連載解説「Deep Learning（深層学習）」〔第3回〕

大規模 Deep Learning（深層学習）の実現技術

Practical Techniques for Large-scale Deep Learning

岡野原　大輔　株式会社 Preferred Infrastructure
Daisuke Okanohara　Preferred Infrastructure Inc.
hillbig@preferred.jp

Keywords: deep learning, machine learning, distributed systems.

1. はじめに

深層学習は近年多くのコンペティションやタスクで著しい成果を上げている．また，深層学習は，アカデミックよりも産業界で先に利用が急速に広がっており，Microsoft が音声認識への適応を進め，Google も画像認識など，従来手法から深層学習を利用した手法に次々と置き換えている．これらを支えているのが大規模な深層学習を実現する技術である．深層学習のニューラルネットワーク（以下，ニューラルネット）はただ階層が深いだけでなく，対象データ数，モデルサイズが非常に大きいのが特徴である．数千万の学習データに対し，数十億のパラメータを備えたニューラルネットを学習させるという，これまでのニューラルネットと比べ，桁違いに大きな学習が実現されている．ほかの機械学習手法と同様に，データやモデルを大規模にすることで精度は著しく向上し，これまで不可能だったタスクが実現できるようになった．量が質を変えたといえる．今度も，大規模なニューラルネットの実現が人工知能実現の大きな鍵であると考えられる．

一口に大規模な深層学習を実現するといってもさまざまな手段がある．Google のチーム [Le 12a, Le 12b] は最先端の分散並列技術をニューラルネットの学習システムに適用し，数千のマシンからなる大規模な深層学習システムを初めて実現した．こうした学習の結果，画像からの教師無学習で，人や猫といった一般物体認識を実現できることを示した [Le 12a]．また，GPU（Graphic Processor Unit）といった並列計算が得意なハードウェアを利用する研究も進められている．多くのコンペティションで優勝した Krizhevsky らのチームは，GPU による効率的な最適化をいち早く実現し，通常の PC クラスタを利用した場合と比べて大規模な学習データを利用可能にした [Krizhevsky 12]．また，今では GPU を用いたとしても1台で処理できるサイズを大きく超えはじめ，複数台のマシンでいかに協調して分散並列処理を実現するかが肝となっている．非常に高速なクラスタ間通信を可能とする Infiniband と GPU を組み合わせて，数十台のマシンで最大規模の深層学習が実現されている [Coates 13]．

このような大規模な深層学習が実現されつつある一方，深層学習が大規模になるにつれ，学習時の最適化や，過学習制御はより難しくなり，小さなデータセットではうまくいっていた最適化法を適用しただけではうまく学習ができない場合が多くなっている．そのため，新しい活性化関数や，過学習制御の手法が登場している．また，ハイパーパラメータの数は人手で制御できる限度を超え，ハイパーパラメータをいかにうまく自動探索，調整するかが重要となる．

実装自体も複雑化し，難易度が高くなっている．深層学習は，各構成部品は単純であるが，全体では複雑な挙動をするため，デバッグや，正常動作の確認は困難である．大規模な深層学習ではなおさらである．機械学習は，ただでさえデバッグが難しいが（うまく動いていなくても，問題が見つかりにくい），深層学習は特に実装が面倒であるので，デバッグや，正常な動作かを確認する方法についてもさまざまな研究が進んでいる．

2. 深層学習の最適化について

大規模な深層学習について述べる前に，深層学習ではどのような処理をするかについて簡単に説明をする．

2・1 深層学習の基本計算

はじめに，深層学習で利用されるネットワーク構造，特に階層型ニューラルネットについて説明をする．内容については第2回 [麻生 13] の3章についても参考にされたい．それぞれのニューロンは多次元の入力 $\boldsymbol{x}=[x_1, \cdots, x_n]$ を受け取り，y を出力するような処理を行う．この処理は具体的には次のような式をとる．

$$y=h\left(\sum_{i=0}^{n} w_i x_i\right)=h(\boldsymbol{w}^T\boldsymbol{x}) \quad (1)$$

ただし，$\boldsymbol{w}=[w_0, \cdots, w_n]$ は各入力の重みである．また x_0 は常に1であり，w_0 はバイアス項，またはしきい値と呼ばれる値である．関数 h は一般に非線形の関数であり，ニューロンの出力関数，または活性化関数と呼ばれ

る．

深層学習では，このようなニューロンが層状に並んでおり，k層目の出力が$k+1$層目の入力となる．例えば，ある層の入力ベクトルが$\boldsymbol{x}$，重み行列がW，活性化関数がhのとき，出力ベクトルは$h(W\boldsymbol{x})$で表される，ただし，ベクトル$\boldsymbol{y}$に対し，$h(\boldsymbol{y})=[h(y_1),h(y_2),\cdots]^T$である．この処理を層ごとに行い，入力から最終的な出力を決定する．

具体的には，パラメータθで特徴付けられた目的関数$f(\theta)$が与えられたとき，これを最小にする$\theta^*=\arg\min_\theta f(\theta)$を求めることで最適化を行う．

この最適化に，最急降下法を利用する場合，現在のパラメータθにおける勾配$v=\partial f(\theta)/\partial\theta$を求め，$\theta:=\theta-\tau v$と更新を行う，ただし$\tau>0$は学習率である．深層学習においてパラメータは，各ニューロンにおける重みである．各重みについての誤差に対する勾配をそれぞれ直接計算するのは計算量が大変大きいが，誤差逆伝播（Back Propagation）法を利用することで効率的に求めることができる．

誤差逆伝播法を説明する前に，チェインルールについて復習をしよう．例えば，$y=g(x), z=f(y)$という関数が与えられたとき，zに対するxの勾配はチェインルールで次のように求められる．

$$\frac{\partial z}{\partial x}=\frac{\partial z}{\partial y}\frac{\partial y}{\partial x} \tag{2}$$

直感的に説明すれば，xの変化量がΔxのとき，yの変化量は$\Delta y=(\partial y/\partial x)\Delta x$であり，$z$の変化量は$\Delta z=(\partial z/\partial y)\Delta y=((\partial z\cdot\partial y)/(\partial y\cdot\partial x))\Delta x$と表される．次に変数の依存関係が複雑になった場合についても考えてみる．

$$y_i=g_i(x),\quad z=f(y_1,y_2,\cdots,y_n) \tag{3}$$

の場合，

$$\frac{\partial z}{\partial x}=\sum_{i=1}^{n}\frac{\partial z}{\partial y_i}\frac{\partial y_i}{\partial x}$$

である．これも同様にxを小さく変化させた場合，各y_iの変化量は$\Delta y_i=(\partial y/\partial x)\Delta x$であり，$z$の変化量は，

$$\Delta z=\sum_{i=1}^{n}\frac{\partial z}{\partial y_i}\Delta y_i=\sum_{i=1}^{n}\frac{\partial z}{\partial y_i}\frac{\partial y_i}{\partial x}\Delta x$$

これを一般化し，変数の依存関係をグラフで表した場合に，閉路がない場合を考えてみる．階層型ニューラルネットがまさにこの例である．あるパラメータxの子孫として存在するパラメータzの勾配は，xの直接の子を$y_1,y_2,\cdots,y_n$としたとき，

$$\frac{\partial z}{\partial x}=\sum_{i=1}^{n}\frac{\partial z}{\partial y_i}\frac{\partial y_i}{\partial x}$$

である．誤差逆伝播法はこの関係を利用し，出力を計算した順とは逆順に，各パラメータの勾配を順に求める．このとき，計算に必要な情報はすべて前のステップで計算済みであるため，効率的に計算することができる．

2・2 確率的勾配降下法

最急降下法において，目的関数はすべての訓練事例を元に定義されているため，本来であれば，勾配計算時はすべての訓練事例を元に計算しなければならない．それに対し，確率的勾配降下法（SGD: Stochastic Gradient Descent）では少数の訓練事例を元に勾配を推定し，パラメータを更新する．これを訓練事例ごとに繰り返す．

SGDは訓練事例が冗長である場合に有効である．例えば，極端な例としてすべての訓練事例を複製してサイズが倍となった新しい訓練事例を考えた場合でも，両者から得られた勾配は等しい．そのため，SGDは少ない訓練事例を用いて勾配情報を求めることができる．また，最適解から大きく離れている場合にSGDは有効である．どの訓練事例を用いても，最適解に近い位置に到達することができるからである．

これらの特徴に加え，SGDでは学習率を適切に設定することによって，目的関数が非凸であり局所最小解が複数存在するような場合でも，そのような解から脱出することも期待できる．

訓練事例が少数であると，勾配の推定が不安定となるため，訓練事例を数十～100程度利用して勾配を求めて，これを利用して確率的勾配降下法を実現する方法が多い．これはミニバッチ法と呼ばれる．

3. 現在の分散並列環境

1台のマシンで処理できないサイズの問題を扱う場合，複数のマシンを利用して分散並列計算を実現する必要がある．マシン並列による分散並列計算の代表例がMapReduceである．MapReduceでは，計算処理をMapとReduceの計算の組合せで実現する．Mapでは入力データからキーKとバリューVのペア(K, V)を定義し，Reduce処理では各キーごとに集められたデータ$(K, \{V\})$に対し計算を定義する．例えば，単語の出現回数を数える場合には，Map処理ではキーは単語，バリューはその単語の出現回数を記録し，Reduce処理では各単語ごとにまとめられた出現回数の和を求めることで実現される．

MapReduceは広い範囲の計算処理を実現可能であり，自明な並列性があり協調処理が必要ない場合には単純に計算機を増やせば性能が向上する，すなわちスケールアウトできるというメリットがある．実際，多くの機械学習手法はMapReduceにより実現可能であることが示されている．しかしMapReduceはオーバヘッドが大きい．一つは，Reduce処理の前にばらまかれたキーを集めるShufflingと呼ばれるフェーズでマシンの全対全の通信が発生しネットワークに負荷が集中するためである．このときにすべてのマシンからすべてのマシンへの通信が発生し，レイテンシ，スループットともに非常に重くなる．もう一つはMapReduce処理では，処理途中（Reduce

直前）で同期をとる必要があり，深層学習の最適化のような逐次的な最適化を行う場合には同期処理のオーバヘッドが大きい．例えば，MapReduce で数十台で処理した場合と 1 台で最適化した処理では同程度の性能が出る場合もある．深層学習の場合，MapReduce とは異なる分散並列処理の実現が必要となる．

4. 分散並列：DistBelief の例

大規模なニューラルネットワークを分散並列処理で実現した例として，Google が開発した DistBelief [Le 12a, Le 12b] を紹介する．

4・1 これまでの分散手法の限界

これまでも，非同期分散による SGD の最適化の手法はいくつか提案されている．この際，異なるマシンからの更新で衝突が起きてしまわないようロックをとってしまうと，計算機を増やしても待ち時間が支配的となり性能が向上しなくなりスケールしなくなってしまう．そのため，ロックをとらずに更新を行う方法も提案されている [Feng Niu 11]．このような一見乱暴に見える最適化でも，特徴ベクトルおよび勾配が疎であるため，更新が衝突する確率は低く問題がなかったと報告されている．

しかし，深層学習による最適化では，特徴ベクトルおよび勾配ベクトルは密であり，同じアプローチはとりにくい．

4・2 DistBelief の概要

Google File System, MapReduce を開発した Jeff Dean らによって DistBelief [Le 12b] が提案された．このシステムはニューラルネット，層状のグラフィカルモデルの学習をサポートし，学習手法としては，オンライン学習とバッチ学習をサポートしている．

DistBelief では後述するような GPU を利用した高速化は目指さず一般的な計算ノードを並べ，大規模化を図る．

並列化といった場合，モデル並列化とデータ並列化の二つが存在する．モデル並列化では，一つのモデルを学習する際に並列化をして高速化を達成する．1 台のマシン内でのマルチスレッドによる並列化，および複数マシンによる並列化である．これに対し，データ並列化では，マスタのモデルからモデルのレプリカを作成し，教師事例を分割したうえで各モデルレプリカ上で勾配を求めることで並列化を実現する．マシン間のアクセスがボトルネックになるためモデルのレプリカは実体のコピーがつくられる．モデル並列化，データ並列化を併用し，分散並列環境下での高速化が行われる．

深層学習の学習においては，各訓練事例ごとに SGD を用いて更新を行うオンライン学習，およびすべての訓練事例を元に更新を行うバッチ学習がある．DistBelief はこの両者をサポートし，前者は Downpour SGD，後者は Sandblaster L-BFGS と呼ばれる．

ユーザは各ニューロンでどのような計算がされるのか，また上方 / 下方にどのような情報を伝播させるのかについて定義する．DistBelief のフレームワークはこれらの情報に基づいて，複数のマシンにまたがる場合にどのように計算を分割するか，情報を伝播するかについて自動的に調整を行う．分散処理にまつわる部分についてユーザは気にする必要はない．

4・3 Downpour SGD

SGD では，すべての訓練事例から勾配を求めずに，ランダムに選ばれた少数（1 ～ 100 程度）の訓練事例を元に勾配を計算し，その勾配を元に最適化を行う．例えば，一つの事例のみから勾配を計算し，その勾配情報を元にパラメータの更新を行う．SGD はいくつかの条件がそろった場合に高速に解に収束する．例えば，訓練事例が冗長である場合，現在のパラメータが最適解から十分離れている場合などである．

この SGD は逐次的な更新を行うため，分散並列化が困難であった．Downpour SGD では，勾配計算，更新計算でモデル並列化を利用し，さらにモデルレプリカを利用しさらなる高速化を図る．システムはパラメータサーバを管理し，全体のパラメータのマスタデータを管理する．学習データはいくつかのグループに分割され，各モデルレプリカ上でデータのサブセットを利用して勾配を計算し，勾配情報をパラメータサーバに送り，更新処理を行う．各モデルレプリカは，常にパラメータサーバに問い合わせて最新のパラメータ情報を得る．

Downpour SGD では，いずれかのモデルレプリカが故障した場合は，ほかのモデルレプリカが更新をし続けるため全体が止まることはない．またパラメータサーバはデータを分割して保持し，耐故障性を達成する．

4・4 Sandblaster L-BFGS

Sandblaster L-BFGS は，各マシンに勾配計算の一部を任せ，これらの勾配情報を集約したうえでまとめて更新を行う．勾配を元に更新を行う L-BFGS（Limited-memory Broyden-Fletcher-Goldfarb-Shanno）はメモリ限定の擬ニュートン法であり，これまでも機械学習の分野で多く利用されてきた．

こうしたバッチ処理において高速化率を制限するのは，異なるマシン間での計算時間のばらつきである．多くのマシンを動かした場合，ハードウェア障害などにより，いくつかのマシンが非常に遅くなる場合があり，それが全体の足を引っ張ることが多くなる．この障害に対応するため，勾配計算は小さいタスクに分割し，これらのタスクを各ノードに提供し，早く終わったノードがあれば次のタスクを割り当てていくことで遅いマシンに足を引

っ張られないようにする．こうした考え方はMapReduceなどでも利用されており，大規模分散システムにおいては不可欠の考え方となっている．大規模分散システムにおいては耐故障性が最重要課題であり，基本的にノードやネットワークは故障し停止もしくは計算時間が非常に長くなり得るものと考えてシステムを構成する．

4·5 実験結果

DistBeliefの評価は音声認識と画像認識のタスクで行われた．音声認識は密なニューラルネットを利用し，画像認識は畳込みニューラルネットという疎なニューラルネットを利用した．一般に密なニューラルネットであるほど，マシンをまたがる通信が増えるため，分散化による高速化が難しい．

音声認識のタスクに用いたニューラルネットは5層，各隠れ層の活性関数がsigmoidである2560個のノード，出力層が8192個のノードから構成される．各層間のノードはすべてつながっており，約4200万個のパラメータからなる．

実験の結果，1台当たり20コアを使ったうえで，8台のマシンを利用したとき，1台のときに比べて2.2倍の高速化を達成した．マシンをこれ以上増やしてもオーバヘッドが大きくなったため，高速化は見られなかった．

その一方で，画像認識のタスクでは，大きく3層からなり，各層の内部はフィルタリング，プーリング，局所正規化からなる．フィルタリングは10×10のパッチに対応する．出力層は21000個の1 vs allで表現した多クラス分類用のロジスティック回帰が使われた．

画像認識では81台のマシンを利用したときに12倍以上の高速化が得られた．これより多くのマシンによる高速化率は限定的であり，ネットワーク間のアクセス速度がボトルネックになっている．

また，YouTubeから得られた200×200の画像1000万枚に対して4000ノード，16000コアでの並列学習を行った．これほど大規模のニューラルネットに対しての学習は初めてである．学習した結果，教師なし学習のみからでも人の姿や顔，猫に反応できるニューロンを学習することができることが報告された．

いまだ，ノード台数に対するスケーラビリティは高くなく，異なるノード間の通信コストのボトルネックが大きい．

今後Jubatusのように学習自体は完全に独立に行ったうえで，学習モデルを後で混ぜる方法や，後述のMaxOutのような疎な勾配を活用することで高速化率を高めることが考えられる．

4·6 GPUを用いた大規模ニューラルネットの実現

もう一方の大規模化の実現手法がGPUを用いた高速化である．GPUは元々コンピュータグラフィクス描画専用装置であったが，現在GPUの仕事はそれにとどまらず汎用化している．GPUとCPUの違いは，GPUは周波数自体はCPUよりも低いがコア数が桁違いに多く，並列性をもった計算処理が得意である．現在のCPUもマルチコア化が進んでいるが，GPUを用いれば数百個程度の演算ユニットを利用でき並列演算性能が圧倒的に高い．そして，GPUはCPUに比べてレジスタ数が圧倒的に多いため，膨大な数のスレッドをレジスタを退避せずに処理することが可能である．また，GPUは，専用プロセッサに比べて，パソコンの部品として量産されているため安価でありコストパフォーマンスも高い．例えば，2013年時点で，CPU（Intel Core i7 Haswell）の性能は224 GFLOPS[*1]（16 FLOPS/Clock × 3.5 GHz × 4コア）であるのに対し，GPU（NVIDIA GeForce GTX 680）の性能は3.0 TFLOPS（2FLOPS/Clock × 1 GHz × 1536コア）であり，約10倍の差がある．

GPUを使う際に気をつけなければならないのは，GPUはCPUとは別のもう1台のコンピュータが載っているのと同様なのでメモリ情報などは共有されない．また，1台のGPUでは2.5億パラメータ（1 GByte）程度しか扱えず，これより大きいニューラルネットを扱う場合には複数のGPUを使わなければならない．しかし，I/O，電力，熱，CPUの計算速度などの問題から計算機1台につきGPUは四つ程度しか利用できないうえに，PCI Expressなどを介しての通信（8～16 GByte/s）は，GPU内のメモリとの転送速度（100 GByte/s）に比べて圧倒的に遅い．また，複数のマシンを利用して処理を行おうとした場合，通常のイーサネットなどを利用した通信はスループット，レイテンシともに性能が不十分であり，この通信部分がボトルネックになってしまう．例えば，GPUを用いれば，例えば一つの訓練事例からの学習は数ミリ秒で終わるのに対し，計算機間の通信を通常のイーサネットなどで行った場合，数秒のオーダが必要であり，通信がボトルネックとなってしまう．

GPUを利用した深層学習の高速化はこれまで多く試みられてきた[Raina 09]．GPUとメインメモリ間の転送速度は遅いが，現在のGPUでは，ホストマシンのメモリを介さずに直接GPU間でデータをやり取りすることが可能であり，1台のGPUでは収まらない大きなモデルを対象に学習を実現することができる．

[Krizhevsky 12]らは，層ごとにGPU並列にした．例えば3層目の入力は2層目の出力を受け取るが，4層目は3層目の結果で同じGPU中にある出力だけを受け取る．こうした最適化は，高速化にはあまり寄与しなかったが，精度向上を達成できた．

ここでは，Ngらにより実現されたGPUを用いた深層学習の実現例[Coates 13]を紹介する．彼らはこうしたGPUの問題を解決するために，InfiniBandを積極的に利用した．InfiniBandではマシン間を最大56 Gbps

*1 FLOPSは1秒間当たり浮動小数点演算を何回実行できるかという性能値．

(bits per second) でつなぎ，レイテンシはマイクロ秒以下である．

また，マシン間の並列計算には MVAPICH2 と呼ばれる MPI (Message Passing Interface) を利用し，GPU 中にあるメモリ間の通信は関数中の引数として渡せば実現できるなど非常に簡易に実現できる．

彼らはまた実装面でも工夫を行った．GPU は疎な行列間の乗算処理などはうまく処理することができず，性能が CPU 並みに落ちてしまう問題が報告されていた．これに対し，ニューラルネットにおける行列中の規則性に着目し，疎な行列から密な部分行列を抜き出し，密な行列同士の演算に変換することで GPU を用いた高速化を実現している．

彼らの実験では YouTube から抽出された 200x200 の画像，1 000 万枚を対象に学習を行った．16 台の計算機からなり，各計算機には二つの quad core プロセッサと四つの NVIDIA GTX 680 GPUs，そして InfiniBand adapter が使われた．学習には 110 億個のパラメータからなるニューラルネットが利用された．これは 4・5 章であげた例よりも 6.5 倍も大きい．ミニバッチ 1 回（訓練事例である画像は 96 枚）当たりの学習時間は 0.6 秒未満，1 000 万枚の画像に対しては 17 時間程度で学習が実現できることが示された．

5. 精度向上のための技術

大規模なニューラルネットではより精緻な学習の制御が必要となる．本章では，精度向上のためのさまざまな技術を紹介する．

5・1 DropOut

ニューラルネットは表現力が高いモデルなので過学習しやすいという問題がある．この問題に対し，これまで入力にノイズを与えたうえで学習する方法や，学習時に正則化を適用する方法などが提案されてきた．

その中でも，DropOut [Hinton 12] は過学習を防ぐ強力な学習方法である．Hinton は近年のニューラルネットの成功の主要因の一つに DropOut をあげている．

DropOut は複数のモデルを学習し，これらの予測結果の幾何平均を利用することで過学習を防ぐ．これまでも，コミッティモデルなど異なるモデルで学習した結果を複数保持し，推論を行う際にはこれらの結果の平均を利用して推論する方法が提案されている．DropOut はこうした手法と同様に複数モデルの推定の平均を利用することで過学習を防ぐことができる．

しかし，複数のモデルを利用するのは，学習，予測ともに計算コストが大きくなってしまうという問題がある．DropOut は毎回ランダムに半数のニューロンを消した状態で学習を行う．図 1 に DropOut の実行例を示す．例えばニューロンが $\{1, 2, 3, \cdots, k\}$ とあるならば，

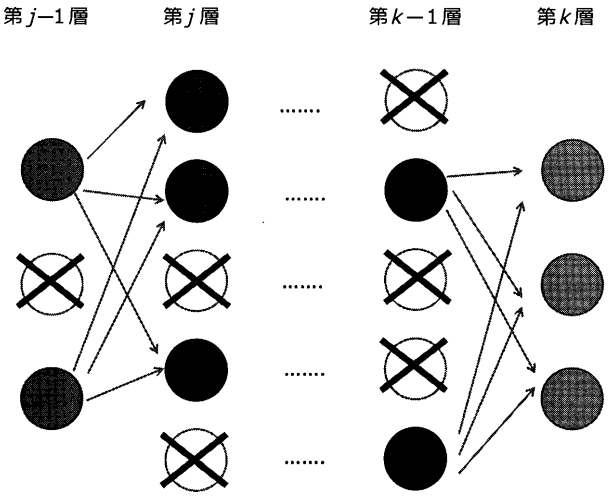

図 1　DropOut の例

最初の訓練事例には $\{2, 5, 6, 7, 9, \cdots\}$，二つ目には $\{1, 2, 4, 6, 8, \cdots\}$ といったようにである．これは，別の言い方をすれば，H 個のニューロンが存在するとき，DropOut の学習時には 2^H 個のモデルからサンプリングを行い，各訓練事例ごとに，違うニューラルネットを利用して学習，各モデルは 1 個だけの訓練事例を受け取って学習をしている．これにより，もし有効な隠れ情報があり，それを担っていたニューロンが DropOut された場合には，ほかのノードがその役割を担って学習するようになる．

学習が終わったら，各モデルの予測結果の幾何平均を最終的な予測結果とするが，学習時のすべての学習結果を保存し，それらの幾何平均を計算するのは計算コストが非常に大きい．ここにも DropOut の巧妙なトリックが使われる．DropOut では，推論時には各ニューロンからの出力を 1/2 にしたうえで推論を行う．実はこれだけで各学習時のモデルからの推論をしたうえで幾何平均をとった場合の近似解が得られる．

これについて以下で直感的な説明を行う．ニューラルネットが次のような 1 層で，活性化関数がソフトマックス関数である場合を考える

$$h(\boldsymbol{x}, \boldsymbol{w}) = \exp(\boldsymbol{w}^T \boldsymbol{x}) \Big/ \sum_{\boldsymbol{x}'} \exp(\boldsymbol{w}^T \boldsymbol{x}') \tag{4}$$

$$\propto \exp(\boldsymbol{w}^T \boldsymbol{x}) \tag{5}$$

なお，バイアス項は $x_0 = 1$ として含まれているとする．また，長さ m の重みベクトル $\boldsymbol{w}$ の各要素を個々に 0 にした，全部で 2^m 個の重みベクトルの集合を W_{do} とする．W_{do} に対する幾何平均は次に比例する．

$$\left(\prod_{\tilde{\boldsymbol{w}} \in W_{do}} \exp(\tilde{\boldsymbol{w}}^T \boldsymbol{x}) \right)^{1/|W_{do}|} \tag{6}$$

この対数をとると

$$(1/|W_{do}|)\sum_{\tilde{\boldsymbol{w}}\in W_{do}}\tilde{\boldsymbol{w}}^T\boldsymbol{x} \tag{7}$$

$$=(1/|W_{do}|)\left(\sum_{\tilde{\boldsymbol{w}}\in W_{do}}\tilde{\boldsymbol{w}}^T\right)\boldsymbol{x} \tag{8}$$

$$=(\boldsymbol{w}/2)^T\boldsymbol{x} \tag{9}$$

であるので，各重みをちょうど1/2とした場合に一致する．

このように，一層の場合は各重みを1/2にしたようなモデルが幾何平均の近似として扱える．また，ニューラルネットが多層で，活性化関数がソフトマックス関数でない場合でも，正確な平均ではないが近い結果が期待される．DropOutは推定時には複数の学習結果の幾何平均を推論に使うことが可能である．

5・2 MaxOut

DropOutは，1層で指数型分布族を利用した場合は正しい幾何平均を計算しているが，複数層の場合やほかの活性化関数を利用した場合には正しい近似とはなっていない．

これに対し，MaxOut[Goodfellow 13]は次のような活性化関数で利用する．

$$h(\boldsymbol{x})=\max z_i \tag{10}$$

$$z_i=\boldsymbol{w}_i^T\boldsymbol{x}+b_i \tag{11}$$

つまり，複数の線形関数の中での最大値を関数の値として採用する．別な言い方をすれば，活性化関数として区分線形な凸関数を利用し，この凸関数自体も学習する．図2にMaxOutの例を示す．

MaxOutを利用することにはさまざまなメリットがある．まず実装が単純であり，順伝播，誤差逆伝播がともに高速になる．次に，更新処理の際に勾配が疎になり（maxが効いているところだけ，勾配が伝播する），分散並列処理の際に書込みで重複が起きにくくなることや，更新時のHessian行列の非対角成分が小さくなるため学習が効率的になる点である．

最大のメリットはMaxOutはほとんどすべての点で線形関数でありDropOut時の近似は，正確な平均の結果とほぼ一致する．これにより，より高い汎化性能が得られる．

MaxOutを利用した分類は多くの分類結果で最高性能を達成しており，現時点で最も有望な学習手法である．

6. 高速化のための技術

6・1 現在の計算機の特性を活かした高速化

近年のコンピュータアーキテクチャの変化に合わせた高速化技術も進んできた．クロック周波数を上げることによる高速化は限界を迎え，コア数を増やす，もしくはノード数を増やすことによる高速化を目指している．もう一つの流れとしてメモリ階層間の速度差が増大するとともに，キャッシュミスのペナルティが大きくなっている．また，パイプラインが長くなるにつれて分岐予測ミスのペナルティも大きくなっている．例えば，キャッシュミスが1回発生するごとに，数百サイクル，分岐予測ミス1回当たり数十サイクルのペナルティが課される．そのため単純に演算回数を減らすのでなく，キャッシュミスや分岐予測ミスの少ない処理にすることが高速化の肝となっている．また，深層学習はうまく実装することでGPUを用いて画一的な並列処理で高速化をはかることができる．

また，これまでの機械学習において，特徴ベクトルが疎であることを活かすことが高速化の肝の一つであった．例えば自然言語処理においては単語wに対応する特徴ベクトルは，wに対応する次元の値だけが1，それ以外は0となるような，いわゆるone-hot representationで表現されてきた．こうした特徴ベクトルが疎であることを利用しない限り，大規模な特徴情報を利用することは困難であった．疎であることを利用して，例えば転置ファイルのような索引を構築し，これを利用して計算の高速化を行っていた．

例として，文書分類において，特徴ベクトルの次元数が100万と多かったとしても，実際に発火（成分が非零の要素）している特徴数は数十～数百であり，学習時，推論時には発火している特徴数に比例した計算量で処理を行うことができる．これにより秒当たり数千～数万といった文書を利用した学習・分類が可能だった．

一方，深層学習では各単語はランダムで密なベクトルに対応する，ただし次元数は50～200と小さい．こうした密なベクトルと行列間の演算を行うことで高速化を達成する．

現在のマルチコアCPUやGPUの特性を活かすためには密なベクトル/行列に対する，同様な並列計算をいかに増やせるかが重要となっている．こうした工夫による深層学習は従来の機械学習と比べても高精度でありながら同程度から数倍高速な推定が可能となっている．

例えば，先ほどの誤差逆伝播法における勾配情報は，ミニバッチにおいては行列として表され，密な行列同士の計算として高速に計算することができる．これによりベクトル計算を行列計算に置き換えることで効率的に実現できる．ミニバッチによる誤差逆伝播法による勾配計算は密な行列同士の計算として高速化できる．

6・2 ReLU

ニューラルネットのモデル自身を変えることでの高速化も図られている．従来のニューロンでは，

$$h_{\tanh}(x)=\tanh(x) \tag{12}$$

$$h_{\mathrm{sigmoid}}(x)=(1+\exp(-x))^{-1} \tag{13}$$

といった活性化関数が主に用いられていた．しかし，こうした関数は学習時，推論時に計算コストが大きいという問題があった．

それに対し，Rectified Linear Units（ReLUs）[Nair 10]では

$$h_{\mathrm{relu}}(x)=\log(1+\exp(x))\simeq\max(0,x) \quad (14)$$

という単純な関数を利用する．これにより，出力計算，勾配計算が高速にできるだけでなく，$\max(0,x)$ という関数の性質上，多くの値がちょうど 0 となり，出力が疎になるため，その部分での高速化できる．この工夫により数倍～ 10 倍近い高速化が図れる．

7. 実装時の技術

深層学習の実装は，パーツ自体は単純ではあるが全体の挙動は複雑であるため，正しく動作しているかのチェック，および正しく動作していない場合のデバッグは困難である．本章ではいかに実装するかについて述べる．このほか，実装時に考慮すべき点については Bengio による素晴らしいサーベイ資料がある [Bengio 12].

7·1 実装の正しさのチェック

誤差逆伝播法は実装が煩雑になりがちであり，多くの誤りが含まれやすい．本章では正しく実装されているかのチェック方法をいくつか紹介する．

まず，誤差逆伝播法で勾配が正しく計算されているのかを一次差分近似により求められた勾配と比較して確認する．目的関数 f に対する，パラメータ x の偏微分の一次差分近似は次のように求められる．

$$\frac{\partial f(x)}{\partial x}\simeq\frac{f(x+\varepsilon)-f(x)}{\varepsilon}+o(\varepsilon) \quad (15)$$

この式は次のように工夫することで誤差項を小さくすることができる．

$$\frac{f(x+\varepsilon)-f(x-\varepsilon)}{2\varepsilon}+o(\varepsilon^2) \quad (16)$$

勾配が正しく計算されていることが確認されれば，次に小さいデータセットで正しく訓練誤差が小さくできるかを確認する．ニューラルネットは表現力が高いので小さいデータセットであれば訓練誤差をいくらでも小さくできるはずである．もしできないのであれば学習が何かしら間違っていることになる．

また，各隠れ層のニューロンがどのような重み列に対応しているのかを分析することも有効である．各隠れ層のニューロンは，どの入力に一番強く反応するかを求めることで分析することができる．例えば畳込みニューラルネットを用いた画像認識の場合，各隠れ層の素子は，入力画像の画像パターンに対応する．自然言語のような場合でも，隠れ層に対応する入力ベクトルを主成分分析などで二次元にプロットすることで，似た単語同士が近くにあるかどうかといったチェックをすることができる．

7·2 ハイパーパラメータの最適化

ニューラルネットには多くのハイパーパラメータが存在し，これらの最適化が重要である．適切なハイパーパラメータが使われなかった場合，学習が全くできないことも起こり得る．例えば，SGD に利用される学習率一つをとったとしても，実際に利用する場合は次のような形をとる．

$$\eta_t=\frac{\eta_0\tau}{\max(t,\tau)} \quad (17)$$

ただし，t は反復回数である．このとき，η_0, τ といったハイパーパラメータは調整が必要である．また，ミニバッチサイズや訓練回数，隠れ層の数，正則化項の重み，活性化関数，重みの初期値といったものも学習の効率を左右するハイパーパラメータである．

こうしたハイパーパラメータの最適化には従来 Grid Search が利用されていた．Grid Search では各ハイパーパラメータごとに決められた候補群があり，それらを組み合わせて最適なハイパーパラメータの組合せを調べる．

これに対し，近年すべてのハイパーパラメータをランダムに設定し，一番良い組合せを採択する Random Search が優れていることが報告されている [Bergstra 12]. Grid Search と比べて，Random Search のほうがなぜ良いかというと，ハイパーパラメータのいくつかは影響が大きく，いくつかは変えても影響が少ないという，いわゆる low effective dimension であるからだと推察されている．

例えば，関数のパラメータが二つ $f(x,y)$ からなり，実際には x だけが関数値に関係があるとする（$f(x,y)=g(x)$ となるような g が存在）．このとき，Grid Search を利用して $x=0, 0.1, 0.2 \cdots, 1$, $y=0, 0.1, \cdots, 1$ と $11\times 11=121$ か所を調べた場合，x だけが有効であるため，11 か所の異なる結果しか調べることができない．これに対し，Random Search であればすべての x は異なるため，121 点の箇所を調べることができる．

これは一つの例だが，実際，多くの実験で同じ試行回数を試すのであれば Random Search のほうが効率良く調べることができることが報告されている．また，ほかの優れたハイパーパラメータ最適化の手法も，実装が面倒であり，Random Search で並列にたくさん試すほうが優れているとの報告がある．

また，SGD による最適化において，学習率の調整が特に重要であり，AdaGrad[Duchi 10] を利用して自動最適化を行う例が増えている [Le 12b]. AdaGrad においては各成分ごとの学習率を次のように設定する．

$$\tau_{i,K}=\gamma\Big/\sqrt{\sum_{j=1}^{K}\partial w_{i,j}^2} \quad (18)$$

ただし，$\partial w_{i,j}$ は i 番目の成分の j 回目の学習時の勾配である．γ はすべての学習率の計算で利用する定数であり，一般的な学習率よりもかなり大きめに設定されている．このほか，学習率の調整手法はいくつか提案されている [Schaul 12, Schaul 13].

8. ま と め

本稿では大規模な深層学習を実現するために必要なフレームワーク，学習手法，高速化手法，実装手法などを紹介した．深層学習はいまだ日進月歩での改良が進み，また利用が産業界で広がっていることもあり実用化の技術の共有はまだ進んでいない．今後，なぜうまくいったのかという解明が進むとともに，より大規模，強力な深層学習の実現が進み，各分野への応用が進むことが期待される．

◇ 参 考 文 献 ◇

[麻生 13] 麻生英樹：多層ニューラルネットワークによる深層表現の学習，人工知能学会誌，Vol. 28, No. 4, pp. 649-659（2013）

[Bengio 12] Bengio, Y.: Practical recommendations for gradient-based training of deep architectures, *Neural Networks: Tricks of the Trade*, pp. 437-478, Springer（2012）

[Bergstra 12] Bergstra, J. and Bengio,Y.: Random search for hyperparameter optimization, *J. Machine Learning Research*, Vol. 13, pp. 281-305（2012）

[Coates 13] Coates, A., Huval, B., Wang, T., Wu, D. J., Ng, A. Y. and Catanzaro, B.: Deep learning with COTS HPC systems, *Int. Conf. in Machine Learning*（2013）

[Duchi 10] Duchi, J., Hazan, E. and Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization, *J. Machine Learning Research*,Vol. 12, pp. 2121-2159（2010）

[Feng Niu 11] Feng Niu, B. R., Ré, C. and Wright, S. J.: Hogwild!: ALock-free approach to parallelizing stochastic gradient descent, *NIPS 2011*（2011）

[Goodfellow 13] Goodfellow, I., Warde-Farley, D., Mirza, M., Courville, A. and Bengio, Y.: Maxout networks, *Proc. 30th Int. Conf. on Machine Learning*, pp. 1319-1327（2013）

[Hinton 12] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R. R.: Improving neural networks by preventing co-adaptation of feature detectors, *arXiv preprint arXiv:1207.0580*（2012）

[Krizhevsky 12] Krizhevsky, A., Sutskever, I. and Hinton, G.: Imagenet classification with deep convolutional neural networks, *Advances in Neural Information Processing Systems*, Vol. 25, pp. 1106-1114（2012）

[Le 12a] Le, Q., Ranzato, M., Monga, R., Devin, M., Chen, K., Corrado, G., Dean, J. and Ng, A.: Building high-level features using large scale unsupervised learning, *Int. Conf. in Machine Learning*（2012）

[Le 12b] Le, Q., Ranzato, M., Monga, R., Devin, M., Chen, K., Corrado, G., Dean, J. and Ng, A.: Building high-level features using large scale unsupervised learning, *Int. Conf. in Machine Learning*（2012）

[Nair 10] Nair, V. and Hinton, G. E.: Rectified linear units improve restricted Boltzmann machines, *Proc. 27th Int. Conf. on Machine Learning*, pp. 807-814, Omnipress Madison, WI（2010）

[Raina 09] Raina, R., Madhavan, A. and Ng, A. Y. :Large-scale deep unsupervised learning using graphics processors, *Proc. 26th Annual Int. Conf. on Machine Learning*, Vol. 382, pp. 873-880, ACM（2009）

[Schaul 12] Schaul, T., Zhang, S. and LeCun, Y.: No more pesky learning rates, *arXiv preprint arXiv:1206.1106*（2012）

[Schaul 13] Schaul, T. and LeCun, Y.: Adaptive learning rates and parallelization for stochastic, sparse, non-smooth gradients, *arXiv preprint arXiv:1301.3764*（2013）

2013年7月8日 受理

著 者 紹 介

岡野原 大輔

2010年東京大学大学院情報理工学系研究科コンピュータ科学専攻博士課程修了，博士（情報理工学）．2006年に株式会社 Preferred Infrastructure を共同で創業，現在同取締役副社長．統計的自然言語処理，機械学習，簡潔データ構造，大規模データ処理などに興味をもつ．