(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第4901097号
(P4901097)
(45)発行日　平成24年3月21日(2012.3.21)　(24)登録日　平成24年1月13日(2012.1.13)

(51)Int.Cl.　F I
G06N 3/00 (2006.01)　G06N 3/00 550G
G06N 3/00 560E

請求項の数 29　(全 31 頁)

(21)出願番号　特願2004-357352 (P2004-357352)
(22)出願日　平成16年12月9日 (2004.12.9)
(65)公開番号　特開2005-182785 (P2005-182785A)
(43)公開日　平成17年7月7日 (2005.7.7)
審査請求日　平成19年12月7日 (2007.12.7)
(31)優先権主張番号　60/528,578
(32)優先日　平成15年12月9日 (2003.12.9)
(33)優先権主張国　米国(US)
(31)優先権主張番号　10/837,382
(32)優先日　平成16年4月30日 (2004.4.30)
(33)優先権主張国　米国(US)

(73)特許権者　500046438
マイクロソフト　コーポレーション
アメリカ合衆国　ワシントン州　９８０５２－６３９９　レッドモンド　ワン　マイクロソフト　ウェイ
(74)代理人　100077481
弁理士　谷　義一
(74)代理人　100088915
弁理士　阿部　和夫
(72)発明者　デビッド　ダブリュ．スタインクラウス
アメリカ合衆国　９８０５２　ワシントン州　レッドモンド　ワン　マイクロソフト　ウェイ　マイクロソフト　コーポレーション内

最終頁に続く

(54)【発明の名称】グラフィックス処理ユニットを使用して機械学習技術の処理を速め、最適化するシステムおよび方法

(57)【特許請求の範囲】
【請求項１】
コンピュータアプリケーションを処理するコンピュータ実施方法であって、
グラフィックス処理ユニットが、機械学習技術のトレーニング可能パラメータ、トレーニングデータ、及びターゲットデータを入力するステップと、
前記グラフィックス処理ユニットが、該入力したトレーニング可能パラメータ及び該入力したトレーニングデータに基づいた関数を計算することにより、第１の結果を取得するステップと、
前記第１の結果と前記ターゲットデータとの誤差が小さくなるように、前記トレーニング可能パラメータの値を更新するステップと、
前記グラフィックス処理ユニットが、クエリデータを入力するステップと、
前記グラフィックス処理ユニットが、該入力したクエリデータ及び該更新されたトレーニング可能パラメータに基づいた関数を計算することにより、第２の結果を取得するステップと、
前記第２の結果を使用して前記コンピュータアプリケーションによって使用できるように解を提供するステップとを備え、
前記計算するステップ及び前記取得するステップは、行列と列ベクトルとの内積を計算するステップをさらに含み、
前記内積を計算するステップは、
前記行列のうちの１２列の行列と前記列ベクトルのうちの１２列の列ベクトルとの内積

を計算するためのピクセルシェーダを使用することにより、複数の列ベクトル部分積を計算するステップと、
該計算された複数の列ベクトル部分積を合計することにより、単一の列ベクトルを取得するステップと
をさらに含むことを特徴とする方法。
【請求項２】
前記コンピュータアプリケーションは（ａ）音声認識アプリケーション、（ｂ）手書き文字認識アプリケーションのうちの一方の対話型使用コンピュータアプリケーションであることを特徴とする請求項１に記載のコンピュータ実施方法。
【請求項３】
前記機械学習技術はニューラルネットワークであることを特徴とする請求項１に記載のコンピュータ実施方法。
【請求項４】
前記グラフィックス処理ユニットに前記更新されたトレーニング可能パラメータの少なくとも一部を格納するステップをさらに含むことを特徴とする請求項１に記載のコンピュータ実施方法。
【請求項５】
前記計算するステップ及び前記取得するステップは、前記ピクセルシェーダを使用して外積を計算するステップをさらに含むことを特徴とする請求項１に記載のコンピュータ実施方法。
【請求項６】
テクスチャマッピングを使用して前記外積を計算するステップをさらに含むことを特徴とする請求項５に記載のコンピュータ実施方法。
【請求項７】
前記計算するステップ及び前記取得するステップは、前記ピクセルシェーダを使用して行列の転置を行うステップをさらに含むことを特徴とする請求項１に記載のコンピュータ実施方法。
【請求項８】
テクスチャマッピングを使用して前記行列の転置を行うステップをさらに含むことを特徴とする請求項７に記載のコンピュータ実施方法。
【請求項９】
請求項１に記載の前記コンピュータ実施方法を実行するためのコンピュータ実行可能命令を格納したことを特徴とするコンピュータ可読記録媒体。
【請求項１０】
グラフィックス処理ユニットを使用して機械学習技術を処理するプロセスであって、
グラフィックス処理ユニットが、前記機械学習技術のトレーニング可能パラメータ、トレーニングデータ、及びターゲットデータを入力するステップと、
前記グラフィックス処理ユニットが、該入力したトレーニング可能パラメータ及び該入力したトレーニングデータに基づいた関数を計算することにより、第１の結果を取得するステップと、
前記第１の結果と前記ターゲットデータとの誤差が小さくなるように、前記トレーニング可能パラメータの値を更新するステップと、
前記グラフィックス処理ユニットが、クエリデータを入力するステップと、
前記グラフィックス処理ユニットが、該入力したクエリデータ及び該更新されたトレーニング可能パラメータに基づいた関数を計算することにより、第２の結果を取得するステップと、
アプリケーションによって使用できるように前記第２の結果を出力するステップとを含み、
前記計算するステップ及び前記取得するステップは、行列と列ベクトルとの内積を計算するステップをさらに含み、

前記内積を計算するステップは、
前記行列のうちの１２列の行列と前記列ベクトルのうちの１２列の列ベクトルとの内積を計算するためのピクセルシェーダを使用することにより、複数の列ベクトル部分積を計算するステップと、
該計算された複数の列ベクトル部分積を合計することにより、単一の列ベクトルを取得するステップと
をさらに含むことを特徴とするプロセス。
【請求項１１】
前記ピクセルシェーダはグラフィックカードに存在することを特徴とする請求項１０に記載のプロセス。
【請求項１２】
前記ピクセルシェーダを使用して、外積、行列の転置のうちの少なくとも１つを計算するステップをさらに含むことを特徴とする請求項１０に記載のプロセス。
【請求項１３】
前記機械学習技術はニューラルネットワークであることを特徴とする請求項１０に記載のプロセス。
【請求項１４】
前記機械学習技術は期待値最大化（ＥＭ）技術を使用することを特徴とする請求項１０に記載のプロセス。
【請求項１５】
前記機械学習技術は（ａ）Ｋ－ｍｅａｎｓ技術、（ｂ）ＬＶＱ（Ｌｅａｒｎｉｎｇ　Ｖｅｃｔｏｒ　Ｑｕａｎｔｉｚａｔｉｏｎ）技術のうちの少なくとも一方であることを特徴とする請求項１０に記載のプロセス。
【請求項１６】
前記グラフィックス処理ユニットを使用して前記トレーニングデータを前処理するステップをさらに含むことを特徴とする請求項１０に記載のプロセス。
【請求項１７】
前記前処理するステップは、（ａ）前記トレーニングデータを標準化してそれを入力のためにより適した形式にするステップと、（ｂ）前記トレーニングデータから情報および特徴を抽出するステップと、（ｃ）入力データを取得して前記入力データを前記グラフィックス処理ユニットにロードするステップとのうちの少なくとも１つを含むことを特徴とする請求項１６に記載のプロセス。
【請求項１８】
１つまたは複数のプロセッサによって実行されると、前記１つまたは複数のプロセッサに請求項１０の前記プロセスを実施させるためのコンピュータ可読命令を格納したことを特徴とする１つまたは複数のコンピュータ可読記録媒体。
【請求項１９】
グラフィックス処理ユニットを使用して機械学習技術を処理するためのコンピュータ実行可能命令を格納したコンピュータ可読記録媒体であって、
グラフィックス処理ユニットが、前記機械学習技術のトレーニング可能パラメータ及びピクセルシェーダを入力するステップと、
前記グラフィックス処理ユニットが、前記機械学習技術のトレーニングデータ及びターゲットデータを入力するステップと、
前記グラフィックス処理ユニットが、該入力したトレーニング可能パラメータ及び該入力したトレーニングデータに基づいた関数を計算することにより、第１の結果を取得するステップと、
前記第１の結果と前記ターゲットデータとの誤差が小さくなるように、前記トレーニング可能パラメータの値を更新するステップと、
前記グラフィックス処理ユニットが、クエリデータを入力するステップと、
前記グラフィックス処理ユニットが、該入力したクエリデータ及び該更新されたトレー

ニング可能パラメータに基づいた関数を計算することにより、対話型使用アプリケーションによって使用するように第２の結果を取得するステップとを含み、
前記計算するステップ及び前記取得するステップは、行列と列ベクトルとの内積を計算するステップをさらに含み、
前記内積を計算するステップは、
前記行列のうちの１２列の行列と前記列ベクトルのうちの１２列の列ベクトルとの内積を計算するための前記ピクセルシェーダを使用することにより、複数の列ベクトル部分積を計算するステップと、
該計算された複数の列ベクトル部分積を合計することにより、単一の列ベクトルを取得するステップと
をさらに含むことを特徴とするコンピュータ可読記録媒体。
【請求項２０】
少なくとも一部のトレーニング可能パラメータを前記グラフィックス処理ユニットに配置するステップをさらに含むことを特徴とする請求項１９に記載のコンピュータ可読記録媒体。
【請求項２１】
前記クエリデータを前処理して入力データを取得するステップと、
前記グラフィックス処理ユニットに前記入力データをロードするステップと
をさらに含むことを特徴とする請求項１９に記載のコンピュータ可読記録媒体。
【請求項２２】
前記ピクセルシェーダを使用して、外積、行列の転置のうちの少なくとも１つを計算するステップをさらに含むことを特徴とする請求項１９に記載のコンピュータ可読記録媒体。
【請求項２３】
前記機械学習技術はニューラルネットワークであることを特徴とする請求項１９に記載のコンピュータ可読記録媒体。
【請求項２４】
前記機械学習技術は勾配降下技術であることを特徴とする請求項１９に記載のコンピュータ可読記録媒体。
【請求項２５】
前記勾配降下技術の調整可能パラメータの少なくとも一部を前記グラフィックス処理ユニットに配置するステップをさらに含むことを特徴とする請求項２４に記載のコンピュータ可読記録媒体。
【請求項２６】
前記対話型使用アプリケーションは手書き文字認識アプリケーションであることを特徴とする請求項１９に記載のコンピュータ可読記録媒体。
【請求項２７】
前記機械学習技術はニューラルネットワークであり、前記ニューラルネットワークを使用して分類を行うステップをさらに含むことを特徴とする請求項２６に記載のコンピュータ可読記録媒体。
【請求項２８】
前記対話型使用アプリケーションは光学文字認識アプリケーションであることを特徴とする請求項１９に記載のコンピュータ可読記録媒体。
【請求項２９】
前記機械学習技術はニューラルネットワークであり、前記ニューラルネットワークを使用して分類を行うステップをさらに含むことを特徴とする請求項２８に記載のコンピュータ可読記録媒体。
【発明の詳細な説明】
【技術分野】
【０００１】

本発明は、一般にコンピュータ処理技術に関し、より詳細にはグラフィックス処理ユニット（ＧＰＵ）を使用して機械学習技術（ニューラルネットワークなど）および他の非グラフィックスのアプリケーションを処理し、その処理を速め、最適化するシステムおよび方法に関する。

【背景技術】

【０００２】

グラフィックス処理ユニット（ＧＰＵ）は、現代のパーソナルコンピュータ（ＰＣ）の不可欠な一部分である。ＧＰＵは、リアルタイムの３次元（３Ｄ）グラフィックスのユーザへの表示を速めるように設計された単一チッププロセッサである。当初ハイエンドのグラフィックスワークステーションの特徴であったＧＰＵは、従来の中央処理装置（ＣＰＵ）では適していなかった、または単に遅すぎたグラフィック機能のアクセラレータとしてパーソナルコンピュータバスに進出した。

【０００３】

コンピュータグラフィックスは、一般に変更されたオシロスコープであるカリグラフィック表示装置上の線画として始まった。こうした表示の計算は、一般の座標変換、表示装置の境界へのクリッピング、および３Ｄ表示の透視変換を含むベクトル演算を必要とした。安価な商品である半導体メモリの出現がきっかけとなって、線画システムの代わりに、フレームバッファメモリを介してテレビのような表示を再生するラスタグラフィックスプロセッサが使用されるようになった。ユーザは一般に、ほとんどのアプリケーションの場合の線画ではなく、陰影付きのベタ塗りの表面を見ることを好むため、ラスタグラフィックスは、線画を迅速に表示した。線画で使用されていたような直線セグメントではなく、ラスタグラフィックスシステムの形状の構成単位（または基本要素）は、複数の三角形の配列から構成される多面体表面であった。表示の基本要素は、フレームバッファメモリに格納されたピクセルの矩形配列であった。配列の行は、ラスタ陰極線管（ＣＲＴ）ディスプレイ上の個別の走査線に対応する。

【０００４】

グラフィックスが線画からラスタグラフィックスに発展していくにつれて、より大きい処理能力が必要となった結果、数学コプロセッサがＰＣに含まれるようになった。数学コプロセッサとは、ホストＣＰＵ命令ストリームを共有し、ＣＰＵメモリにアクセスできる一体型の浮動小数点コプロセッサである。他のタイプの一体型コプロセッサは、パラレルデータパスを有し、非同期実行を行い、またＣＰＵメモリにアクセスすることができるＭｕｌｔｉｍｅｄｉａ Ｅｘｔｅｎｓｉｏｎ（ＭＭＸ）、Ｓｔｒｅａｍｉｎｇ ＳＩＭＤ Ｅｘｔｅｎｓｉｏｎ（ＳＳＥ）などのＣＰＵの拡張機能である。

【０００５】

３Ｄグラフィックスでの品質の高さおよび現実感の要求が絶えず高まっていった結果、より大きなグラフィックス処理能力が必要となった。この必要性を満たすために、計算集約的なグラフィックタスクを行うＧＰＵが導入された。これによってＣＰＵを楽にし、開放して、ＣＰＵが別の処理タスクを行えるようにした。現在の具体化では、ＧＰＵは、メインＣＰＵメモリへのそれ自体の専用パス、およびそれ自体の専用のグラフィックスメモリを備えるＰＣの重要な構成要素である。数学コプロセッサとは対照的に、ＧＰＵは、それ自体の命令ストリーム、データパス、および専用メモリを備える自律した専用プロセッサである。

【０００６】

ＧＰＵの設計および構成の現在の傾向によって、専用メモリがより大きくなり、グラフィックスメモリに対する帯域幅がより高くなり、また内部並列が向上している。さらに現在のＧＰＵは、絶えず高まるプログラム可能性の度合いを考慮して設計されている。プログラム可能性の導入によって、ＧＰＵは、非グラフィックスのアプリケーションでの使い道を見つけるほど十分な柔軟性を得た。さらに、ＧＰＵのデータ並列アーキテクチャは、計算集約型アプリケーションでは、ＣＰＵに比べて劇的なパフォーマンスの向上をもたらす。代替のグラフィックスアルゴリズムへの拡張および科学的なコンピューティング問題

がいくつかの実例で探求されている。

【０００７】

しかし、対話型使用（音声認識や手書き文字認識など）を対象としたアプリケーションには、相対的にあまり関心が集まっていない。この１つの理由は、ＧＰＵによって処理するためのこうしたアルゴリズムの実装が困難であり、いくつかの制限があったからである。汎用コンピューティングでは、ＧＰＵは本質的に制限を有するストリームプロセッサである。こうした制限を扱い、回避するには、明白でも直観的でもないプログラミングおよび処理のスタイルが必要である。

【０００８】

こうした対話型使用アプリケーション（ｉｎｔｅｒａｃｔｉｖｅ　ｕｓｅ　ａｐｐｌｉｃａｔｉｏｎ）は、一般に自明ではない解を有し、大量のデータを扱う。こうした状況では、機械学習技術が好ましいソリューション技術である。機械学習技術は、トレーニング後、入力が正しく分類されるように、アルゴリズムのパラメータを自動的に調整することによって動作する。例えば、タスクが「Ａ」のピクセル画像に正しいＡＳＣＩＩラベルを割り当てることであると仮定する。残念ながら、トレーニングは一般に、それら自体何十万もの演算を有するアルゴリズムに何十万もの対（入力、ターゲット）を提示することを伴う。その結果、最も早い使用可能なマシンでさえ、トレーニングには多大な時間がかかる可能性がある。また、実社会の様々な状態でアルゴリズムをテストし、または使用することは、法外に費用がかかる可能性がある。

【発明の開示】

【発明が解決しようとする課題】

【０００９】

したがって、ＧＰＵを使用して機械学習技術を処理するシステムおよび方法が必要である。さらに、ＧＰＵの制限を回避することによって機械学習技術の処理を速め、最適化するシステムおよび方法が必要である。このことによって、機械学習技術を使用した対話型使用アプリケーション（音声認識、手書き文字認識など）の解決にＧＰＵの十分な処理能力が確実に利用されるようになる。

【課題を解決するための手段】

【００１０】

本明細書で開示した本発明は、グラフィックス処理ユニット（ＧＰＵ）を使用して機械学習技術を処理するシステムおよび方法に関する。本発明は、ＣＰＵ処理の一部をＧＰＵに移植することによって中央処理装置（ＣＰＵ）の計算の制限を軽減する。より具体的には、本発明は、従来ＣＰＵによって処理されていた機械学習アーキテクチャをＧＰＵに移植する。機械学習技術は、ＧＰＵへの移植に特に適している。というのは、ＧＰＵは一般にＣＰＵより強力であり、機械学習技術は、自明ではない解を有し、音声認識、手書き文字認識など、データアクセスよりかなり多くの計算を必要とする問題の解決を伴うからである。これは、計算に比べてデータアクセスが多いメモリベースの分類または検索とは異なる。ＧＰＵへのデータ転送にかなりの時間がかかる場合、ＧＰＵで計算を行うメリットは低下する。言い換えれば、計算に必要とされるのと同じ量のデータがＧＰＵに転送される場合、データの転送がネックとなり、どんなネットの改良も無駄になる。

【００１１】

本発明の方法は、ＧＰＵでのその後のテスト無しにＧＰＵを使用して機械学習技術をトレーニングするステップと、ＧＰＵでの事前のトレーニング無しにＧＰＵを使用して機械学習技術をテストするステップと、ＧＰＵでトレーニングおよびテストを行うステップとを含む。特に、本明細書で開示した本発明は、対話型使用のコンピュータアプリケーションを処理するコンピュータ実施方法を含む。この方法は、グラフィックス処理ユニットを使用して機械学習技術を処理して関数の解を得るステップと、対話型使用のコンピュータアプリケーションによって使用できるようにその解を出力するステップとを含む。この方法は、グラフィックス処理ユニットを使用して機械学習技術のトレーニング用学習可能パラメータを得てトレーニング済み学習可能パラメータを得るステップと、トレーニング済

み学習可能パラメータを使用して解を得るステップとをさらに含む。対話型使用のコンピュータアプリケーションは、音声認識アプリケーションおよび手書き文字認識アプリケーションのうちの１つとすることができる。さらに機械学習技術は、ニューラルネットワークとすることができる。

【００１２】

この方法は、学習可能パラメータを中央処理装置に格納するステップと、学習可能パラメータの少なくとも一部をグラフィックス処理ユニットに格納するステップとをさらに含むことができる。関数の解は、ピクセルシェーダを使用して、ベクトル内積および行列内積のうちの少なくとも一方である内積を計算することによって取得することができる。この方法は、内積をサブ問題に分解するステップと、ピクセルシェーダを使用してサブ問題にわたって複数回のパスを実行するステップとをさらに含むことができる。関数の解の取得は、ピクセルシェーダを使用して外積を計算することによって達成することができる。この方法は、テクスチャマッピングを使用して外積を計算するステップを含むこともできる。関数の解は、ピクセルシェーダを使用して行列の転置を行うことによって取得することができる。この方法は、テクスチャマッピングを使用して行列の転置を行うステップを含むこともできる。

【００１３】

本明細書で開示した本発明は、グラフィックス処理ユニットを使用して機械学習技術を速め、最適化するプロセスも含む。この方法は、グラフィックス処理ユニット上でピクセルシェーダ（ｐｉｘｅｌ　ｓｈａｄｅｒ）を使用して機械学習技術の学習可能パラメータをトレーニングするステップと、グラフィックス処理ユニット上でピクセルシェーダを使用し、またトレーニング済み学習可能パラメータを使用して機械学習技術から結果を取得するステップと、アプリケーションによって使用できるようにその結果を出力するステップとを含む。

【００１４】

このプロセスは、グラフィックス処理ユニット上で学習可能パラメータのうちの少なくとも一部を探し出すステップと、ピクセルシェーダを使用して（ａ）ベクトル内積、（ｂ）行列内積、（ｃ）外積、（ｄ）行列の転置のうちの少なくとも１つを計算するステップとを含む。機械学習技術はニューラルネットワークとすることができ、ニューラルネットワークは、（ａ）マルチプレーヤの完全接続ニューラルネットワーク、（ｂ）重畳ニューラルネットワーク（ｃｏｎｖｏｌｕｔｉｏｎａｌ　ｎｅｕｒａｌ　ｎｅｔｗｏｒｋ）のうちの少なくとも一方とすることができる。機械学習技術は、期待値最大化（ＥＭ）アルゴリズムおよびＫ－ｍｅａｎｓ技術およびＬＶＱ（Ｌｅａｒｎｉｎｇ　Ｖｅｃｔｏｒ　Ｑｕａｎｔｉｚａｔｉｏｎ）技術を使用することもできる。このプロセスは、グラフィックス処理ユニットを使用してデータを前処理するステップも含む。

【００１５】

本明細書で開示した本発明は、グラフィックス処理ユニットを使用して機械学習技術を処理するコンピュータ実行可能命令を有するコンピュータ可読媒体も含む。媒体は、シェーダをグラフィックス処理ユニットにロードするステップと、クエリデータを中央処理装置にロードするステップと、シェーダを呼び出して、機械学習技術を使用してグラフィックス処理ユニット上でトレーニング済み関数を処理するステップと、対話型使用アプリケーションによって使用できるように結果を取得するステップとを含む。

【００１６】

コンピュータ可読媒体は、少なくとも一部の学習可能パラメータをグラフィックス処理ユニットに配置するステップと、入力データを取得するためにクエリデータを前処理し、入力データをグラフィックス処理ユニットにロードするステップとをさらに含む。さらに媒体は、シェーダを使用してグラフィックス処理ユニット上で基本的な演算（ｐｒｉｍｉｔｉｖｅ　ｏｐｅｒａｔｉｏｎ）を計算するステップを含む。基本的な演算は、（ａ）ベクトル内積、（ｂ）行列内積、（ｃ）外積、（ｄ）行列の転置のうちの少なくとも１つを含む。

【００１７】
機械学習技術は、ニューラルネットワーク、および勾配降下技術（ｇｒａｄｉｅｎｔ ｄｅｓｃｅｎｔ ｔｅｃｈｎｉｑｕｅ）とすることができる。コンピュータ可読媒体は、勾配降下技術の調整可能パラメータのうちの少なくとも一部をグラフィックス処理ユニットに配置するステップをさらに含む。対話型使用アプリケーションは、手書き文字認識アプリケーションとすることができる。さらに、機械学習技術は、ニューラルネットワークとすることができ、ニューラルネットワークを使用して分類を行うステップをさらに含む。また、対話型使用アプリケーションは、光学文字認識アプリケーションとすることもでき、機械学習技術は、ニューラルネットワークとし、ニューラルネットワークを使用して分類を行うステップをさらに含む。機械学習技術は、勾配降下技術とすることができ、グラフィックス処理ユニットに勾配降下パラメータを格納するステップと、勾配降下パラメータを２倍にすることによって勾配降下パラメータの仮数を拡張するステップとを含むことができる。
【発明を実施するための最良の形態】
【００１８】
本発明は、以下の説明および本発明の態様を示した添付図面を参照することによってさらに理解することができる。他の特徴および利点は、本発明の以下の詳細な説明を添付図面と併せ読めば明らかになる。添付図面は、一例として本発明の原理を示している。
【００１９】
図面を参照すると、図中、同様の参照番号は、図面を通じて対応する部分を表している。
【００２０】
本発明の以下の説明では、その一部を構成し、本発明を実施できる特定の例を実例として示した添付図面を参照する。他の実施形態を使用してもよく、また本発明の範囲から逸脱することなく構造的な変更を加えてもよいことを理解されたい。
【００２１】
Ｉ．導入
グラフィックス処理ユニット（ＧＰＵ）は、従来、リアルタイム３Ｄグラフィックス表示を速めるために使用されてきた。しかし、処理能力およびプログラム可能性の向上の結果、ＧＰＵは、他の非グラフィックス関連のプロセスを効率的に処理することもできる。こうしたプロセスは一般に、大量のデータおよび処理（計算流体力学など）を伴う代替のグラフィックスアルゴリズムおよび特定のコンピュータ問題に限定されている。しかし、音声認識および手書き文字認識などの対話型使用アプリケーションには、ＧＰＵアーキテクチャの制限および特徴のために、相対的にあまり関心が集まっていない。
【００２２】
本明細書に記載した機械学習ＧＰＵ実行システムおよび方法は、ＣＰＵ処理の一部またはすべてをＧＰＵに移植することによって、ＣＰＵの計算の制限を軽減する。より具体的には、このシステムおよび方法は、様々な機械学習技術に使用できるアーキテクチャをＣＰＵからＧＰＵに転送する。ＧＰＵへの処理の転送は、制限を克服し、ＧＰＵアーキテクチャのフレームワーク内でよく働くいくつかの新しい技術を使用して達成される。こうした制限が克服された状態では、機械学習技術は、ＧＰＵでの処理に特に適している。というのは、一般にＧＰＵは、一般のＣＰＵよりかなり強力であるからである。さらに、グラフィックス処理と同様に、機械学習技術の処理は、自明ではない解および大量のデータの解決に関する問題を伴う。
【００２３】
ＩＩ．動作環境の例
本明細書で開示した機械学習ＧＰＵ実行エンジンおよび方法は、コンピューティング環境で動作するように設計されている。次の説明は、機械学習ＧＰＵ実行エンジンおよび方法を実施できる適したコンピューティング環境の簡単な概要を提供するためのものである。

【００２４】
図１は、機械学習ＧＰＵ実行エンジンおよび方法を実施できる適したコンピューティングシステム環境の例を示している。コンピューティングシステム環境１００は、適したコンピューティング環境の一例にすぎず、本発明の使用または機能の範囲に関する限定を示唆するものではない。また、コンピューティング環境１００を、動作環境１００の例に示した構成要素のいずれか１つ、またはその組合せに関連する依存性または必要条件を有しているものと解釈すべきではない。
【００２５】
機械学習ＧＰＵ実行エンジンおよび方法は、他の多くの汎用または専用コンピューティングシステム環境または構成で動作可能である。機械学習ＧＰＵ実行エンジンおよび方法との使用に適したよく知られているコンピューティングシステム、環境、および／または構成の例には、それだけには限定されないが、パーソナルコンピュータ、サーバコンピュータ、例えばセルラー式電話やＰＤＡなどのハンドヘルド、ラップトップ、またはモバイルコンピュータまたは通信装置、マルチプロセッサシステム、マイクロプロセッサベースのシステム、セットトップボックス、プログラム可能家庭用電化製品、ネットワークＰＣ、ミニコンピュータ、メインフレームコンピュータ、上記の任意のシステムまたは装置を含む分散コンピューティング環境などがある。
【００２６】
機械学習ＧＰＵ実行エンジンおよび方法は、コンピュータによって実行されるプログラムモジュールなどのコンピュータ実行可能命令の一般的な文脈で説明することができる。一般にプログラムモジュールは、特定のタスクを実行する、または特定の抽象データ型を実装するルーチン、プログラム、オブジェクト、構成要素、データ構造などを含む。また、機械学習ＧＰＵ実行エンジンおよび方法は、通信ネットワークによってリンクされているリモート処理装置によってタスクが実行される分散コンピューティング環境でも実施することができる。分散コンピューティング環境では、プログラムモジュールを、メモリ記憶装置を含むローカルおよびリモートのコンピュータ記憶媒体に配置することができる。図１を参照すると、機械学習ＧＰＵ実行エンジンおよび方法を実施するシステムの例は、汎用コンピューティング装置をコンピュータ１１０の形で含んでいる。
【００２７】
コンピュータ１１０の構成要素は、それだけには限定されないが、処理ユニット１２０（中央処理装置、すなわちＣＰＵなど）、システムメモリ１３０、およびシステムメモリを含む様々なシステム構成要素を処理ユニット１２０に結合するシステムバス１２１を含み得る。システムバス１２１は、様々なバスアーキテクチャのうちの任意のものを使用するメモリバスまたはメモリコントローラ、周辺バス、およびローカルバスを含むいくつかのタイプのバス構造のうちどんなものでもよい。こうしたアーキテクチャには、それだけには限定されないが一例として、ＩＳＡ（Ｉｎｄｕｓｔｒｙ Ｓｔａｎｄａｒｄ Ａｒｃｈｔｅｃｔｕｒｅ）バス、ＭＣＡ（Ｍｉｃｒｏ Ｃｈａｎｎｅｌ Ａｒｃｈｉｔｅｃｔｕｒｅ）バス、ＥＩＳＡ（Ｉｎｈａｎｃｅｄ ＩＳＡ）バス、ＶＥＳＡ（Ｖｉｄｅｏ Ｅｌｅｃｔｒｏｎｉｃｓ Ｓｔａｎｄａｒｄｓ Ａｓｓｏｃｉａｔｉｏｎ）ローカルバス、およびメザニンバスとしても知られているＰＣＩ（Ｐｅｒｉｐｈｅｒａｌ Ｃｏｍｐｏｎｅｎｔ Ｉｎｔｅｒｃｏｎｎｅｃｔ）バスなどがある。
【００２８】
コンピュータ１１０は、一般に様々なコンピュータ可読媒体を含む。コンピュータ可読媒体は、コンピュータ１１０からアクセスできる使用可能な任意の媒体とすることができ、揮発性および不揮発性媒体、リムーバブルおよび非リムーバブル媒体を含む。コンピュータ可読媒体は、それだけには限定されないが一例として、コンピュータ記憶媒体および通信媒体を含み得る。コンピュータ記憶媒体には、コンピュータ可読命令、データ構造、プログラムモジュール、他のデータなど、情報を記憶するための任意の方法または技術で実施される揮発性および不揮発性のリムーバブルおよび非リムーバブル媒体がある。
【００２９】

コンピュータ記憶媒体には、それだけには限定されないが、ＲＡＭ、ＲＯＭ、ＥＥＰＲＯＭ、フラッシュメモリまたは他のメモリ技術、ＣＤ－ＲＯＭ、デジタル多用途ディスク（ＤＶＤ）または他の光ディスク記憶装置、磁気カセット、磁気テープ、磁気ディスク記憶装置または他の磁気記憶装置、または所望の情報の格納に使用でき、コンピュータ１１０からアクセスできる他の任意の媒体などがある。通信媒体は一般に、コンピュータ可読命令、データ構造、プログラムモジュール、または他のデータを搬送波または他の移送機構などの変調されたデータ信号に組み込む。これには任意の情報配送媒体がある。

【００３０】

「変調されたデータ信号」という用語は、信号に情報を符号化するように１つまたは複数のその特性が設定または変更された信号を意味することに留意されたい。通信媒体には、それだけには限定されないが一例として、有線ネットワーク、直接配線された接続などの有線媒体、および音響、ＲＦ、赤外線、その他の無線媒体などの無線媒体がある。また、上記のどんな組合せでもコンピュータ可読媒体の範囲内に含まれるものとする。

【００３１】

システムメモリ１３０は、読取り専用メモリ（ＲＯＭ）１３１やランダムアクセスメモリ（ＲＡＭ）１３２など、揮発性および／または不揮発性メモリの形のコンピュータ記憶媒体を含む。基本入出力システム１３３（ＢＩＯＳ）は、例えば起動中など、コンピュータ１１０内の要素間での情報の転送を助ける基本ルーチンを含み、一般にＲＯＭ１３１に格納されている。ＲＡＭ１３２は一般に、処理ユニット１２０から直接アクセス可能な、かつ／または処理ユニット１２０が現在処理しているデータおよび／またはプログラムモジュールを含む。図１は、それだけには限定されないが一例として、オペレーティングシステム１３４、アプリケーションプログラム１３５、他のプログラムモジュール１３６、およびプログラムデータ１３７を示している。

【００３２】

コンピュータ１１０は、他のリムーバブル／非リムーバブル、揮発性／不揮発性コンピュータ記憶媒体を含むこともできる。一例にすぎないが、図１は、非リムーバブル不揮発性磁気媒体から読み取り、あるいはそこに書き込むハードディスクドライブ１４１、リムーバブル不揮発性磁気ディスク１５２から読み取り、あるいはそこに書き込む磁気ディスクドライブ１５１、およびＣＤ－ＲＯＭや他の光媒体など、リムーバブル不揮発性光ディスク１５６から読み取り、あるいはそこに書き込む光ディスクドライブ１５５を示している。

【００３３】

動作環境の例で使用できる他のリムーバブル／非リムーバブル、揮発性／不揮発性コンピュータ記憶媒体には、それだけには限定されないが、磁気テープカセット、フラッシュメモリカード、デジタル多用途ディスク、デジタルビデオテープ、半導体ＲＡＭ、半導体ＲＯＭなどがある。ハードディスクドライブ１４１は一般に、インターフェース１４０などの非リムーバブルメモリインターフェースを介してシステムバス１２１に接続され、磁気ディスクドライブ１５１および光ディスクドライブ１５５は一般に、インターフェース１５０などのリムーバブルメモリインターフェースによってシステムバス１２１に接続される。

【００３４】

上述し、図１に示したドライブおよびその関連のコンピュータ記憶媒体は、コンピュータ可読命令、データ構造、プログラムモジュール、およびコンピュータ１１０の他のデータの記憶域を提供する。図１では例えば、ハードディスクドライブ１４１は、オペレーティングシステム１４４、アプリケーションプログラム１４５、他のプログラムモジュール１４６、およびプログラムデータ１４７を格納するものとして示されている。これらの構成要素は、オペレーティングシステム１３４、アプリケーションプログラム１３５、他のプログラムモジュール１３６、およびプログラムデータ１３７と同じであっても、異なっていてもよいことに留意されたい。オペレーティングシステム１４４、アプリケーションプログラム１４５、他のプログラムモジュール１４６、およびプログラムデータ１４７は

少なくとも異なるコピーであることを示すために、ここではそれらに異なる番号を付している。ユーザは、キーボード１６２、および一般にマウス、トラックボール、またはタッチパッドと呼ばれるポインティング装置１６１などの入力装置を介してコマンドおよび情報をコンピュータ１１０に入力することができる。

【００３５】

他の入力装置（図示せず）には、マイクロフォン、ジョイスティック、ゲームパッド、衛星パラボラアンテナ、スキャナ、無線受信機、またはテレビまたはブロードキャストビデオ受信機などがある。これらおよび他の入力装置は、しばしばシステムバス１２１に結合されているユーザ入力インターフェース１６０を介して処理ユニット１２０に接続されるが、パラレルポート、ゲームポート、ユニバーサルシリアルバス（ＵＳＢ）など他のインターフェースおよびバス構造で接続してもよい。モニタ１９１または他のタイプの表示装置もまた、ビデオインターフェース１９０などのインターフェースを介してシステムバス１２１に接続される。モニタに加えて、コンピュータは、出力周辺インターフェース１９５を介して接続できるスピーカ１９７、プリンタ１９６などの他の周辺出力装置を含むこともできる。

【００３６】

コンピュータ１１０は、リモートコンピュータ１８０など１つまたは複数のリモートコンピュータへの論理接続を使用してネットワーク式環境で動作することができる。リモートコンピュータ１８０は、パーソナルコンピュータ、サーバ、ルータ、ネットワークＰＣ、ピア装置、または他の一般のネットワークノードでよく、一般にコンピュータ１１０に関連して上述した多くまたはすべての要素を含むが、図１にはメモリ記憶装置１８１のみを示している。図１に示した論理接続は、ローカルエリアネットワーク（ＬＡＮ）１７１および広域ネットワーク（ＷＡＮ）１７３を含むが、他のネットワークを含んでいてもよい。こうしたネットワーキング環境は、オフィス、全社規模のコンピュータネットワーク、イントラネット、およびインターネットではごく一般的である。

【００３７】

ＬＡＮネットワーキング環境で使用する場合、コンピュータ１１０は、ネットワークインターフェースまたはアダプタ１７０を介してＬＡＮ１７１に接続される。ＷＡＮネットワーキング環境で使用する場合、コンピュータ１１０は一般に、モデム１７２、またはインターネットなどＷＡＮ１７３を介して通信を確立する他の手段を含む。モデム１７２は、内蔵のものでも外付けのものでもよく、ユーザ入力インターフェース１６０または他の適切な機構を介してシステムバス１２１に接続することができる。ネットワーク式環境では、コンピュータ１１０に関連して示したプログラムモジュール、またはその一部をリモートメモリ記憶装置に格納することができる。図１は、それだけには限定されないが一例として、リモートアプリケーションプログラム１８５をメモリ装置１８１上に存在するものとして示している。図示したネットワーク接続は例であり、コンピュータ間の通信リンクを確立する他の手段を使用してもよいことは理解されよう。

【００３８】

ＩＩＩ．概要

ＧＰＵは、リアルタイム３Ｄグラフィックス表示を速めるように設計されている。より優れたグラフィックスに対する要求が増えるにつれて、ＧＰＵは急速により強力でプログラム可能なものになりつつある。プログラム可能性の向上の結果、ＧＰＵは、他の多くのタイプの非グラフィックス関連プロセスの効率的な処理を行うこともできる。本明細書で開示した機械学習ＧＰＵ実行エンジンおよび方法は、ＣＰＵ処理の一部をＧＰＵに移植することによってＣＰＵの計算の制約を軽減する。より具体的には、本発明は、従来ＣＰＵによって処理されていた機械学習アーキテクチャをＧＰＵに移植する。以下で詳述するように、これは、ＧＰＵのいくつかの制限を克服し、機械学習技術のＧＰＵ処理を速め、最適化するためのいくつかの実装技術の使用を必要とする。汎用コンピューティングでは、機械学習は、ＧＰＵへの移植に特に適している。というのは、ＧＰＵは一般のＣＰＵより強力だからである。

【００３９】
図２は、本明細書で開示した機械学習ＧＰＵ実行エンジンおよび方法の実装例を示すブロック図である。図２は、機械学習ＧＰＵ実行エンジンおよび方法を実施し、使用できるいくつかの方法の１つにすぎないことに留意されたい。機械学習技術は、トレーニング段階およびテスト（または使用）段階中にＧＰＵによって処理することができる。トレーニング段階とは、アルゴリズムのパラメータがトレーニングデータを使用して調整される計算を指す。テスト段階とは、システムを使用して、入力データおよびトレーニング済みパラメータに応じて有用な情報を計算する計算を指す。トレーニングは一般に時間がかかるが、行うのは一度だけでよい。テストは、配置したシステムを使用することを指し得るが、例えば文字認識または音声認識などの場合、時として非常に迅速な応答時間を必要とすることがある。トレーニングが長時間かかり、しかし最大限のハードウェア独立性が望まれる分野では、トレーニングはＧＰＵで行われ、テストはＣＰＵで行われる。また、テストアルゴリズムは簡単であるが、トレーニングアルゴリズムは非常に複雑でＧＰＵでは稼動しない可能性もある（例えばニューラルネットワークにおけるトレーニング可能な重畳レイヤ（ｃｏｎｖｏｌｕｔｉｏｎ　ｌａｙｅｒ）など）。この場合、学習アルゴリズムをＣＰＵでトレーニングし、しかしテスト段階中はＧＰＵを使用することができる。一部の場合には、当然、ＧＰＵでトレーニングおよびテストの両方を稼動させることが好ましいこともある。ＧＰＵの実施は、トレーニングおよびテストの両方について、機械学習処理速度を、ＣＰＵのみの実施に比べて最高で１桁スピードアップする。ＧＰＵ技術は新しく、古いＣＰＵ技術より進化が早いため、この比率は増加している。

【００４０】
より具体的には、図２に示すように、コンピューティング装置１１０は、機械学習ＧＰＵ実行エンジン２００を含む。コンピューティング装置は、ＣＰＵ１２０およびＣＰＵメモリ１３０をさらに含む。ＣＰＵは、高速バス２１０（ａｃｃｅｌｅｒａｔｅｄ　ｂｕｓ）を介してビデオインターフェース１９０と通信する。このバス２１０は、３Ｄグラフィックスのスループットに対する要求のために特に設計されたＡｃｃｅｌｅｒａｔｅｄ　Ｇｒａｐｈｉｃｓ　Ｐｏｒｔ（ＡＧＰ）またはより新しいＰＣＩ　Ｅｘｐｒｅｓｓであることが好ましい。

【００４１】
ビデオインターフェース１９０は、ＧＰＵ２２０およびＧＰＵメモリ２３０を含む。ＧＰＵ２２０は、バス２１０を介してＣＰＵ１２０にデータを転送することができる。また、機械学習ＧＰＵ実行エンジン２００もＧＰＵ２２０およびＧＰＵメモリ２３０と通信する。機械学習ＧＰＵ実行エンジン２００は、トレーニングモジュール２４０およびテストモジュール２５０を含む。トレーニングモジュール２４０は、機械学習技術のトレーニング段階中に、ＧＰＵを使用して技術のパラメータを訓練するために使用される。テスト（または使用）モジュール２５０は、トレーニング済みパラメータおよび入力の所与の関数を計算するために使用される。計算の結果は、機械学習ＧＰＵ実行エンジン２００からＣＰＵ１２０に転送されて、音声認識など、対話型使用アプリケーションで使用される。

【００４２】
ＩＶ．システムの構成要素
図３は、一例として勾配降下アルゴリズムを使用する３層学習機械のトレーニングモジュールを示すブロック／フロー図である。図３は、一例として示したにすぎず、本発明を勾配降下アルゴリズムに限定するためのものではない。この例では、勾配降下を使用して学習機械の最適化が行われる。他の機械学習技術では、ベイジアンネットワーク、グラフィカルモデルなどのように、勾配の代わりに確率を伝えることもある。層の数または構成もまた制限的なものではなく、本発明はより多くの層、より少ない層、または異なる層の構成（不連続）を有することができる。パラメータを$W=(W_1, W_2, W_3)$とする。モジュール全体の入力はＸ、出力は$Y=Y_3$である。

【００４３】
各層は、その入力Ｙ、Ｘ、Ｗの関数Ｇを計算する。層の関数の観点から見ると、Ｗまた

はＹは対称的であり、区別できない。これは、W_1、W_2、W_3は、それ自体他の層によって計算することができることを意味する。Ｇは、ＹおよびＷより多くの入力を有することもできる。各層は、その入力の関数を計算し（前方パス）、ｄＥ／ｄＹで示した出力勾配にヤコビの転置を掛けることによって得られるその出力の導関数を伝えることによって隣接した層と通信する。これは後方パスである。定義上、Ｇのヤコビ Ｊは、行列J_{ki}＝dY_k/dX_iとなる。式中、ｋは関数Ｇのｋ番目の出力、およびｉは関数Ｇのｉ番目の入力の指数を表す。誤差関数Ｅは、費用関数を使用して最後の層の出力をターゲットＴと比較する。異なる費用関数の例には、平均二乗誤差（ＭＳＥ）、クロスエントロピー（ＣＥ）などがある。費用関数は、勾配、すなわち各変数が費用にどれだけ影響を与えるか、またどの方向に影響を与えるかを変数ごとに提供している。この勾配は、システムにおける変数およびパラメータごとに計算される。パラメータＷは、費用関数を低減するように更新される。図４は、図３に示した３層学習機械のテストモジュールを示すブロック／フロー図である。

【００４４】

Ｖ．動作の概要

本明細書で開示した機械学習ＧＰＵ実行エンジン２００は、機械学習ＧＰＵ実行方法を使用して、ＧＰＵによる機械学習技術の処理を可能にする。一般にこの方法は、ＧＰＵを使用して機械学習技術をトレーニングするトレーニング段階、およびトレーニング済み関数を使用して適用された問題を解決するテスト段階に分けることができる。次にこれらの各段階について説明する。

【００４５】

トレーニング段階

トレーニング段階では、機械学習ＧＰＵ実行方法を使用して、機械学習技術の学習可能パラメータをトレーニングする。図５は、機械学習ＧＰＵ実行方法のトレーニング段階の動作の概要を示すフロー図である。図５は、ＧＰＵで実行されるトレーニングセッションのデータフローおよび制御構造を表す。図は、任意のタイプの機械学習技術に固有のものではないことに留意されたい。多くのタイプの機械学習技術（ＳＶＭ、Ｋ－ｍｅａｎｓ、ＬＶＱ（Ｌｅａｒｎｉｎｇ Ｖｅｃｔｏｒ Ｑｕａｎｔｉｚａｔｉｏｎ）、期待値最大化（ＥＭ）など）がこのアーキテクチャを使用することができる。上記の機械学習技術は当分野ではよく知られており、詳しい説明は行わない。図５に示すように、ＣＰＵは、命令をＧＰＵに与え、ＧＰＵがこれらの命令を完了するまでブロックする。あるいは、ＣＰＵおよびＧＰＵに同時に計算させることもできる。

【００４６】

図５を参照すると、機械学習ＧＰＵ実行方法のトレーニング段階は、シェーダプログラム（Ｐ）、重みなど最初のトレーニング可能パラメータ（Ｗ）、および他の学習パラメータ（Ｌ）をロードする（ボックス５００）ことによって開始する。このデータは次いでＣＰＵからＧＰＵに転送される（矢印５０５）。ＣＰＵからＧＰＵへのデータ転送は、比較的費用がかかるため、ＣＰＵは、トレーニングループに入る前に、ＧＰＵにできる限りプリロードしておく。トレーニング可能パラメータ（Ｗ）は、ＧＰＵメモリ２３０に格納される（ボックス５１０）。好ましい実装では、学習パラメータ（Ｌ）は、ニューラルネットワーク層のそれぞれの重み、および各ユニットの閾値である。学習パラメータ（Ｌ）は、学習率と呼ばれる単一のスカラである。シェーダプログラム（Ｐ）は、順伝搬（ｆｏｒｗａｒｄ ｐｒｏｐａｇａｔｉｏｎ）および逆伝搬（ｂａｃｋｗａｒｄ ｐｒｏｐａｇａｔｉｏｎ）および重みの更新に使用する様々なタイプのシェーダのためのものである。

【００４７】

ＣＰＵは次いでトレーニングデータに対してループを開始し、トレーニングデータ（Ｘ，Ｔ）のバッチをロードすることによってトレーニングデータの組を蓄積する（ボックス５１５）。（Ｘ）はピクセル画像、（Ｔ）はターゲットラベルを表す。グループ（またはバッチ）を使用する理由は、ＣＰＵとＧＰＵとの間のデータ転送を開始する費用があるからである。いくつかのパターンのグループごとに同時にデータの転送を行うことがより効

率的である。

【００４８】

次いでピクセル画像（Ｘ）は、（Ｘ）が（Ｘ’）に変換されるように前処理される（ボックス５２０）。次に（Ｘ’）および（Ｔ）はＧＰＵに送信される（矢印５２５）。次いでトレーニングデータ（Ｘ’）は、トレーニングターゲットデータ（Ｔ）（ボックス５３５）とともにＧＰＵメモリ２３０に格納される（ボックス５３０）。前処理には、いくつかあげると、データをよりよい形式にするためのデータの標準化、インテリジェントなまたは複雑な特徴の抽出、およびデータセットの質を高めるための歪みの生成など、多くの様々な機能があり得る。理論上、前処理は、ＧＰＵまたはＣＰＵのいずれかで行うことができる。しかし実際には、ＧＰＵよりＣＰＵでプログラムする方がかなり容易である。これは、前処理が計算的に高価でない場合、ＣＰＵで前処理を稼動させる方がかなり容易であることを意味する。一部の場合、演算の中には、ＧＰＵでは容易ではなく、非効率で、まったく実施できず、したがってＣＰＵで稼動させなければならないものもある。一部の場合、前処理は、トレーニング段階およびテスト段階で異なっていてもよい。例えばトレーニング中、元のデータから人工データ（変換、回転など）を生成することができる。このプロセスは、テスト時に生成を向上させることで知られている。

【００４９】

前処理にトレーニング可能パラメータを含めることができることに留意されたい。これは、トレーニング可能パラメータが大域的最適化の一部であり、ＧＰＵに常駐する学習パラメータとともにトレーニングされる場合でさえも当てはまる。しかし、次のパターンが更新された前処理から利益を得るように、場合によっては、各パターンの後に、ＣＰＵに常駐するパラメータを更新するために情報（誤差勾配、負のフィードバックなど）をＧＰＵからＣＰＵに戻す必要があるという難問が生じる。現在、ＧＰＵからＣＰＵへのデータフローは最適化されない。というのは、一般にグラフィックカードは、データを画面には送信するが、ＣＰＵには戻さないように設計されているからである。その結果、現在のアーキテクチャでは、すべてのトレーニング可能パラメータをＧＰＵで保持することが好ましい。他のすべてのデータをＣＰＵで保持し、処理はＧＰＵで行うことが好ましい。あるいは、トレーニング可能パラメータをＣＰＵおよびＧＰＵの両方に配置することができる。

【００５０】

トレーニングデータ（Ｘ’）がＧＰＵにロードされると、ＣＰＵは、機械学習技術の処理に必要な様々なシェーダを稼動させるようＧＰＵに命令する（ボックス５４０）。一般の機械学習技術は、ＧＰＵ２２０にロードされた学習モジュール５４５によって表される。学習モジュール５４５は、前処理された入力（Ｘ’）およびトレーニング可能パラメータ（Ｗ）に応じて関数Ｇ（Ｘ’，Ｗ）を計算する学習機械５５０を含む。目的は、この出力をターゲット値（Ｔ）にできるだけ近づけることである。Ｇ（Ｘ’，Ｗ）と（Ｔ）との間の誤差５５５が計算され、誤差信号（Ｗに対する勾配など）が学習機械５５０に送り返される。次いでＧ（Ｘ’，Ｗ）と（Ｔ）との間の誤差を減らすために重み（Ｗ）が更新される。

【００５１】

一例として、２層ニューラルネットワークをトレーニングするとき、前方および後方の伝達は、約２０の異なるシェーダ（その一部は複数回呼び出される）に相当する。シェーダの数および複雑度は、当然、使用するアルゴリズムによって異なり得る。シェーダは、１つのグループ内のパターンごとに呼び出される（例えば一部の場合では５００回）。確率的勾配降下の場合、グループ内の各パターンを処理した後で学習パラメータが更新される。バッチ勾配降下の場合、学習パラメータが更新される前に学習パラメータの勾配がいくつかのパターンにわたって蓄積される。確率的勾配降下を使用すべきか、バッチ勾配降下を使用すべきかは、アプリケーションおよび学習アルゴリズムにかなり依存する。ＳＶＭなど一部の学習アルゴリズムでは、問題は生じない。手書き文字認識およびニューラルネットワークでは、確率的勾配降下機械学習技術が好ましい機械学習技術である。

【００５２】
ＣＰＵがトレーニングループ中のＧＰＵからフィードバックを受け取ることができるように、次の２つのプロセスを普遍性のために追加することができる。例えば、図５に示すように、ＣＰＵは、トレーニング統計値を収集し、トレーニング進捗データ（ｔｒａｉｎｉｎｇ　ｐｒｏｇｒｅｓｓ　ｄａｔａ）を取得することができる（ボックス５６０）。これは、ＧＰＵが更新済みのトレーニング可能パラメータ（Ｗ）および誤差をＣＰＵに送信することによって達成される（矢印５６５）。このプロセスは、図５の点線のボックスで示すように、オプションである。トレーニング進捗データを使用して、学習パラメータ、またはいくつかの種類のパターンの提示の頻度さえも調整することができる（ボックス５７０）。例えば、ニューラルネットワークでは、時として学習が進行するにつれて学習率を低下させるのが望ましいことがある。「ブースティング（ｂｏｏｓｔｉｎｇ）」と呼ばれる別のクラスのアルゴリズムでは、いくつかのパターンの頻度、またはその学習の影響は、システムによる誤差に応じて変更することができる。

【００５３】
前処理でのトレーニング可能パラメータも、ＧＰＵからの誤差信号に応じて変更することができる。これは、誤差信号がＣＰＵに戻り、ＣＰＵで学習パラメータを更新することができるようにすることによって達成される。グループのサイズは、それに応じて変更することができる。極端には、グループのサイズは１であり、これは、前処理のセクションで上述したように、誤差信号がＧＰＵから戻るとすぐに、ＣＰＵのトレーニングパラメータがそれぞれＧＰＵに提示された後で更新されることを意味する。

【００５４】
次いで、トレーニングが完了したかどうかに関する決定が行われる（ボックス５７５）。これは、一定の回数の反復の後、トレーニングデータのバッチのすべてが処理されたとき、または所望の誤差閾値が達成されたときに決定することができる。トレーニングが完了していない場合、トレーニングループがデータのロードから再開する（ボックス５１５）。そうでない場合、最後にトレーニングされたパラメータ（Ｗ）が取得される（ボックス５８０）。これは、ＧＰＵがパラメータ（Ｗ）をＣＰＵに送信することによって達成される（矢印５８５）。次いでトレーニングが完了する（ボックス５９０）。

【００５５】
テスト段階
テスト段階では、機械学習ＧＰＵ実行方法を使用して入力およびトレーニング可能パラメータの関数を計算する。図６は、機械学習ＧＰＵ実行方法のテスト段階の動作の概要を示すフロー図である。図６に示すように、機械学習ＧＰＵ実行方法は、一部のトレーニング可能パラメータＷおよび入力Ｘの所与の関数Ｇを計算する。トレーニング可能パラメータは、ＧＰＵ、ＣＰＵ、または他の任意の手段によって計算されている可能性があることに留意されたい。さらに、トレーニング可能パラメータは、トレーニングの結果であっても、またはそうでなくてもよい。この点で、図６は、図５に示したトレーニング可能パラメータのトレーニングには依存していない。

【００５６】
図６では、図５のアーキテクチャと類似のアーキテクチャが使用されている。特に、図５および図６のアーキテクチャは、データ構造およびシェーダプログラムＰを共有する。しかし１つの重要な違いは、トレーニングがすでに完了しているため、逆伝搬シェーダはもはや必要ないということである。機械学習ＧＰＵ実行方法のテスト段階は、シェーダプログラムＰおよびトレーニング可能パラメータＷをＧＰＵにロードする（ボックス６００）ことによって開始する。トレーニング段階と同様、テスト段階は、認識／使用ループ外にできる限りダウンロードし、事前に計算しようとする。次いでシェーダプログラムＰおよびトレーニング可能パラメータＷは、ＧＰＵに送信される（矢印６０５）。特に、シェーダプログラムＰは、ＧＰＵ２２０に送信されて処理を行い、トレーニング可能パラメータＷ６１０はＧＰＵメモリ２３０に格納される。

【００５７】

次に、関数Ｇが計算される１組のパターンＸが収集され、ロードされる（ボックス６１５）。ＣＰＵとＧＰＵとの間の通信を開始するための費用があるため、グループ分けを行うことによって、この費用をいくつかのパターンにわたって消却することができる。次いでパターンＸは、ＣＰＵからＧＰＵに送信される前に前処理されてＸ’になる（ボックス６２０）。次いでＸ’はＧＰＵに送信される（矢印６２５）。入力データＸ’６３０は、ＧＰＵメモリ２３０に格納される。

【００５８】

前処理は、ＣＰＵまたはＧＰＵのいずれかで行うことができる。しかし、前処理は、計算的に費用がかかりすぎない限りＣＰＵで行われることが好ましい。前処理は、標準化、および重要な情報および特徴をデータから抽出するなどの機能に有用である。前処理後、パターンＸ’は、グループとしてＧＰＵに送信される（矢印６２５）。

【００５９】

次に、ＣＰＵは、シェーダプログラムＰを使用するようＧＰＵに命令する（ボックス６３５）。関数モジュール６４０は、ＧＰＵに存在し、トレーニング済み関数６４５を含む。トレーニング済み関数６４５をシェーダプログラムとともに使用して、関数Ｙ＝Ｇ（Ｘ’，Ｗ）を計算する。ＧＰＵメモリ２３０内の一時計算スペース６５０を使用して、この計算を助けることができる。この計算から、グループの各パターンの結果Ｙ６５５がＧＰＵメモリ２３０に蓄積され、関数モジュールの出力として送信される（ボックス６６０）。次いで結果Ｙは、ＣＰＵに送り返され（矢印６６５）、結果ＹはＣＰＵで取得される（ボックス６７０）。繰り返しになるが、ＧＰＵからＣＰＵへの転送はかなり費用がかかるため、グループ分けを行い、Ｙのサイズを最低限に抑えることが有利である。手書き文字の分類の例では、Ｙは画像Ｘのクラスであるにすぎず、したがって非常に小さい。クエリデータＸの追加のバッチがあるかどうかに関する決定が行われる（ボックス６７５）。バッチがある場合は、認識／使用ループが再開する。そうでない場合、結果Ｙが出力として送信される（ボックス６８０）。

【００６０】

ＶＩ．動作の詳細

機械学習の問題の説明

いくつかの機械学習技術は、非常に大きい行列演算として実行することができる計算を核として含んでいる。特に興味深いのは、大きい行列／ベクトル乗算である。一例として、手書き文字認識に使用されるニューラルネットワークは、一般に１対の層を含んでいる。図７は、ニューラルネットワークにおける１対の層の詳細図である。

【００６１】

図７を参照すると、次のように、隠れ変数（ｈｉｄｄｅｎ　ｖａｒｉａｂｌｅ）の計算が行列乗算として実行され、次いでマッピングステップが行われる（１層の場合の計算）。

［ｗ］［ｉ］＝［ｏ’］

［ｏ］＝ｆ（［ｏ’］）

【００６２】

式中ｆはシグモイド関数

【００６３】

【数１】

$$f(x)=\frac{1}{e^{-x}+1}$$

【００６４】

またはｔａｎｈ関数

【００６５】

【数２】

$$f(x)=\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$$

【００６６】

である。こうした２つの層はカスケードされて、結果が生成される。入力のベクトルサイズ、隠れ変数、および出力は何千にも及ぶ。図７および上記はニューラルネットワークに基づいているが、他のいくつかの機械学習問題は、上述した大きい浮動小数点ベクトルに対する反復演算の一般的なモデルに適合することに留意されたい。

【００６７】

ＧＰＵでの基本的な演算

幸いにも、ニューラルネットワーク、期待値最大化、および他の多くの勾配降下ベースのアルゴリズムなどの多くの機械学習技術は、簡単な基本要素（または構成単位）から成る。基本要素は、

・内積（ベクトルまたは行列とベクトルとの間）

・外積（ベクトル間）

・線形代数（ベクトルまたは行列に対するスカラによる加算、減算、乗算）

・ベクトルまたは行列に適用される非線形（ｔａｎｈ、シグモイド、閾値処理（ｔｈｒｅｓｈｏｌｄｉｎｇ））

・行列の転置

【００６８】

機械学習ＧＰＵ実行方法は、ピクセルシェーダを使用してこれらの各演算を実施する。さらに、この方法によって、これらの各演算を共にトレーニングのために使用したり、実際の設定に使用したりできるようになる。この方法は、ニューラルネットワーク、または後述したものと同様の基本要素またはその単なる拡張から成る他の任意の学習アルゴリズムとともに使用することができることに留意されたい。

【００６９】

ピクセルシェーダを使用したＧＰＵ計算

機械学習ＧＰＵ実行方法は、１つまたは複数のピクセルシェーダを使用して上記のすべての演算を実施する。ピクセルシェーダは、グラフィックス処理パイプラインで使用される。ピクセルシェーダは、ピクセルレンダリングハードウェア上で実行される個々のプログラムである。特に、三角形をピクセルに変えるプロセスは、「ラスタ化」と呼ばれる。ハードウェアは、三角形を入力としてみなし、三角形がレンダリングされる前にロードすることができるプログラムで指定する各ピクセルをレンダリングするよう命令することができる。これらのプログラム可能な三角形レンダラを「ピクセルシェーダ」と呼ぶ。シェーダでのプログラムの命令は、それぞれ直接的なハードウェアの実施を有するため、アセンブリ言語に近い。例えばＨｉｇｈ　Ｌｅｖｅｌ　Ｓｈａｄｅｒ　Ｌａｎｇｕａｇｅ（ＨＬＳＬ）、Ｎｖｉｄｉａ（登録商標）社のＣ　ｇｒａｐｈｉｃｓ（Ｃｇ）、ＤｉｒｅｃｔＸなど、シェーダの競合する言語（およびハードウェア）がある。ピクセルシェーダによって導入された新しい柔軟性によって、表面の写実的なレンダリングだけではなく、ＧＰＵを汎用並列プロセッサにより近づけることができるようになる。

【００７０】

機械学習ＧＰＵ実行方法は、ＧＰＵを汎用並列プロセッサとして使用する。さらにこの方法は、ピクセルシェーダを使用して、機械学習技術で使用される様々な基本的な演算を実施する。並列専用ＧＰＵハードウェアの結果、これらのアルゴリズムのトレーニングおよび使用は、ＧＰＵでは、ＣＰＵに比べて１桁速く稼働する。

【００７１】

ＤｉｒｅｃｔＸのＤｉｒｅｃｔ３Ｄ構成要素では、頂点シェーダおよびピクセルシェーダと呼ばれる２つの要素がある。ＤｉｒｅｃｔＸは、ＧＰＵのハードウェア加速機能をアプリケーションに利用させるＭｉｃｒｏｓｏｆｔ（登録商標）社によって開発された１組

のアプリケーションプログラムインターフェース（ＡＰＩ）である。現世代のピクセルシェーダは、プログラム可能性が高い。両方のタイプのシェーダは、三角形（グラフィックオブジェクトの基本単位）の出力装置へのレンダリングと関係がある。頂点シェーダは、三角形の頂点の空間変換および動画化などのタスクに使用することができる（したがってこのように名付けられている）。単一の三角形がラスタ化されると、ピクセルシェーダを使用して個々のピクセルに陰影を付ける（または個々のピクセルのカラー値を計算する）。

【００７２】

ピクセルシェーダは、ＤｉｒｅｃｔＸで定義された限られたハードウェア依存型言語であるＤｉｒｅｃｔＸシェーダアセンブリ言語で一連の命令として表現される。シェーダでのコードは、レンダリングされる三角形のピクセルごとに一度実行され、その唯一の影響は、そのピクセルの４ベクトルの値を設定することである。シェーダ言語の制限、および副作用の欠如は、ＧＰＵが任意の順序で、そのハードウェアがサポートできるだけの並列処理を使用して自由にピクセルをレンダリングでき、その結果非常に高いパフォーマンスが得られることを意味する。ピクセルが４ベクトルであることから、さらに別の種類の並列処理を行うことができ、ピクセルシェーダの各実行によって、４つの要素（ベクトルの４つの隣接する要素など）を同時に計算することができる。

【００７３】

アセンブリ言語プログラミングで使用する機能の多くは、ピクセルシェーダ内で使用することができる。これらには、定数、レジスタ、加算、減算、乗算、逆数、小さい１組の超越関数などがある。しかし、他のよく知られているルーピング、ブランチングなどの構成体は、一般には使用できない。これは、シェーダ当たりの命令の数が制限されており（一般的なＧＰＵで９６個の命令）、シェーダはレンダリングされるピクセル以外のメモリを変更することができないからである。これらの制限は、一部のアルゴリズムはＧＰＵによる処理に適しておらず、仮に実施できたとしてもＣＰＵに比べてＧＰＵでの稼働の方が遅いことを意味する。特定の機械学習アルゴリズムをＧＰＵで実行することができるかどうかを評価するために、アルゴリズムを構成する個々の演算のそれぞれを検査する必要がある。

【００７４】

単位演算（ｕｎｉｔ　ｏｐｅｒａｔｉｏｎ）の実施

機械学習ＧＰＵ実行方法は、１つまたは複数のピクセルシェーダを使用して黒丸付きにした（ｂｕｌｌｅｔｅｄ）基本的な演算のそれぞれを実施する。第１の問題は、シェーダに結果を計算させることである。結果とはＧＰＵメモリ内の矩形である。言い換えれば、結果は浮動小数点値（４ベクトル）の２次元行列でなければならない。しかし、ピクセルシェーダは、所与の三角形のすべてのピクセルをレンダリングする。この問題を克服するために、機械学習ＧＰＵ実行方法は、三角形で矩形のビューポートを指定する。図８は、機械学習ＧＰＵ実行方法によって使用される矩形メモリ技術を示している。図８は特に、レンダリングされるピクセルを含んでいる三角形８００、および三角形８００内の矩形ビューポイント８１０を示している。機械学習ＧＰＵ実行方法によれば、ＧＰＵは、ビューポイント８１０と三角形８００との共通部分のピクセルを計算するだけでよい。したがって所望の矩形領域を計算するために、領域がビューポイント８１０として指定され、三角形８００で囲まれる。これがレンダリングターゲットとなる。

【００７５】

行列の代わりにベクトルをレンダリングする必要がある場合、同じ技術が適用される。唯一の違いは、わずか１ピクセルの高さのビューポイントの矩形が指定されることである。同様に、スカラ値をレンダリングするために、１×１ピクセルのビューポイントを使用することができる。

【００７６】

この技術およびピクセルシェーダ内で使用可能な汎用プログラミング機能では、ベクトルおよび配列についてのｘ＝Ｆ（ｘ）のような単項演算を実施することができる。他のオ

ペランドから読み取る必要がある演算では、ＧＰＵメモリ構成を検査する必要がある。特に、ＤｉｒｅｃｔＸでは、メモリの矩形領域を作業領域として割り当てることができる。図９は、機械学習ＧＰＵ実行方法のＧＰＵメモリ構成を示している。図９に示すように、作業領域９００は、ＧＰＵメモリ内に生成されている。現在のハードウェアでは、作業領域は、２０４８×２０４８ピクセル（各４値）の正方形領域の最小割り振りから生成することができる。このサブ矩形９１０から、シェーダ演算は、t_1　９２０、t_2　９３０、t_3　９４０など他のサブ矩形からオペランドをレンダリングし、フェッチすることができる。これらのフェッチは、オペランドリード（ｏｐｅｒａｎｄ　ｒｅａｄ）９５０を介して達成される。

【００７７】

このメモリ構成を容易に行えるようにするＤｉｒｅｃｔＸの機構がテクスチャマッピングである。テクスチャマッピングは、レンダリングする面上に画像を置く必要性から生じる。例えば、木星、シマウマ、レンガ壁などのテクスチャの表面をレンダリングするとき、その面に描かれたパターン、光線の当たり方、角度、反射などを考慮する必要がある。この演算では、矩形ルックアップテーブルへのインデックスがラスタ化された三角形のターゲットエリアにわたって双線形に挿入される。（レンダリングされるエリア以外の）作業領域の矩形エリアをテクスチャとみなすことによって、こうした領域を宛先矩形にマッピングすることができる。したがって、宛先矩形内の任意のｘ，ｙ位置のピクセルを計算すると、テクスチャ矩形の形状的に対応するピクセルの値へのアクセスがある。例えば、サイズａ×ｂピクセルの宛先矩形がレンダリングされている場合、同じくａ×ｂピクセルの別の領域をテクスチャマッピングすることができる。これによって、宛先のｉ番目のピクセルに対応するa_iおよびb_iのソース矩形値へのシェーダコード内での直接アクセスが得られる。この技術の簡単な適用によって、任意の行列、ベクトル、またはスカラ値を、作業領域内の同じサイズの行列、ベクトル、またはスカラなどにコピーできるようになる。あるいは、ｘピクセル×１ピクセルのテクスチャ領域を宛先矩形にマッピングすることができる。これは、レンダリングされる宛先ピクセルのｙ座標ではなくｘ座標に応じてその値が決まるルックアップテーブルへのアクセスを提供する。

【００７８】

テクスチャの有用性は、シェーダ内のレジスタ値に対する演算を使用することによって拡張することができる。レジスタは、シェーダが所与のピクセルをレンダリングするために使用できるローカル変数である。それらの値は、ピクセル間では共有することはできない（これは並列処理の前提を破ることになる）が、中間結果として（ローカルに）使用することはできる。例えば、その値を新しい位置にコピーしながら配列またはベクトルを転置することができる。その左、右、上、および下の座標がｌ、ｒ、ｔ、およびｂのソース矩形を仮定する。次いで、その座標がｔ、ｂ、ｌ、およびｒのテクスチャ矩形を指定する。ピクセルシェーダ内でｘおよびｙのテクスチャ座標が交換された後で、それらを使用して値をソースからフェッチし、宛先にコピーする。レンダリングの最後で、宛先はソースの転置を含む。

【００７９】

図１０は、テクスチャ三角形を使用したテクスチャマッピングの使用を示している。図１０に示すように、宛先１０００の現在のピクセルのレンダリングは、指定されたソースベクトル１０２０のテクスチャ矩形１０１０からテクスチャ座標（１５，７）を取り出す。テクスチャ値をフェッチする前に、転置される実際の真のソースベクトル１０４０内に配置されている位置（７，１５）を有するテクスチャ矩形１０３０からの値が実際に読み取られるように、行および列の座標が逆にされる。

【００８０】

ＤｉｒｅｃｔＸによって、テクスチャマッピングを使用して、複数のソース矩形を現在の宛先にマップすることができる。現在のハードウェアでは、各パスで少なくとも８つのこうしたマッピングを使用することができる。複数のソースでは、（ベクトルＡ－ベクトルＢ－＞ベクトルＣ）などの演算を実施することができる。Ｃでの各ピクセルで、Ａおよ

びＢからテクスチャマッピングされた値がフェッチされ、レジスタ値に対して基本的な計算が行われ、結果が格納される。

【００８１】

２つのベクトルから行列へのテクスチャマッピングも、外積（ベクトルＡ×ベクトルＢ－＞Ｃ）を実施する方法を提供する。ベクトルＡを１ピクセル幅の行ベクトル、およびＢを１行高さの列ベクトルであると仮定する。これらの縮退矩形（ｄｅｇｅｎｅｒａｔｅ ｒｅｃｔａｎｇｌｅ）を行列Ｃの矩形にテクスチャマッピングする。次いでＣのピクセルｘ、ｙをレンダリングすると、テクスチャのサンプリングによって、Ａのｙ番目の要素およびＢのｘ番目の要素が得られる。これらは単に、乗算し、格納するのに必要な値である。

【００８２】

４つの成分の使用

内積の実施を説明する前に、ＧＰＵ作業領域が４つの成分を有しているという点から各単位演算について説明する。各ピクセルがｘ、ｙ、ｚ、およびｗの値から成るためにこれらの成分が生じる。ラベルｘおよびｙは、ここでは、上記の説明でピクセル座標を参照するｘおよびｙと混同しないものとする。４つの成分は従来、４次元物体空間に座標を格納するために使用されている。機械学習ＧＰＵ実行方法は、ＧＰＵプログラミングモデルおよびハードウェアのこの特徴を活用して、より速く計算を行う。成分のうちの３つを無視して、すべての計算を例えばｘ平面で行うこともできるが、その結果得られるプログラミングの単純化は、パフォーマンスの面で代償が高くなる。

【００８３】

成分を利用するために、ピクセル平面の数学的配列またはベクトル内の位置へのマッピングが定義される。ベクトルの場合、最も簡単なマッピングは、

ピクセル０：ｘ－＞要素０
ピクセル０：ｙ－＞要素１
ピクセル０：ｚ－＞要素２
ピクセル０：ｗ－＞要素３
ピクセル１：ｘ－＞要素４

などである。

【００８４】

これを行列に拡張するため、行列の各行（または列）がベクトルであることが観察される。上記のマッピングは、各行（または列）に適用される。４つの成分を行数（ｒｏｗ ｄｉｍｅｎｓｉｏｎ）にまとめる（ｃｏｌｌａｐｓｅ）か、列数（ｃｏｌｕｍｎ ｄｉｍｅｎｓｉｏｎ）にまとめるかは選択であり、この選択は、プログラミングを単純化する方法で行列ごとに個々に行うことができる。

【００８５】

行列またはベクトルの要素からピクセルおよび成分へのマッピングが与えられている場合、コピー演算はまったく影響を受けないことがわかる。シェーダ命令ｔｅｘｌｄおよびｍｏｖは、多くの他のものと同様、一度に１つのピクセルに対して作用するため、それぞれ４つの値をテクスチャピクセルからレジスタに、またレジスタから宛先ピクセルに移動させる。

【００８６】

ベクトルに対する転置演算も変わらない。成分は常にベクトルの寸法にまとめられる。行列の場合、コードは変わらないが、まとめる方向がデータとともに転置されることに留意されたい。

【００８７】

数学的演算を適用することもできる。多くのシェーダ命令を適切な構文とともに４ベクトルの値、または単一の値に使用することができる。例えば、指数ｒ１．ｘ，ｒ０．ｘは、レジスタ０のｘ平面をフェッチし、それをべき乗し、その結果をｒ１のｘ平面に格納する。

【００８８】
外積
２つのベクトルの外積を、４つすべての成分の使用で達成することができるが、別の新しい技術を導入する必要がある。この技術は、機械学習ＧＰＵ実行方法によって使用され、インデクサテクスチャ技術（ｉｎｄｅｘｅｒ　ｔｅｘｔｕｒｅ　ｔｅｃｈｎｉｑｕｅ）と呼ばれる。一般にこれは、上述したように要素にマップされたベクトルの４つの成分のうちのただ１つの成分の値を選択する方法である。
【００８９】
一例として、ベクトルＡがサイズａの列ベクトルであると考える。ベクトルＢはサイズｂの行ベクトルである。外積Ｃ、すなわちａ行高さおよびｂ列幅の行列を計算することが望ましい。Ｃは、その４つの成分を列数すなわちｙにまとめることである。言い換えれば、ピクセルに関しては、Ｃのメモリ矩形はｂ列幅であり、しかしａ／４行高さである（各ピクセル行は行列の４つの行を格納するため）。ａは４の倍数である必要がある。
【００９０】
Ｃの各ピクセルを計算するシェーダルーチンが必要である。テクスチャマッピングは、Ａについては単純であるが、Ｂについては単純ではない。Ｃのｉ番目のピクセルを計算するとき、Ｂのｉ番目の値（ピクセルではない）へのアクセスが必要である。例えば、Ｃのピクセル０，０の４つの成分では、以下の値を計算する必要がある。

$C_{0,0}=A_0*B_0$
$C_{1,0}=A_1*B_0$
$C_{2,0}=A_2*B_0$
$C_{3,0}=A_3*B_0$

【００９１】
この計算は、１つのＧＰＵ　ｍｕｌ（乗算）命令で行うことができる。これには、（ｒ２にA_0～A_3を、ｒ３のｗ成分にB_0を格納するなど）何らかの方法で値B_0をレジスタのｗ平面に置くことが必要である。したがって次のようになる。
ｍｕｌ　ｒ１，ｒ２，ｒ３．ｗｗｗｗ
【００９２】
もう１ステップ戻って、そのピクセル内の隣接するB_1～B_3からB_0を選び出す「ビットマスク」があった場合、ｄｐ４（ドット積）命令を使用してｒ３のｗ成分に値を抽出することができることになる。ｒ５が（１，０，０，０）を含み、ｒ４が（Ｂのテクスチャマッピングによってロードされた）B_0～B_3を含むと仮定する。次いでこの命令は、（ｒ４．ｘ＊１＋ｒ４．ｙ＊０＋ｒ４．ｚ＊０＋ｒ４．ｗ＊０）を計算する。これはｒ４．ｘに等しいか、B_0：
ｄｐ４　ｒ３．ｗ，ｒ４，ｒ５
となる。
【００９３】
内積は、行列Ｃの列０（０，１，０，０）がレンダリングされるとき、列１（０，０，１，０）がレンダリングされるとき、列２がレンダリングされるときなど、値（１，０，０，０）が使用可能な場合に計算することができる。これがインデクサテクスチャ技術の目的である。インデクサテクスチャ技術は、４ピクセル幅および１ピクセル高さで、次の値に初期設定される非常に小さいテクスチャを使用する。
【００９４】
【表１】

1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

【００９５】
これら４つのピクセル値は、上述した技術を使用して個々の値をベクトルＢから抽出するのに必要なすべての「ビットマスク」から成る。残るのは、インデクサテクスチャ技術の正しいピクセルをシェーダの各呼出で使用可能にする方法を見つけることである。このタスクを達成するために、機械学習ＧＰＵ実行方法は、テクスチャマッピングのさらに別のアプリケーションを使用する。
【００９６】
この時点まで、テクスチャピクセルと宛先ピクセルとの間の１対１マッピングを保つ方法でテクスチャマッピングが使用されてきた。これは、同じ数のピクセルを含むテクスチャ座標矩形および宛先矩形を定義することによって行われた。インデクサテクスチャ技術では、次のようにインデクサテクスチャ自体のサイズ（定義上１．０×１．０テクスチャ座標単位）の倍数であるテクスチャ座標矩形が定義される。行列Ｃの幅がｗピクセルの場合、その幅がｗ／４（必ず整数）であるインデクサテクスチャのテクスチャ座標矩形が指定される。Ｃの幅にわたる各ピクセルでシェーダが呼び出されるので、インデクサテクスチャ座標は０からｗ／４に及ぶ。言い換えれば、Ｃのピクセルを４つ通過するごとに、インデクサ座標がある整数値から次の整数値に変わる。Ｃの４ピクセルごとにテクスチャ座標の端数部分を考える場合、これは０から１に及ぶ。Ｃの各ピクセルで正しい「ビットマスク」を得るために、インデクサテクスチャをサンプリングするのに必要な値がまさにこれらの値である。
ｆｒｃ　ｒ３，ｔ３　／／インデクサテクスチャ座標の端数部分をレジスタ３にロードする
ｔｅｘｌｄ　ｒ４，ｒ３，ｓｌ　／／レジスタ３のテクスチャ座標を使用してインデクサテクスチャをサンプリングする
上記の技術を使用して、２つのベクトルの外積を計算するピクセルシェーダが作成される。
【００９７】
内積
内積とは、（行列Ａ＊ベクトルＢ－＞ベクトルＣ）と表すことができる演算である。内積は問題を提起する。というのは、いくつかの値にわたってループしている間、合計を累積することが必要となるからである。これは、隣接するピクセル間に通信がないことを前提とするシェーダアーキテクチャに反する。このため、内積は、単一のピクセルシェーダとして実施することができない。しかし、以下に示すように、一連のシェーダおよび一部の追加メモリを使用することによって内積を実施することができる。
【００９８】
サイズａ行×ｂ列の行列Ａに、サイズａの行ベクトルであるベクトルＢを掛ける。Ｂは通常、列ベクトルとして表されるが、行ベクトルの使用によってＧＰＵの計算が単純化することに留意されたい。ベクトルの転置済みのコピーは、機械学習ＧＰＵ実行方法を使用して作成することができるため、これは障害ではないことを思い出されたい。その結果のＣは、高さｂの列ベクトルである。行列Ａは、上記のように、ピクセル当たりその４つの成分がｙ（行）数にまとめられた状態で構成される。
【００９９】
図１１は、内積を示すブロック図である。図１１の対応する式は
$C_0 = A_{0,0} * B_0 + A_{0,1} * B_1 + A_{0,2} * B_2 + A_{0,3} * B_3$
$C_1 = A_{1,0} * B_0 + A_{1,1} * B_1 + A_{1,2} * B_2 + A_{1,3} * B_3$
などである。
【０１００】
レンダターゲットＣは１次元であることに留意されたい。これは、前の手法を無効にする。というのは、定義された任意のテクスチャマップはそれぞれ、Ｃの任意の所与のピクセルの１つの値しか有することができないからである。しかしＣの各値は、Ｂ、およびＡの１行内のすべての値に応じて決まる。したがって別の手法を使用して、シェーダのＡお

よびＢから複数の値にアクセスする必要がある。この手法は、まず、複数のテクスチャマップ（現在のハードウェアでは８個が妥当な数である）を使用して、複数のオペランドを効率的に読み取る。次に、シェーダ内でアドレス計算を使用して、追加のオペランドを読み取る。

【０１０１】

これらの技術でさえ、単一のレンダリングパスで大きい内積を計算することはできない。これは、ピクセルシェーダがブランチングやルーピングを使用することができず、また、限られた数（現在の１回の実施では９６）のアセンブリ命令しか含めないからである。従って各シェーダは、命令制限に到達するまでにある量の計算しか行うことができない。幸いなことに、内積は、一度に考えられるＡの列数を制限することによって１組のサブ問題に分解することができる。これは、１組の列ベクトル部分積を生成する。次いで内積結果を含む単一の列ベクトルを取得するまでこれらの部分積を繰り返し減らすことができる。

【０１０２】

分解の技術は次のとおりである。まず、Ａの１２列のサブ行列にＢの１２列のサブベクトルを掛けるシェーダが作成される。８つのテクスチャマップが使用可能であり、６つはＡに、２つはＢに割り振られる。Ａの６つのマップはすべて同じサイズのソース矩形を、しかし０ピクセルから５ピクセルまでの６つの異なる横方向のオフセットで有している。これによって行列の最初の６列への直接アクセスが得られる（横方向では１ピクセルは１列に等しいことを思い出されたい）。Ｂの２つのマップも１ピクセルだけオフセットしており、Ｂの最初の８列への直接アクセスが得られる（この場合、１ピクセルは４列に等しい）。

【０１０３】

シェーダ手順は、ＧＰＵの一時（スクラッチ）メモリを使用する宛先矩形を用い、１２列の部分積について実行される。Ｃの４行（１ピクセル）を一度に計算するシェーダ手順は、次のとおりである。

【０１０４】

ａ）４つのテクスチャマップが互いに水平方向に１ピクセルだけオフセットしている状態でｔｅｘｌｄ（テクスチャのロード）命令を４回使用して、Ａの最初の４つのピクセルをレジスタにロードする。各ピクセルは４つの行値を含む。別のｔｅｘｌｄを使用してＢの１ピクセル（４列）をロードする。

【０１０５】

ｂ）ｍｕｌを使用して最初の４つの積をレジスタ０に格納する。これは、レンダリングされた第１のピクセルに関して$A_{0,0}$＊B_0、$A_{1,0}$＊B_0、$A_{2,0}$＊B_0、および$A_{3,0}$＊B_0を計算する。表記ｘｘｘｘは、４つすべての積についてレジスタ１のｘ成分（つまりB_0）を使用することを意味する。

```
mul　r0、r1.xxxx、r2
```

【０１０６】

ｃ）ｍａｄ（乗算／加算）を使用して次の４つの積をレジスタ０に蓄積する。これは、（ｒ３を介して）Ａの第２の水平方向のピクセルにアクセスし、前の結果に積$A_{0,1}$＊B_1、$A_{1,1}$＊B_1、$A_{2,1}$＊B_1、および$A_{3,1}$＊B_1を追加する。B_1へのアクセスにｙｙｙｙを使用する。

```
mad　r0、r1.yyyy、r3、r0
```

【０１０７】

ｄ）ｍａｄを同じように２回以上使用して次の８つの積を累積する。

```
mad　r0、r1.zzzz、r4、r0
mad　r0、r1.wwww、r5、r0
```

【０１０８】

ｅ）ここで、５番目の列から８番目の列（列番号４～７）の準備をする。列４および５はすでに、Ａに割り振られた６つのうちの残りの２つのテクスチャマップによってアクセ

ス可能である。列６および７については、列６の座標をとり、定数ｃ０を２回追加する。これは、１ピクセル（または作業領域幅の１／２０４８）に等しくなるように設定されている。列６および７のこれらの座標は、追加のレジスタに格納される。次いで４つのｔｅｘｌｄ命令を使用して、これらの値、つまりＡ$_{0,4}$からＡ$_{3,7}$をレジスタに格納する。

【０１０９】

ｆ）１ピクセルだけオフセットされているＢに割り振られた第２のテクスチャマップを使用して、Ｂ$_{4}$からＢ$_{7}$（１ピクセルの幅）の値をロードする。

【０１１０】

ｇ）４つのｍａｄ命令を使用して、ステップ（ｂ）から（ｄ）で行われたように、ｒ０に格納されている現在の４つの合計に１６以上の積を累積する。

【０１１１】

ｈ）次に、部分積の最後の４つの列について準備がなされる。この時点で、すべてのテクスチャマップが使用されている。したがってまだレジスタ内にある列７のアドレスが取得され、Ｃ０が連続的にこの値に４回追加される。この結果がレジスタに格納される。同じ手順が行われて、Ｃ０がＢの第２のピクセルのアドレスを含むレジスタに追加される。これは、Ｂ$_{8}$からＢ$_{11}$にアクセスするように１ピクセルだけそれを進めるステップを含む。正しい座標がレジスタに存在すると、ｔｅｘｌｄを再度使用してＡ$_{0,8}$からＡ$_{3,11}$およびＢ$_{8}$からＢ$_{11}$の値を他のレジスタにロードする。

【０１１２】

ｉ）４つ以上のｍａｄ命令を使用して、最後の１６個の積をｒ０に累積する。ここでｒ０は、１２列の部分内積の要素Ｃ０からＣ３の値を含む。次いでシェーダルーチンが完了する。より多くの列がこのシェーダで処理されない理由は、シェーダごとの命令の制限に到達しているからである。

【０１１３】

上記は、１２列部分内積を計算して一時メモリ内の列ベクトルにする方法である。この方法は、次のように、１２列より広い行列に拡張することができる。まず、残りの幅が１２以上である限り、１２列部分積が処理される。次に、第１の部分結果に直接隣接するこれらのパスの結果が一時メモリの連続ブロックに配置される。最後に、残りの８つまたは４つ（行列幅は４の倍数でなければならないことを思い出されたい）の列がある場合、８つまたは４つの列を収容するように書き込まれた変更されたシェーダが使用される。これらの技術は、１２列シェーダの単純化である。

【０１１４】

この結果、部分結果を表す１つまたは複数の隣接する列ベクトルのブロックが得られる。これらを「低減する」、つまり合計して、最後の内積を含む（また「スクラッチ」メモリの代わりに、他のルーチンによってアクセスできる明確な場所に配置される）単一の列ベクトルにする必要がある。効率上、（レジスタにおけるアドレス計算とは対照的に）テクスチャマップを使用して、低減する必要のある部分結果にアクセスする。８つのテクスチャマップをソースとして使用することによって、８つもの部分結果を一度に低減することができる。このシェーダコードは、非常に単純であり、ｔｅｘｌｄ命令およびａｄｄ命令から成る。プロセスを単純化するための１つの方法は、最終結果ではなく、低減された結果の位置の選択によるものである。つまり、低減された結果を次の列の既存の部分結果の右に入れることができる。このように、まさに最後の低減ステップまで、低減される１組の部分結果は常に一続きの矩形である。これらの技術および十分な一時メモリを使用して、任意のサイズの内積を計算することができる。

【０１１５】

上記では、機械学習ＧＰＵ実行方法が、ニューラルネット（および同じ演算で構成される他の任意の機械学習アルゴリズム）のトレーニングに必要なすべての演算を含んでいることを示してきた。さらに、これらの演算は、ＧＰＵでピクセルシェーダとして実施することができ、いずれの場合にもアルゴリズムは高度の並列処理に役立つ。

【０１１６】

ＶＩＩ．実施例

本明細書に開示した機械学習ＧＰＵ実行システムおよび方法を完全に理解するために、実施例の動作上の詳細を提示する。この実施例は、機械学習ＧＰＵ実行システムおよび方法を実施できる方法を１つだけ示していることに留意されたい。

【０１１７】

この実施例では、機械学習ＧＰＵ実行システムおよび方法は、単一のプロセッサ２．８ＧＨｚ　Ｐｅｎｔｉｕｍ（登録商標）４、ＡＴＩ（登録商標）Ｒａｄｅｏｎ９８００グラフィックスカード付きで稼働させた。９８００グラフィックスカードは、９４箇所の命令メモリ、および２５６ＭＢの作業領域を有しており、その１６ＭＢは、読み取り／書き込みの作業領域に使用し、残りを読み取り専用データに使用した。ＣＰＵのみの参照実装は、Ｐｅｎｔｉｕｍ（登録商標）４のＳＳＥ　ＳＩＭＤ拡張を利用するようにすでに最適化されているため、ＳＩＭＤ対ＳＩＭＤでの比較を行う。さらにＧＰＵは、テストされた例についてほぼ１桁高速である。

【０１１８】

この実施例は、手書き文字認識を行うために使用される機械学習ＧＰＵ実行システムおよび方法を示す。タスクは、手書きの数字のピクセル画像のクラス（「０」から「９」）を見つけるためのものであった。トレーニングデータベースは、正しいラベルを有する６０，０００のトレーニング画像で構成されていた。このトレーニングセットは、時としてＭＮＩＳＴと呼ばれることもあり、Ｗｅｂ上で入手可能であり、機械学習アルゴリズムの資料ではよく知られている。このタスクを解決するために、ニューラルネットワーク手法が使用された。特に、ニューラルネットワークは２層完全接続ニューラルネットワークであった。

【０１１９】

逆伝搬による２層完全接続ニューラルネットワークのトレーニングが次のパスに分解された。

【０１２０】

【表２】

順伝搬	$W_1 I = H' \xrightarrow{F_1(x)} H$
順伝搬	$W_2 H = O' \xrightarrow{F_2(x)} O$
誤差計算	$\Delta_2 = \alpha(\text{target} - O) * D_2(O)$
重みの更新	$W_2 += (\Delta_2 \otimes H)$
誤算の伝搬	$\Delta_1 = (W_2^T \Delta_2) * D_1(H)$
重みの更新	$W_1 += (\Delta_1 \otimes I)$

【０１２１】

ここで順伝搬は、行列W_1×ベクトル１、次いで関数マップ（F_1（ｘ）。この場合F_1（ｘ）＝ｔａｎｈ）によりベクトルＨを算出する。このニューラルネットワークには２層あるため、W_2、Ｈ、およびF_2（Ｈ）（この場合F_2（ｘ）＝シグモイド）について同じ演算が繰り返されて、Ｏが得られる。同じようにしてより多くの層を実装することができる。W_1およびW_2は、ニューラルネットのレベルごとの重みの行列、Ｉは入力ベクトル、Ｈは隠れ層ベクトル、およびＯは出力ベクトルである。また、閾値も一般に隠れユニット（ｕｎｉｔ）および出力ユニットのそれぞれに追加される。閾値は、その値が常に１であるユニットを入力および隠れ層に追加することによってエミュレートすることができる。一定ユニットを他のユニットに接続している重みは、実質上こうしたユニットの閾値である。層ＩおよびＨが一定ユニットによって増大した場合、上記の式は、隠れ層および出力層ごとに閾値を正しく実施する。閾値パラメータは、W_1およびW_2に埋め込まれる。誤差計算は、ターゲットベクトルと、Ｆ（ｘ）関数の導関数Ｄ（ｘ）を掛けたニューラルネットワークからの出力ベクトルとの差である。Ｆ（ｘ）および対応するＤ（ｘ）の

式は次のとおりである。

【０１２２】

【表３】

	関数	導関数
シグモイド	$F(x)=\frac{1}{1+e^{-x}}$	$D(x)\frac{e^{-x}}{(1+e^{-x})^2}=F(x)*(1-F(x))$
Tanh	$F(x)=\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	$D(x)=\frac{4}{(e^{x}+e^{-x})^2}=(1+F(x))*(1-F(x))$

【０１２３】

シェーダ言語は、成分ごとにベクトルに適用でき、またはシェーダの観点からいえば、ピクセルごとに適用することができる指数関数を有することに留意されたい。

【０１２４】

重みの更新は、その層の誤差ベクトルおよび入力ベクトルの外積を実行することによって実行された。結果として得られた行列は、学習係数（ｌｅａｎｉｎｇ　ｃｏｅｆｆｉｃｉｅｎｔ）αによってスケールされ、既存の重み行列に追加された。誤差をネットワークに伝搬するために、別の行列ベクトル乗算が計算された。つまり、誤差ベクトル付きの重み行列の転置が計算され、再度Ｄ（ｘ）関数によってスケールされた。

【０１２５】

まとめると、機械学習ＧＰＵ実行方法を使用してＧＰＵで次の演算が実施された。

１．行列＊ベクトル－＞ベクトル（内積）

２．ベクトルの要素ごとにｘ＝ｆ（ｘ）（この場合ｆはｔａｎｈまたはロジスティック関数、またはそれらの導関数）

３．ベクトルＡ－ベクトルＢ－＞ベクトルＣ

４．ベクトル＊ベクトル－＞行列（外積）

５．行列Ａ＋定数＊行列Ｂ－＞行列Ａ

６．行列の置換

７．ＧＰＵメモリ内の新しい場所へのコピー（行列またはベクトル）

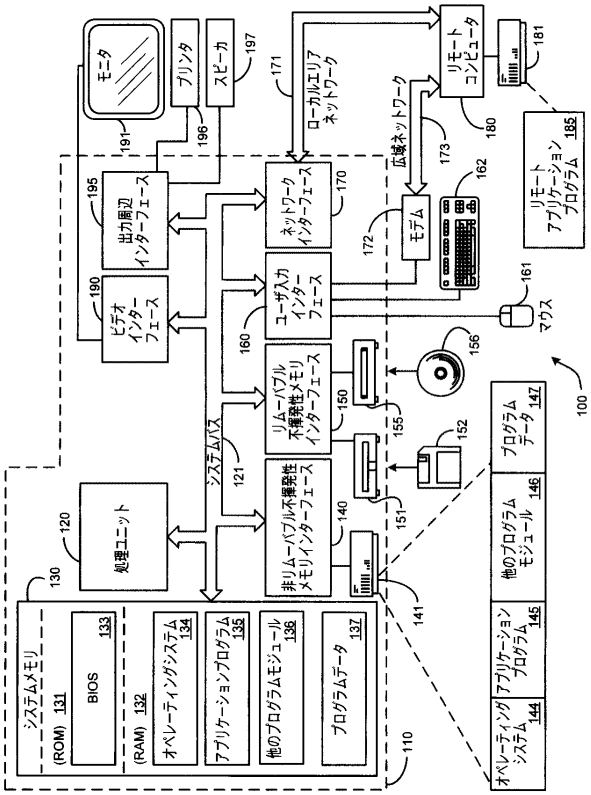
８．ＣＰＵメモリからＧＰＵメモリへの、またはその逆のコピー（行列またはベクトル）（これはＤｉｒｅｃｔＸによって直接サポートされ、完全を期すためのみに言及される）

【０１２６】

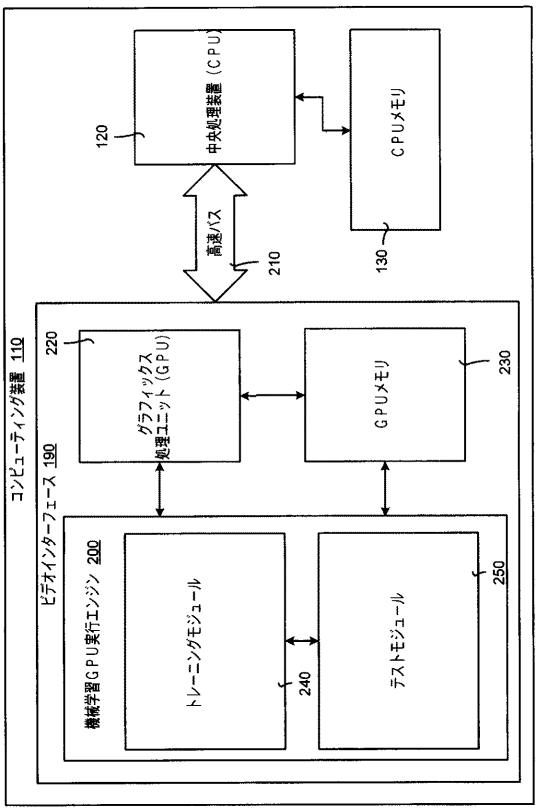
これらの演算のそれぞれは１回のパスで行われ、時として上述した制限によって単一のパスで計算できないときには、複数のパスで行われる。ＧＰＵでの実際の実行は、パスのリストが前処理で作成されることを必要とする。このリストは、手動または自動で作成することができる。さらに、個々の関数がコンパイルされ、ＧＰＵにダウンロードされ（これもまた前処理で）、入力値およびトレーニング値をグラフィックスメモリの作業領域にダウンロードし、三角形およびビューポイントをダウンロードして、関数をパスごとに指定する必要がある。こうした各ステップは、Ｄｉｒｅｃｔ３Ｄ（Ｄ３Ｄ）グラフィックスＡＰＩへの呼出を介して実施された。三角形およびビューポイントのダウンロードは、パスの実行を開始することに留意されたい。上述したように、ＧＰＵでのクリップされた三角形のレンダリングは、黙示的なピクセルごとのＤＯループを構成する。パスごとの特定のフラグメントシェーディングプロセス（ｆｒａｇｍｅｎｔ　ｓｈａｄｉｎｇ　ｐｒｏｃｅｓｓ）が各多角形の特性としてコード化された。反復型のトレーニングでは、反復ごとにこのシーケンスが繰り返される。プログラマが進捗を監視したい場合を除いて、反復を通じてグラフィックスメモリからＣＰＵにデータを読み戻す必要はない。グラフィックスメモリからホストメモリへの転送は、現在のハードウェアでは遅く、プログラマは一般にこうした操作を避ける。

【０１２７】
本発明の上記の説明は、例示および説明のために提示したものである。網羅的ではなく、また本発明を開示した正確な形状に限定するためのものでもない。上記の教示に鑑みて、多くの変更および変形が可能である。本発明の範囲は、本発明のこの詳細な説明によって限定されるものではなく、本明細書に添付した特許請求の範囲によって限定されるものである。
【図面の簡単な説明】
【０１２８】
【図１】機械学習ＧＰＵ実行エンジンおよび方法を実装できる適したコンピューティングシステム環境の例を示す図である。
【図２】本発明の一実施形態に係る機械学習ＧＰＵ実行エンジンおよび方法の実装例を示すブロック図である。
【図３】一例として勾配降下アルゴリズムを使用した３層学習機械のトレーニングモジュールを示すブロック／フロー図である。
【図４】図３で示した３層学習機械のテストモジュールを示すブロック／フロー図である。
【図５】機械学習ＧＰＵ実行方法のトレーニング段階の動作の概要を示すフロー図である。
【図６】機械学習ＧＰＵ実行方法のテスト段階の動作の概要を示すフロー図である。
【図７】ニューラルネットワークでの１対の層を示す詳細図である。
【図８】機械学習ＧＰＵ実行方法によって使用される矩形メモリ技術を示す図である。
【図９】機械学習ＧＰＵ実行方法のＧＰＵメモリ構成を示す図である。
【図１０】テクスチャ三角形（ｔｅｘｔｕｒｅ　ｔｒｉａｎｇｌｅ）を使用したテクスチャマッピングの使用を示す図である。
【図１１】内積を示すブロック図である。
【符号の説明】
【０１２９】
１１０　コンピューティング装置
１２０　中央処理装置（ＣＰＵ）
１３０　ＣＰＵメモリ
１９０　ビデオインターフェース
２００　機械学習ＧＰＵ実行エンジン
２１０　高速バス
２２０　グラフィックス処理ユニット（ＧＰＵ）
２３０　ＧＰＵメモリ
２４０　トレーニングモジュール
２５０　テストモジュール

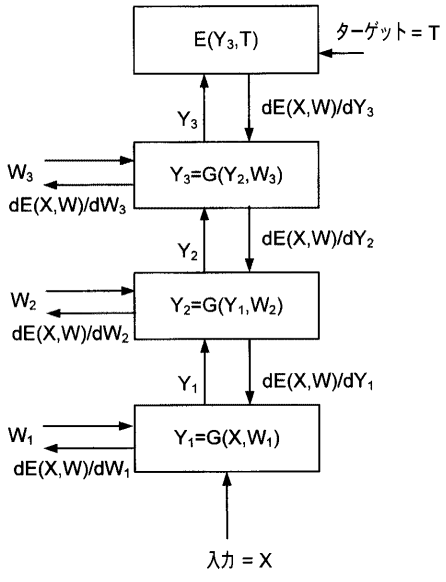
【図１】


モニタ
プリンタ
スピーカ
ローカルエリアネットワーク
リモートコンピュータ
広域ネットワーク
リモートアプリケーションプログラム
出力周辺インターフェース
ネットワークインターフェース
モデム
ビデオインターフェース
ユーザ入力インターフェース
マウス
リムーバブル不揮発性メモリインターフェース
システムバス
処理ユニット
非リムーバブル不揮発性メモリインターフェース
システムメモリ
(ROM)
(RAM)
BIOS
オペレーティングシステム
アプリケーションプログラム
他のプログラムモジュール
プログラムデータ
プログラムデータ
他のプログラムモジュール
アプリケーションプログラム
オペレーティングシステム

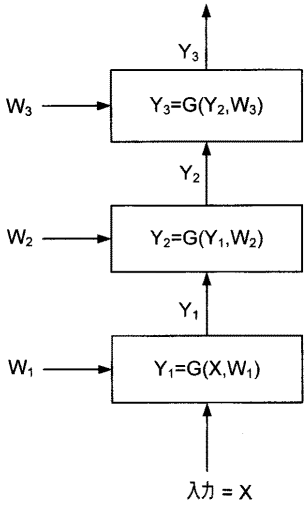
【図２】


中央処理装置（CPU）
CPUメモリ
高速バス
コンピューティング装置
グラフィックス処理ユニット（GPU）
GPUメモリ
ビデオインターフェース
機械学習GPU実行エンジン
トレーニングモジュール
テストモジュール

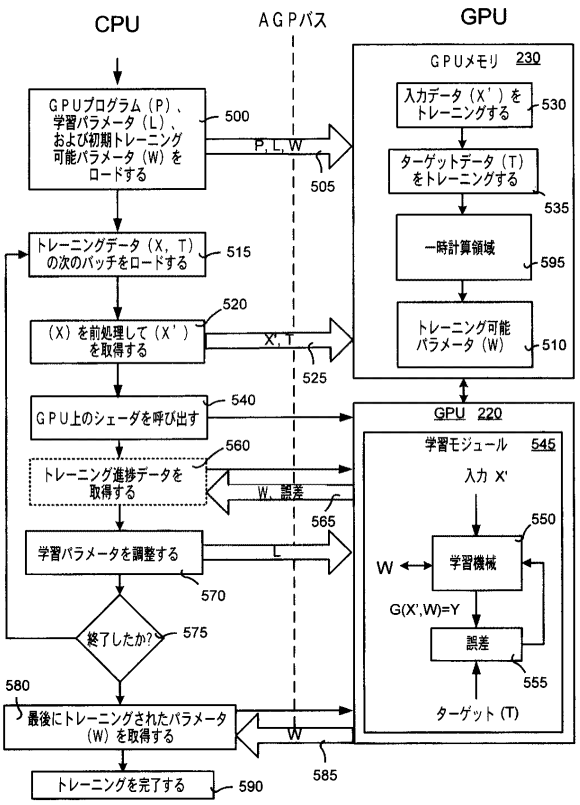
【図３】


E(Y3,T)
ターゲット = T
dE(X,W)/dY3
Y3=G(Y2,W3)
dE(X,W)/dW3
dE(X,W)/dY2
Y2=G(Y1,W2)
dE(X,W)/dW2
dE(X,W)/dY1
Y1=G(X,W1)
dE(X,W)/dW1
入力 = X

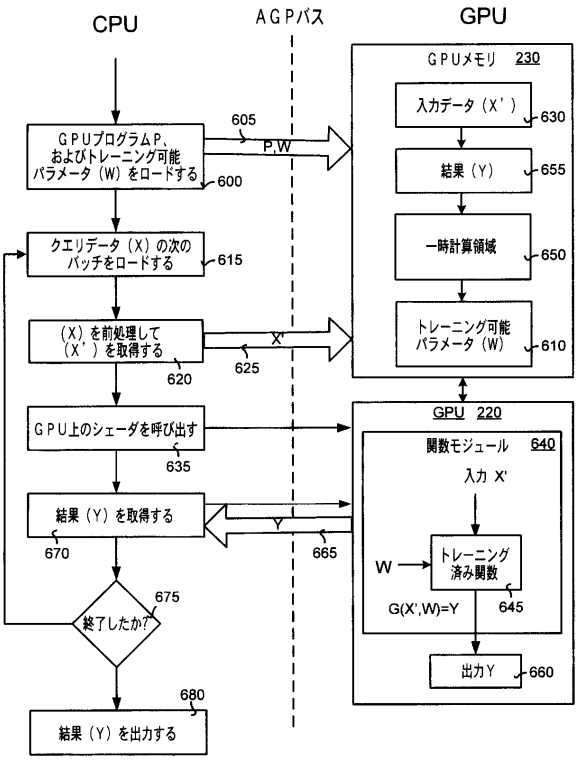
【図４】


Y3=G(Y2,W3)
Y2=G(Y1,W2)
Y1=G(X,W1)
入力 = X

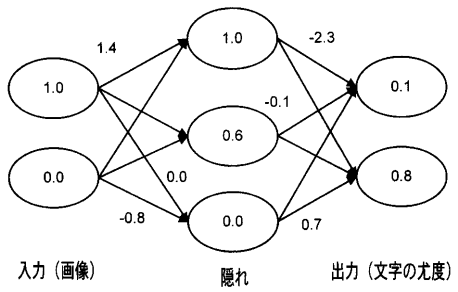
【図５】


CPU
AGPバス
GPU
GPUメモリ 230
GPUプログラム（P）、学習パラメータ（L）、および初期トレーニング可能パラメータ（W）をロードする
500
P, L, W
505
入力データ（X'）をトレーニングする
530
ターゲットデータ（T）をトレーニングする
535
一時計算領域
595
トレーニング可能パラメータ（W）
510
トレーニングデータ（X, T）の次のバッチをロードする
515
（X）を前処理して（X'）を取得する
520
X', T
525
GPU上のシェーダを呼び出す
540
GPU 220
学習モジュール 545
入力 X'
トレーニング進捗データを取得する
560
W, 誤差
565
学習パラメータを調整する
L
570
W
550
学習機械
G(X',W)=Y
誤差
555
ターゲット（T）
終了したか?
575
580
最後にトレーニングされたパラメータ（W）を取得する
W
585
トレーニングを完了する
590

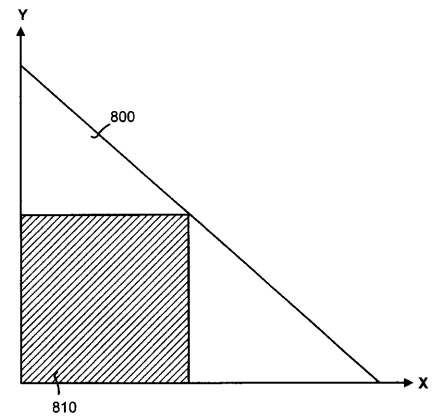
【図６】


CPU
AGPバス
GPU
GPUメモリ 230
GPUプログラムP、およびトレーニング可能パラメータ（W）をロードする
605
P,W
600
入力データ（X'）
630
結果（Y）
655
一時計算領域
650
トレーニング可能パラメータ（W）
610
クエリデータ（X）の次のバッチをロードする
615
（X）を前処理して（X'）を取得する
X'
620
625
GPU上のシェーダを呼び出す
635
GPU 220
関数モジュール 640
入力 X'
結果（Y）を取得する
Y
670
665
W
トレーニング済み関数
G(X',W)=Y
645
終了したか?
675
出力Y
660
680
結果（Y）を出力する

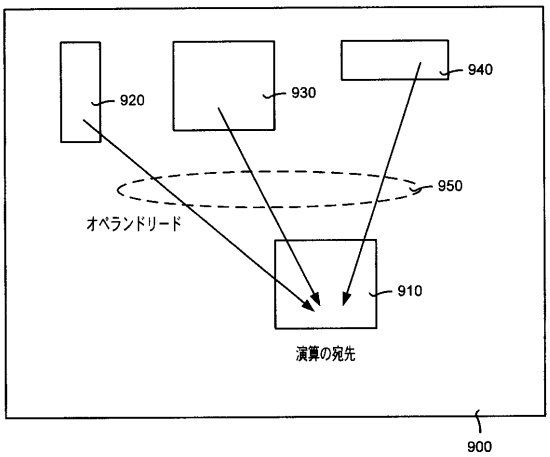
【図７】


1.4
1.0
-2.3
1.0
-0.1
0.1
0.6
0.0
0.0
0.8
-0.8
0.0
0.7
入力（画像）
隠れ
出力（文字の尤度）

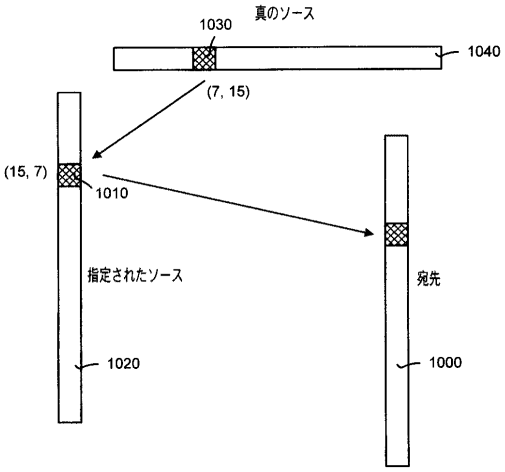
【図８】


Y
800
X
810

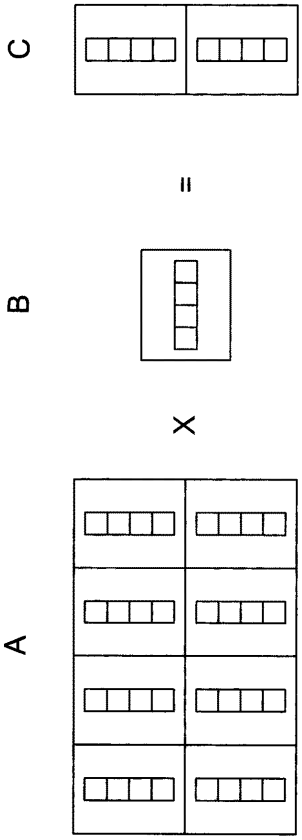
【図９】


920
930
940
950
オペランドリード
910
演算の宛先
900

【図１０】



【図１１】



フロントページの続き

(72)発明者 イアン エー. バック
アメリカ合衆国 ９８０５２ ワシントン州 レッドモンド ワン マイクロソフト ウェイ マイクロソフト コーポレーション内
(72)発明者 パトリス ワイ. シマード
アメリカ合衆国 ９８０５２ ワシントン州 レッドモンド ワン マイクロソフト ウェイ マイクロソフト コーポレーション内

審査官 長谷川 篤男

(56)参考文献 Optical Character Recognition on Graphics Hardware, integrative paper, UNCCH, Department of Computer Science., ２００２年１１月, 第１－９頁, ＵＲＬ, http://www.cs.unc.edu/~adyilie/IP/Final.pdf
Christian-A. Bohn, Kohonen Feature Mapping through Graphics Hardware, Proceedings of Int. Conf. on Comp. Intelligence and Neurosciences, １９９８年, 第1-4頁, ＵＲＬ, http://www.fh-wedel.de/~bo/db/pubs/bohn_KFMGPU1998.pdf

(58)調査した分野(Int.Cl., ＤＢ名)
Ｇ０６Ｎ ３／００－３／０６３
ＪＳＴＰｌｕｓ（ＪＤｒｅａｍＩＩ）
ＩＥＥＥ Ｘｐｌｏｒｅ