

(19)日本国特許庁(JP)

(12)特許公報(B2)

(11)特許番号

特許第7344259号

(P7344259)

(45)発行日 令和5年9月13日(2023.9.13)

(24)登録日 令和5年9月5日(2023.9.5)

(51)国際特許分類

F I

G 0 6 N 3/08 (2023.01)

G 0 6 N 3/08

G 0 6 N 3/04 (2023.01)

G 0 6 N 3/04

請求項の数 17 (全18頁)

(21)出願番号 特願2021-143795(P2021-143795)
 (22)出願日 令和3年9月3日(2021.9.3)
 (65)公開番号 特開2021-192282(P2021-192282 A)
 (43)公開日 令和3年12月16日(2021.12.16)
 審査請求日 令和3年9月3日(2021.9.3)
 (31)優先権主張番号 202011242902.8
 (32)優先日 令和2年11月9日(2020.11.9)
 (33)優先権主張国・地域又は機関
 中国(CN)

(73)特許権者 514322098
 ベイジン バイドゥ ネットコム サイエ
 ンス テクノロジー カンパニー リミテ
 ッド
 Beijing Baidu Netco
 m Science Technolog
 y Co., Ltd.
 中華人民共和国 ペキン 100085,
 ハイディアン ディストリクト, シャン
 ディ テンス ストリート, 10番, バ
 イドゥ キャンパス 2階
 2/F Baidu Campus, N
 o. 10, Shangdi 10th
 Street, Haidian Dis
 trict, Beijing 1000
 最終頁に続く

(54)【発明の名称】 深層学習フレームワークにおけるパターン変換方法、装置、電子デバイス、コンピュータ
 ー記憶媒体およびコンピュータプログラム製品

(57)【特許請求の範囲】

【請求項1】

第1のパターンにおける第1の文法要素と前記第1のパターンにおける第1の文法要素との間のコンテキスト関係に基づいて前記第1のパターンを更新し、更新された第1のパターンを取得することと、

前記更新された第1のパターンにおける更新された第1の文法要素と第2のパターンシステムにおける第2の文法要素との間のマッピング関係に基づいて、前記更新された第1の文法要素に対応する第2の文法要素を生成することと、

前記更新された第1の文法要素のコンテキスト関係に従って前記第2の文法要素を組み合わせて第2のパターンを生成することと、を含み、

前記更新された第1の文法要素の数は、前記第1の文法要素の数よりも少ない
 ことを特徴とする深層学習フレームワークにおけるパターン変換方法。

【請求項2】

前記第1の文法要素は第1の演算子を含み、前記第2の文法要素は第2の演算子を含み、前記方法は、

前記第2のパターンが前記第2の演算子に対応する第1の演算子によって実行されることをさらに含む

ことを特徴とする請求項1に記載の方法。

【請求項3】

前記第1のパターンの第1の識別子は、高速記憶装置に記憶され、

前記方法は、

前記第 1 のパターンの第 1 の識別子を取得することと、

別の第 1 のパターンの第 2 の識別子を決定することと、

前記第 1 の識別子が前記第 2 の識別子とマッチングすると決定すると、前記第 2 のパターンを前記第 2 パターンシステムにおける前記別の第 1 パターンに対応するパターンに決定することと、をさらに含む

ことを特徴とする請求項 1 に記載の方法。

【請求項 4】

前記第 1 の文法要素がループ演算子および条件演算子を含み、

前記第 1 のパターンにおける第 1 の文法要素と前記第 1 のパターンにおける第 1 の文法要素との間のコンテキスト関係に基づいて前記第 1 のパターンを更新することは、

前記条件演算子と前記ループ演算子との前文関係が前記第 1 のパターンで隣接していると決定すると、前記条件演算子と前記ループ演算子を別のループ演算子に更新することを含む

ことを特徴とする請求項 1 に記載の方法。

【請求項 5】

前記第 1 の文法要素は定数を含み、

前記第 1 のパターンにおける第 1 の文法要素と前記第 1 のパターンにおける第 1 の文法要素との間のコンテキスト関係に基づいて前記第 1 のパターンを更新することは、

前記第 1 のパターンにおける重複な定数を除去することを含む

ことを特徴とする請求項 1 に記載の方法。

【請求項 6】

前記第 1 のパターンは、命令型プログラミングを介して実施され、

前記第 2 のパターンは、宣言型プログラミングを介して実施される

ことを特徴とする請求項 1 に記載の方法。

【請求項 7】

前記第 1 のパターンが実施されるのに必要なマシンコストは、前記第 2 のパターンが実施されるのに必要なマシンコストよりも大きい

ことを特徴とする請求項 1 に記載の方法。

【請求項 8】

第 1 のパターンにおける第 1 の文法要素と前記第 1 のパターンにおける第 1 の文法要素との間のコンテキスト関係に基づいて前記第 1 のパターンを更新し、更新された第 1 のパターンを取得するように配置される第 1 の更新モジュールと、

前記更新された第 1 のパターンにおける更新された第 1 の文法要素と第 2 のパターンシステムにおける第 2 の文法要素との間のマッピング関係に基づいて、前記更新された第 1 の文法要素に対応する第 2 の文法要素を生成するように配置される文法要素変換モジュールと、

前記更新された第 1 の文法要素のコンテキスト関係に従って前記第 2 の文法要素を組み合わせ第 2 のパターンを生成するように配置されるパターン生成モジュールことと、を含み、

前記更新された第 1 の文法要素の数は、前記第 1 の文法要素の数よりも少ない

ことを特徴とする深層学習フレームワークにおけるパターン変換装置。

【請求項 9】

前記第 1 の文法要素は第 1 の演算子を含み、前記第 2 の文法要素は第 2 の演算子を含み、

前記パターン変換装置は、

前記第 2 のパターンを前記第 2 の演算子に対応する第 1 の演算子によって実行させるように配置されるパターン実行モジュールをさらに含む

ことを特徴とする請求項 8 に記載のパターン変換装置。

【請求項 10】

前記第 1 のパターンの第 1 の識別子は、高速記憶装置に記憶され、

10

20

30

40

50

前記パターン変換装置は、

前記第 1 のパターンの第 1 の識別子を取得するように配置される第 1 の識別子決定モジュールと、

別の第 1 のパターンの第 2 の識別子を決定するように配置される第 2 の識別決定モジュールと、

前記第 1 の識別子が前記第 2 の識別子とマッチングすると決定すると、前記第 2 のパターンを前記第 2 パターンシステムにおける前記別の第 1 パターンに対応するパターンに決定するように配置されるパターン決定モジュールと、をさらに含む

ことを特徴とする請求項 8 に記載のパターン変換装置。

【請求項 1 1】

前記第 1 の文法要素がループ演算子および条件演算子を含み、

前記第 1 の更新モジュールは、

前記条件演算子と前記ループ演算子との前文関係が前記第 1 のパターンで隣接していると決定すると、前記条件演算子と前記ループ演算子を別のループ演算子に更新するように配置される第 2 の更新モジュールを含む

ことを特徴とする請求項 8 に記載のパターン変換装置。

【請求項 1 2】

前記第 1 の文法要素は定数を含み、

前記第 1 のパターンにおける第 1 の文法要素と前記第 1 のパターンにおける第 1 の文法要素との間のコンテキスト関係に基づいて前記第 1 の更新モジュールは、

前記第 1 のパターンにおける重複な定数を除去するように配置される第 3 の更新モジュールを含む

ことを特徴とする請求項 8 に記載のパターン変換装置。

【請求項 1 3】

前記第 1 のパターンは、命令型プログラミングを介して実施され、

前記第 2 のパターンは、宣言型プログラミングを介して実施される

ことを特徴とする請求項 8 に記載のパターン変換装置。

【請求項 1 4】

前記第 1 のパターンが実施されるのに必要なマシンコストは、前記第 2 のパターンが実施されるのに必要なマシンコストよりも大きい

ことを特徴とする請求項 8 に記載のパターン変換装置。

【請求項 1 5】

1 つ又は複数のプロセッサと、

前記 1 つ又は複数のプロセッサによって実行されると、前記 1 つ又は複数のプロセッサに請求項 1 ～ 7 のいずれか一項に記載の方法を実現させる 1 つ又は複数のプログラムを記憶する記憶装置と、を含む

ことを特徴とする電子デバイスである。

【請求項 1 6】

プロセッサによって実行されると請求項 1 ～ 7 のいずれか一項に記載の方法を実現するコンピュータプログラムを記憶する

ことを特徴とするコンピュータ読み取り可能な記憶媒体。

【請求項 1 7】

プロセッサによって請求項 1 ～ 7 のいずれか一項に記載の方法を実現するコンピュータプログラム命令を含む

ことを特徴とするコンピュータプログラム。

【発明の詳細な説明】

【技術分野】

【0001】

本開示の実施例は、マシン学習分野に関し、且つより具体的には、パターン変換方法、装置、電子デバイス、コンピューター記憶媒体及びコンピュータープログラム製品に関する。

10

20

30

40

50

【背景技術】

【0002】

現在の深層学習フレームワークは、使いやすさと性能の配置問題をできるだけ同時に解決している。現在、主要な深層学習フレームワークには、命令型プログラミング（動的グラフとも呼ばれる）と宣言型プログラミング（静的グラフとも呼ばれる）の2つのパターンがある。命令型プログラミングとは、書きパターンが計算センテンスを実行した直後に解釈して実行し、すぐにこの計算結果を所有することを意味する。このメカニズムは、ユーザのプログラミングの習慣により適合し、パターンデバッグを行いやすく、ユーザが記述しやすくなるが、マシン実行コンピューティングリソースの消費が多い。宣言型プログラミングとは、プログラムをコンパイルして実行するときに、最初に計算のグラフ構造を構築し、次に実行ユニットを使用してグラフ構造操作を実行することを意味する。このメカニズムは、ユーザが記述しやすくないが、マシン実行コンピューティングリソースの消費が少なく、産業モデルの配置に適している。

10

【発明の概要】

【発明が解決しようとする課題】

【0003】

したがって、命令型プログラミング（動的グラフとも呼ばれる）と宣言型プログラミング（静的グラフとも呼ばれる）との間にパターン変換を可能にするパターン変換案が必要である。

【課題を解決するための手段】

20

【0004】

本開示の実施例によれば、パターン変換案が提供される。

本開示の第1の方面に、パターン変換方法が提供されており、第1のパターンにおける第1の文法要素と第1のパターンにおける第1の文法要素との間のコンテキスト関係に基づいて第1のパターンを更新し、更新された第1のパターンを取得することと、更新された第1のパターンにおける更新された第1の文法要素と第2のパターンシステムにおける第2の文法要素との間のマッピング関係に基づいて、更新された第1の文法要素に対応する第2の文法要素を生成することと、更新された第1の文法要素のコンテキスト関係に従って第2の文法要素を組み合わせて第2のパターンを生成することと、を含む。

【0005】

30

本開示の第2の方面に、パターン変換装置が提供されており、第1のパターンにおける第1の文法要素と第1のパターンにおける第1の文法要素との間のコンテキスト関係に基づいて第1のパターンを更新し、更新された第1のパターンを取得するように配置される第1の更新モジュールと、更新された第1のパターンにおける更新された第1の文法要素と第2のパターンシステムにおける第2の文法要素との間のマッピング関係に基づいて、更新された第1の文法要素に対応する第2の文法要素を生成するように配置される文法要素変換モジュールと、更新された第1の文法要素のコンテキスト関係に従って第2の文法要素を組み合わせて第2のパターンを生成するように配置されるパターン生成モジュールと、を含む。

【0006】

40

本開示の第3の方面に、電子デバイスが提供されており、1つ又は複数のプロセッサと、1つ又は複数のプロセッサによって実行されると、1つ又は複数のプロセッサに本開示の第1の方面の方法を実現させる1つ又は複数のプログラムを記憶する記憶装置と、を含む。

【0007】

本開示の第4の方面に、コンピュータ読み取り可能な記憶媒体が提供されており、プロセッサによって実行されると本開示の第1の方面の方法を実現するコンピュータプログラムを記憶する。

【0008】

本開示の第5の方面に、コンピュータプログラム製品が提供されており、プロセッサに

50

よって本開示の第1の方面の方法を実現するコンピュータプログラム命令を含む。

【0009】

この部分で説明した内容は、本開示の実施例の中心な又は重要な特徴を表記するものではなく、本開示の範囲を限定するものでもないと理解すべきである。本開示の他の特徴は、以下の説明によって理解し易くなるであろう。

【0010】

添付の図面を組合せ、以下の詳細な説明を参照して、本開示の各実施例の上記および他の特徴、利点および態様がより明らかになるであろう。図面では、同じまたは類似の符号は、同じまたは類似の要素を示す。

【図面の簡単な説明】

【0011】

【図1】図1は、本開示の複数の実施例が実現される例示的な環境の概略図を示している。

【図2】図2は、本開示のいくつかの実施例によるパターン変換のプロセスのフローチャートを示している。

【図3】図3は、本開示のいくつかの実施例によるパターンを更新するプロセスのフローチャートを示している。

【図4】図4は、本開示のいくつかの実施例によるパターン変換の例示的なプロセスの概略図を示している。

【図5】図5は、本開示のいくつかの実施例によるパターン変換の装置の概略ブロック図を示している。

【図6】図6は、本開示の複数の実施例を実施することができるコンピューティングデバイスのブロック図を示している。

【発明を実施するための形態】

【0012】

以下、本開示の好ましい実施例を、添付の図面を参照してより詳細に説明する。本開示の好ましい実施例が図面に示されているが、本開示が様々な形態で実現することができ、本明細書に記載の実施例によって限定されるべきではないことを理解されたい。逆に、これらの実施例は、本開示をより明瞭かつ完全にし、そして本開示の範囲を当業者に完全に伝えるために提供される。

【0013】

本明細書で使用されている「含む」という用語とその変形は、制限のない包含、つまり「含むがこれに限定されない」ことを意味する。特に明記しない限り、「または」という用語は「および／または」を意味する。「に基づく」という用語は、「少なくとも部分的に基づく」ことを意味する。「例示的な実施例」および「一実施例」という用語は、「少なくとも1つの例示的な実施例」を意味する。「別の実施例」という用語は、「少なくとも1つの別の実施例」を意味する。

【0014】

「第1」、「第2」などの用語は、異なるまたは同じオブジェクトを指すことができる。以下には、他の明示的および暗黙的な定義も含まれる場合がある。

【0015】

前述のように、使いやすさと配置性の両方を併有する深層学習フレームワークを実現するように、命令型プログラミング（動的グラフとも呼ばれる）と宣言型プログラミング（静的グラフとも呼ばれる）の間でパターン変換を実行する技術案が必要である。従来技術は、動的グラフから静的グラフに変換する（以下、動的から静的）モジュールを開発することによって、フレームワークシステムのユーザが動的グラフパターンを使用して深層学習フレームワークを便利に構築し、静的グラフに変換して配置性能の利点を取得する。具体的には、動的グラフパターンを静的グラフパターンまたは静的グラフの中間表現IRに変換する。上記の2つの変換では、共にコンパイラに関連する技術を使用する。実現手順は、最初に動的グラフパターンのフロントエンドコードを抽象文法ツリー（abstract syntax tree、ASTと呼ばれる）に解析し、そして、抽象文法ツリー

10

20

30

40

50

から静的グラフパターン又は静的グラフの中間表現IRを生成するということである。従来技術には主に以下の欠点がある。

【0016】

(1) 動的から静的へのプロセスにおける高級言語の最適化を欠けており、動的グラフパターンにおける一部のセンテンスが冗長であるか、ループのネストが複雑であり、最適化せずに静的グラフに直接変換すると、静的グラフで実行することにコンピューティングリソースを大量に消費する。

【0017】

(2) ソフトウェア工学では、動的グラフと静的グラフとのパターンの一致性または運行方式の一貫性に比較的高い制限がある。動的グラフと静的グラフとのパターン差分が小さいほど、変換作業は少なくなる。この場合、変換前後のパターンが動的グラフと静的グラフで同じ方式で実行できるように、いくつかの深層学習フレームワークでは、静的グラフと動的グラフの運行インターフェイスを一致になるように定義する必要がある。動的グラフパターンと静的グラフパターンのインターフェイスを厳密に同じにするために、深層学習フレームワークに対する要求は高くなる。

【0018】

本開示の実施例によれば、パターン変換の技術案が提出される。この技術案は、パターン間の変換プロセスに最適化を行う。この技術案では、まず、第1のパターンを更新する。次に、マッピング関係によって、更新されたパターンの文法要素を第2のパターンの文法要素に変換する。最後に、コンテキスト関係に基づいて、第2の文法要素を結合して第2のパターンを生成する。

【0019】

パターン変換プロセスに最適化プロセスを追加することにより、変換されたパターンの実行速度を向上させることができ、異なる最適化およびマッピングモデルを確立することにより、パターンのデバッグを容易にすることができる。

【0020】

以下、添付の図面を参照して、本開示の基本原則およびいくつかの例示的な実施例を説明する。

【0021】

図1は、本開示の複数の実施例が実現される例示的な環境100の概略図を示している。図1に示される環境100が単なる例示であり、本開示に記載される実現の機能および範囲を制限するべきではないことを理解されたい。図1に示すように、環境100は、第1のパターン110、第1の文法要素120、更新された第1の文法要素170、更新された第1のパターン130、第2のパターン140、第2の文法要素150、およびコンピューティングデバイス160を含む。

【0022】

第1のパターン110は、動的グラフパターンであり得る。本開示の実施例では、説明および理解の便宜上、「パターン」という用語は、コードの形態で表現され、PYTHONのネイティブ文法要素(for、if、else、while)を介してネットワークを構築することができる。第1のパターン110は、命令型プログラミングによって実施され、マシンによって実行することは容易ではなく、すなわち、マシンによってこの第1のパターン110を実行するために必要なマシンコストが多く、例えば、多くのグラフィックス処理ユニット(GPU)リソースまたは大量の時間が必要である。

【0023】

コンピューティングデバイス160は、第1のパターン110を解析することができる。例えば、コンピューティングデバイス160は、パーサーを使用して、第1のパターン110における第1の文法要素120およびそれらの間のコンテキスト関係を解析することができる。ここでの第1の文法要素120は、パターンの基本構成ユニットを指すことができる。本願の実施例では、文法要素は、各パターン言語における基本構成要素を指す。例えば、アプリケーションプログラムのソースパターンファイルがPYTHON言語で

10

20

30

40

50

ある場合、パターンにおける文法要素は、制御フロー（`for`、`if`、`else`、`while`）、入力及び出力（`input`、`print`）、プログラム形式、データ類型、変数などであることができ、例えば J A V A（登録商標）言語である場合、クラス、属性、パターンブロック、アノテーション、コメントなどであることができる。コンピューティングデバイスは、第 1 のパターン 1 1 0 に対する解析結果に従って、第 1 のパターンを更新（最適化）し、第 2 のパターン 1 4 0 に変換することができ、具体的なプロセスを以下に説明する。上記の P Y T H O N 言語は単なる例示であり、第 1 のパターン 1 1 0 はまた、ユーザの書き込みに適した任意の言語システムに基づいて作成されることができる。

【0024】

コンピューティングデバイス 1 6 0 は、コンピューティング機能を備えた任意のデバイスであり得る。非限定的な例として、コンピューティングデバイス 1 6 0 は、任意のタイプの固定コンピューティングデバイス、モバイルコンピューティングデバイス、またはポータブルコンピューティングデバイスであることができ、デスクトップコンピュータ、ラップトップコンピュータ、ノートブックコンピュータ、ネットブックコンピュータ、タブレットコンピュータ、マルチメディアコンピュータ、携帯電話などを含むが、これらに限定されない。コンピューティングデバイス 1 6 0 の構成要素の全部または一部は、クラウドに分散され得る。計算、記憶、通信、および制御などの機能を実現するために、コンピューティングデバイス 1 6 0 は、少なくとも、プロセッサ、メモリ、および汎用コンピュータに存在する他のコンポーネントを含む。

【0025】

一実施例では、コンピューティングデバイス 1 6 0 は、事前にトレーニングされた文法パーサー、文法オブティマイザー、文法コンバーター、パターンジェネレーターなどを含み得る。

【0026】

更新された第 1 のパターン 1 3 0 は、第 1 のパターン 1 1 0 と同じ言語システムのパターンであり、更新された第 1 のパターン 1 3 0 における更新された第 1 の文法要素 1 7 0 の数は、第 1 のパターン 1 1 0 の第 1 の文法要素 1 2 0 の数よりも少ない。更新（最適化）プロセスについては、以下で詳細に説明する。

【0027】

第 2 のパターン 1 4 0 は、静的グラフパターンであり得、実行時に各ステップで結果を出力しないが、最終的に結果を全体的に出力する。その実行に必要なマシンコストは、第 1 のパターン 1 1 0 よりも少ない。第 2 のパターン 1 4 0 は、C++または中間表現（Intermediate Representation）パターンであり得、本開示では限定しない。マシンによって実行されるのに適した任意のパターン言語であり得る。第 2 のパターン 1 4 0 は、第 2 の文法要素 1 5 0 を含み得る。

【0028】

以下、図 2～4 を結合して、詳細なパターン変換プロセスをさらに説明する。図 2 は、本開示の実施例によるパターン変換方法 2 0 0 のフローチャートを示している。方法 2 0 0 は、図 1 のコンピューティングデバイス 1 6 0 によって実施され得る。説明の便宜上、方法 2 0 0 は、図 1 を参照して説明される。

【0029】

ブロック 2 1 0 において、コンピューティングデバイス 1 6 0 は、第 1 のパターン 1 1 0 における第 1 の文法要素 1 2 0 と第 1 のパターンにおける第 1 の文法要素 1 2 0 との間のコンテキスト関係に基づいて、第 1 のパターン 1 1 0 を更新して、更新された第 1 のパターン 1 3 0 を取得する。

【0030】

一実施例では、更新された第 1 のパターン 1 3 0 における更新された第 1 の文法要素 1 7 0 の数は、第 1 のパターン 1 1 0 における第 1 の文法要素 1 2 0 の数よりも少ない。

【0031】

例えば、コンピューティングデバイス 1 6 0 は、第 1 のパターンに対して文法解析を実

10

20

30

40

50

行し、第1のパターンを、抽象文法ツリー (Abstract Syntax Tree、AST) レベルで第1のパターンの抽象文法ツリーに変換することができる。コンピューティングデバイス160は、ある特定の形式文法に従って第1の文法要素120によって形成された第1のパターン110を分析し、その文法構造、すなわち第1の文法要素120およびそのコンテキスト関係を決定することができる。

【0032】

一実施例では、コンピューティングデバイス160は、その内部または外部の文法パーサーを使用して、第1のパターン110に対する文法解析を実行することができる。文法パーサーは、通常、コンパイラまたはインタープリターのコンポーネントとして使用される。その役割は、文法チェックを実行し、入力された単語で構成されるデータ構造（一般的には、文法解析ツリーや抽象文法ツリーなどの階層化のデータ構造）を構築することである。文法パーサーは、通常、独立した字句アナライザーを使用して、入力文字ストリームから個々の単語を分離し、単語ストリームを入力として使用する。実際の開発では、文法パーサーは手動で作成してもよく、ツールを使用して（半）自動で生成してもよい。

【0033】

抽象文法ツリーは、ソースパターンの文法構造の抽象表現であり、プログラミング言語の構造をツリーのような形で表現し、ツリーの各ノードASTNodeは全てソースパターンにおける一つの構造を表す。抽象文法ツリーは、第1のパターン110における各種の文法要素を相応的な対象として定義し、ツリーの各ノードは、ソースパターンに現れる構造を表す。文法は、実際の文法に表示されるすべての詳細を表さずに構造や内容に関する詳細のみであるので、抽象的なものである。例えば、グループ化括弧はツリー構造に暗黙的に含まれ、3つのブランチを持つ単一のノードによって、if-conditional式と類似する構文構造を表現できる。通常、パーサーによって、ソースアプリケーションパターンの変換およびコンパイルプロセスに構築される。構築されると、後続の処理（例えばコンテキスト分析）によって添付情報をASTに追加する。

【0034】

次に、コンピューティングデバイス160は、前記で得られた第1の文法要素120およびそのコンテキスト関係に従って、第1のパターン110を更新することができる。更新のプロセスは、図3を結合してさらに説明される。図3は、本開示のいくつかの実施例によるパターンを更新するプロセスのフローチャートを示している。

【0035】

310において、コンピューティングデバイス160は、第1の文法要素110に含まれるループ演算子と条件演算子との前文関係が前記第1のパターンで隣接しているかどうかを決定する。例えば、PYTHONを例にとると、第1のパターン110には次のパターンが存在する。

【0036】

```
while i < n:
    if some_condition:
        break
```

コンピューティングデバイス160は、ループ演算子while（ループオペレーター (operator) と呼びでもよい）および条件演算子ifという第1の文法要素120が存在することを解析し、且つコンピューティングデバイス160は、それらが第1のパターン110において隣接していると決定する。

【0037】

320において、コンピューティングデバイス160は、条件演算子と前記ループ演算子を別のループ演算子に更新する。例えば、上記の例示の説明を続けると、コンピューティングデバイス160は、上記のパターンを等価のPYTHONパターンに更新する。

【0038】

```
while i < n and some_condition:
```

その中、一つのループ演算子whileのみがあり、上記の更新された第1のパターン

10

20

30

40

50

130に基づいて変換された第2のパターン140にはループ演算子が1つだけあり、最適化されない前に比べてその後の実行速度が数倍（例えば少なくとも7回）向上する。

【0039】

一つの代替の実施例では、第1の文法要素120が定数を含む場合、コンピューティングデバイス160は、第1のパターン110における重複な定数を除去することができる。例えば、第1のパターンに、 $a = 1$ 、 $b = a$ 、および $c = b$ のパターンが含まれる場合、コンピューティングデバイス160は、それを $a = b = c = 1$ に更新（最適化）することができる。

【0040】

上記の例示は単なる例示であり、それ以外、第1のパターンに対する更新と最適化、例えば定数最適化、冗長な文句の除去の最適化、等価であるがより高速なパターン置換の最適化などがさらにあり、これにより、更新された第1のパターン130における更新された第1の文法要素170の数は、第1のパターン110における第1の文法要素120の数よりも少なくなる。各更新または最適化を、一つの変換関数またはモジュールに制限することもでき、その後で各種の関数またはモジュールを追加および削除して最適化することができ、テストの最適化も非常に柔軟である。第1のパターン、つまり動的グラフパターンの階層レベルで分析及び最適化を行なうにより、変換された静的グラフの実行速度を大幅に向上させることができる。

【0041】

図2に戻り説明を続けると、ブロック220において、コンピューティングデバイス160は、更新された第1のパターン130における更新された第1の文法要素170と第2のパターンシステムにおける第2の文法要素150との間のマッピング関係に基づいて、更新された第1の文法要素170に対応する第2の文法要素150を生成する。

【0042】

一例では、コンピューティングデバイス160は、更新された第1の文法要素170と相応的な第2の文法要素150との間に、例えば `IfElse` という文法要素を変換する `IfElse` コンバーター、ループを変換する `Loop` コンバーター、標準出力を変換する `Print` コンバーターなどの、異なる変換器カテゴリを作成することができる。

【0043】

一実施例では、更新された第1のパターン130における各更新された第1の文法要素170の抽象文法ツリーは、1つのコンバーターの入力とし、当該コンバーターは、変換を行って一つの新しい抽象文法ツリーを第2の文法要素150として出力する。それらの間に文法的な違いがある更新された第1の文法要素170と第2の文法要素150との変換が実行完了した後、更新された第1のパターン130における更新された第1のパターン要素170は、第2の文法要素150に変換される。この構造の利点は、ソフトウェアエンジニアリングで複数の文法要素コンバーターを個別にデバッグでき、異なるタイプの文法要素コンバーターを最適化でき、テスト中に文法要素ごとにテストできることである。

【0044】

ブロック230において、コンピューティングデバイス160は、更新された第1の文法要素170のコンテキスト関係に従って第2の文法要素150を組み合わせて、第2のパターン140を生成する。例えば、コンピューティングデバイスは、更新された第1のパターン130におけるパラメータ、呼び出し関数間の論理関係に従って、変換されたパラメータ、呼び出し関数を結合して、第2のパターン140を生成することができる。

【0045】

一実施例では、第1の文法要素120は第1の演算子を含み、第2の文法要素150は第2の演算子を含み、コンピューティングデバイス160は、第2のモード140を、第2の演算子に対応する第1の演算子によって実行させる。

【0046】

例えば、コンピューティングデバイス160は、第2のパターン140を第1のパターン110の第1の演算子に実行して結合することができ、当該第1の演算子は、第2のパ

10

20

30

40

50

ターン140に変換される演算を含む。当該第1の演算子は、ニューラルネットワークの順方向計算と逆方向計算を実行できる。一つの例示では、例えば第一のパターンは次のとおりである。

【0047】

```
def op__add(x, y);  
return x + y
```

ただし、第1の演算子は加算演算子 `add` であり、コンピューティングデバイス160は、上記の第1のパターンを第2のパターン「第2のパターンの `add`、`x`、`y`」に変換する。「第2のパターンの `add`、`x`、`y`」は、第2のパターン言語で記述された第2の演算子（第2のパターン言語で記述された加算演算子 `add`）を含む。次に、コンピューティングデバイス160は、第2の演算子に対応する第1のパターンの演算子 `add` により第2のパターンを実行し続けることができる。

【0048】

```
def op__add(第2のパターンのadd, x, y):  
sum=第2のパターン__add(x, y)を実行する;  
return sum
```

最後に、実行の結果 `sum` が取得される。

【0049】

上記の例示は、第1の演算子と第2の演算子が単一の演算子、つまり加算演算子 `add` であることを示している。別の例示では、第1の演算子は、結合演算子であり得る。例えば、第1のパターンは、以下であり得る。

【0050】

```
def func(x):  
return softmax(w*x+b)
```

すなわち、第1のパターンの第1の演算子は、行列乗算 $w * x$ にオフセット $+b$ を加えたものであり、最後に `softmax` 処理を行い、コンピューティングデバイス160は、上記の第1のパターンを第2のパターン「第2のパターンの `func(x)`」に変換することができる。ここで、「第2パターンの `func(x)`」には、第2パターン言語で記述された第2演算子（第2パターン言語で記述された `softmax(w*x+b)` 操作）が含み、そして、コンピューティングデバイス160は、第1のパターンの演算子によって第2のパターンを実行し続けることができる。即ち

```
def func(第2のパターンのfunc(x)):  
return 第2のパターンのfunc(x)を実行する。
```

【0051】

変換された第2のパターンを第1のパターンの一部に完全にネストすることにより、異なるパターン間のインターフェースの統一に対する需要を減らすことができる。第1のパターン（動的グラフパターン）と第2のパターン（静的グラフパターン）が非常に異なる深層学習フレームワークで実行されている場合、動的グラフパターンが静的グラフパターンに変換された後、実行パターンが異なるため、静的グラフパターンが元の動的グラフパターンを基礎として実行されることはできない。上記の方式は、この部分のソフトウェア管理の負担を軽減している。実行パターンが異なっても、動的グラフ演算子の方式にして、動的グラフパターンで実行することもできる。また、動的グラフパターンを静的グラフパターンに部分的に変換できる。例えば、ユーザは一部の動的グラフパターンの性能に不満を持っているだけで、他の部分について動的グラフの使いやすさを保持したい。上記の方式は、すべての動的グラフパターンを静的グラフパターンに変換することによりも柔軟である。

【0052】

本開示では、深層学習フレームワークの使いやすさと配置性能の両方を併有している。変換前に命令型プログラミング（動的グラフとも呼ばれる）パターンを更新および最適化することにより、変換後の宣言型プログラミング（静的グラフとも呼ばれる）パターンの

10

20

30

40

50

実行効率を向上させることができる。変換された静的グラフパターンを動的グラフオペレーター（operator）として運行することにより、同じインターフェイスを制限する必要がなく、深層学習フレームワークの許容性が向上する。さらに、異なるタイプの文法要素コンバーターを限定し、コンバーターを最適化することにより、パターン変換を柔軟に分業およびテストでき、コンバーターの追加と削除が便利で、システムを柔軟かつ拡張可能にする。

【0053】

図4は、本開示のいくつかの実施例によるパターン変換の例示的なプロセスの概略図を示す。まず、コンピューティングデバイス160は、第1のパターン110に対する変換を実行するかどうかを判断する。ユーザが指定した第1のパターンについて、深層学習の第1のパターンのアプリケーションプログラミングインターフェイス（Application Programming Interface）の運行は、通常、ビッグデータトレーニングを行なうために、複数のサイクル複数の実行を必要とする。運行ごとにパターンが変換されると、システムの速度が低下する。変換プロセスを高速化するために、システムは、まず、高速メモリ（例えばキャッシュ）を使用して、この部分のパターンが以前に変換されたかどうかを判断する。キャッシュがヒットした場合、つまりこの部分のパターンが以前に変換された場合は、変換された静的グラフパターンを直接に返し、そうでなければ、正式な変換プロセスに入り、今回の変換結果をキャッシュに記憶する。

【0054】

一実施例では、コンピューティングデバイスは、前記第1のパターンの第1の識別子を取得し、別の第1のパターンの第2の識別子を決定し、前記第1の識別子が第2の識別子とマッチングすると決定される場合、前記第2のパターンを前記第2パターンシステムにおける、別の第1パターンに対応するパターンとして決定する。

【0055】

例えば、コンピューティングデバイス160は、履歴における第1のパターンの第1の識別子およびそれに対応する変換された第2のパターンの識別子を識別子ペアとしてキャッシュ410に記憶する。コンピューティングデバイス160は、別の第1のパターンを受信すると、まず、当該別のパターンの第2の識別子を取得する。次に、コンピューティングデバイス160は、当該第2の識別子を上記の識別子ペアにおける第1の識別子と比較する。第1の識別子が第2の識別子とマッチングする場合、別の第1のパターンが第1のパターンと同じまたは機能的に同等であることを意味し、コンピューティングデバイスは、上記の識別子ペアにおける第2のパターンの識別子によって、履歴的に変換された第2のパターンを決定することができる。その後、当該第2のパターンを直接に別の第1のパターンの変換されたパターンとする。

【0056】

コンピューティングデバイス160が、第1の識別子が第2の識別子とマッチングしないと決定した場合、コンピューティングデバイスは、第1のパターン110を第1のパターンの抽象文法ツリー410に解析する。そして、この抽象文法ツリーを更新または最適化し、更新された第1のパターンの抽象文法ツリー420を決定する。次に、コンピューティングデバイス160は、第2のパターンの抽象文法ツリー430を決定する。コンピューティングデバイス160は、最後、第2のパターン140を生成する。詳細なプロセスについては、上記の図2の説明を参照し、ここでは贅言しない。

【0057】

上記の更新または最適化プロセスは、第1パターンに対する操作として示されているが、第2パターンを最適化または更新することもできる。または、第1パターンと第2パターンを同時に更新および最適化することができる。本開示はこれを制限しない。

【0058】

図5は、本開示のいくつかの実施例による、深層学習フレームワークにおけるパターン変換の装置の概略ブロック図を示している。装置500は、図1のコンピューティングデバイス160に含んでもよく、コンピューティングデバイス160として実現されてもよ

10

20

30

40

50

い。

【0059】

図5に示すように、装置500は、第1のパターンにおける第1の文法要素と第1のパターンにおける第1の文法要素との間のコンテキスト関係に基づいて第1の文法要素を更新し、更新された第1のパターンを取得するように配置される第1の更新モジュール510と、更新された第1のパターンにおける更新された第1の文法要素と第2のパターンシステムにおける第2の文法要素との間のマッピング関係に基づいて、更新された第1の文法要素に対応する第2の文法要素を生成するように配置される文法要素変換モジュール520と、更新された第1の文法要素のコンテキスト関係に基づいて第2の文法要素を結合して第2のパターンを生成するように配置されるパターン生成モジュール530と、を含む。

10

【0060】

いくつかの実施例では、更新された第1の文法要素の数は、第1の文法要素の数よりも少ない。

【0061】

いくつかの実施例では、第1の文法要素は第1の演算子を含み、第2の文法要素は第2の演算子を含み得る。装置500は、第2のパターンを第2の演算子に対応する第1の演算子によって実行させるように配置されるパターン実行モジュールをさらに含んでもよい。

【0062】

いくつかの実施例では、第1のパターンの第1の識別子は、高速記憶装置に記憶され、装置500は、第1のパターンの第1の識別子を取得するように配置される第1の識別子決定モジュールと、別の第1のパターンの第2の識別子を決定するように配置される第2の識別子決定モジュールと、第1の識別子が第2の識別子マッチングすると決定されると、第2のパターンを第2パターンシステムにおける、別の第1パターンに対応するパターンに決定するように配置されるパターン決定モジュールと、をさらに含んでもよい。

20

【0063】

いくつかの実施例では、第1の文法要素がループ演算子および条件演算子を含み得る。第1の更新モジュール510は、条件演算子とループ演算子との前文関係が第1のパターンで隣接していると決定すると、条件演算子とループ演算子を別のループ演算子に更新するように配置される第2の更新モジュールを含んでもよい。

30

【0064】

いくつかの実施例では、第1の文法要素は、定数を含み、第1のパターンにおける文法要素と第1のパターンにおける第1の文法要素との間のコンテキスト関係に基づいて、第1の更新モジュール510は、第1のパターンにおける重複な定数を除去するように配置される第3の更新モジュールを含んでもよい。

【0065】

いくつかの実施例では、第1のパターンは、命令型プログラミングによって実施され、第2のパターンは、宣言型プログラミングによって実施される。

【0066】

いくつかの実施例では、第1のパターンが実施されるのに必要なマシンコストは、第2のパターンが実施されるのに必要なマシンコストよりも大きくてもよい。

40

【0067】

図6は、本開示の実施例を実施するための例示デバイス600の概略ブロック図を示している。例えば、図1に示される例示環境100におけるコンピューティングデバイス160は、デバイス600によって実施され得る。図面に示すように、デバイス600は、リードオンリーメモリ（ROM）602に記憶されたコンピュータプログラム命令または記憶ユニット608からランダムアクセスメモリ（RAM）603にロードされたコンピュータプログラム命令に従って、各種の適切な動作及び処理を実行する中央処理ユニット（CPU）601を含む。RAM603には、デバイス600の操作に必要な様々なプログラムおよびデータをさらに記憶することもできる。CPU601、ROM602および

50

RAM 603は、バス604を介して互いに接続されている。入出力（I/O）インターフェース605もまた、バス604に接続されている。

【0068】

デバイス600における複数の部品は、I/Oインターフェース605に接続されており、例えばキーボード、マウスなどの入力ユニット606と、例えば様々なタイプのディスプレイ、スピーカーなどの出力ユニット607と、例えば磁気ディスク、光ディスクなどの記憶ユニット608と、例えばネットワークカード、モデム、無線通信トランシーバなどの通信ユニット609とを含む。通信ユニット609は、デバイス600が、例えばインターネットのコンピューターネットワークおよび／または様々な電気通信ネットワークを介して他のデバイスと情報／データを交換することを許す。

10

【0069】

例えば方法200および300などの上記の各プロセスおよび処理は、処理ユニット601によって実行され得る。例えば、いくつかの実施例では、方法200および300は、例えば記憶ユニット608というマシン読み取り可能な媒体に有形的に含まれるコンピュータソフトウェアプログラムとして実現され得る。いくつかの実施例では、コンピュータプログラムの一部または全部は、ROM 602および／または通信ユニット609を介してデバイス600にロードおよび／またはインストールされ得る。コンピュータプログラムがRAM 603にロードされ、CPU 601によって実行されるとき、上記の方法200および300における1つまたは複数の動作を実行することができる。

【0070】

本開示は、方法、装置、システム、および／またはコンピュータプログラム製品であり得る。コンピュータプログラム製品は、本開示の各態様を実行するためのコンピュータ読み取り可能なプログラム命令がロードされたコンピュータ読み取り可能な記憶媒体を含み得る。

20

【0071】

コンピュータ読み取り可能な記憶媒体は、命令実行デバイスによって使用される命令を保持および記憶することができる有形デバイスであり得る。コンピュータ読み取り可能な記憶媒体は、電気記憶装置、磁気記憶装置、光記憶装置、電磁記憶装置、半導体記憶装置、または上記の任意の適切な組み合わせであり得るが、これらに限定されない。コンピュータ読み取り可能な記憶媒体のより具体的な例（網羅的ではないリスト）は、ポータブルコンピュータディスク、ハードディスク、ランダムアクセスメモリ（RAM）、リードオンリーメモリ（ROM）、消去可能プログラマブルリードオンリーメモリ（EPROMまたはフラッシュ）、スタティックランダムアクセスメモリ（SRAM）、ポータブルコンパクトディスクリードオンリーメモリ（CD-ROM）、デジタル多用途ディスク（DVD）、メモリスティック、フロッピー（登録商標）ディスク、マシンエンコーディングデバイス、例えば命令が保存されたホールカードまたは溝の突出構造、および上記の任意の適切な組み合わせを含む。ここで使用されるコンピュータ読み取り可能な記憶媒体は、無線電波または他の自由に伝播する電磁波、導波路または他の伝送媒体を介して伝播される電磁波（例えば、光ファイバーケーブルを通る光パルス）、またはワイヤーを介して伝播される電気信号などの過渡信号自体として解釈されない。

30

【0072】

ここに記載のコンピュータ読み取り可能なプログラム命令は、コンピュータ読み取り可能な記憶媒体から各コンピューティング／処理デバイスにダウンロードするか、またはネットワーク、ローカルエリアネットワーク、広域ネットワーク、および／またはワイヤレスネットワークを介して外部コンピュータまたは外部記憶装置にダウンロードすることができる。ネットワークには、銅線伝送ケーブル、光ファイバー伝送、無線伝送、ルーター、ファイアウォール、スイッチ、ゲートウェイコンピューター、および／またはエッジサーバーを含むことができる。各コンピューティング／処理デバイスのネットワークアダプタカードまたはネットワークインターフェースは、ネットワークからコンピュータ読み取り可能なプログラム命令を受信し、このコンピュータ読み取り可能なプログラム命令を転

40

50

送して、各コンピューティング／処理デバイスのコンピュータ読み取り可能な記憶媒体に記憶する。

【0073】

本開示の操作を実行するためのコンピュータプログラム命令は、アセンブリ命令、命令セットアーキテクチャ（ISA）命令、マシン命令、マシン関連命令、マイクロコード、ファームウェア命令、状態設定データ、または1つ以上のプログラミング言語の任意の組み合わせで記述されたソースコードまたはオブジェクトコードであってもよい。前記プログラミング言語は、例えばSmalltalk、C++などのオブジェクト指向プログラミング言語と、「C」言語または同様のプログラミング言語などの通常のプロセス型プログラミング言語を含む。コンピュータで読み取り可能なプログラム命令は、完全にユーザのコンピュータで、一部がユーザのコンピュータでスタンドアロンソフトウェアパッケージとして、一部がユーザのコンピュータで、一部がリモートコンピュータで、または完全にリモートコンピュータまたはサーバーで実行できる。リモートコンピュータの場合、リモートコンピュータは、ローカルエリアネットワーク（LAN）やワイドエリアネットワーク（WAN）など、あらゆる種類のネットワークを介してユーザのコンピュータまたは、外部コンピュータに接続することができる（例えば、インターネットサービスプロバイダーを使用してインターネット経由で接続する）。いくつかの実施例では、例えばプログラマブル論理回路、フィールドプログラマブルゲートアレイ（FPGA）、またはプログラマブルロジックアレイ（PLA）などのコンピュータ読み取り可能なプログラム命令を実行することができる電子回路は、コンピュータ読み取り可能なプログラム命令のステータス情報を使用することによってカスタマイズすることができる。これにより、本開示の様々な側面を実現する。

【0074】

本明細書では、本開示の様々な態様が、本開示の実施例による方法、装置（システム）、およびコンピュータプログラム製品のフローチャートおよび／またはブロック図を参照して説明される。フローチャートおよび／またはブロック図の各ブロック、およびフローチャートおよび／またはブロック図のブロックの組み合わせは、コンピュータ読み取り可能なプログラム命令によって実現することができることを理解されたい。

【0075】

これらのコンピュータ読み取り可能なプログラム命令は、汎用コンピュータ、専用コンピュータ、または他のプログラム可能なデータ処理デバイスの処理ユニットに提供することができ、それによって、これらの命令をコンピュータまたは他のプログラム可能なデータ処理デバイスの処理ユニットによって実行する時に、フローチャートおよび／またはブロック図の1つまたは複数のブロックで指定された機能／動作を実現する装置が生成されるようにマシンを製造する。これらのコンピュータ読み取り可能なプログラム命令をコンピュータ読み取り可能な記憶媒体に格納することも可能であり、これらの命令は、コンピュータ、プログラム可能なデータ処理装置、および／または他のデバイスを特定の方式で動作させる。したがって、命令を保存するコンピュータ読み取り可能な媒体は、フローチャートおよび／またはブロック図の1つまたは複数のブロックで指定された機能／動作の様々な側面を実現する命令を含む製造品を含む。

【0076】

一連の操作ステップをコンピュータ、他のプログラム可能なデータ処理装置、または他のマシンで実行するようにコンピュータ読み取り可能なプログラム命令をコンピュータ、他のプログラム可能なデータ処理装置、または他のマシンにロードしてもよい。これにより、コンピュータが実現するプロセスを発生して、コンピュータ、他のプログラム可能なデータ処理装置、または他のマシンで実行される命令が、フローチャートおよび／またはブロック図の1つまたは複数のブロックで指定された機能／動作を実現する。

【0077】

添付の図面のフローチャートおよびブロック図は、本開示の複数の実施例による、システム、方法、およびコンピュータプログラム製品の可能な実現アーキテクチャ、機能、お

10

20

30

40

50

よび動作を示している。これに関して、フローチャートまたはブロック図の各ブロックは、モジュール、プログラムセグメント、または命令の一部を表すことができ、前記モジュール、プログラムセグメント、または命令の一部は、指定された論理機能を実現するための1つまたは複数の実行可能な命令を含む。いくつかの代替の実現では、ブロックでマークされた機能は、図面でマークされた順序と異なる順序で発生する場合もある。例えば、2つの連続するブロックは、関連する機能に応じて、実際には並行して実行することも、逆の順序で実行することもできる。ブロック図および／またはフローチャートの各ブロック、およびブロック図および／またはフローチャートのブロックの組み合わせは、指定された機能または動作を実行する専用のハードウェアベースのシステム又は専用のハードウェアとコンピュータの命令との組み合わせによって実現できる。

10

【0078】

本開示の実施形態は上記に記載されており、上記の説明は例示的であり、網羅的ではなく、開示された実施形態に限定されない。説明された各実施形態の範囲および精神から逸脱することない場合、多くの修正および変更が当業者には明らかである。本明細書で使用する用語の選択は、各実施形態の原理、実際の応用、または市場における技術の改善を最もよく説明すること、または本技術分野の他の当業者が本明細書に開示される様々な実施形態を理解できるようにすることを意図している。

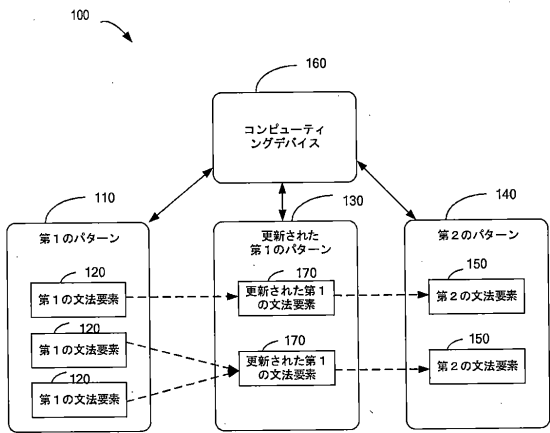
20

30

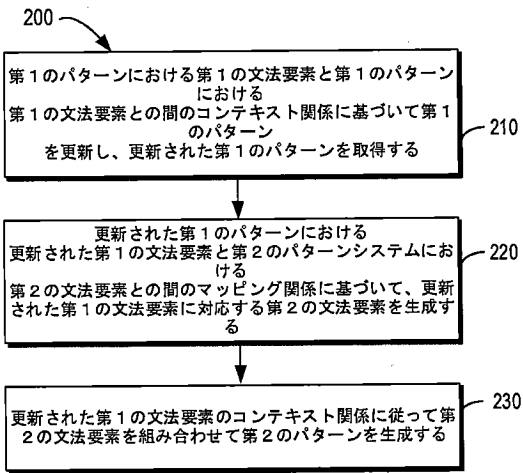
40

50

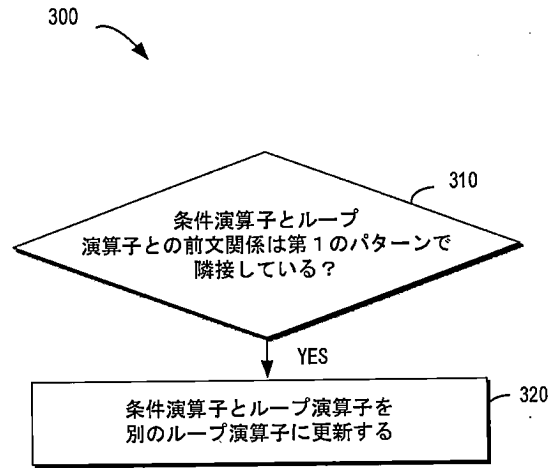
【図面】
【図 1】



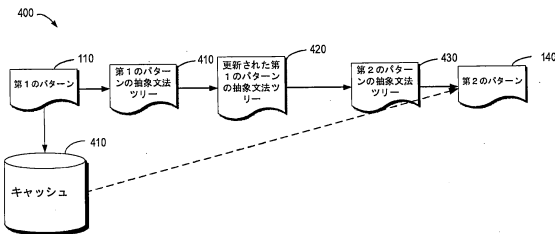
【図 2】



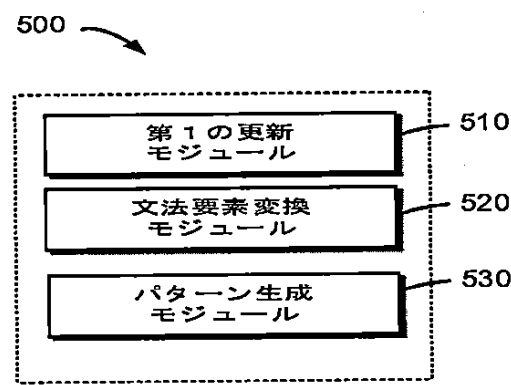
【図 3】



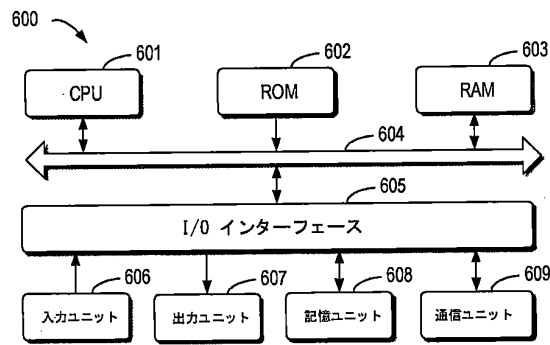
【図 4】



【図 5】



【図 6】



10

20

30

40

50

フロントページの続き

- 85, China
- (74)代理人 110001195
弁理士法人深見特許事務所
- (72)発明者 張 留 傑
中華人民共和国100085北京市海淀区上地十街10号百度大厦2層
- (72)発明者 李 雅 美
中華人民共和国100085北京市海淀区上地十街10号百度大厦2層
- (72)発明者 鄭 輝 煌
中華人民共和国100085北京市海淀区上地十街10号百度大厦2層
- (72)発明者 劉 紅 雨
中華人民共和国100085北京市海淀区上地十街10号百度大厦2層
- (72)発明者 藍 翔
中華人民共和国100085北京市海淀区上地十街10号百度大厦2層
- (72)発明者 于 佃 海
中華人民共和国100085北京市海淀区上地十街10号百度大厦2層
- (72)発明者 馬 艷 軍
中華人民共和国100085北京市海淀区上地十街10号百度大厦2層
- (72)発明者 吳 甜
中華人民共和国100085北京市海淀区上地十街10号百度大厦2層
- (72)発明者 王 海 峰
中華人民共和国100085北京市海淀区上地十街10号百度大厦2層
- 審査官 青木 重徳
- (56)参考文献 特開2017-033446 (JP, A)
特開2008-009463 (JP, A)
特開2007-158687 (JP, A)
米国特許出願公開第2016/0117364 (US, A1)
LOOKS, Moshe, DEEP LEARNING WITH DYNAMIC COMPUTATION GRAPHS, arxiv:1702.02181v2, 2017年02月22日, [検索日:2022年08月19日], インターネット <<https://arxiv.org/pdf/1702.02181.pdf>>
得居 誠也, ニューラルネットワーク研究のフロンティア, 人工知能 第31巻 第2号, 日本, 人工知能学会 JSAI, 2016年03月01日, 第31巻, pp.216-222
大津 真, これから学ぶPython, 第1版, 株式会社インプレス, 2020年08月20日, p.150-152
菱沼佑香, 第11回条件分岐はif文でwith elif・break・pass・continue(菱沼佑香), [online], 2020年08月31日, [検索日:2022年08月19日], インターネット <URL: <https://www.school.ctc-g.co.jp/columns/hishinuma/hishinuma11.html>>
note.nkmk.me, Pythonのwhile文によるループ処理(無限ループなど), [online], 2019年02月02日, [検索日:2022年08月19日], インターネット <URL: <https://note.nkmk.me/python-while-usage/>>
松岡 恭正, 原理から学ぶ言語処理プログラミング, インターフェース 第15巻 第3号 Interface, 日本, CQ出版株式会社, 1989年03月01日, 第15巻, pp.196-210
- (58)調査した分野 (Int.Cl., DB名)
G06N 3/08
G06N 3/04
JSTPlus/JMEDPlus/JST7580 (JDreamIII)
IEEE Explore