI開発者は隠蔽されたAIブロックだけに注目すれば、制御と関係なく開発を進められる。さらに、AIのネットワーク構造を入力層、畳み込み層などの階層で抽象化すれば、そのさらに下の階層は平易な数値演算の組み合わせに単純化できる。この階層はマイコン実装技術者が並列化やメモリ配置などを工夫して高速化するために利用される。このことから、ネットワークモデルのブロック線図変換はAIを車載マイコン実装するためだけでなく、開発プロセスを役割ごとに分離し、品質および性能の保証範囲を明確にする効果も期待できる。

4. ネットワークモデルの軽量化とECU搭載制約

前章で述べた手法によりブロック線図に変換したAIはSimulink®の機能により車載マイコンへ実装可能なC言語コードを生成することができる。ただし、これだけでは実装したプログラムが実行可能な処理性能に収まるかは保証できない。処理性能の問題が実行後に判明すればAI設計からやり直すことになり、手戻りのインパクトは大きい。したがって、できるだけ早い段階で要求処理性能の概算を行う方法が必要となる。

そこで、ネットワークモデルのレイヤごとに標準的な演算規模を定式化し、その積算結果をAIの要求処理性能として定義する手法を提案する。併せて、マイコンごとに処理性能を定義しておくことで、設計中のAIがどのマイコンに実装可能かを見積もることが可能となる。

例えば、次のような畳み込み層の演算規模について考える（**図4**）。畳み込みの演算ではフィルタの各要素を掛け合わせてから加算して特徴マップを作成し、これを入力のサイズ分だけスライドさせる。なお、フィルタは複数枚用いることもできる。

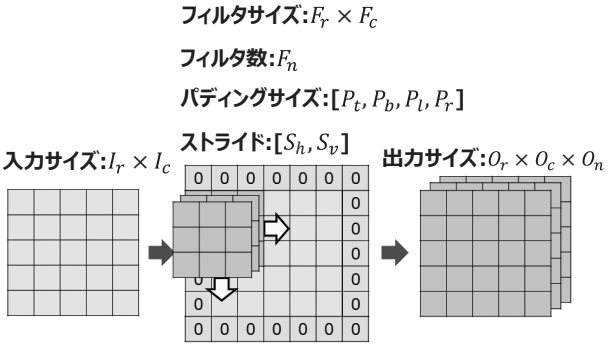

図4　畳み込み層の演算規模

このことを単純に考えると、演算規模Nは入力サイズ$I_r \times I_c$とフィルタサイズ$F_r \times F_c$、フィルタ数F_nに比例することは明らかである。ここに、入力の外周を0で埋めるパディング$[P_t, P_b, P_l, P_r]$とフィルタのスライド時のストライド$[S_h, S_v]$を考慮すると、次の式(5)が導出できる。

$$N = (F_r \times F_c) \times \frac{I_r + P_l + P_r}{S_h} \times \frac{I_c + P_t + P_b}{S_v} \times F_n \cdots (5)$$

この手法のポイントは、AI設計段階で要求処理性能が概算できることに加えて、どのレイヤがボトルネックになっているかを定量的に説明することができることにある。また、デバイスに依存しない指標であるため、ターゲットに拘わらず利用がしやすい。

AIの演算規模を縮小する手法としては大きく分けて四つ考えられ、手法①アプリケーションに適したモデルを選定すること、手法②入力および中間パスを出力に影響の高いものに絞り込むこと、手法③計算精度を必要最低に抑えること、手法④ターゲットに応じたコードに変換することが挙げられる。

手法①については、近しい先行事例で利用されたAIから転移学習する方法がある。ただし、前例で演算規模が最適化されていたかどうかは確認しておく必要がある。

手法②については、さらに二つの手法に分けられ

る。一つは、出力に寄与することが期待できる入力を選別することである。これは、対象アプリケーションの知見が求められるため、ECU への AI 搭載ではサプライヤとしての専門性が発揮できるポイントにもなる。もう一つは、重みの小さなパスを削除してしまうことで、スパースモデリングとして知られている。

手法③については、量子化などの手法が有効とされる。通常、ニューラルネットワークで使用するデータは 16bit 又は 32bit の浮動小数点数であるが、これを 8bit の固定小数点数や、極端な例では 1bit の真偽値にしてしまうことで軽量化する。

手法④についてはターゲットとなるデバイスに搭載されている高速化機能によって対応は異なるが、通常、依存関係のない演算をどの程度並列化できるかがカギになる。具体的には、全結合層や畳み込み層の内積をどれだけ効率的に演算できるかが軽量化に貢献する。

これら四つのうち、手法①～③は AI の設計段階で効果が計上できる。つまり、前述の演算規模で定式化が可能である。一方、手法④はデバイス依存の性能であるため、個別に検討する必要がある。こちらについては、ブロック線図から生成されたコードが参照する AI 専用ライブラリをターゲットごとに準備することで解決する。つまり、マイコン実装技術者の課題として明確に切り分けてしまうわけである。

5. AI専用ライブラリの開発

GPU サーバ環境上で設計・学習した AI が、構造的にそれ以上演算を削減できないとすると、マイコン実装時の課題としてはいかにして CPI（Clocks Per Instruction）を改善するかに絞り込める。レイテンシの小さな命令やメモリアクセスを用いるなど、方法はさまざま知られているが、最も性能に寄与するのは並列化である。そのため、ライブラリ開発に際しては演算の依存関係をどこまでなくせるかを検討する必要がある。

演算のどの部分に依存関係があるかはコード生成を行う前のブロック線図の構造を有向グラフと見做せば明確に分かる。これを利用して、ブロック線図を用いてグラフィカルに並列化を検討する手法を提案する。

例えば次のようなブロック線図を想定する。矢印で繋がるブロック同士は明確に依存関係があるため、分岐ごとに並列化の対象を選定する（**図 5**）。

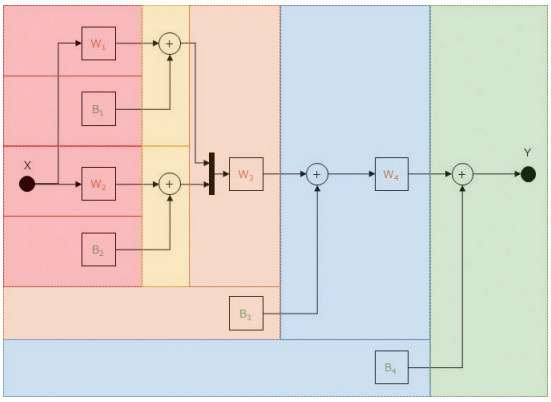

図 5　ブロック線図の並列化検討

分割した領域は横軸が時間軸、縦軸が並列数を表しており、この時点で最大並列化かつ最短時間の状態を表している。ターゲットの並列数が十分であれば、この構成のまま実装するが、そうでない場合は割り当てを検討する必要がある。そこで、この分割した領域を有向グラフとして、終了ノードから順に割り当てを行う（**図 6**）。

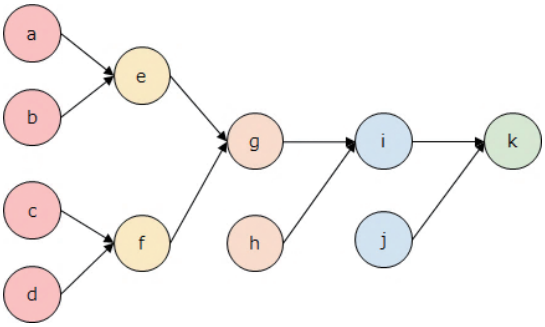

図 6　処理依存関係のグラフ化

今、ターゲットをデュアルコアCPUと仮定すると、その並列数は2である。わかり易くするためにa～kの全ての処理が同様の処理時間と仮定すると、並列数×時系列の箱を用意してなるべく時系列が短くなるように処理を並べればよい。そのアルゴリズムは次のとおり。手順①各ノードが直接参照するほかノードの一つ前になるように時系列方向に整理する。手順②各ノードを参照する全てのノード数である「被依存度」を求める。手順③被依存度の大きいものから空いている箱の内、最も右に配置する。ただし、手順①で整理した位置より左になるように配置すること（**図7**）。

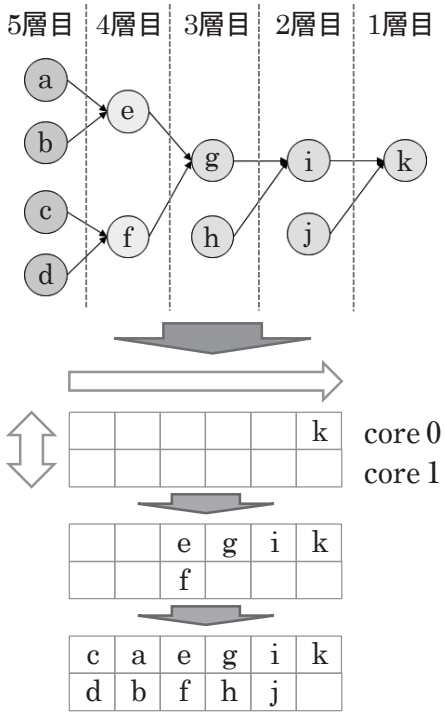

図7　並列処理の割り当て手順

6. おわりに

以上の工夫により、ターゲットに応じたAIの規模を見極めて実装するプロセスを構築することができた。この手法では、AIという新たなアイテムに対しても従来同様に成熟したMBDプロセスを適用できる点でも成果があったと考える。

今後はこの手法による開発試行を行い、AI適用ECUの量産開発に向けて継続して改善を行っていきたい。

・Simulink®は、MathWorks社の登録商標です。

筆者紹介

横山 夏軌
よこやま なつき

AE事業本部
先行システム開発部
技術開発室

森山 裕
もりやま ゆう

AE事業本部
先行システム開発部
技術開発室