Vol.2021-ARC-245 No.19
2021/7/21

アルゴリズム・ハードウェア協調設計による ベイジアン畳み込みニューラルネットワークの高速化

藤原 良樹[1,a)]　高前田 伸也[1,b)]

概要：従来のニューラルネットワークにおける不確実性の問題を解決するために，ベイジアン畳み込みニューラルネットワーク（BCNN）が提案されている．BCNN では，重みを決定的な値ではなく分布として扱うことで，オーバーフィッティング，少量のデータでの学習，不確実性評価の問題に対処することができるようになる．しかし，BCNN の出力分布を計算するには，複数のフォワードパスを計算する必要があるため，多くの時間とエネルギーを消費してしまう．この問題を解決するため，近似アルゴリズムとハードウェアサポートを備えた新しいアルゴリズム・ハードウェア協調設計による BCNN の高速化を提案する．各層の入力の値と複数のフォワードパス間の入力の差の絶対値を観測した結果，これらの値のほとんどは，それ以外の一部の大きな値に比べて著しく小さくなることがわかった．提案アルゴリズムでは，これらの小さな値を 0 として扱い，スパースにする．このようにして抽出されたスパース性により，ほとんどの乗算を省略することができる．その結果，分類タスクでは 81.1%，回帰タスクでは 77.7%の計算の削減を実現した．さらに，アルゴリズムレベルの近似をハードウェア上でサポートするために，本アルゴリズムに特化した新しいデータフローを提案し，アルゴリズムによって抽出されたスパース性を扱うことができる新しいアクセラレータアーキテクチャ ASBNN を開発した．評価の結果，ASBNN はアルゴリズムによる計算削減を効率良く活用し，BCNN アクセラレータの素朴な実装に比べて，計算速度を 3.3 倍，エネルギー効率を 3.7 倍改善することがわかった．

1. はじめに

機械学習の重要な一分野として深層学習が挙げられる．その応用分野は，自然言語処理 [21]，自動運転 [15]，ロボット工学 [17] など多岐にわたる．しかし，その応用分野の増加に伴い，いくつかの問題が発生している．その一つは，訓練されたモデルが訓練データに適合しすぎてしまうオーバーフィッティングである．また，多くの応用分野では小さなデータセットでの学習が求められるが，従来のニューラルネットワークでの実現は困難である．さらに，推論結果だけでなく，その信頼度を知るために不確実性評価が必要であるが，従来のニューラルネットワークでは対応することが難しい．ベイジアンニューラルネットワーク（BNN）は，重みを分布として扱うことで，上記の問題に対処することができる．そのようなアドバンテージから BNN は，病気の検出 [14] や自動運転 [2] などの用途に利用されている．

BNN の重みは，学習データと事前分布によって条件付けられた事後分布として表現される．それに加えて，BNN の出力も分布として表現される．これらの分布を解析的に計算することは困難であるため，様々な近似アプローチが提案されている．代表的な手法として，変分推論 [4] やマルコフ連鎖モンテカルロ法（MCMC）[3] がある．本稿では，MCMC よりも高速でスケーラブルであることから，変分推論に着目した [5,18]．また，その中でも事前分布をガウス混合分布またはガウス分布に制限する，ガウス変分推論と呼ばれる手法を使用する [6]．本稿では，性能とパラメータ数のバランスを考慮して，事前分布としてガウス分布を用いる．

ガウス変分推論には，計算速度やエネルギー消費量の観点で課題がある．これは，出力分布を得るために，同じ入力値を用いてフォワードパスを複数回計算しなければならないからである．この計算速度とエネルギー消費量に関する問題を解決するために，VIBNN [7] というアクセラレータが提案された．しかし，アルゴリズムとハードウェアの協調最適化ではないという点で改善の余地があり，複数回のフォワードパスの必要性についても言及されていない．

本稿では，ベイジアンニューラルニューラルネットワークの中でも，ベイジアン畳み込みニューラルネットワーク

[1] 東京大学 大学院情報理工学系研究科　〒 113–8656 東京都文京区本郷 7 丁目 3–1
[a)] fujiwara-yoshiki064@g.ecc.u-tokyo.ac.jp
[b)] shinya@is.s.u-tokyo.ac.jp

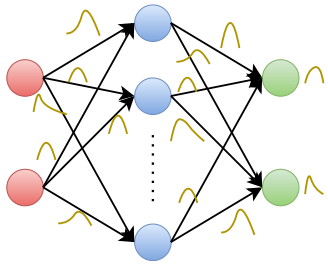


図 1: BNN の重みと出力の表現.

(BCNN) に注目する．速度とエネルギー消費の問題を解決するために，BCNN の計算を高速化するための近似アルゴリズムとハードウェアサポートを備えた新しいアルゴリズム・ハードウェア協調設計アプローチを提案する．アルゴリズムの観点からは，BCNN における分布の観察に基づき，各層の入力分布の特性と複数のフォワードパス間の差異を利用した近似手法を提案する．さらに，アルゴリズムレベルの近似をハードウェアでサポートするために，新しいデータフローとアクセラレータアーキテクチャを提案する．

本稿の主な貢献は以下のようにまとめられる．

- BCNN における分布の特徴に基づいた，ガウス変分推論のための新しい近似アルゴリズムを提案した．
- このアルゴリズムに特化した新しいデータフローの提案と，このアルゴリズムによって抽出されたスパース性を扱うことができる新しいアクセラレータアーキテクチャ，ASBNN を提案した．

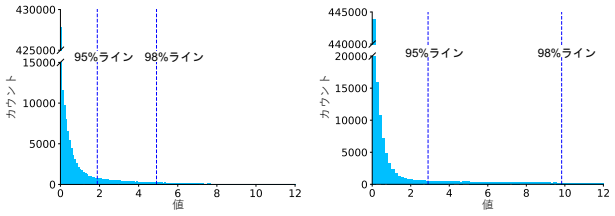
2. ASBNN アルゴリズム

2.1 BNN における推論

従来のニューラルネットワークとは異なり，BNN の重みは図 1 に示すように分布として表されるため，出力の分布を解析的に計算することは困難である．そこで，モンテカルロサンプリングを用いて，フォワードパスを複数回計算することで，出力の分布を近似計算する．計算されるフォワードパスの数を num_passes と定義する．それぞれのフォワードパスの計算に用いる重みの値は，対応する重みの分布に従う乱数である．特に今回対象としているガウス変分推論では，重みとバイアスの分布はガウス分布である．したがって，畳み込み (CONV) 層のフォワードパスは，ベクトル $\boldsymbol{\epsilon_w}, \boldsymbol{\epsilon_b} \sim \mathcal{N}(0, \mathrm{I})$ を用いて次のように書くことができる．

$$\mathbf{output} = Conv(\mathbf{input}, \mathbf{W}, \mathbf{b}).$$
$$(\mathbf{W} = \overline{\mathbf{W}} + \boldsymbol{\epsilon_w} * \boldsymbol{\Sigma_W},\ \mathbf{b} = \overline{\mathbf{b}} + \boldsymbol{\epsilon_b} * \boldsymbol{\Sigma_b})$$

ここで，$\overline{\mathbf{W}}$ と $\overline{\mathbf{b}}$ は，それぞれ層の重みの分布とバイアスの分布の平均である．$\boldsymbol{\Sigma_W}$ と $\boldsymbol{\Sigma_b}$ はそれらの標準偏差である．ただし，異なるフォワードパスに対しては，$\boldsymbol{\epsilon_w}, \boldsymbol{\epsilon_b}$ として異なる値をサンプリングする．



(a) $|\mathbf{input_k} - \mathbf{input_{(k,first)}}|$. (b) $|\mathbf{input_k}|$.

図 2: 特徴の視覚化.

2.2 中間層の分布の観察

アルゴリズムレベルでの計算高速化を狙い，まず BCNN の内部挙動を観察する．観察したところ，いくつかの興味深い特徴が見つかった．$\mathbf{W}$ と $\mathbf{b}$ を使って推論する際の第 k 層への入力を $\mathbf{input_k}$ とする．また，$\overline{\mathbf{W}}$ と $\overline{\mathbf{b}}$ で表される平均値のみを用いて推論を行った場合の第 k 層への入力を $\mathbf{input_{(k,first)}}$ とする．

図 2a に $|\mathbf{input_k} - \mathbf{input_{(k,first)}}|$ の分布を，図 2b に $|\mathbf{input_k}|$ の分布を示す．$|\mathbf{input_k} - \mathbf{input_{(k,first)}}|$ と $|\mathbf{input_k}|$ の分布がロングテールになることを発見した．これらのグラフには，VGG11 [19] をベースにした BCNN の第 4 番目の CONV 層を使用している．可視化のため，12 より大きな値は表示していない．グラフよりこれらの値のほとんどは，他の大きな値に比べて著しく小さいことがわかる．

2.3 複数フォワードパスにおける近似

ここでは，前節で得られた知見をもとに，近似アルゴリズムを紹介する．上で説明したように，BCNN ではフォワードパスを num_passes 回計算する．我々のアルゴリズムでは，$num\_passes + 1$ 回のフォワードパスを計算し，最初のフォワードパスの計算を「初期パス」，それ以降の計算を「後続パス」と呼ぶことにする．

2.3.1 初期パス

初期パスでは，乱数を使わずにフォワードパスを計算する．第 k 層の重みとバイアスの平均をそれぞれ $\overline{\mathbf{W_k}}$ と $\overline{\mathbf{b_k}}$ とし，$\mathbf{input_{(k,first)}}$ をこの時の入力とする．この層の出力である $\mathbf{output_{(k,first)}}$ は以下のように計算される．

$$\mathbf{output_{(k,first)}} = Conv(\mathbf{input_{(k,first)}}, \overline{\mathbf{W_k}}, \overline{\mathbf{b_k}}).$$

$\mathbf{input_{(k,first)}}$ と $\mathbf{output_{(k,first)}}$ は後の計算のために保存される．

2.3.2 後続パスの近似

この近似手法では，まず乱数に依存する部分と依存しない部分を分け，そして，乱数に依存する部分の 0 の数を増やす．0 のオペランドとの乗算結果は 0 になるので，スパース性が高めることができ，ほとんどの乗算を省略することに繋がる．

$\mathbf{input_k}$ を第 k 層の入力，$\mathbf{output_k}$ を出力する．$\boldsymbol{\Sigma_{(W,k)}}$

と $\mathbf{\Sigma_{(b,k)}}$ をそれぞれ重みとバイアスの標準偏差とする．CONV 層の計算に使用される重みとバイアスは，$\boldsymbol{\epsilon_w}, \boldsymbol{\epsilon_b} \sim \mathcal{N}(0, \mathrm{I})$ を用いて以下のように表される．

$$\mathbf{W_k} = \overline{\mathbf{W_k}} + \boldsymbol{\epsilon_w} * \mathbf{\Sigma_{(W,k)}},\ \mathbf{b_k} = \overline{\mathbf{b_k}} + \boldsymbol{\epsilon_b} * \mathbf{\Sigma_{(b,k)}}.$$

この層の出力は，以下のように変形される．

$$\begin{aligned}
&\mathbf{output_k} = Conv(\mathbf{input_k}, \mathbf{W_k}, \mathbf{b_k}) \\
&\simeq Conv(\mathbf{input_k}, \mathbf{W_k}, \overline{\mathbf{b_k}}) \qquad (1) \\
&= Conv(\mathbf{input_k}, \overline{\mathbf{W_k}}, \overline{\mathbf{b_k}}) + Conv(\mathbf{input_k}, \boldsymbol{\epsilon_w} * \mathbf{\Sigma_{(W,k)}}, \mathbf{0}) \\
&= Conv(\mathbf{input_{(k,first)}}, \overline{\mathbf{W_k}}, \overline{\mathbf{b_k}}) \\
&\quad + Conv(\mathbf{input_k} - \mathbf{input_{(k,first)}}, \overline{\mathbf{W_k}}, \mathbf{0}) \\
&\quad + Conv(\mathbf{input_k}, \boldsymbol{\epsilon_w} * \mathbf{\Sigma_{(W,k)}}, \mathbf{0}) \\
&= \mathbf{output_{(k,first)}} + Conv(\mathbf{input_k} - \mathbf{input_{(k,first)}}, \overline{\mathbf{W_k}}, \mathbf{0}) \\
&\quad + Conv(\mathbf{input_k}, \boldsymbol{\epsilon_w} * \mathbf{\Sigma_{(W,k)}}, \mathbf{0}). \qquad (2)
\end{aligned}$$

式 (1) は BCNN のバイアスの標準偏差がすべて 0 に近い値であるという観測結果に基づいている．(2.2 章には記載していない)

次に，式 (2) の第二項を近似する．$|\mathbf{input_k} - \mathbf{input_{(k,first)}}|$ のほとんどの値は 0 または 0 に近い値であるため，閾値 α を用いて α 未満の値を 0 にし，その結果を $\mathbf{x_1}$ と定義する．

$$Conv(\mathbf{input_k} - \mathbf{input_{(k,first)}}, \overline{\mathbf{W_k}}, \mathbf{0}) \simeq Conv(\mathbf{x_1}, \overline{\mathbf{W_k}}, \mathbf{0})$$

$$\mathbf{x_1}[i] = \begin{cases} 0 & (|\mathbf{input_k}[i] - \mathbf{input_{(k,first)}}[i]| < \alpha) \\ \mathbf{input_k}[i] - \mathbf{input_{(k,first)}}[i] & (otherwise) \end{cases}$$

さらに，式 (2) の第三項を近似する．$|\mathbf{input_k}|$ のほとんどの値は 0 または 0 に近い値であるため，閾値 β を用いて β 未満の値を 0 にし，その結果を $\mathbf{x_2}$ と定義する．

$$Conv(\mathbf{input_k}, \boldsymbol{\epsilon_w} * \mathbf{\Sigma_{(W,k)}}, \mathbf{0}) \simeq Conv(\mathbf{x_2}, \boldsymbol{\epsilon_w} * \mathbf{\Sigma_{(W,k)}}, \mathbf{0})$$

$$\mathbf{x_2}[i] = \begin{cases} 0 & (|\mathbf{input_k}[i]| < \beta) \\ \mathbf{input_k}[i] & (otherwise) \end{cases}$$

最終的に 第 k 層の出力 $\mathbf{output_k}$ を次のように書き換えることができる．

$$\begin{aligned}
\mathbf{output_k} &\simeq \mathbf{output_{(k,first)}} \\
&\quad + Conv(\mathbf{x_1}, \overline{\mathbf{W_k}}, \mathbf{0}) + Conv(\mathbf{x_2}, \boldsymbol{\epsilon_w} * \mathbf{\Sigma_{(W,k)}}, \mathbf{0}).
\end{aligned}$$

アルゴリズム 1 に提案アルゴリズムを要約する．アルゴリズム中の “gaussian(0,1)” はガウス分布に従う乱数を表す．

アルゴリズム 1 ASBNN アルゴリズム．

```
1: function DROP(x,γ)                    ▷ γ 未満の値を drop する関数
2:   for i = 0, ..., len(x)-1 do
3:     if abs(x[i]) < γ then
4:       x[i] ← 0
5:                                                    ▷ 初期パス
6: output[0] = CONV(input[0], W_mean, b_mean)
7:                                                    ▷ 後続パス
8: for pass = 1, ..., num_passes do
9:   x1 = DROP((input[pass] - input[0]), α)
10:  x2 = DROP(input[pass], β)
11:  output[pass] = output[0] + CONV(x1,W_mean,0)
                   + CONV(x2, gaussian(0,1)*W_std,0)
```

2.4 近似のためのハイパーパラメータ

2.3 章で提案したアルゴリズムは，新しい変数 α と β を導入するものであり，これらの値によって挙動が変化する．これらのパラメータを適切に設定するために，アルゴリズムを用いる前にこれらの値を決定する．あるハイパーパラメータの値における性能を評価するために，元の分布と近似された分布の間の KL ダイバージェンスまたは L2 ノルムを用いて，近似の精度を見ることができる．また，不確実性の評価指標を用いてハイパーパラメータを評価することもできる．さらに，実験結果より，層間で同じ α と β を使用しても良いことが確認されているため，ハイパーパラメータのチューニングをより簡単に行うことができる．5 章にて，ハイパーパラメータを決定するためのワークフローの例を示す．

なお，本節の議論は，カーネルサイズが 1×1 の CONV 層と等価である，全連結（FC）層についても成立する．

3. ASBNN データフロー

2 章の近似アルゴリズムから得られるスパース性を効果的に利用するために，新しいデータフロー，ASBNN データフローを提案する．ASBNN データフローは，Planner-Tiled-Input-Stationary-Cross-Production（PT-IS-CP）データフロー [16] を拡張したものである．PT-IS-CP データフローと同様に，ASBNN データフローでは，複数の Processing Element（PE）を使用して，畳み込み計算を並列に行う．すべての PE は乗算の累積和の計算を行う．

この章では，$num\_passes \times num\_ich \times W \times H$ 次元の実数配列の input を受け取り，$num\_passes \times num\_och \times (W - kx + 1) \times (H - ky + 1)$ 次元の実数配列の output を出力する CONV 層を考える．num_ich と num_och はそれぞれ入力チャネルと出力チャネルを表している．さらに，カーネルサイズを $kx \times ky$ とする．

3.1 並列化

3.1.1 PE 間並列化

ASBNN データフローは，PT-IS-CP データフローと同じマッピング戦略を採用する．入力平面を $W_t \times H_t$ の小さなサブセットに分割し，各 PE に対応する入力をマッピングする．重みは各 PE にブロードキャストされる．なお，ASBNN データフローでは $\mathbf{input_k}$ の代わりに $\mathbf{x_1}$ と $\mathbf{x_2}$ を

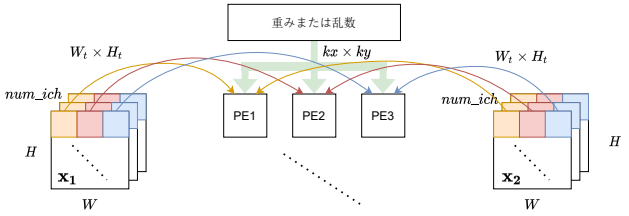


図 3: マッピング全体像

マッピングすることで第 k 層を計算している．各 PE は次の層の入力である **new_x1** と **new_x2** を出力する．もし PE 内での計算を処理できないほど *num_och* が大きい場合は，出力チャネルを *num_og* 個のグループに分割する．

3.1.2 PE 内並列化

各 PE では，$\mathbf{x_1}$ と $\mathbf{x_2}$ の一部を用いた 2 つの畳み込みが計算される．入力と出力にそれぞれ二つのバッファが用意され，それらは “in_buf1” と “in_buf2”，“out_buf1” と “out_buf2” としてアルゴリズム 2 に記載されている．“in_buf1” と “out_buf1” は $\mathbf{x_1}$ と **new_x1** に，“in_buf2” と “out_buf2” は $\mathbf{x_2}$ と **new_x2** に対応している．各 PE ではまず，長さが K の重みまたは乱数のベクトルと，長さが I の入力活性ベクトルが，それぞれのバッファから取り出され，乗算器アレイに分配される．次に，乗算器アレイが直積を計算し，同時に出力座標が計算される．その後，計算された出力座標を用いて，積を累積バッファに累積する．

これらの計算は，アルゴリズム 2 において，**parallel1** と **parallel2** で表されている．また，アルゴリズム内の “rand_generator” はガウス分布に従う乱数発生器を表している．“Ocoord” と “Xcoord” と “Ycoord” は各直積に対応する累積バッファの位置を計算する関数である．さらに，二つの畳み込み計算，**parallel1** と **parallel2** は並列に実行され，それらは異なるバッファに累積される．

3.2 ポストプロセッシング

二つの畳み込みの計算に続き，次の層の入力を作るための計算が処理され，この計算をポストプロセッシングと呼ぶ．まず，二つの畳み込みに対応する二つの累積バッファの値が足される．次に，現在のパスが初期パスであればバイアスの平均が足され，ReLU の前と後の値が後続パスのために保存される．これらの保存される値は，アルゴリズム 2 において “out_preserve” と “in_preserve” で表されている．そして，$\mathbf{new\_x_1} = \mathbf{0}, \mathbf{new\_x_2} = \mathbf{input_{(k+1,first)}}$ がセットされる．もしパスが初期パスでなければ，初期パスで保存された値を用いて，“out_buf1” と “out_buf2” が計算される．さらに “out_buf1” と “out_buf2” のスパース性を高めるために，α と β を用いた drop 関数が適用される．

3.3 Pass Loop の配置

従来の CNN と BCNN の最も重要な違いは，pass loop

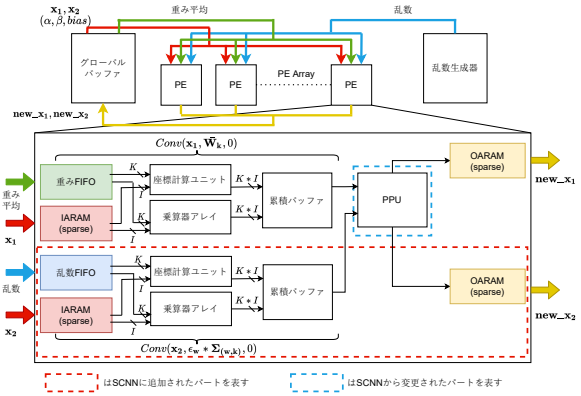


図 4: ASBNN アーキテクチャ．

（複数回のフォワードパスを行うループ）が含まれているかどうかである．ASBNN データフローでは，ロードとストアの数を最小限にするように pass loop を追加している．pass loop 間で値の変わらない部分に着目し，出力チャネル群ループと入力チャネルループの間に pass loop を挿入する．

3.4 境界計算

CONV 層の性質上，入力サイズと出力サイズは必ずしも一致しない．この問題を解決するには，入力サイズを大きくする必要がある．この手法は，PT-IS-CP データフローにおいて，input halo として提案されたものである．

3.5 スパース性の活用

スパース性を利用するために，PT-IS-CP データフローと同様に，よりハードウェアに適した方法で行列を表現する compressed sparse format を用いる．ここでは，行列またはベクトルを，0 以外の要素の値を表すベクトルと 0 以外の要素の間にある 0 の数を表すベクトルに分ける．0 の数は 4 ビットで表される．非ゼロの要素の前に 15 個以上の 0 が現れた場合，非ゼロの値として 0 を追加する．ASBNN データフローでは，$\mathbf{x_1}$, **new_x1**, $\mathbf{x_2}$, **new_x2** がこのフォーマットで表現される．このフォーマットにより，ロードとストアの数を減らすことができる．さらに，0 以外の値だけを計算することで **parallel1** と **parallel2** に必要なステップ数を減らすことができる．

4. ASBNN アーキテクチャ

我々は 2 章におけるアルゴリズム，3 章におけるデータフローをベースに，我々のアルゴリズムに特化した新しいアーキテクチャを提案する．

4.1 アーキテクチャ全体図

3 章のデータフローを採用した ASBNN アクセラレータは，図 4 のように，相互接続された PE，バッファ，乱数生

アルゴリズム 2 ASBNN データフロー

```
wt_mean_buf[num_ich][num_og*kx*ky/K][K]
bias_mean_buf[num_och/num_og][num_og]
in_buf1[num_passes+1][num_ich][Wt*Ht/I][I]
in_buf2[num_passes+1][num_ich][Wt*Ht/I][I]
acc_buf1[num_og][Wt][Ht]
acc_buf2[num_og][Wt][Ht]
out_buf1[num_passes+1][num_och/num_og][num_og*Wt*Ht]
out_buf2[num_passes+1][num_och/num_og][num_og*Wt*Ht]
for og = 0, ..., num_och/num_og − 1 do
 for pass = 0, ..., num_passes do
                          ▷ parallel1 と parallel2 は並列計算
  parallel1 {
  for ich = 0, ..., num_ich − 1 do
   for a = 0, ..., Wt ∗ Ht/I − 1 do
    in1[0:I-1] = in_buf1[pass][ich][a][0:I-1];
    for w = 0, ..., num_og ∗ kx ∗ ky/K − 1 do
     wt1[0:K-1] = wt_mean_buf[ich][w][0:K-1];
     for (i = 0, ..., I − 1) × (k = 0, ..., K − 1) do
      o = Ocoord(w,k);
      x = Xcoord(a,i,w,k);
      y = Ycoord(a,i,w,k);
      acc_buf1[o][x][y] += in1[i]*wt1[k];
  }
  parallel2 {
  for ich = 0, ..., num_ich − 1 do
   for a = 0, ..., Wt ∗ Ht/I − 1 do
    in2[0:I-1] = in_buf2[pass][ich][a][0:I-1];
    for w = 0, ..., num_og ∗ kx ∗ ky/K − 1 do
     wt2[0:K-1] = rand_generator();
     for (i = 0, ..., I − 1) × (k = 0, ..., K − 1) do
      o = Ocoord(w,k);
      x = Xcoord(a,i,w,k);
      y = Ycoord(a,i,w,k);
      acc_buf2[o][x][y] += in2[i]*wt2[k];
  }
                                   ▷ ポストプロセッシング
  acc_buf = acc_buf1 + acc_buf2;
  if pass == 0 then
   out_preserve = acc_buf + bias_mean_buf[og];
   in_preserve = ReLU(out_preserve);
   out_buf1[pass][og] = {0};
   out_buf2[pass][og] = in_preserve;
  else
   out_buf1[pass][og] =
   drop(ReLU(acc_buf + out_preserve) - in_preserve);
   out_buf2[pass][og] =
   drop(ReLU(acc_buf + out_preserve));
```

成器から構成されている．各 PE は，入力，重み，乱数を受け取るチャネルと，出力を送信するチャネルを持っている．さらに，モデルにバイアスがある場合は，それも受信し，また，ハイパーパラメータ α と β も PE に渡される．

ASBNN は SCNN [16] をベースに設計されており，さらに，SCNN は我々のデータフローの元となった PT-IS-CP データフローをベースに設計されている．5.2 章で我々は ASBNN を SCNN と比較して評価するため，図 4 にてその違いを明示している．さらに，我々は VIBNN と同様に，乱数生成器として，BNNWallace-GRNG 法を用いた．

4.2 PE デザイン

各 PE は，入力活性 RAM（IARAM），出力活性 RAM（OARAM），重み FIFO，乱数 FIFO，乗算器アレイ，座標計算ユニット，ポストプロセッシングユニット（PPU）から構成される．IARAM と OARAM は，それぞれの活性値を，3.5 章で提案したスパースフォーマットで保存する．

PE の設計の概要を図 4 に示す．各 PE は，ASBNN のデータフローの順に重みと入力活性を用いて計算する．並列処理とその計算については前節で説明したため，ここではデータフローを実現する PE の各構成要素の設計について説明する．PPU 以外の要素は，SCNN の PE コンポーネントをベースに構成されている．

まず，長さが K の重みベクトルと，長さが I の圧縮された入力活性ベクトルが，重み FIFO と IARAM から取り出される．これらは，$K \times I$ 個の乗算器からなる乗算器アレイに分配される．さらにそれらの座標は，座標計算ユニットに分配される．次に乗算器アレイで直積が計算され，同時に，座標計算ユニットで出力座標が計算される．別の畳み込みは，もう一つの乗算器アレイを用いて，同じ方法で同時に計算される．

各ステップでは，$K \times I$ 個の直積が累積バッファに蓄積される．このバッファは，入力のスパース性による不連続なアクセスの影響を軽減するために，小規模な累積バンクの分散アレイに分割されている．小規模な累積バンクは，クロスバースイッチのように構成されたネットワークを介してアクセスされ，加算器と小さなエントリのセットで構成されている．累積する場所は，座標計算ユニットの出力に基づいて決定される．[16] の結果に基づけば，$2 \times K \times I$ 個の累積バッファが十分な大きさであり，競合を減らすことができる．さらに，値の更新と PPU への値の送信を同時に行うことができるように，累積バッファはダブルバッファリングされている．

二つの畳み込みが計算された後，累積和の値が PPU に送られる．PPU は図 5 のように表される．ここでは，バイアス加算，ReLU 関数，drop 関数，sparse format への変換が行われえる．はじめに，二つの累積和が計算され，バイアスまたは $\mathbf{output_{(k,first)}}$ に保存されている値

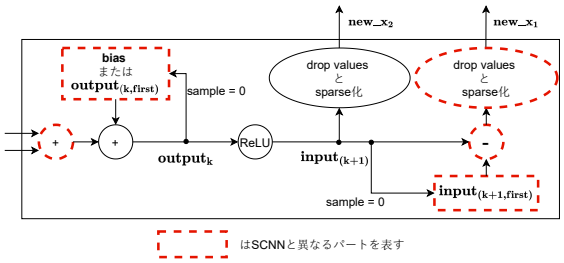
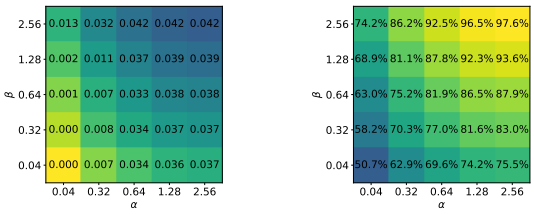


図 5: PPU.



(a) L2 ノルムとパラメータの関係を表すヒートマップ　(b) スキップされる乗算の割合とパラメータの関係を表すヒートマップ

図 6: パラメータを選択するためのヒートマップ

が加算される．次に，ReLU 関数が適用される．さらに，$\mathbf{input_{(k+1)}} - \mathbf{input_{(k+1,first)}}$ が計算され，次の層の入力となる．最後に，計算された次の層の入力の一部が α と β を使った drop 関数によって 0 となり，sparse format へと変換される．もし，パスが初期パスであれば，$\mathbf{output_k}$ が $\mathbf{output_{(k,first)}}$ に格納され，$\mathbf{input_{(k+1)}}$ が $\mathbf{input_{(k+1,first)}}$ に格納される．

5. 評価

5.1 アルゴリズム評価

5.1.1 分類

この章では，我々のアルゴリズムの分類タスクへの適用可能性を示し，ハイパーパラメータ α と β を選択する方法を示す．

ここでは，VGG11 をベースにした BCNN を用いて，我々の近似手法の性能を検証する．他の多くの場合と同様に，バッチ正規化層を挿入し，活性化関数として ReLU を使用する．また，CIFAR10 データセット，Bayes by Backprop，確率的勾配降下法を用いてモデルを学習させる．

適切なハイパーパラメータを選択するために，100 回のフォワードパスを異なる重みサンプリングで計算した時の，学習済みモデルからの 2 種類の出力分布を比較する．1 つは近似なしの出力分布で，もう 1 つは我々の近似手法を用いた近似出力分布である．出力分布の結果がハイパーパラメータの値によってどのように変化するかを観察するために，2 つの分布の間の L2 ノルム（図 6a）と削減された乗算の平均割合（図 6b）を，パラメータごとにヒートマップを用いて描く．

近似に必要な精度は，どこで必要とされ，適用されるかによって異なる．今，「L2 ノルムが約 0.012 より小さいこと」という要求があると仮定する．「L2 ノルムが約 0.012 より小さい」というのは，直感的に「100 回フォワードパスを計算したときの結果の数が，どのクラスについても約 11 以上の差があってはならない」という意味を持つ．この要求される精度の下で，省略される乗算が最大となるように，ハイパーパラメータを決定する．図 6a と図 6b の結果から，ハイパーパラメータ $\alpha = 0.32$ と $\beta = 1.28$ が得られる．図 6b より，これらのパラメータを用いることで，全体の 81.1%の乗算をスキップすることができる．

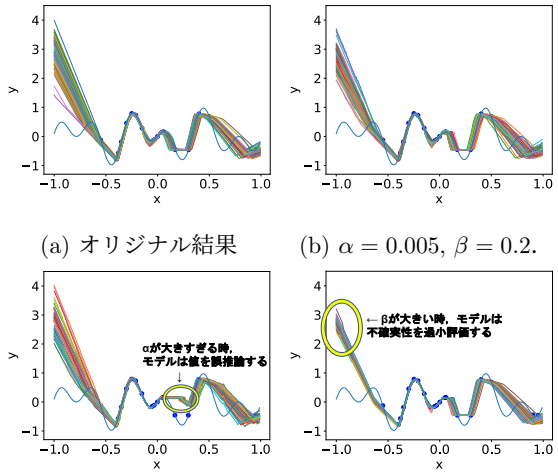
また，test log likelihood（TestLL） [9] を用いて，そのパラメータ下での近似が精度と不確実性を保持しているのかを調べることもできる．TestLL は，予測の精度とその不確実性の両方に依存する評価指標であり，値が大きいほど良い指標である．今回のパラメータ設定では，近似なしの結果の TestLL は-0.23，近似ありの結果の TestLL は-0.20 となっている．よって，このパラメータ設定における近似手法は，TestLL のスコアを若干向上させ，精度や不確実性を悪化させていないことがわかる．また，精度だけで言えば，0.6%の向上となる．これらの指標は，今回のパラメータの選択が，TestLL のスコアと精度を維持しつつ，計算量を削減することに成功していることを示している．

5.1.2 回帰

さらに，回帰タスクを用いて，提案近似手法の有効性を検証する．目的関数 $f(x)$ を $f(x) = \sin(4x) * \cos(14x)$ とし，観測点 X の集合を $X = \{x_1, x_2, ..., x_{20}\}$ とする．対応するターゲットの集合 T を $T = \{t_1, t_2, ..., t_{20}\}$ とする．$i \in \{1, 2, ..., 20\}$ において，$t_i = f(x_i) + noise$ が成立する．$noise$ は標準偏差が 0.05 のガウスノイズである．実装された BNN は，1 つの入力，1 つの出力，そして 3 つの隠れ層を持っており，すべての層は FC 層（1 - 512 - 1024 - 512 - 1）である．モデルの学習には，Bayes by Backprop と確率的勾配降下法を用いる．図 7 中の点は，学習用に用意された観測点 (X, T) を示している．これらの図では，目的関数曲線（青線）と，重みを 50 回サンプリングして得られた回帰曲線（カラフルな線）を示している．

図 7 では，オリジナルの結果（図 7a）と近似された結果（図 7b）の類似性を視覚的に示している．この近似には，$\alpha = 0.005$ and $\beta = 0.2$ を用い，77.7% の計算を削減している．このように，我々の近似手法では，近似なしの場合とほぼ同じ出力分布が得られることがわかる．また，近似手法を用いない場合の TestLL は -0.65，近似手法を用いた場合の TestLL は -0.64 となる．このことから，提案近似手法は TestLL を若干改善し，精度や不確実性を悪化させていないと結論づけることができる．

図 7c と図 7d では，α と β が出力分布に与える影響を視



(a) オリジナル結果　(b) $\alpha = 0.005, \beta = 0.2$.

(c) $\alpha = 0.09, \beta = 0.2$.　(d) $\alpha = 0.005, \beta = 3$.

図 7: 異なるパラメータによる近似.

表 1: アーキテクチャパラメータ設定

パラメータ	値
乗算のビット幅	16 bits
累積のビット幅	24 bits
乗算器アレイ $(I \times K)$	4×4
PE の数	16

覚的に見ることができる．α を大きくしすぎると出力値を誤推論してしまいやすくなり（図 7c），β を大きくしすぎると，不確実性を過小評価してしまいやすくなる（図 7d）ことがわかる．

5.2　ハードウェア評価

5.2.1　評価設定

上で説明したように, ASBNN は PT-IS-CP データフローと SCNN をベースにしている．ASBNN データフローとアーキテクチャを評価するために, ASBNN を 2 つの SCNN ベースラインと比較する．1 つは，PT-IS-CP データフローと SCNN を用いて，素朴にフォワードパスを *num_passes* 回計算する “SCNN (dense)”．もう一つは PT-IS-CP データフローと SCNN を用いてフォワードパスを *num_passes* 回計算するが，3.5 章で説明したスパース性を活用するメカニズムを追加している，“SCNN (sparse)” である．これらを ASBNN と，速度向上とエネルギー効率の観点で比較を行う．ASBNN は，リアルタイム推論を必要とする低レイテンシー重視のアプリケーションを対象としているため，ここでいう速度向上は，レイテンシーの改善を意味する．

アーキテクチャを評価するためには，アーキテクチャのパラメータ設定が必要である．ASBNN は SCNN をベースにしているので，SCNN の設定を参考にして，アーキテクチャのパラメータを設定する．設定したパラメータは，表 1 の通りである．

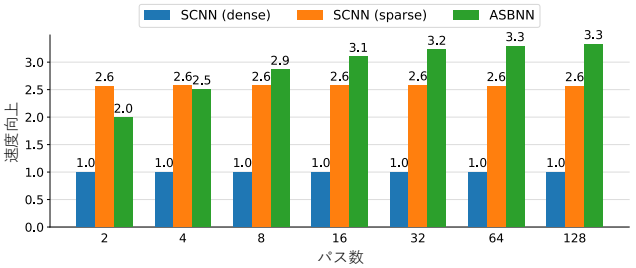


図 8: SCNN (dense) と SCNN (sparse) と ASBNN の速度向上の比較.

ASBNN の PE は, 図 4 のように, PPU 以外の部分は 2 つずつ用意されており，それらの部分は単純に SCNN の PE の 2 倍とみなすことができる．ASBNN の PPU と SCNN の PPU との大きな違いは，ASBNN では，$\mathbf{input}_{(\mathbf{k+1,first})}$ と $\mathbf{output}_{(\mathbf{k,first})}$ を格納するレジスタが追加されていることである．なお，これらを合わせたサイズは常に，ダブルバッファリングされた累積バッファと同じか，それよりも小さくなる．また，SCNN の PE では，ダブルバッファリングされた累積バッファが PE の 29%を占めていることから，ASBNN の PE は SCNN の PE の $2 + 0.29 = 2.29$ 倍以下の大きさになることが言える．以上から，公平な評価を行うために，SCNN には 36 個の PE を用意するのに対し，ASBNN には 16 個の PE を用意する．

5.2.2　パフォーマンス

評価のために，16 個の PE の ASBNN と 36 個の PE の SCNN のサイクルアキュレイトなシミュレータを開発し，CONV 層に必要なサイクル数をカウントする．シミュレータでは，PE 内のローカルバッファやグローバルバッファを含むメモリサブシステムにおけるデータの動きも正確にモデル化されている．評価には，5.1 章で使用した学習済みの VGG11 ベースの BCNN モデルとパラメータ設定を使用する．

なお，BNN の出力分布を評価する際には，2.1 章で説明したように，異なる重みで複数回フォワードパスを計算することに注意する．ASBNN は計算するフォワードパスの回数によって性能が異なる．そのため，我々は ASBNN を異なるフォワードパスの回数で比較する．評価に用いたパスの数は 2, 4, 8, 16, 32, 64, 128 である．結果は図 8 に記されている．

図 8 より，ASBNN はどのパス数でも SCNN (dense) より高速である．また，出力分布を 128 回のパスで比較した場合，ASBNN は SCNN (dense) よりも 3 倍以上高速になる．さらに，8 回以上のパスを計算した場合，ASBNN は SCNN (sparse) よりも高速である．しかし，パスの数が 2 または 4 の場合，初期パスのオーバーヘッドが相対的に大きくなるため，ASBNN は SCNN (sparse) よりも遅くなる．

表 2: 評価に用いられたエネルギー消費量．これらは，[11] や [12] や [8] で提案されている．ここでの数字は [12] の結果を用いて計算されている．

	エネルギー (pJ)	相対コスト
加算 (16 bit)	0.4	1×
乗算 (16 bit)	1.1	2.75×
ロード/ストア (16 bit)	9.6	24×

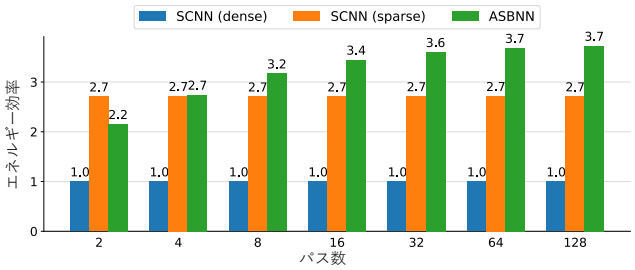


図 9: SCNN (dense) と SCNN (sparse) と ASBNN のエネルギー効率の比較．

5.2.3 エネルギー効率

エネルギー効率の評価には，性能評価と同じモデルを使用する．まず，先程のシミュレータを用いて乗算，加算，ロード/ストアの回数をカウントし，その回数を使ってエネルギー消費量を計算する．演算処理数とロード/ストア数に，それぞれのエネルギー消費量の比率を乗じてエネルギー消費量を算出する．各演算のエネルギー消費量は表 2 のように定める．

ASBNN のエネルギー効率は，出力分布を計算するためのパスの数に依存するため，我々は複数のパス数でエネルギー効率の比較を行った．結果は図 9 のようになった．

その結果，ASBNN は SCNN (dense) と比較して，どのようなパスの数でもエネルギーを節約できることがわかる．パスが 2 回しか計算されない場合は，最初のパスのオーバーヘッドが比較的大きく，ASBNN のエネルギー効率の妨げになる．しかし，出力分布を計算するパスの数が多い場合，ASBNN はより多くのエネルギーを節約することができる．

5.3 議論

アルゴリズムの評価では，密なニューラルネットワークに比べて，分類タスクで 81.1%，回帰タスクで 77.7%の計算量を削減できることを示している．ハードウェアの評価では，パスの数を変えた ASBNN の性能とエネルギー効率を観察する．BCNN を用いて出力分布を計算する場合，2 回のパスや 4 回のパスでは出力分布が十分に近似できないため，10〜100 回のパスを計算することが多い．したがって，ASBNN を 64 回のパスの場合で評価することは合理的である．この場合，ASBNN は密な BCNN アクセラレータの素朴な実装と比較して，計算速度を 3.3 倍，エネルギー効率を 3.7 倍改善すると結論づけることができる．

6. 関連研究

我々の知る限りでは，二つの BNN アクセラレータが存在する．VIBNN は，始めに提案された BNN アクセラレータである．彼らは，我々の研究と同様に，ガウス変分推論の高速化を試みており，乱数生成器の設計を複数提案し，比較している．さらに，BNN のための空間アーキテクチャを提案している．しかし，彼らのハードウェアは FC 層のみを対象としているが，[1, 8] によると，VGG11 や AlexNet [13] など，最も広く使われている CNN では，CONV 層が全体の計算のほぼ 90%を占めている．そのため，本稿では主に CONV 層を考慮している．

もう一つのアクセラレータは Fast-BCNN [20] で，最初の BCNN アクセラレータである．著者らは，BNN を実現する手段としてドロップアウト [10] を用い，影響を受けないニューロンを予測することで，不要な計算をスキップできるアクセラレータを提案している．しかし，ドロップアウトでは，事後分布として表現される分布がベルヌーイ分布に限定されてしまい，一般性が損なわれてしまう．我々のアプローチであるガウス変分推論は，より多くのクラスの分布を事後分布として表現することができるため，より一般的で応用性の高いものとなっている [6]．さらに，VIBNN や Fast-BCNN は，BNN が推論を行うために複数のフォワードパスを計算する必要があるという事実を適切に考慮していない．我々の手法は，ガウス変分推論における複数のフォワードパスの類似性を利用した高速化を提案している．

7. まとめ

本稿では，BCNN の高速計算を実現するために，近似アルゴリズムとハードウェアのサポートによる新しいアルゴリズム・ハードウェア協調設計手法を提案した．BCNN の特性を利用した近似アルゴリズムでは，評価の結果，分類タスクで 81.1%，回帰タスクで 77.7%の計算を省略することができるとわかった．さらに，このような計算省略の機会を利用するために，BCNN 用のアクセラレータを開発した．我々は，提案アルゴリズムに特化した新しいデータフローを提案し，我々のアルゴリズムによって抽出されたスパース性を扱うことができる新しいアクセラレータアーキテクチャ，ASBNN を開発した．最後に，評価の結果，ASBNN は計算量を減らすことに成功し，BCNN アクセラレータの素朴な実装と比較して，計算速度を 3.3 倍，エネルギー効率を 3.7 倍向上させた．

謝辞 本稿の一部は，JSPS 科研費 19H04075, 18H05288, および JST さきがけ JPMJPR18M9 の支援により行われたものである．

参考文献

[1] Akhlaghi, V., Yazdanbakhsh, A., Samadi, K., Gupta, R. K. and Esmaeilzadeh, H.: SnaPEA: Predictive Early Activation for Reducing Computation in Deep Convolutional Neural Networks, *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pp. 662–673 (2018).

[2] Amini, A., Soleimany, A., Karaman, S. and Rus, D.: Spatial uncertainty sampling for end-to-end control, *arXiv preprint arXiv:1805.04829* (2018).

[3] Andrieu, C., De Freitas, N., Doucet, A. and Jordan, M. I.: An introduction to MCMC for machine learning, *Machine learning*, Vol. 50, No. 1-2, pp. 5–43 (2003).

[4] Barber, D. and Bishop, C. M.: Ensemble learning in Bayesian neural networks, *Nato ASI Series F Computer and Systems Sciences*, Vol. 168, pp. 215–238 (1998).

[5] Blei, D. M., Kucukelbir, A. and McAuliffe, J. D.: Variational inference: A review for statisticians, *Journal of the American statistical Association*, Vol. 112, No. 518, pp. 859–877 (2017).

[6] Blundell, C., Cornebise, J., Kavukcuoglu, K. and Wierstra, D.: Weight Uncertainty in Neural Networks, ICML'15, JMLR.org, p. 1613–1622 (2015).

[7] Cai, R., Ren, A., Liu, N., Ding, C., Wang, L., Qian, X., Pedram, M. and Wang, Y.: VIBNN: Hardware Acceleration of Bayesian Neural Networks, ASPLOS '18, New York, NY, USA, Association for Computing Machinery, p. 476–488 (2018).

[8] Chen, Y., Emer, J. and Sze, V.: Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks, *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pp. 367–379 (2016).

[9] Gal, Y.: Uncertainty in deep learning, *University of Cambridge*, Vol. 1, No. 3 (2016).

[10] Gal, Y. and Ghahramani, Z.: Dropout as a bayesian approximation: Representing model uncertainty in deep learning, *international conference on machine learning*, pp. 1050–1059 (2016).

[11] Han, S., Liu, X., Mao, H., Pu, J., Pedram, A., Horowitz, M. A. and Dally, W. J.: EIE: Efficient Inference Engine on Compressed Deep Neural Network, *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pp. 243–254 (2016).

[12] Horowitz, M.: 1.1 Computings energy problem (and what we can do about it), *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pp. 10–14 (2014).

[13] Krizhevsky, A., Sutskever, I. and Hinton, G. E.: ImageNet Classification with Deep Convolutional Neural Networks, *Commun. ACM*, Vol. 60, No. 6, p. 84–90 (2017).

[14] Kwon, Y., Won, J.-H., Kim, B. J. and Paik, M. C.: Uncertainty quantification using Bayesian neural networks in classification: Application to biomedical image segmentation, *Computational Statistics and Data Analysis*, Vol. 142, No. C (2020).

[15] Milz, S., Arbeiter, G., Witt, C., Abdallah, B. and Yogamani, S.: Visual SLAM for Automated Driving: Exploring the Applications of Deep Learning, *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops* (2018).

[16] Parashar, A., Rhu, M., Mukkara, A., Puglielli, A., Venkatesan, R., Khailany, B., Emer, J., Keckler, S. W. and Dally, W. J.: SCNN: An accelerator for compressed-sparse convolutional neural networks, *ACM SIGARCH Computer Architecture News*, Vol. 45, No. 2, pp. 27–40 (2017).

[17] Pierson, H. A. and Gashler, M. S.: Deep learning in robotics: a review of recent research, *Advanced Robotics*, Vol. 31, No. 16, pp. 821–835 (2017).

[18] Salimans, T., Kingma, D. and Welling, M.: Markov chain monte carlo and variational inference: Bridging the gap, *International Conference on Machine Learning*, pp. 1218–1226 (2015).

[19] Simonyan, K. and Zisserman, A.: Very Deep Convolutional Networks for Large-Scale Image Recognition, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (Bengio, Y. and LeCun, Y., eds.) (2015).

[20] Wan, Q. and Fu, X.: Fast-BCNN: Massive Neuron Skipping in Bayesian Convolutional Neural Networks, *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 229–240 (2020).

[21] Young, T., Hazarika, D., Poria, S. and Cambria, E.: Recent Trends in Deep Learning Based Natural Language Processing [Review Article], *IEEE Computational Intelligence Magazine*, Vol. 13, No. 3, pp. 55–75 (online), DOI: 10.1109/MCI.2018.2840738 (2018).