C-009

Zynq における機械学習を用いた TRAX ソルバーの実装

鶴田 千晴 †　金田 隆大 †　中原 浩 †　志村 英樹 †　酒井 諒太郎 †　天野 英晴 †
† 慶應義塾大学大学院理工学研究科

1　はじめに

近年、FPGA の技術の発展により科学技術計算や画像処理などの広い分野において研究、開発が盛んに行われている。ここ数年での高位合成ツールの発展や CPU との協調動作を手軽に行える Zynq(Zynq-7000 AP SoC) プラットフォームの登場により、開発の難易度が下がり、システム全体を FPGA で実行する例も増えてきた。そのような環境の中で FPGA を用いたゲーム AI コンテストが開催されており、現在は TRAX[1] と呼ばれるボードゲームが対象となっている。ゲーム AI を作るうえでの方法の一つとして機械学習が挙げられ、機械学習は近年のコンピュータ技術の向上や、DNN(Deep Neural Network) の登場から注目が集まっている。

本論文の構成は次の通りである。2 章において TRAX のルールを説明した後、3 章にて機械学習について述べる。次に 4 章において実装について述べたあと、5 章にて評価、6 章にて結論を述べる。

2　TRAX のルール

Trax は二人対戦型ボードゲームで，プレイヤーは二種類のタイルを交互に置いていき，先に自分の色の線でループまたはビクトリーラインを作ることを目標とする。ゲームで使用するタイルと勝利条件であるループ，ビクトリーラインの例を図 1 に載せる。またタイルを置く際の制約は以下の通りである。

- すでに置かれているタイルに繋げること
- 接しているタイルの線の色が繋がっていること
- 繋げる線は自分の色でも相手の色でも良い

これらに加え，連鎖ルールというものが存在する。これは新らたにタイルが置かれた時に、置き方が 1 つに決まってしまうタイルについては自動的に置かれるものである。連鎖ルールによって置かれたタイルがさらに連鎖を生むこともあり、連鎖の発生がすべて処理されたら次のターンに移る。

3　Machine Learning

3.0.1　DQN(Deep Q-Network)

DQN(Deep Q-Network) は深層学習と Q 学習を組み合わせた機械学習アルゴリズムで，一部のゲームでは，人間のエキスパートを上回る結果を出している。また Trax はラインの繋がりを認識することが重要なゲームであるため，画像分類などの分野で高い識別率を達成する CNN(Convolution Neural Network) は Trax に有効な手法であると考えられる。そこで我々は DQN から出力される Q 値を評価値として使用する。

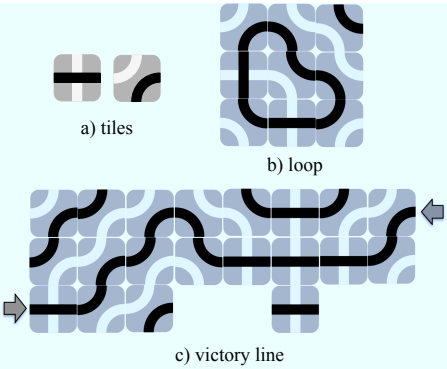

図 1: Trax tiles and win conditions

3.1　DQN on Caffe

Caffe とは一般公開されている機械学習用フレームワークである [2]。我々は Caffe 上に DQN を実装し学習を行った。Caffe は十分に最適化されている上に、学習に GPU を用いることができ、非常に高速な学習が可能である。学習に用いる DQN の構成を図 2 に載せる。Trax は盤面サイズに制限はないが、今回盤面サイズは 20×20 で固定した。タイルの置き方は 3 パターン存在するので、出力される Q 値は 1200 通りとなる。DQN への入力は盤面状態を表す画像データで、入力画像は 84×84 の 8bit グレースケールとした。また trax のタイルは 4×4 で表現している。実装した DQN 内の主な Layer は Convolution Layer が 2 層、Inner Product Layer が 2 層となっている。1 層目の Convolution Layer(conv1) では 8×8 の kernel 16 種類を用いて畳み込み積分を行い、2 層目の Convolution Layer(conv2) では 4×4 の kernel 32 種類を用いて畳み込み積分を行った。最後に Eltwise Layer を通すことで、現在盤面に対する合法手以外の Q 値を強制的に 0 にしている。

4　implementation

CPU と FPGA を組み合わせて効率の良い実装を行うために、Zynq を採用した。まずはアルゴリズムをソフトウェアで記述し Zynq CPU 上で動作させ、ボトルネックとなる処理を FPGA にオフロードしていく。

4.1　Off-Loading Target

DQN の実行時間の大半は Convolution Layer が占める。Convolution Layer では BLAS(Basic Linear Algebra Subprograms) ライブラリに含まれる GEMM(GEneral Matrix Multiply) と呼ばれる演算を

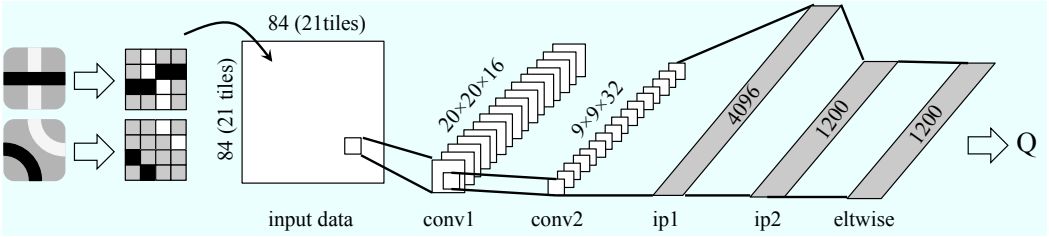

図 2: trax tiles and win conditions

表 1: gemm utilization

	LUT	FF	BRAM	DSP48E
conv1 gemm	5413(10%)	6350(5%)	34(12%)	28(12%)
conv2 gemm	10724(20%)	12518(11%)	66(23%)	56(25%)
xc7z020	53200	106400	280	220

表 2: gemm performance

	Frequency	Latency	Interval
conv1 gemm	150 MHz	628 cycle	629 cycle
conv2 gemm	150 MHz	228 cycle	229 cycle

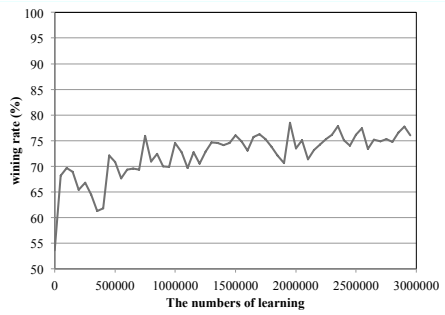

図 3: result of machine learning

利用する。GEMM で行われる演算は下記の通りである。

$$C = alpha * AB + beta * C \quad (1)$$

A, B, C はそれぞれ $M \times K, N \times K, N \times M$ 行列である。Convolution Layer では $alpha = 1.0, beta = 0.0$ として使用される。まず im2col 関数で入力画像を Patch を行とする行列に変換する。Patch はフィルタをあてられる部分を指す。そして GEMM を用いて Patch Matrix(A) と Kernel Matrix(B) を掛けあわせて畳み込み積分を行っている。この GEMM 演算を FPGA 実行することで高速化を試みる。

5 評価

5.1 GEMM

Vivado, Vivado HLS, SDK 2014.2 を用いて Digilent ZED BOARD(zc7x020) 上に実装した。

5.1.1 Resource Utilization

実装した 2 種類の GEMM モジュールのリソース使用量は表 1 のようになった。

リソースはまだ十分に余っているため、今後さらなる高速化を試みることが可能である。

5.1.2 Execute time

実装した gemm モジュールの周波数、レイテンシ、インターバルを表 2 に示す。これらの値と入力データの大きさから、conv1，conv2 での畳込み積分に用する時間は以下の様になる。

$$t_{conv1} = 6.67ns \times 629cycle \times 20 \times 20 \approx 1.68ms \quad (2)$$

$$t_{conv2} = 6.67ns \times 229cycle \times 9 \times 9 \times 16 \approx 1.98ms \quad (3)$$

Zynq に使用されている CPU(ARM Cortex-A9) で実行した場合、conv1 は 45.2 ms、conv2 は 64.0 ms となった。よって GEMM に関して CPU よりも約 26 倍の高速化を達成した。

5.2 学習結果

学習回数と勝率の関係を図 3 にまとめた。学習会数は 50000 回刻みで最大 300000 回とした。学習の相手にはパターンマッチングを用いたフィルターを搭載した AI を使用し、学習したデータセットを DQN に配置し、Q 値が最大になるように手を選ぶ方法を用いた。初期状態ではほぼ勝率は 50 %程度であったが、2300000 回学習後から勝率は 75 %を超えるようになっている。

6 結論

本論文では Zynq を用いた TRAX ゲーム AI のシステムについて提案した。ゲーム AI の評価関数では DQN を採用し、機械学習によって AI を成長させた。また、システムは ZedBoard を用いて実装し、一部演算をオフロードを行い、オフロード部に関して ARM Cortex-A9 とくらべて 26 倍の高速化を達成した。

参考文献

[1] Traxgame.com, “Official trax website”. http://www.traxgame.com/

[2] theBVLC, “Caffe”. http://caffe.berkeleyvision.org/