Web 工学で応用するための Deep Learning 利用法と知見の体系化

2013 年度卒業論文

東京大学工学部システム創成学科知能社会システムコース
03120929 黒滝 紘生

指導教官: 松尾 豊 准教授

2013 年 2 月 日

概要

近年機械学習の分野において、Deep Learning と呼ばれるアルゴリズム群が優れた成果を納めている。Web 工学でも、Deep Learning を応用することによる発展が期待される。

しかし、Deep Learning は歴史の浅い発展途上の技術であり、どのアルゴリズムを定番とすれば良いのか、試行錯誤の段階にある。これは、各アルゴリズムの改良点が次々と見つかっていることに加え、各アルゴリズムの得手不得手や、学習性能が高くなる原理の詳細など、解明されていない部分が多いことが、主な原因である。

また、現在の Deep Learning 技術では、他のアルゴリズムに比べて学習にかかる時間が長いことが多く、ハードウェア性能が低いマシンでは、アルゴリズムを実用的な時間で実行すること自体が容易ではない。実行時間の長さをカバーするため、GPU を用いて演算をスピードアップさせる手法が確立されつつあるが、まだ広く一般に浸透した技法にまでは至っていないため、参入障壁の一つになっている。

このような原因により、Deep Learning 技術に関心を持っていても、まず実際の問題に Deep Learning を試行すること自体が困難であり、応用技術開発のハードルは更に高くなっている。アルゴリズムが開発途上で確定できていないため、公開されているライブラリも、現状では、開発用途や実験的なものが多くなってしまっている。そもそも有力なアルゴリズムに対応する実装が用意されていない場合や、問題に応じて自らアルゴリズムの細部を調整しなければならない場合もある。ライブラリが GPU 専用に書かれていることが徒となり、GPU を持っていないと実行自体ができなくなることも考えられる。例え実行するところまで到達できたとしても、提供されているのがアルゴリズムのダイジェスト版でしかなく、論文にて示されている精度を手元で再現することが出来ない場合も多い。標準と言える公開ライブラリが確立していない状況なので、Web 工学など応用分野に Deep Learning を適用したいと考えても、プログラム開発に長い時間がかかってしまい、開発における大きな障壁となっている。特に、国内での研究開発は遅れており、早急なキャッチアップが必要である。

このような現状を踏まえ、本研究では、Web 工学における応用を見据えつつ、Deep Learning を様々な問題に応用するための方法論を整理する。具体的な目標としては、Deep Learning の特徴である高い学習性能や分類精度を確実に利用できて、その上で出来る限り、実行時間の短さ、実行プログラムの使いやすさ、アルゴリズムの調整・改良の容易さを兼ね備えた方法を確立する。

目次

図目次

表目次

第 1 章

はじめに

1.1 Web 工学と機械学習

近年 Web 工学の領域においては、広告の効率アップや文章の分析、検索エンジンの性能向上などが求められている。このため、ユーザの性格や行動の分析、ネットワーク構造の未来予測、文章の意味や感情の理解といった技術が必要とされている。

例えば、Web 上で物を販売しているサイトにおいて、ユーザが購入しそうな商品を勧め、売り上げを増大させることを考える。ユーザが今まで見た商品、購入した商品を分析して、ユーザの求めている商品のジャンルを知ったり、Web ショッピングにおける性格を知ることが出来れば、ユーザが買いたい商品を先回りして、広告表示することが出来る。ユーザに対して、より購入してくれそうな広告を多く見せて、広告の効率をアップさせることで、サイトからの収益が上がることが期待できる。

Web 工学では、Web 上のデータを元に知識を学習し、次に得た知識を用いて予測や分類を行う、というアルゴリズムが必要になっている。例えば、ユーザ行動やソーシャルネットワーク構造の傾向を学習して、将来の動向を予測したり、ユーザに送られたメールの意味を理解できるようになって、迷惑メールとそうでないものを分類できるようになる、といった具合である。こういった知識学習のためには、機械学習 (マシンラーニング) と呼ばれる手法が広く用いられている。

機械学習とは、対象となる知識を数学的なモデルで表し、より適切なモデルを求めて微修正を重ねることで、知識を洗練させていくアプローチである。使用する数学的なモデルの大筋は、経験を基にあらかじめ決めておく。ユーザ行動、リンク構造、メールの文章といった元データは、全て数値に変換されて、モデルに入力される。モデルは、入力データから予測や識別を実際に行いつつ、より良い結果が得られるように、モデル内の係数を調整していく。最終的に、多種多様なデータに対して、適切な予測や識別を行うことが出来る数学的モデルを得ることができるのである。

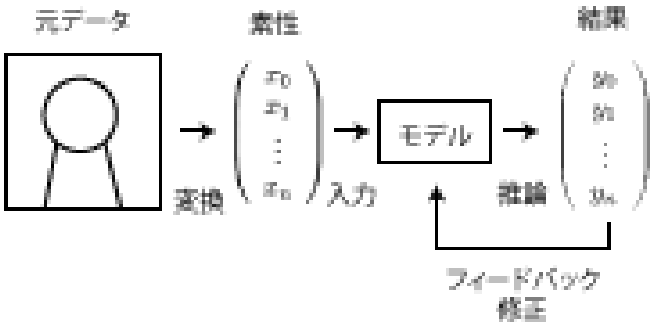

図 1.1　機械学習の一般的プロセス

1.2　深層学習の台頭

機械学習における大きなポイントの 1 つに、元データをどのような数値データに変換するか、という問題がある。

機械学習の一般的なプロセスを、図 1.1 に示す。変換された数値データのことを、 特徴量、あるいは素性 (feature) と呼ぶ。機械学習は、「生データから素性への変換」「素性をモデルに入力」「モデルによる数値計算で、結果を出力」「結果がより正確になるよう、モデルを修正」というプロセスの繰り返しで成り立っている。このうち、「素性への変換」だけは、画像・文章・リンク構造といった、データの種類に依存している。いったん素性が数値の形で得られれば、残りのプロセスは、データの種類に依存せず、全て入力数値とモデルの関係だけで解決することができる。つまり、機械学習のプロセスは、データの種類に依存する素性変換と、汎用的に使えるモデル学習の部分に分かれているのである。

機械学習の性能を上げるためには、素性への変換部分を工夫する方法と、使用する数学的モデルを洗練する方法の 2 つがある。素性への変換部分の改良は、対象となるデータの種類に強く依存している。例えば、画像データを素性数値に変換する場合、単純な RGB 画素データをそのまま使っても良いが、SIFT 特徴量 [61] や SURF 特徴量 [4]、フィッシャーベクトル [74] など、より画像の特徴を捉えた特徴量を用いることが定石となっている。音声データに対しては、時間領域や周波数領域の波形をそのまま用いても良いが、メル周波数ケプストラム係数と呼ばれる特徴量も用いられる。問題とデータ種別に応じて、利用する特徴量を工夫することにより、機械学習の識別精度や実行時間などの性能が向上することが知られていた。このとき、特徴量はそれぞれのデータの専門家による、謂わば職人芸によって作られていた。

2000 年代半ばに、Deep Learning と呼ばれる一群の手法が台頭した。Deep Learning とは機械学習の手法の 1 つで、人間の脳の構造に類似した、多層ニューラルネットワーク構造をモデルに用いて、学習を行わせる方法である。Deep Learning は、画像認識や音声認識、化合物の生成予測といっ

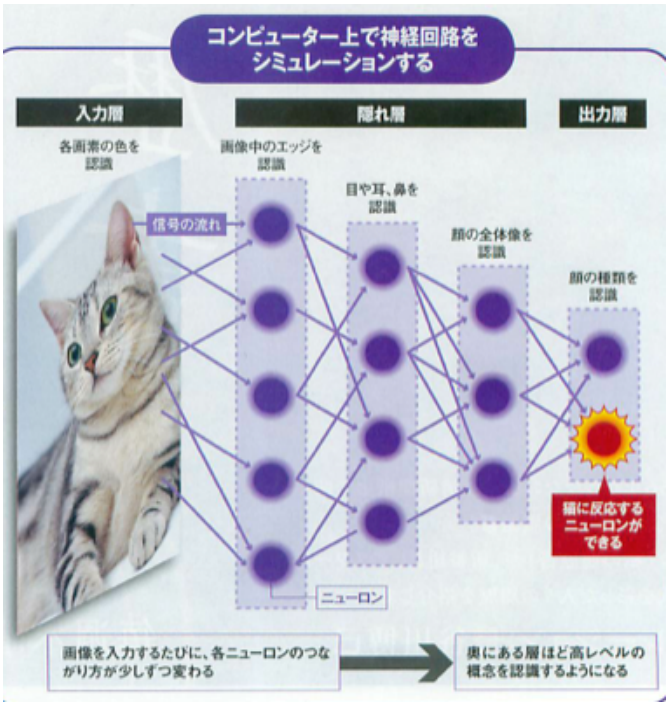

図 1.2　ディープラーニングで画像を認識する流れ

た分野で優れた性能を示し、注目を浴びるようになった。図 1.2 は、多層ニューラルネットワークが人間の脳の思考回路である、神経細胞のつながりを模倣する様子を表している。(日経ビジネス 2013 年 4 月号より一部抜粋)

Deep Learning の特徴の一つとして、生の入力データから自動的に素性を作り、抽象表現を習得する働きがあると考えられている。例えば人間の顔画像データを学習させた場合、多層ニューラルネットワークを構成する複数のレイヤーのうち、入力に近い低レイヤーのニューロンは、画像のエッジ部分に対して強く反応し、出力に近い高レイヤーのニューロンでは、目口鼻、さらに顔全体など、より抽象度の高い要素に対して反応することがわかった。これは、見方を変えれば、低レイヤーでは中間表現 (素性にあたる) を抽出しており、低レイヤーで得た素性を高レイヤーに入力することにより、より高レベルな抽象的概念に対しても優れた識別精度を実現している、と考えられる。どのような素性を使えば良い結果が出るのか、素性の抽出方法自体を同時に機械学習していることから、Deep Learning は Representation Learning(表現学習) とも呼ばれている。

図 1.3 は、2009 年、Lee らによる Deep Convolutional Belief Net の実験によって学習された顔の要素である。上が 2 レイヤー目、下が 3 レイヤー目で、2 レイヤー目では、鼻や口、目といった顔を構成するパーツが学習されている。これは、単なる画素の数値に比べて、明らかに抽象度の高い情報に反応していると言える。さらに、3 レイヤー目では人間の顔を学習することに成功している。2 レイヤー目で作り出した、「顔のパーツ」という素性を元にして、さらに抽象度の高い、人間の顔という要素を推論しているのである。[59]

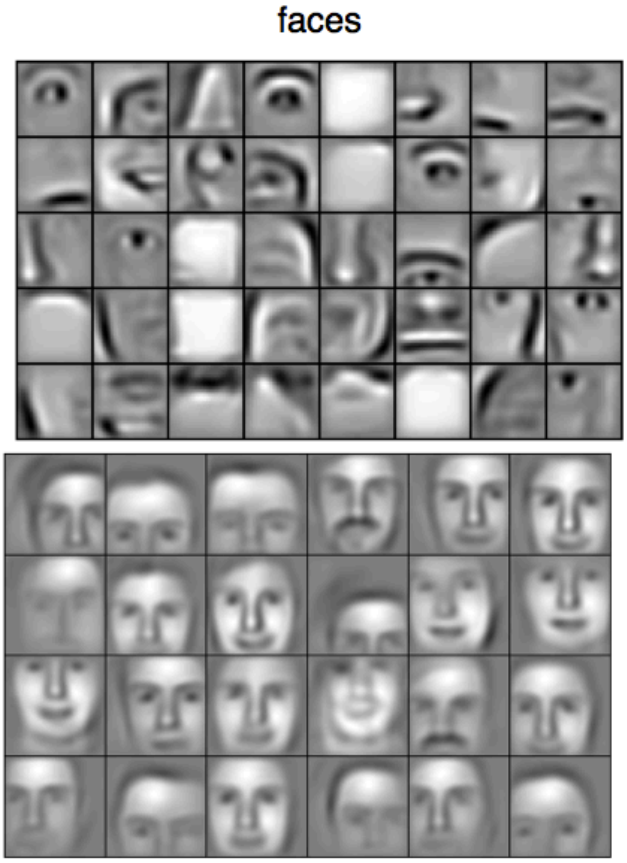

図 1.3　Deep Learning による、顔の構成要素の学習結果

Google の Deep Learning 研究グループは 2012 年、Youtube のビデオによる学習を長時間行わせることで、ニューラルネットワークの 1 つのニューロンが、猫の画像 (1.4、Google の研究紹介記事より引用[*1]) を認識するようになったと発表し、大きな話題を呼んだ [56]。この Youtube ビデオ学習の間、「この画像は猫という名前のものである」「猫を認識できるよう学習を行いなさい」といった、認識ラベルの付け方における指針は一切学習器に与えられなかった (教師無し学習)。それにも関わらず、多層ニューラルネットワークは、自然と「この (我々の言葉でいう猫の) 画像は覚えておくべきである」というように、何を認識すると学習効率が良いのか、何を素性とすれば学習がしやすいのか、という情報自体を学習することが出来た。また、同じ論文にて、この多層ニューラルネットワークが人間の顔を学習できることも示された (1.5、[56] より引用)。

1.3　深層学習の課題と、研究の目的

Deep Learning が高い識別性能を持つことがわかり、Deep Learning を身近な問題に適用して、良い成果を得たいという機運が高まっている。例えば、Web 工学の分野では機械学習が大きな役割を果たしており、この学習プロセスに Deep Learning を組みこむことで、学習精度が向上したり、よ

[*1] http://googleblog.blogspot.jp/2012/06/using-large-scale-brain-simulations-for.html

図 1.4　猫を認識するニューロン (教師無し Youtube ビデオより学習された。)

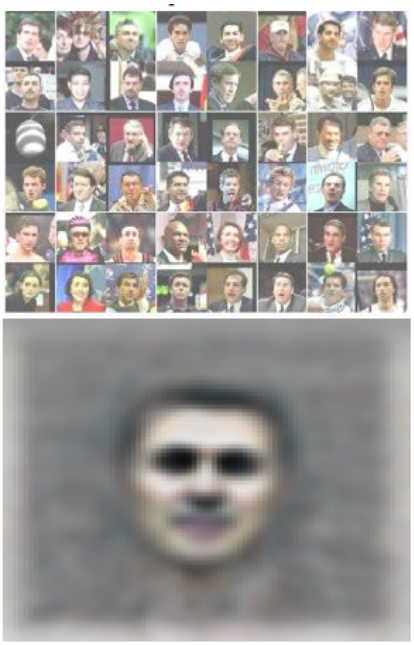

図 1.5　上：入力画像の中で、ニューロンが最も強く反応した 48 枚　下：計算上、最もニューロンが強く反応する画像

り多様な情報を扱えるようになる可能性がある。出来るだけ簡易に、Deep Learning を様々な問題に応用するための方法論が求められている。

しかし、Deep Learning は歴史の浅い発展途上の技術であり、どのアルゴリズムを定番とすれば良いのか、試行錯誤の段階にある。これは、各アルゴリズムの改良点が次々と見つかっていることに加え、学習性能が高くなる原理や、各アルゴリズムの得手不得手など、解明されていない部分が多いことが、主な原因である。アルゴリズムが開発途上で確定できていないため、公開されているライブラリも、現状では、開発用途や実験的なものが多くなってしまっている。実験的なライブラリでは、一部の種類のデータにのみ適用されることを想定して書いている場合があり、他の種類のデータを扱うためには、データ変換用のソースコードを記述しなければならないケースが多い。そもそも有力なアルゴリズムに対応する実装が公開されていない場合もあり、この場合、アルゴリズムの部分も含めて

全ての実装を用意しなければならない。また、問題に応じて自らアルゴリズムの細部を調整しなければならない場合もある。例えば、学習の繰り返し回数 (エポック数) や、どの種類のレイヤーを何回重ねるべきか (レイヤー構造)、1 つのレイヤーに含まれる学習素子 (ニューロン) の個数はどうするか、などである。これらはソースコードの作成者が経験的に手作業で調整しているケースが多く、標準と言える公開ライブラリが確立していない状況なので、Web 工学など応用分野に Deep Learning を適用したいと考えても、プログラム開発に長い時間がかかってしまう。開発における大きな障壁となっている。

さらに、現在の Deep Learning 技術では、他のアルゴリズムに比べて学習にかかる時間が長いことが多く、ハードウェア性能が低いマシンでは、アルゴリズムを実用的な時間で実行すること自体が容易ではない。実行時間の長さをカバーするため、GPU を用いて演算をスピードアップさせる手法が確立されつつあるが、特殊なプログラミングが要求され、開発における障壁の 1 つとなっている。また、ノート PC の大部分など、並列演算に利用可能な GPU を搭載していない PC を使っている場合には、ライブラリが GPU を利用しているために、却ってその実行が不可能になってしまうこともある。

手元のパソコンにて実行可能な Deep Learning のソースを探し当てたとしても、このソースが実際に、論文にて示されているような高い精度を実現できるかどうかは、定かではない。Deep Learning の分野では、実験に用いたソースコードが提供されている場合が多いが、単にアルゴリズムの要点を示すためのものだったり、ハイパーパラメータのチューニングが言わばお試し用の状態になっていて、論文に記述してある精度を手元の PC で実現することが難しい場合がある。

以上に挙げた原因により、Deep Learning 技術に関心を持っていても、まず実際の問題に Deep Learning を試行すること自体が困難であり、応用技術開発のハードルは更に高くなっている。特に、国内での研究開発は遅れており、早急なキャッチアップが必要である。

本研究では、このような現状を踏まえ、Deep Learning 技術を実際の問題に応用するための方法論を確立すると共に、実装における障壁を出来る限り取り除くことを目指す。特に Web 工学における応用を目標とする。

2 章では、従来の Web 工学における機械学習の応用方法と、Deep Learning 出現以前に良く用いられていた、機械学習による識別器学習方法について、俯瞰する。

3 章と 4 章では、現在の Deep Learning 技術について紹介する。3 章では、Deep Learning を構成する個々のアルゴリズムについて、その原理と詳細を述べる。4 章では、実際に Deep Learning アルゴリズムを利用する上で、どのように実装を進めれば、Deep Learning の特徴である高い学習性能を確実に利用できるのか、また、現実の問題を解決するときに重要となる、実行時間の短さ、実行プログラムの使いやすさ、アルゴリズムの調整・改良の容易さなどは、どのようにすれば確保できるの

か、といった Deep Learning を使っていく上でのノウハウを記述する。

5 章からは、4 章までの内容を実際の問題に応用することで、その有用性を確かめる。5 章では、Deep Learning のプログラムを、汎用的なベンチマークにかけることで、その性能を測定・比較する。6 章では、5 章の結果を受けて、Deep Learning を利用していく上でのベストプラクティスを考察する。7 章では、Deep Learning の利用例として、ソーシャルネットワークにおけるプロフィール推定を行う。

第 2 章

関連研究

2.1 Web 工学と機械学習

この節では、Web 工学における課題をいくつか例示し、それらの課題を解決するために、機械学習がどのように用いられてきたかを概観する。具体的には、Recommendation System, Link Prediction, Sentiment Analysis, Learning to Rank の 4 つを取り上げる。

2.1.1 Recommendation System

主に Web ショッピングを含むサイトや、Web 広告の配信において求められる技術である。Web サイトを閲覧しているユーザに対し、そのユーザが購入したいと商品を予測して、Web 上の広告などの形で推薦する。適切な広告を表示することにより、ユーザの購買行動を促進することが出来る。Recommendation System の研究は、1992 年の Tapestry システムに始まる [36]。また、少なくとも 90 年代の終わりには、Amazon.com, CDNOW, eBay, Levis, GroupLens など、様々なサイトにて Recommendation System は利用されていた [77]。Recommendation System を実現するための、機械学習のテクニックとしては、大きく 2 種類が挙げられる [50]。1 つは、Content Filtering と呼ばれ、ユーザや商品の属性や購買傾向を学習することことで、推薦を行う。もう 1 つは Collaboraitive Filtering と呼ばれており、ユーザや商品の属性を扱う代わりに、購入や評価といった、ユーザの過去の行動を基にして、推薦を行う。

2.1.2 Link Prediction

Link Prediction とは、図 2.1 のように、現在の時間 t のグラフ構造 (ノード間接続の様子) が与えられたとき、未来の時間 $t+1$ における、同じグラフの構造変化を予測する問題である。テストデータでは、グラフ中の一部のエッジが隠されており、その値を

最初に提唱された論文 [60] では、ソーシャルネットワークの状態が与えられたとき、ユーザ間で交わされる将来の行動を予測する問題として述べられている。この論文では、ノード (ユーザ) 間の近

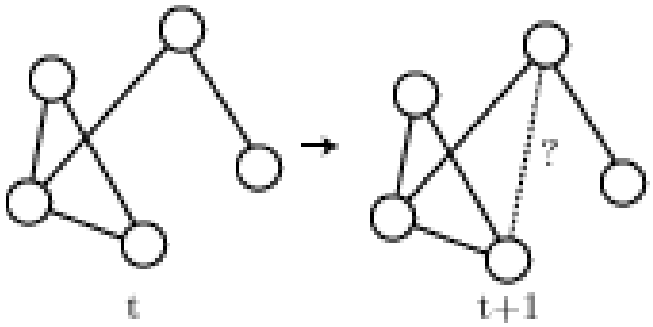

図 2.1　Link Prediction の問題設定

graph distance	(negated) length of shortest path between x and y
common neighbors	$\|\Gamma(x) \cap \Gamma(y)\|$
Jaccard's coefficient [44]	$\left\|\frac{\Gamma(x) \cap \Gamma(y)}{\Gamma(x) \cup \Gamma(y)}\right\|$
Adamic / Adar [1]	$\sum_{z \epsilon \Gamma(x) \cap \Gamma(y)} \frac{1}{log\|\Gamma(x)\|}$
preferential attachment	$\|\Gamma(x)\| \cdot \|\Gamma(y)\|$

表 2.1　Link Prediction にて、ノード同士の近さを定義する方法

さを定義する方法や、全てのエッジ (接続) を通しで見て判断する方法、クラスタリングによる方法などが用いられている。論文で用いられているノード間の近さの定義を図 2.1 に挙げる ([60] より引用)。ノード x に対し、x とつながっているノードの集合を $\Gamma(x)$ とする。

LInk Prediction には様々な応用があり、例えばショッピングサイトのデータ分析に役立てることが出来る [20]。Web 工学の問題以外にも応用が可能であり、タンパク質の反応予測 (Protein to Protein Interaction, PPI) に用いられたり [3]、テロリストのネットワークや草原の食物連鎖の分析にも役に立っている [19]。全ての部分グラフをカウントする方法は NP 困難であり、計算時間的に解くことが難しいことが知られている [33] が、この問題は GraphHopper Kernel を用いることで、現実的な時間で解くことができる [26]。また、Noisy OR を用いて局所的な情報を統合することにより、効率良く様々な情報を扱わせることができる [16]。

2.1.3　Sentiment Analysis

この場合の Sentiment とは、ユーザが持っている感情のことである。Web 上の情報収集の自由度が高まるにつれて、ユーザが書いた文章を取得できる機会が増えてきている。その中には、新製品やサービスへの感想やレビューも含まれている。ユーザの属性や行動といった明確な事実だけでなく、他の人がどのような感情を抱いているのかを分析したいという要望が高まり、2000 年代に入ってから、人間が書いた文章から人間の感情を読み取るための研究が盛んになった [73]。なお、opinion mining という語も、多少の差異はあれど、広義には似た内容を指していることが多い。

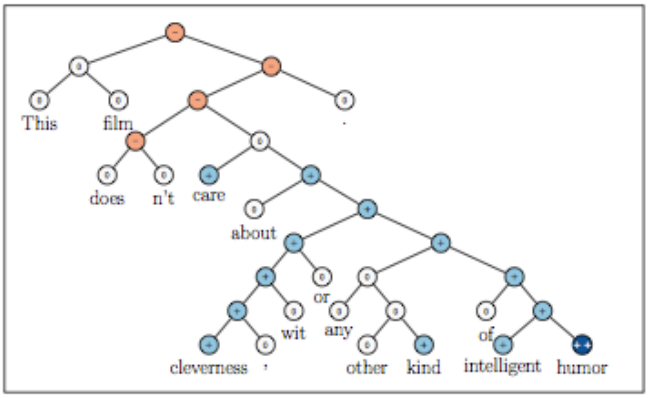

図 2.2　Stanford 大が製作した、Deep Learning による Sentiment Analysis の例

[72] では、最小カット法を用いることで良い成果が出たとされている。また、[95] では、文脈を考慮した単語の感情 (positive, negative) を分析して、性能を挙げることに成功している。Google 社も、検索エンジンの性能向上のため、Sentiment Analysis の研究を行っている [35]。

Deep Learning との関連で重要な研究として、Stanford 大のチームによる文章解析データセットの制作及び、Web アプリの開発がある [89]。この研究にて彼らは、感情分析のための大規模なデータセットがそもそも不足しているという問題を解決するため、The Stanford Sentiment Treebank という Sentiment Analysis のためのデータセットを制作した。さらに、Recursive Neural Tensor Network (RNTN) という Deep Learning のモデルを構築した。このモデルにより、与えられた文章を構文木に分解した後、各単語が 5 段階の感情のうちどれにあたるかを分析している。感情は、”very negative”、”negative”、”neutral”、”positive”、”very positive” の 5 種類に分類される。このモデルは、ソースコードが公開されている他、Web ページ上にて実際に分析を試してみることができる[*1]。図 2.2 は、論文より引用したもので、文章を実際にこの web アプリで分析した様子を表している。赤が強いと negative な言葉、青が強いと positive な言葉、というように色分けされている。

2.1.4　Learning to Rank

これは、情報検索システムを作る際に用いられる技術である。ユーザが検索をかけるときは、自分が求めた情報だけを、出来るだけ取りこぼし少なく取得したい (precision と recall という。[97])。従来は tf-idf や PageRank[12][70] などの単一の指標によって、検索結果を表示させていたが、その後多様なランキング素因を組み合わせるという方法が出現した。どの素因をどのような割合で組み合わせるべきかが問題であり、これに対し機械学習によってランキング関数を作成するアプローチが、Learning to Rank である。

RankNet[13] では、Deep な構造こそ使っていないが、ニューラルネットワークによってランキン

[*1] http://nlp.stanford.edu/sentiment/index.html

グ関数の性能をアップさせており、Deep Learning の応用による性能向上が期待できる。

2.2 機械学習で利用される、代表的な分類器

機械学習のプロセスは、「入力データを、数学的モデルで使える素性に変換する」「素性を数学的モデルに入力して、出力値を得る」「出力を見ながら、モデルを修正する」という行程に大きく分けられる。データの分類問題を機械学習で解く場合、モデルによる出力値が分類結果に対応するよう、モデルを学習させることになる。この場合、モデルのことを分類器とも呼ぶ。

機械学習において、素性への変換部分は、データの種類に大きく依存する。一方、分類器に用いる数学的モデルと、モデルの改修法、つまり学習法は、汎用的に使うことができる。あるいは、画像や音声、文章といったデータの多様性を、素性という一般的な数値に落とし込むことで吸収して、汎用的分類モデルでも学習できるようにしている。

Deep Learning、あるいは Deep Neural Network(多層ニューラルネットワーク) は、汎用的分類モデルの一種である。ここでは、Deep Learning の他にどのような分類器が存在するのか、代表的なものを述べる。

2.2.1 Support Vector Machine

Support Vector Macine(SVM) は、図 2.3 のように、データを 2 つのクラスに分類する能力を持ったモデルである [22]。

SVM のメインとなる原理は、マージン最大化である。図 2.3 は最も単純な SVM の問題を表しており、グラフ上に散らばった黒と白の点を、直線を 1 本引くことで分けることが目標である。言い換えれば、どの点が黒で、どの点が白なのかを識別する、2 クラス分類問題を解こうとしている。緑の線は、分類に失敗している。青の線は分類に成功しているが、後から点線で表された新しい白点が加わると、やはり分類に失敗してしまう。赤の線は、現在見えている点を分類できるだけでなく、新しい点が加わっても正しく識別できる可能性が高い。SVM のマージン最大化とは、各点からの距離 (マージン) の和が、最大になるように直線の引き方を決めることである。こうすることにより、汎化性能 (訓練時に見えていなかった、新しいデータを正しく分類する性能) が最大になることが知られている。

この図の SVM は、データが直線で完全に分類できることを前提にしている。しかし、現実のデータは、2.4 のように、必ずしも 1 本の直線にて分類が可能ではない。(3 次元以上の場合で言えば、1 つの超平面にて分類が可能とは限らない。) このような場合に対応するため、まず非線形関数にてデータを高次元の空間に移し、超平面による分類が出来るように変換する方法が知られている [14]。ただし、一般に複雑な分類ほど、変換先の次元数が高くなる傾向があり、次元数が増えると、「次元の

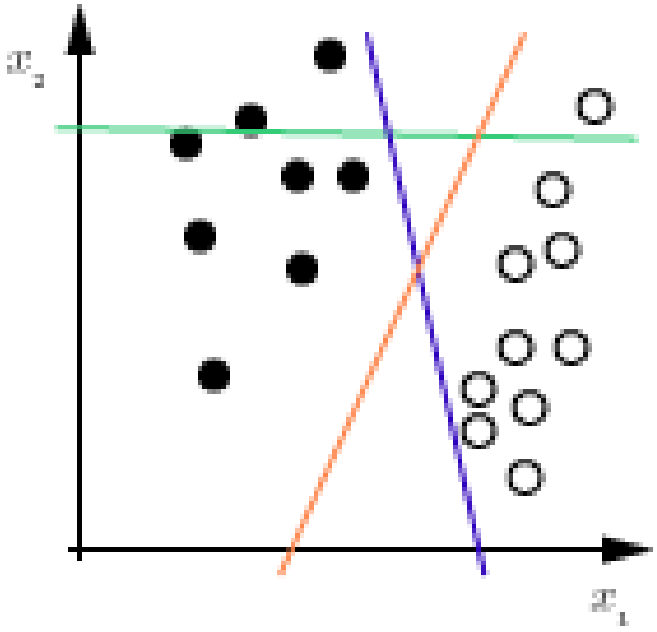

図 2.3　Support Vector Machine のマージン最大化

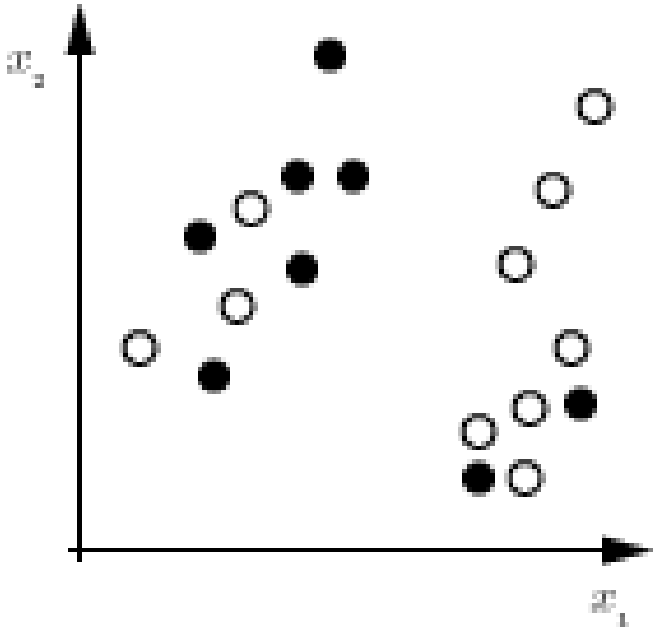

図 2.4　直線にて分類できない場合

呪い」と言って、内積計算にかかる時間が指数関数的に増大することがわかっている [5]。この計算時間を減少させるため、数式処理によって内積を計算する必要がなくなるように設計された、カーネル関数と呼ばれる変換関数を用いる方法が使われている (カーネルマジック)。 SVM は、広くその信頼性が認められたモデルの 1 つであり、ライブラリの利用方法も確立している。例えば、libsvm[*2]や liblinear[*3]は使用が容易なライブラリとしてよく知られている。これらのライブラリを使うと、簡単

[*2] http://www.csie.ntu.edu.tw/~cjlin/libsvm/
[*3] http://www.csie.ntu.edu.tw/~cjlin/liblinear/

な所定の方式に沿って入力データファイルを用意し、CUI 上で 2,3 回の操作をするだけで、SVM による分類を行わせることができる。このとき、利用者が自分でプログラムを書く必要は全くない。プログラムを書かないで済むと、手軽に利用することができ、またバグを起こす危険性が非常に少なく安全に使うことができる。

Deep Learning については、このようなライブラリはまだ存在していないため、Deep Learning の代表的なアルゴリズムについて、プログラム無しで利用できるようなライブラリの整備が望まれる。

2.2.2 ニューラルネットワーク

ニューラルネットワークは、人間の脳の構造を模倣した数学的モデルである。人間の脳は、ニューロンと呼ばれる神経細胞が大量に接続されて出来ている。ニューロンが電気信号を伝達することで、様々な脳の働きが行われている、と考えられている。プログラム上で表現されるニューラルネットワークには、人間の脳の動作との相違点もあり、脳が行っている計算を正しくシミュレートしているとは言い難い面もある [98] が、機械学習のモデルとしては広く使われてきた。

ニューラルネットワークの伝達方式

以後、機械学習におけるニューラルネットワークのニューロンを、慣例に従ってユニットと呼ぶことにする。ニューラルネットワークでは、ユニットからユニットへの接続が網目のように広がっている。ユニットは入出力の機能を持つ。これは、生体におけるニューロンが、他のニューロンから化学的な刺激を受け取り、受け取った刺激に応じて自らも他のニューロンを刺激する構造を真似ている。ユニットは入力を受け取ると、活性化関数 (activation function) を使って入力値を変換し、接続先のユニットへ出力する。このとき、ユニットからユニットへの結線に、重み (weight) と呼ばれる係数を付与して、出力を変化させることが普通である。これは、生体のニューロン同士の接続が、脳の学習につれて強固になり、刺激がより伝わりやすくなっていくことに対応させている。

パーセプトロン

パーセプトロンとは、ニューラルネットワークの一種である。複数のユニットをまとめた層 (レイヤー) が、いくつか重なって出来ており、値の伝達は入力レイヤーから出力レイヤーへの一方向に限られているものを指す (feed-forward)。狭義には、単層かつ活性化関数にヘビサイド関数を用いた 2 クラス分類モデルのみを指すこともある。

始めに提唱されたのも、入力層と出力層の 2 層から成る、単層パーセプトロンと呼ばれるモデルだった [81]。このモデルは、後に線形関数しか近似できないことがわかり、いったん下火になった [67]。例えば、単層パーセプトロンでは、非線形関数である XOR 関数を学習させることが出来なかった。

しかし、隠れ層を追加し、活性化関数にシグモイド関数などの非線形関数を用い、さらにバックプロパゲーションという方法で学習を行わせることにより、非線形関数を近似可能となることがわかり、再び有用な識別モデルとして脚光を浴びた [82][31]。これを単層パーセプトロンと区別して、多層パーセプトロン (Multi Layer Perceptron, 以下 MLP) とも呼ぶ。このモデルは 2 クラス分類モデルの範囲を逸脱しており、元々のパーセプトロンとはやや異なるものだが、慣例的にこのように呼ばれている。

バックプロパゲーションは教師有り学習の一種である。出力層におけるモデルが出した推定値と、教師データの結果が異なっている場合、教師データからの推定誤差 (error) が減少するように、モデルのパラメータを修正する。MLP の場合、モデルのパラメータとはユニット間の伝達にかかる重み係数のことである。パラメータの修正値の仕方にはバリエーションがあるが、最もよく使われるのは Stochastic Gradient Descnet(以下 SGD) と呼ばれる方法である。これは確率変数に拡張された一種の再急降下法である。直感的に言えば、微分によって、その場で推定誤差が最も急速に減少するパラメータの修正方向を算出し、その方向に向かってパラメータを変化させる。SGD を少し変化させ、複数のデータによる誤差を一度に処理する Batch Gradient Descent(BGD) や、SGD の 1 次精度に対し、2 次精度までの情報を使う共役勾配法 (Conjugate Gradient Method, CG) も存在する。

バックプロパゲーションを行うためには、活性化関数が微分できることが重要である。シグモイド関数は、元々使われていたヘビサイド関数に形が似ている上に、微分が容易という点でバックプロパゲーションとの親和性が高く、有利である。しかし、シグモイド関数のデメリットとして、入力と重みの積が大きくなるにつれて、誤差への反応が小さくなってしまい、学習の進行が遅くなるという問題を抱えていた。この問題は、特に隠れ層を 2 層以上にしたとき顕著になった。このため、多層ニューラルネットワークを学習させる方法は長い間課題となっていた。

第 3 章

深層学習の成果とアルゴリズム

Deep Learning は、多層ニューラルネットワーク (MLP) のうち、普通隠れ層が 2 つ以上のものをいう。2006 年に Hinton らによって、このような Deep な構造を学習させる方法が発見され、発展への道が開けた [39, 40]。従来よりも多くのレイヤーを扱うことで、より複雑な関数を学習できるようになった。Deep Learning は、数々の分類タスクにて、従来手法を大きくしのぐ成果を収め、注目を浴びている。この章では、Deep Learning で使われているアルゴリズムの詳細について述べる。

3.1 Deep Learning の成果

第 1 章でも触れたが、2012 年には、Google が Supervision と題して Youtube ビデオの静止画を使って多層ニューラルネットワークの教師なし学習を行わせ、結果として、猫を認識するユニットや人を認識するユニットを学習させることに成功した。また、ImageNet Large Scale Visual Recognition Competition(ILSVRC) というデータセット [25] の分類にて、state of the art の結果を残した。ImageNet[*1] とは、WordNet[*2] を真似して作られたデータセットである。2009 年より、100 を超える様々な論文にて、画像認識のベンチマークに利用されてきている[*3]。図 3.1 は、ImageNet の画像の例である。画像はツリー状に分類されており、例えば上を見ると、mammal(哺乳類) → palcental(胎盤) → carnivore(肉食) → canine(イヌ科) → dog(イヌ) → working dog(盲導犬やそり犬など使役される犬) → husky(ハスキー) というように、徐々に分類が細かくなっていくことがわかる。また、一つのカテゴリーには、9 枚の画像がひもづけられている。

Google の supervision による、画像分析の結果の例を図 3.2 に載せる ([52] より引用)。画像の分類に失敗している場合でも、妥当なラベルをつけていることがわかる。

Deep Learning は画像分析だけでなく、自然言語処理 (NLP) の分野でも有効なことがわかっている。Google が公開している”word2vec” というライブラリを使うと、入力されたテキスト文書から、

[*1] http://www.image-net.org/
[*2] http://wordnet.princeton.edu/
[*3] http://www.image-net.org/about-publication

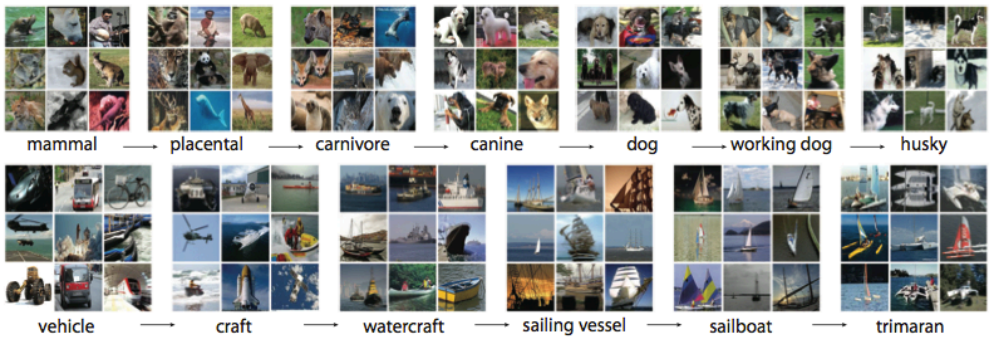

図 3.1　Imagenet の画像とラベル構造の例

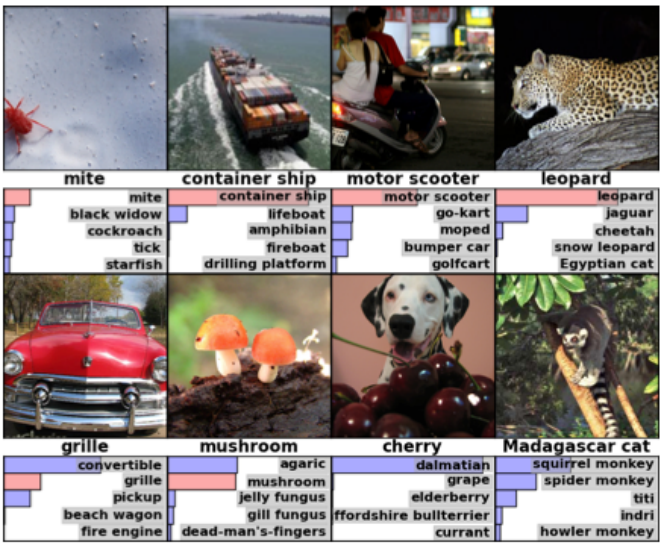

図 3.2　Supervision による画像分類の結果の例

単語のベクトル化やクラスタリングを行うことができる [63][65]。このベクトル化により、

$$vector('Paris') - vector('France') + vector('Italy') = vector('Rome') vector('king') - vector('man') + vector('woman') \tag{3.1}$$

のような計算が行えるようになった。また、この計算を行う過程で、全ての単語同士の内積の和を計算していたところを止め、数個だけサンプルを取って計算することで、計算時間が短縮できるだけでなく、分類精度まで向上することが判明した [64]。

word2vec を応用した研究も出現している。Deep Visual-Semantic Embedding model(DeViSE)[28] と名付けられたシステムでは、zero-shot learning と呼ばれる問題を解かせるため、Supervision と word2vec を組み合わせている。zero-shot learning とは、訓練時に一度も見たことがないクラスの画像に対し、正しく分類を行えるかという問題である。この研究では、1000 カテゴリの画像による訓練を行っただけで、まだ見たことが無い 20000 カテゴリに属する画像を適切にラベリングしている。画像以外に全く情報がなければ分類は不可能なので、wikipedia のテキスト内容を補助データとして使っているが、これは WordNet や意味データベースの情報を直接用いていた従来の研究

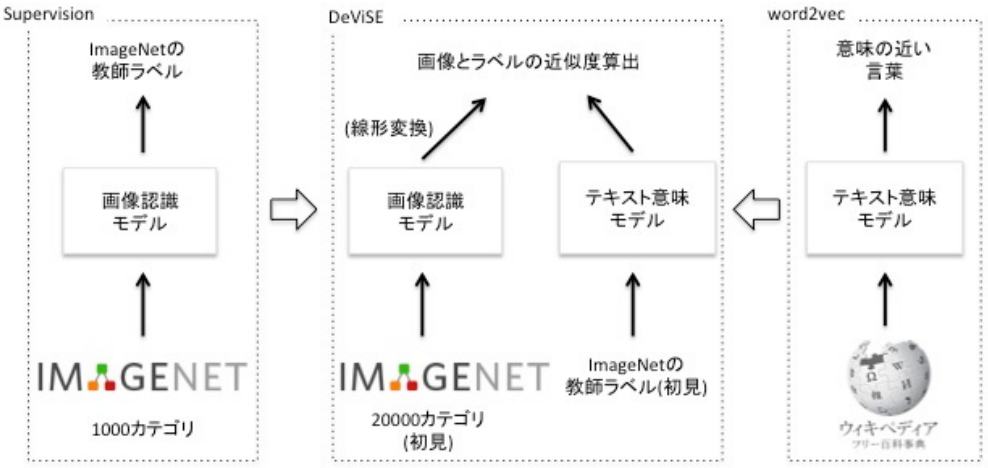

図 3.3　DeViSE の学習方法

[62][80][71] に比べて、より生データに近い情報で処理に成功していると言える。

図 3.3 は、DeViSE の学習方法を示している (ImageNet と Wikiedia のロゴは、公式 Web ページより引用した[*4],[*5])。DeVise では、画像データを意味付けするために Supervision を使い、wikipedia の情報処理に word2vec を用いている。まず Supervision に ImageNet のデータを、word2vec に wikipedia のデータを与えて学習させたあと、実際の画像及び未見カテゴリのラベルを与え、画像とラベルの意味ベクトルが、出来るだけ近くなるように分類している。

Deep Learning を強化学習 (reinforced learning) と組み合わせることで、ゲームを上手くプレイできる AI を作る研究も成功している [68]。強化学習では、環境に対して AI が行動を起こし、環境から得られる報酬を最大化できるように、アルゴリズムを修正していく。多くの場合、「行動と報酬の間に時間差があり、どの行動が報酬に結びついたのか分かりにくい」「行動によって環境が変化してしまうため、分析が難しい」といった状況設定がなされる。また、Bandit 問題というカテゴリでは、スロットゲームを題材にして、「知識を活用すると報酬は上がるが、もっと良い行動があるかもしれない」「行動の探索を優先すると、既存知識は活かせず、一時的な報酬は下がりやすい」というジレンマの解決が研究されている [15]。[68] の研究では、ゲーム内のキャラクターを AI で行動させ、報酬としてのスコアを最大化させるような行動を学習させる。Deep Learning は、ゲーム内の状況をモデル化して、そこから最適な行動を導き出すために用いられている。

[*4] http://www.image-net.org/splash/logo.jpg
[*5] http://upload.wikimedia.org/wikipedia/commons/a/ad/Wikipedia-logo-v2-ja.png

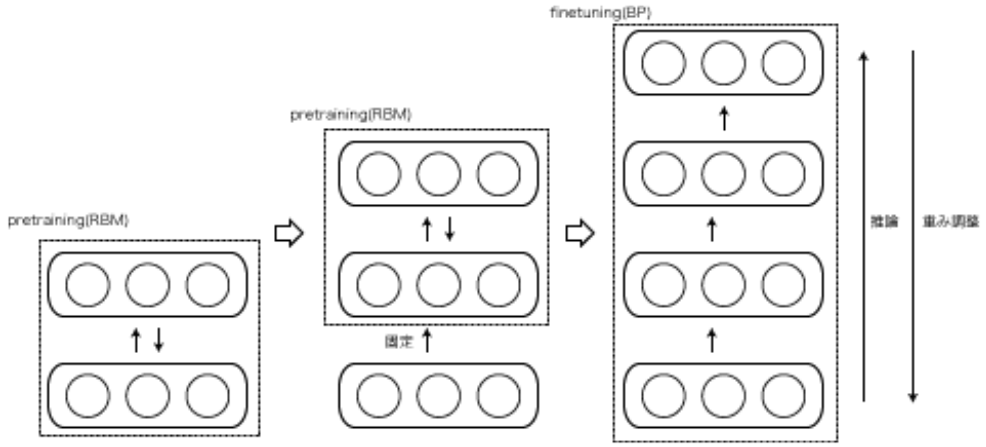

図 3.4　Deep Belief Network の構造と、学習過程

3.2　Deep Learning のアルゴリズム

3.2.1　Deep Belief Network

最初に MLP の学習のブレイクスルーとなったのは、2006 年の Hinton らによる Deep Belief Nets(DBN)[39, 40] である。このモデルについて解説する。

unsupervised pretraining と finetuning

この研究では、MLP の重みをランダムに初期化した後、すぐバックプロパゲーション学習にかけるのではなく、各レイヤーの重みをあらかじめ教師無し (unsupervised) 学習で調整して、隠れ層が効率の良い素性を学習できるよう仕込んでおくというアイデアが使われた。この事前学習のことを、pretraining と呼び、バックプロパゲーションにて学習する段階の方は、finetuning と呼ぶ。pretraining の段階では、入力側から 1 レイヤーずつ学習を行い、学習済のレイヤーは固定して動かさない (Greedy Layer-wise pretraining)。finetuning の段階では、全てのレイヤーをバックプロパゲーションにて同時に変化させていく。DBN の構造を、図 3.4 に示す。

unsupervised pretraining をどのような基準で行うかが問題だが、この研究では、Restricted Boltzman Machine(以下 RBM) というモデルを用いている。RBM とは、ニューラルネットワークの一種で、可視変数層と隠れ変数層の 2 つのレイヤーから成り立っている。DBN においては、可視変数=入力データと考えておけばよい。RBM モデルの学習を進めることで、入力データ→隠れ変数への推論モデルを得ることができる。これが Deep Learning の最大の特徴の 1 つである、「抽象度の高い素性への変換方法自体を学習する」ことに対応している。DBN では、RBM を 1 レイヤーずつ学習させながら、RBM 同士を「前のレイヤーの隠れ層 = 次のレイヤーの入力層」となるようにつなげていく。これにより「入力層→抽象度の低いレイヤー→... →抽象度の高いレイヤー→出力」とい

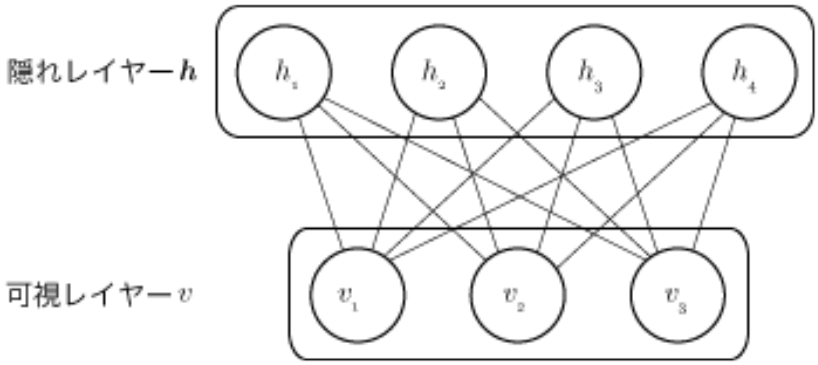

図 3.5　Restriceted Boltzman Machine のネットワーク構造

う、複数の隠れ層をもつ Deep な構造を作り、上手く pretraining させることに成功している。RBM を積み重ねた最も上のレイヤーに、普通の線形レイヤーを 1 つ置いて、「最も抽象度の高い素性→出力値」の推測を行わせることが多い。なお、finetuning におけるバックプロパゲーションでは、RBM にあった「隠れ層→入力層」の方のつながりは、最終レイヤーにおける誤差に影響しない。よって、以前からある feed-forward な MLP と同様に学習させることができる。

Restricted Boltzman Machine

DBN の pretraining に用いられる Restricted Boltzman Machine[87] とは、ニューラルネットワークを用いた生成モデルの一種である。まず生成モデルについて説明する。

クラス分類問題を確率的アプローチで解く方法は、生成モデル (generative model) と識別モデル (判別モデル、discriminative model) の 2 つに大別できる [10][*6]。クラス分類問題では、まずクラス事後確率 $p(C_k|x)$ を各クラス k に対して求める。基本的には入力 x に対して最もクラス事後確率が大きくなったクラスに分類することになる[*7]。識別モデルでは、クラス事後確率 $p(C_k|x)$ のみを直接学習するのに対し、生成モデルでは、まずクラスの条件付き密度 $p(x|C_k)$ を学習する。これと、別途求めたクラスの事前確率 $p(C_k)$ を用いることで、クラス事後確率は

$$p(C_k|x) = \frac{p(x|C_k)p(C_k)}{p(x)} = \frac{p(x|C_k)p(C_k)}{\sum_k p(x|C_k)p(C_k)} \tag{3.2}$$

によって求められる。これは、入力とクラスの同時確率 $p(C_k, x)$ を求めることと等価である。生成モデルの利点は、クラス分類のモデルだけでなく、入力データの性質に関する情報をも同時に得られる点にある。しかし、一般に識別モデルよりも複雑で困難な問題を解く必要が生じるため、常に生成モデルが用いられるわけではない。

さて、前述したように、RBM は可視層と隠れ層の 2 レイヤーで構成されるニューラルネットワー

[*6] 確率モデルを介さず、入力からクラスを得る識別関数を直接導くことも出来る。
[*7] 医療における誤診断など、誤分類のタイプに応じて重要度が異なる場合、損失関数 (loss function) を導入することもある。

クである。RBM の構造を、図 3.5 に示す。入力層の各ユニットの値を v_j, 隠れ層の各ユニットの値を h_i とおく。これらをまとめたベクトルをそれぞれ $\boldsymbol{v} = \{v_1, v_2, ...\}, \boldsymbol{h} = \{h_1, h_2, ...\}$ とおく。これは、例えば $\boldsymbol{v}$ が画像や音声などの入力データをベクトル化したもので、$\boldsymbol{h}$ がこれらの変数の関係を説明する変数だと考えれば良い。ただし、$\boldsymbol{h}$ が即 $\boldsymbol{v}$ のクラス分類に対応するとは限らないことに注意する。RBM の学習を進めると、$p(\boldsymbol{v}|\boldsymbol{h})$ と $p(\boldsymbol{h}|\boldsymbol{v})$ との、2 方向の条件付き確率のモデルが同時に得られる。

RBM の学習は、自由エネルギーと呼ばれる従属変数を最小にするように行われる[*8]。ここでは、最も簡単かつ重要な、$v_j, h_i \epsilon \{0, 1\}$ のケースについて述べる。W を v と h の間の重み、b と c を、それぞれ v と h に対応するバイアス項とする。このとき、RBM における自由エネルギー $F(v)$ は、

$$F(v) = -b'v - \sum_i log(1 + e^{(c_i + W_i v)}) \tag{3.3}$$

となる。この自由エネルギーを、SGD などのアルゴリズムを用いて最小化することで、目標となる生成モデルを得ることができる。なお、自由エネルギーの導入と最小化というアプローチは、Energy-Based Model と呼ばれるアルゴリズム群に共通して用いられる概念であり、自由エネルギーにもさらに様々なモデルで使える一般的な定義が存在するが、割愛する。

DBN の性能

この研究では、評価実験として MNIST の手書き数字認識データセットを用いている。彼らは各レイヤーのユニット数が”784-500-500-2000-10”という構造の DBN によって、学習を行った。そして、Permutation invariant と呼ばれるタスクにて、当時の state fo the art を塗り替えたことにより、注目された。ここで、Permutation invariant とは、データを単なる 1 次元の値の集まりと見なし、2 次元の画像という事前情報の利用を禁止した状態で、分類実験を行うことである。具体的には、画像的変形による水増しや、2 次元的畳み込み (Convolutional Layer の節にて詳述) などが当てはまる。Permutation invariant の制約下で良い成果を出した DBN は、画像の性質を利用した Convolutional Net に比べたとき、画像以外の 1 次元のデータに対しても適用しやすいアルゴリズムだと考えられる。

3.2.2 Stacked Denoising Autoencoder

Deep Belief Network と共に、1 次元のデータに対して広く用いることができるモデルが、Stacked Denoising Autoencoder である。これは 2007 年に提唱された Stacked Autoencoder[8] というモデルを基に、2008 年に発表された [92]。基本的なアイデアは、Deep Belief Nets における RBM を、Denoising Autoencoder というモデルに変更することである。

[*8] 自由エネルギーという呼び名は、物理学から採られている。

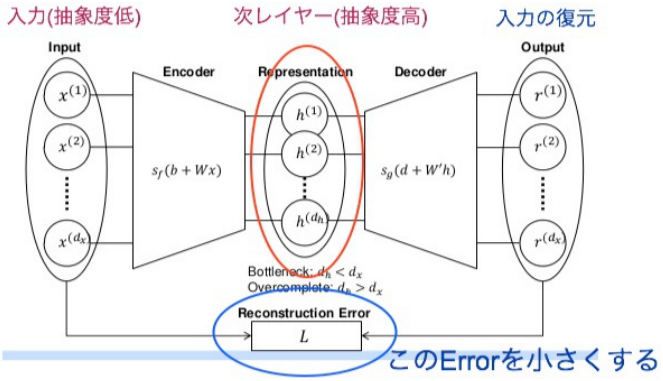

図 3.6　Autoencoder の構造図

Autoencoder

Denoising Autoencoder は、Autoencoder の特殊なバージョンなので、まずは Autoencoder について説明する。Autoencoder は、ニューラルネットワークの一種であり、入力されたデータを再現するようなモデルを学習する。[11, 42, 84]

Autoencoder の構造を、3.6 に示す。Autoencoder は、この図でいう Input, Representation, Output の 3 つのレイヤーから構成されている。Output と Input の値が一致するように、重み W と W' を学習させていく。この一致に成功すれば、Representation は Input を復元 (decode) するための情報を含んでいる、つまり Representation は Input の別表現である、という論理が成り立つ。Input に与えられたデータを、別の素性を用いた Representation に符号化 (encode) していることから、自分自身の符号化方法を覚えるという意味で、Autoencoder という名前がつけられている。

Autoencoder を用いることで、Input を別の素性に変換することができる。しかし一方で、必ずしも得られた素性が抽象度の高いものになっているとは限らない。例えば、最も自明な Autoencoder は、Encoder と Decoder の双方に恒等写像を用いることである。このとき、Input と Representation と Output の値は一致し、確かに Reconstruction Error は 0 になっている。しかし、Input と Representation の抽象度は同じであり、素性学習としての有用性は全くない。実用上は、SGD を用い、入力レイヤーよりも表現レイヤーのユニット数を多くとり、非線形な関数を Encoder や Decoder に使うことで、抽象度が高くスパースな素性が獲得されることがわかっている [8][58]。しかし、理論的な理由は解明されていない。この素性学習の不確実性の問題を緩和するのが、次に述べる Denoising Autoencoder である。

Denoising Autoencoder

Denoising Autoencoder は、Autoencoder を拡張したものである。Input 層に渡す入力データの一部をランダムに隠し (corrupted input)、元の隠されていない入力データを復元するように学習さ

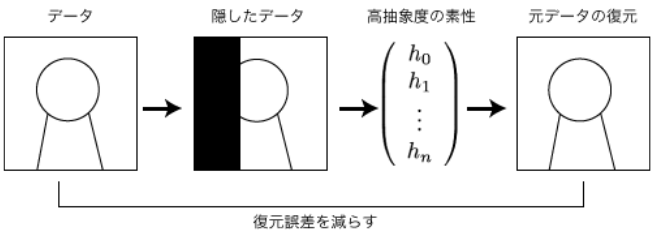

図 3.7　画像に対する Denoising Autoencoder

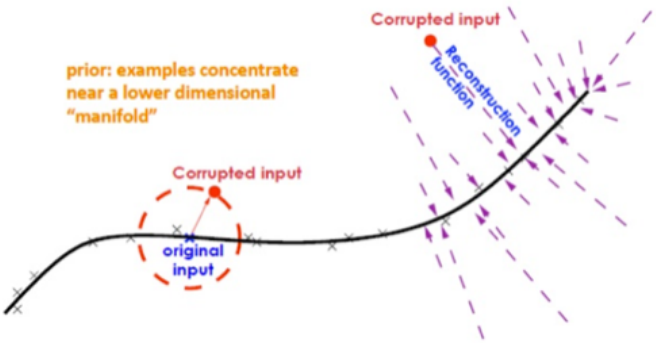

図 3.8　Denoising Autoencoder の模式図

せる。データを隠す割合を corruption rate といい、例えば corruption rate30% の場合、100 次元の画像データだとしたら、そのうち 30 次元分をランダムに選んで、数値を 0 にしてしまうことになる。図 3.7 は、Denoising Autoencoder が働く様子を、画像の入力レイヤーの場合について表している。

また、図 3.8 は、元の入力データ群から外れた入力データを作り、これを復元している様子を模式的に表している (*9より引用)。データの一部を隠すことによって、より robust な素性の作り方のモデルを得ることが出来ると考えられている。

なお、Denoising Autoencoder の学習過程には、Restricted Boltzman Machine のような Energy-Based Model との類似性があることがわかっている [91]。

Stacked Denoising Autoencoder

Stacked Denoising Autoencoder(SDA) は、Deep Belief Nets の pretraining で、RBM の代わりに Denoising Autoencoder(以下 DA) を用いたものである。DBN 同様、前の層の Representation レイヤー = 次の層の Input レイヤーとなるように、DA で学習したレイヤーを積み重ねていく。最終レイヤーには、DA ではなく Softmax レイヤーなどを置き、最終的な確率を出して、分類を行わせる。図 3.9 は、SDA にて、どのように pretraining が繰り返され、DA が積み上げられていくのかを表している。いったん pretraining が済んだら、そのレイヤーの入力は隠さず、学習された重みをそのまま使って高い抽象度の素性を作る。そして、その新しい素性を入力と捉えて、新たに Denoising

*9 http://www.iro.umontreal.ca/~bengioy/talks/icml2012-YB-tutorial.pdf

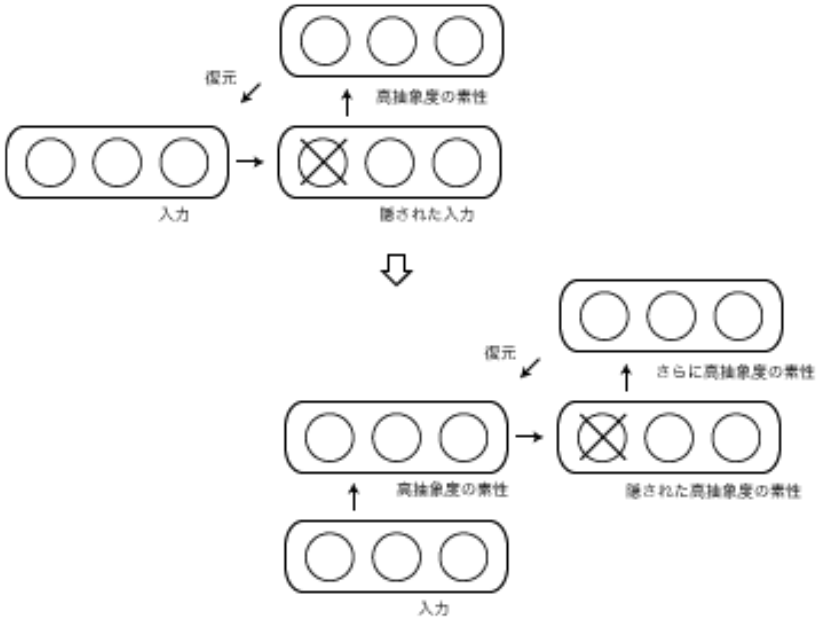

図 3.9　Stacked Denoising Autoencoder における学習の進行

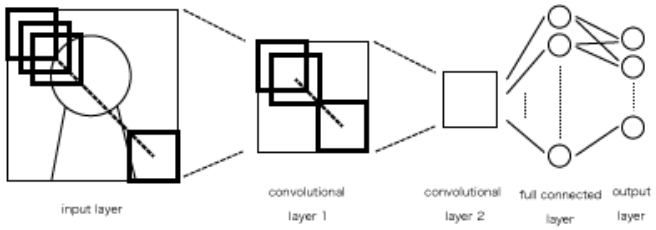

図 3.10　Convolutional Net の仕組み

Autoencoder の学習を行わせる。これを繰り返してレイヤーを積み上げ、最終的に良い素性を学習しやすい MLP を作ることで、finetuning 段階でも学習がスムーズに進み、精度のよい推論ができるようになる。

3.2.3　Convolutional Network

Convolutional Network(畳み込みネットワーク) は、画像認識にて使われるニューラルネットワークの一種である。DBN や SDA のようにユニットが 1 対 1 で接続されるのではなく、画像上で距離が近いユニットを一まとめに考えるが特徴である。具体的には、フィルターと呼ばれる小さな四角形を、画像の上で動かしていき、フィルターが覆っている部分の性質を抽出して次のレイヤーに渡していく。フィルターは行列で表現されており、フィルター行列の各要素と、画像の各画素を掛け算し、その和を取ることで、次のレイヤーに渡す値が得られる。基本的な構造は、図 3.10 のようになっている。この図では、2 回の convolution を行ったあと、十分に画像の素性が得られたところで、従来の MLP で見られる隠れレイヤーを 1 つ介して、出力を得ている。

Convolutional Net の発祥は DBN や SDA よりも古い。画像上の各部分ごとに性質を抜き出し

ニューラルネットワークを構成するというアイデアは、人間の視覚野の仕組みを参考にしたもので、1980 年の Neocognitron を始めとして [29][30] 様々な形で提唱されている [57][85]。2003 年に提唱された Convolutional Net のモデルが、画像認識ベンチマークの MNIST にて、今でも 5 位の精度を保っており [86]、2012 年のものは 2 位となっている [18]。Fisher ベクトルなど既存の画像認識手法との組み合わせでも、良い性能を発揮できることがわかっている [69]。ここで紹介するのは、1998 年に LeCun らが紹介した方法 [57] であり、ほとんどの Convolutional Network に使われているアルゴリズムの基本的な部分である。

Feature Map

実際の Convolutional Network では、Feature Map と呼ばれる方法が使われている。これは、filter が 1 つだけでなく、複数種類の filter を、複数枚の画像の上で動かしていく方法である。図 3.10 の構造に、Feature Map の考え方を適用すると、3.11 のようになる。この図では、フィルターが移動する様子は省略している。四角形は Feature Map1 つを、円は MLP のユニット 1 つを表している。Feature Map 同士では、filter による畳み込みによって情報が伝達される。Feature Map から MLP の隠れレイヤーにつながる部分では、まず全ての Feature Map をつなげ、1 次元のベクトルに並び替えて入力したと考えている。残りは、通常の MLP と同じように処理される。(なお、図 3.10 では、簡単のため、Feature Map は表現されていない。)

1 枚の map が、1 書類の filter に対応しているので、filter の数と同じだけ次のレイヤーの Feature Map が用意されている。それぞれの filter を、全ての Feature Map の上で動かして、畳み込み計算を行っていく。最終的に、普通の MLP に帰着させるところは変わらない。

Feature Map は、カラー画像を扱うときにも利用できる。カラー画像データは、ほとんどの場合 Red、Green、Blue の光の 3 原色に対応する 3 つの値にて表現される。これを省略して RGB と呼ぶ。Convolutional Net の入力レイヤーにて、画像サイズと同じ大きさの、RGB の 3 色に対応する 3 つの Feature Map を設定することで、色情報を簡単に扱うことができる。

Max-pooling

Convolutinal Net にて、もう 1 つ広く使われている方法が、Max pooling である。Max pooling layer は普通 Convolutional Layer 同士の間に置かれる。Max pooling では、convolutional layer で得られた画素を、部分ごとにたたみ込んでいく。ただし、convolutional layer の計算に使う filter とは違い、重み行列は用いず、単純に対象領域の画素値の max を取って、次のレイヤーに渡していく。また、convolutional layer では filter を 1 ピクセルずつ動かしていくが、max pooling layer では、一度に filter のサイズと同じだけ動かし、畳み込み対象領域が重ならないようにする。また、filter のサイズは 2x2 ピクセルに設定されることが多い。図 3.12 の左側は、max pooling による畳み込みの

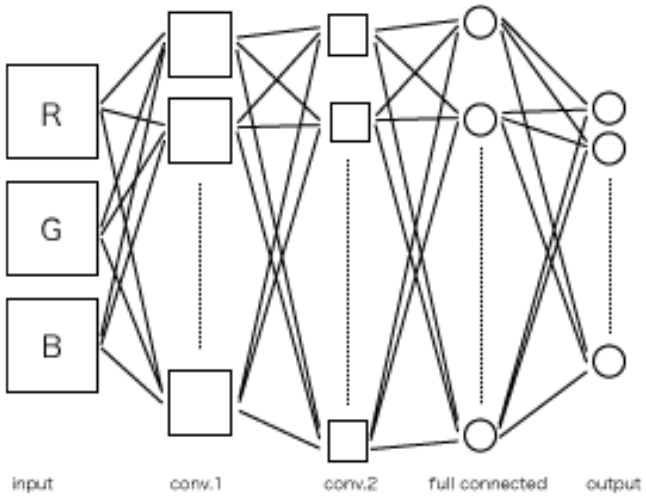

図 3.11　Convolutional Net と feature map

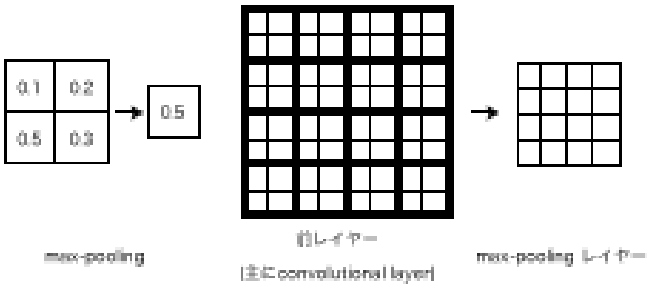

図 3.12　Maxpooling と、Maxpooling Layer の作用

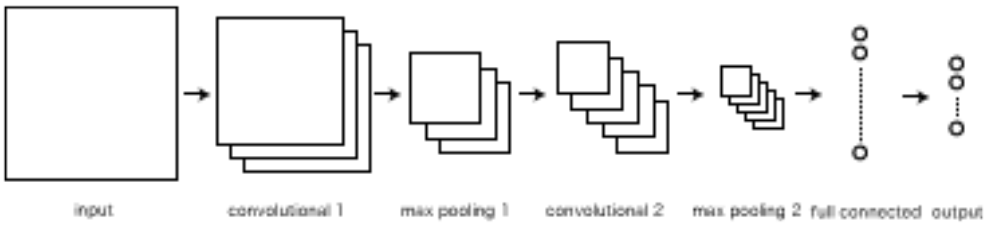

図 3.13　max pooling layer を取り入れた Convolutional Network

様子を、右側では Max-pooling Layer による畳み込みの全体の様子を表している。

Max-pooling を用いることにより、各レイヤーの次元数を効率良く下げることが出来る。また同時に、ピクセルの位置が少し変化しても同じ結果が得られるので、位置的な robustness を上げることも出来る。

Max pooling Layer を併用した場合の、一般的な Convolutional Network の全体図を、図 3.13 に挙げる。なお、この図では、各 Feature Map ごとのつながりは省略している。

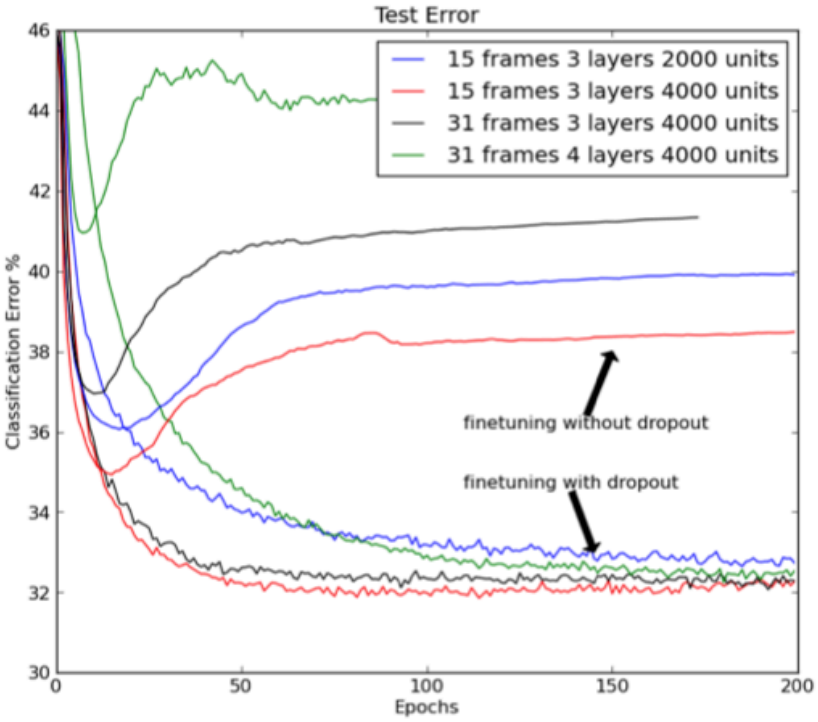

図 3.14　Dropout による TIMIT データベースの識別結果

3.2.4　Dropout と DropConnect

Dropout とは、2012 年に Hinton 氏らによって提案された、過学習を防ぐための技術である [41]。過学習とは、機械学習全般において使われる用語で、学習モデルが訓練中に見たデータへの適応を重視し過ぎたために、未知のデータをうまく識別できなくなる現象のことを指す。Dropout では、過学習を防ぐため、各レイヤーからの出力をランダムに消去してしまう。これによって各ユニットには、他のユニットに頼らず自力で学習をする必要が生じ、より様々な入力に対応しやすい堅固 (robust) なモデルが獲得される、と考えられている。図 3.14 は、[41] より引用した図で、音声認識の TIMIT データベース [27][*10]に対する識別精度である。Dropout を使うことで、使わなかった場合に比べ、特に Epoch 数 (学習の繰り返し回数) が大きい場合に、分類誤差を減少出来ていることがわかる。

DropConnect は、Dropout をさらに一般化した手法である。DropConnect では、各ユニットからの出力ではなく、ユニット同士のつながりをランダムにカットしている。図 3.15 は、DropConnect の著者による紹介 Web ページ[*11]に掲載されている画像の引用である。ニューラルネットワークにおけるニューロン同士の接続が、ランダムに切断されている様子を表している。

[*10] http://catalog.ldc.upenn.edu/LDC93S1
[*11] http://cs.nyu.edu/~wanli/dropc/

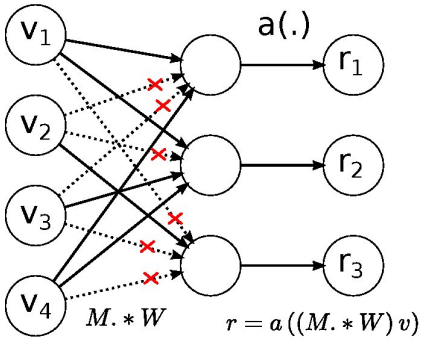

図 3.15　DropConnect の模式図

3.2.5　活性化関数

各ユニットに入力された値は、ある一定の関数によって増幅変換される。これは人間の神経細胞を真似た仕組みで、神経細胞の化学物質による活性化にちなんで、活性化関数 (activation function) と呼ばれる。この項では、ニューラルネットにおける様々な活性化関数を紹介する。

heaviside 関数

始めにニューラルネットワークの活性化関数として用いられたが Heaviside の階段関数である。この関数は負値に 0、正値に 1 を返す。数式では、

$$H(x) = \begin{cases} 0 & (x < 0) \\ 1 & (x > 0) \end{cases} \tag{3.4}$$

となる。ただし、これだけでは $x = 0$ の点で不連続になるので、$H(x) = 0$ や $H(x) = 0.5$ などの定義が用いられる。ヘビサイド関数のグラフを 3.16 に示す。

heaviside 関数は、ニューロンが受けた刺激の総和を測り、閾値を超えていれば次のニューロンにも情報を伝達し、超えていなければ何もしないという性質を、そのまま表現している。しかし、この関数は微分不可能なため、バックプロパゲーションを行う上では不都合であり、特に Deep Learning のネットワークモデルにおいて使われることは少ない。

sigmoid 関数と hyperbolic tangent

シグモイド関数は、ヘビサイド関数に似ているが、全域にて非線形かつ微分可能である。この 2 つの特性により、バックプロパゲーションに適している。以下の式で表される。

$$S(x) = \frac{1}{1 + e^{-x}} \tag{3.5}$$

また、微分した形が同じシグモイド関数で表されるという特徴があり、実装が容易である。

$$\frac{d}{dx} S(x) = \left(\frac{1}{1 + e^{-x}} \right) \left(\frac{-e^{-x}}{1 + e^{-x}} \right) \tag{3.6}$$

$$= S(x)(1 - S(x)) \tag{3.7}$$

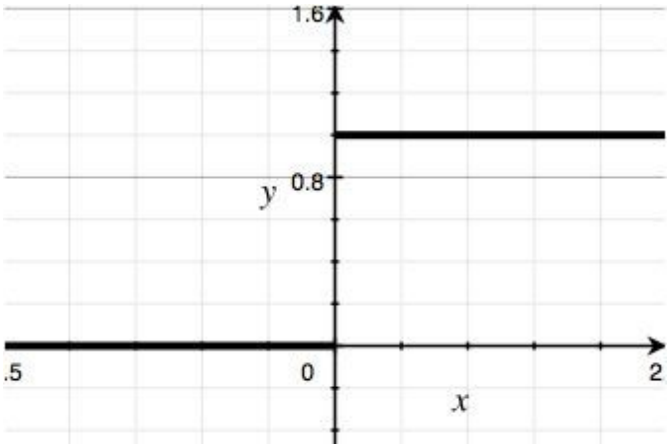

図 3.16　Heaviside の階段関数

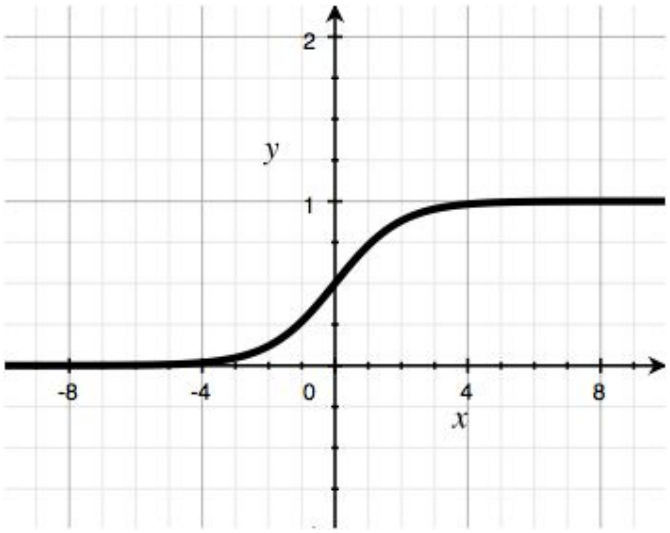

図 3.17　Sigmoid 関数

シグモイド関数のグラフが図 3.17 である。Heaviside 関数を $x = 0$ の付近のみ滑らかにしたような形をしている。この関数は、x の絶対値が大きくなるにつれ、y の変化が非常にに小さくなっている。そのため、ユニット間伝達において、入力と重みの積が大きくなってしまうと、その部分は誤差のバックプロパゲーションに対する反応が小さくなってしまい、学習が進みにくくなってしまうという弱点がある。なお、hyperbolic tangent 関数、通称 tanh 関数を用いることもある。sigmoid 関数と tanh 関数は線形の差しかないため、ニューラルネットワークの性能という面では、どちらを使用しても等価である。

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3.8}$$

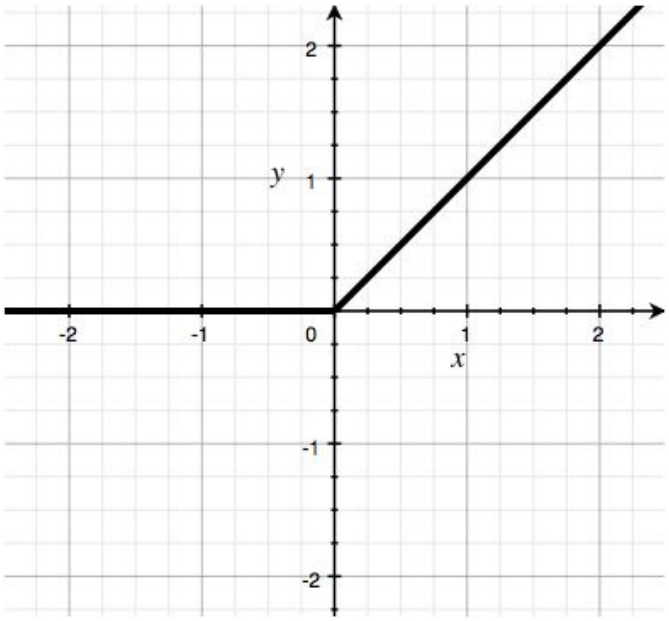

図 3.18　Rectifier 関数

$$S(x) = \frac{tanh(x/2) + 1}{2} \tag{3.9}$$

Rectifier Unit

Rectifier 関数は、最近のニューラルネットワークにおいてよく用いられている活性化関数である。

$$f(x) = max(0, x) = \begin{cases} 0 & (x < 0) \\ x & (x > 0) \end{cases} \tag{3.10}$$

Rectifier 関数が性能向上に役立つ理由は、まだ解明されていない。

Maxout Unit

Maxout Network とは、バックプロパゲーションを伴う単純な多層 MLP 構造に、3.2.4 で述べた Dropout 法を組み合わせ、さらに Maxout Unit という特殊な活性化関数を併用したニューラルネットワークである [38]。Maxout Network の 1 レイヤー分の構造を、図 3.19 に示す。

Maxout Unit では、複数の線形ユニットの出力について、max を取っている。これにより、任意の凸関数を近似することが出来る。図 3.20 は、Maxout Unit が凸関数を近似する様子を、1 次元の入力について示している。そして、複数の maxout unit の線形和を取ることは、複数の近似された凸関数の線形和を取ることである。これによって、最終的に任意の連続関数を近似できることが証明されている。

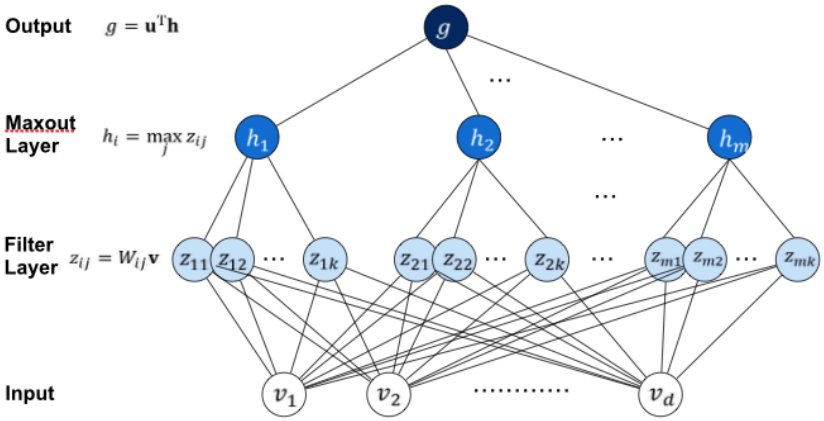

図 3.19　Maxout Network の構造図

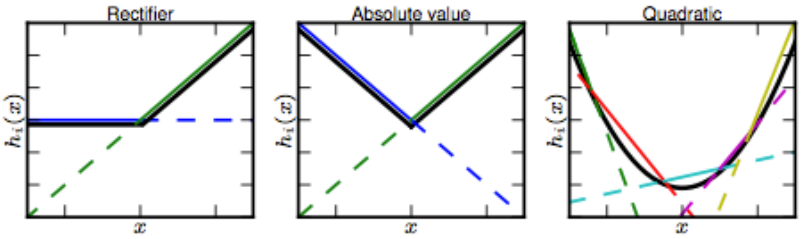

図 3.20　Maxout Unit が凸関数を近似する様子 (1 次元の場合)

第 4 章

深層学習の実装における技術

この章では、Deep Learning のアルゴリズムを実装して、実際の問題に適用するに当たって、どのようにすれば良い結果が得られるか、紹介する。

4.1 評価基準

Deep Learning のアルゴリズムを使用するための具体的な手段を選ぶにあたり、次のような判断基準を設けた。優先順位は、1 → 2 → 3 の順に高い。基準 1. Deep Learning が注目された大きな理由である、高い識別精度を再現できる。基準 2. 学習にかかる時間が、他のアルゴリズムに比べて、極端に長くならない。基準 3. 利用にあたって必要なプログラミング量が出来るだけ少なく、バグが混入しにくい。基準 1 は最優先目標である。今回 Deep Learning を選択した理由は、Web 工学のタスクにおいて、高い分類精度を実現するための最も有力な方法だと思われるからである。つまり、例え Deep Learning のアルゴリズムとして正しいプログラムだったとしても、分類精度が従来の分類器より劣っていれば意味はない。従来の分類器をそのまま使い続ければよいことになってしまう。基準 2 と 3 は、現実の問題を Deep Learning で解決する際に、必要となってくる視点である。プログラムの作成や、学習の実行にかかる時間が、あまりにも長くなってしまうと、実際のビジネスや、刻一刻と変化する Web サービスに対して応用するのは非現実的だろう。

4.2 既存ソースコード使用の利点

Deep Learning のプログラムを実行するために、まず大きく分けて、「自分でソースコードを全て書く」方法と、「主に既存のソースコードを利用する」方法の 2 つが考えられる。今回は、既存ソースコードをベースに使うことを選択した。以下、その理由を説明する (図 4.1 参考)。

「既存のソースコードを利用する」場合のメリットとして、「開発期間は基本的にゼロで済む」「新たにバグが混入する危険性が無い」「自分の改造コードが、他のライブラリ利用者によって使ってもらえるチャンスが大きい」という点が挙げられる。デメリットとして、「ソースコード中にブラック

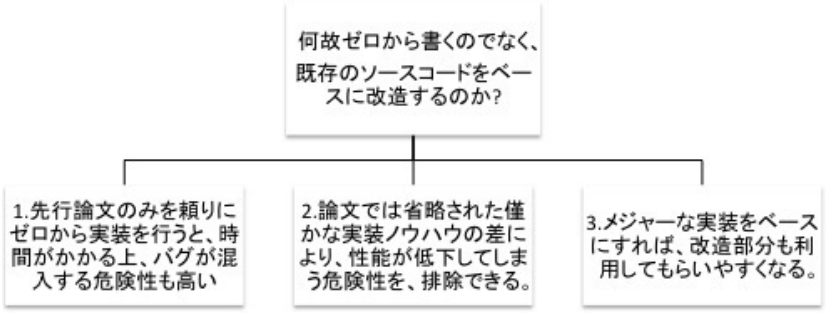

図 4.1　既存コードの利用における、完全オリジナルコードの作成に対する利点

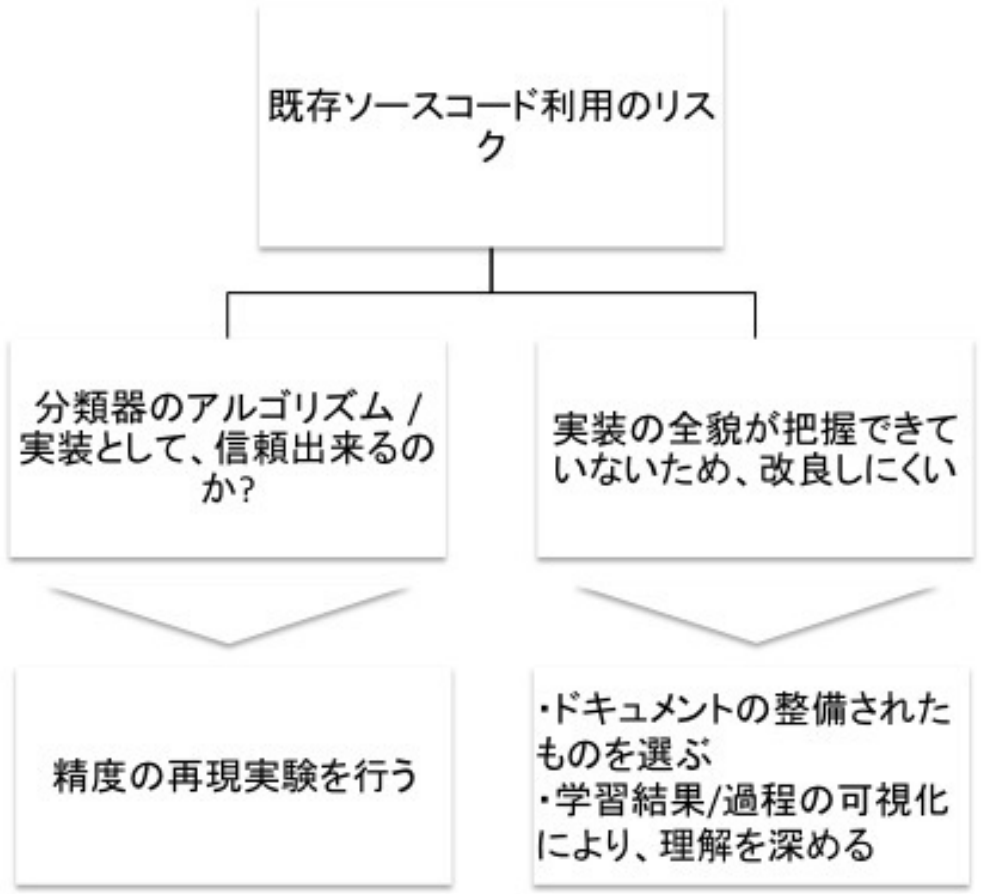

図 4.2　既存ソースコード利用のリスクと、対処方法

ボックスが増え、改造にかかる時間が短くなる」「全く新しいアルゴリズムを実装する場合、ゼロからスタートした方が早く書けるケースもある」などが想定される。しかし、基準 3 の「プログラミング量が少なくバグのリスクが低い」という点において優れているため、今回は既存のソースコードを探して利用していくことにした。既存ソースコードを利用する際に生じるリスクについては、図 4.2 のように対処できる。

なお、「自分でソースコードを全て書く」場合のメリットとデメリットは、上記「既存のソースコード」の場合の逆となる。自分で新たなコードを書くので時間がかかり、バグの混入リスクも大きい。

また、ソースコードを公開した場合の、ライブラリとしての信頼度も、ゼロから築かなければならない。ただし、コードの詳細部分を改造する段階では、自分の手で書いたコードを使う方が、より深い理解を得やすく、確実で素早い実装ができる可能性もある。

4.3 Graphics Processing Unit の利用による高速化

現代の GPU は、単にグラフィック処理に特化したプロセッサというだけでなく、その並列演算能力を活かして、大量の演算を行う科学計算用途にも使われている。GPU 設計・製作メーカーの NVIDIA からは、グラフィック出力機能を敢えて外し、科学計算用途に特化した、Tesla というハイエンド GPU シリーズも発売されている[*1]。

Deep Learning の研究においても、GPU による並列計算が有効とされている。これは、入力ベクトルと重み行列の積など、ニューラルネットワークの計算にて中心を占める行列演算が、並列計算の恩恵を受けやすいからである [7]。例えば、Dropconnect の研究に拠れば、GPU で演算した場合、テクスチャメモリを利用しなくても CPU より 97.2 倍高速化でき、テクスチャメモリを利用すれば 414.8 倍の速度を出すことも可能である [93]。

また、Deep Learning の実装によっては、はじめから GPU で高速化することを前提に、GPU 専用のコードを書いている場合がある。例えば、後述する cuda-convnet[*2]や、Maxout Network[38] のコードの一部、DropConnect[93][*3]が該当する。この場合、そもそも GPU を搭載したマシンを使わないと、コンパイルや実行が全く出来なくなってしまう。

GPU が高速に演算を実行できる理由は、その構造にある。1〜数個のプロセッサで様々な命令をこなす CPU に対し、GPU は数百〜数千個の core を持っており、同一の命令を高速に並列演算する用途に向いている。これは、大量の点の変換処理が必要となるグラフィック処理環境から生まれた GPU の強みである。かつて GPU を科学演算に用いるには、OpenGL や Cg のようなグラフィック処理用ライブラリを使ってアルゴリズムを記述しなくてはならず、非常に難易度が高かった[*4]。そこで、NVIDIA は CUDA という GPU による演算の開発環境を開発した [32]。CUDA は C/C++ 言語を拡張した作りになっており、グラフィックライブラリに比べて記述は非常に容易である。また、Python にて CUDA コードを生成するライブラリも存在している [49][48]。

GPU による並列演算は高速だが、GPU のメモリと CPU のメモリとの間で起こる通信は、比較して低速であり、ボトルネックになることが多い。通信の回数を減らし、GPU 上のみで完結する演算を増やすことが重要である [7]。

[*1] http://www.nvidia.co.jp/object/tesla-supercomputing-solutions-jp.html
[*2] https://code.google.com/p/cuda-convnet/
[*3] http://cs.nyu.edu/~wanli/dropc/
[*4] http://www.nvidia.co.jp/object/what-is-gpu-computing-jp.html

4.4 Deep Learning のライブラリ

Deep Learning に利用できるライブラリは、まだ標準が固まっていないものの、論文と共にソースコードを公開するパターンが多く、いくつか出回っている。この節では、Deep Learning 関係のライブラリやソースコードを紹介する。

4.4.1 cuda-convnet

CUDA と C++ による Convolutional Network の実装が、CIFAR データセットの制作者である Krizhevsky 氏によって公開されている[*5]。max-pooling layer や最終レイヤー用の softmax layer など、Convolutional Net の実装に必要なパーツが一通り用意されている。なお、後述する pylearn2 にも、この cuda-convnet の Python ラッパクラスが実装されている。

4.4.2 word2vec

何度か触れてきたが、word2vec のコードも全て公開されており、誰でもテキストデータを入れて、意味ベクトルを生成できるようになっている[*6]。word2vec は、入力されたテキストデータから、まず単語の辞書を作り、次にそれぞれの単語の意味をベクトル化していく。ベクトル化が済めば、単語同士の類似度を計測したり、クラスタリングに用いることができる。

4.4.3 数値計算ライブラリ Theano

Theano は、python 上で記述される数式処理/数値計算ライブラリである。Theano は、数式のコンパイルと実際のデータによる数値計算の 2 段階で動作する [9]。図 4.3 に、Theano が動作する仕組みを示した。Theano では、数式は文字シンボルを用いた文字式で記述できる。記述された文字式は、内部的には計算木 (グラフ) として蓄積されている。Theano は、実際の計算を行う前に、このグラフをコンパイルして、C による CUDA のコードに変換している。このときデバッグの難しい数値計算の誤差絡みの部分、0 による除算など危険な処理を、自動的に解析してくれる。ニューラルネットワークの演算過程では、非常に小さい数値が出現することがあり、また、文字式を分析することで、計算グラフも最適化してくれる。プログラマは、プログラム上の些末な計算テクニックに囚われることなく、数式の本質的な部分の記述に集中することができる。また、微分も自動で行ってくれる。ニューラルネットワークでは、様々な文字の微分が必要になり、多層にすると爆発的に式が複雑になる。これを手計算による文字式の微分をしてから書いていくと、層を増やす度に手計算が必要に

[*5] https://code.google.com/p/cuda-convnet/
[*6] https://code.google.com/p/word2vec/

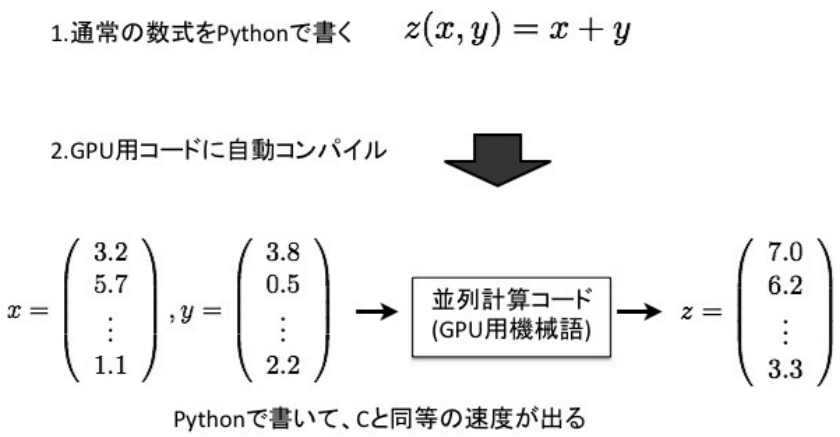

図 4.3　Theano の動作過程

なり、非常に煩雑で、開発効率が落ちてしまう。Theano の自動微分と自動最適化は、pylearn2 の内部記述の簡略化を大いに助けているおいて、多層のニューラルネットワークを使うときでも、文字式に対する大量の手計算と、大量のプログラムの式を書くことなく、簡単に微分やニューラルネットワークを拡張することができる。

4.4.4　Deep Learning Tutorial

Deep Learning Tutorial は、Theano を用いて記述されたソースコードで、Theano と同じく LISA.Lab のメンバーによって書かれている。内容的には、単純な Logistic 回帰から始まり、最終的に Convolutional Net, Stacked Denoising Autoencoder, Deep Belief Net の 3 つの原理が理解できるよう構成されている。ソースコードのほぼ全ての部分に、詳細なコメントが書いてあり、Deep Learning のアルゴリズムと同時に Theano を用いた実装上の注意点なども理解できるような作りになっている。

一方で、提供されているコードは簡単のため、アルゴリズムの基本的な部分に絞られており、最新の研究成果が反映されているわけではないため、注意が必要である。例えば、Convolutional Net のソースコードには、1998 年の LeCun らによる研究 [57] をベースにしていると書いてある。しかし、元の論文の段階で既に、2012 年に発表された最も優れている Convolutional Neural Net[18] に比べて、3 倍以上誤差が大きくなってしまっている。またあくまで Tutorial という位置づけのため、Epoch 数などのハイパーパラメータが最も誤差が低くなるようには設定されていない。

なお、Deep Architecture の形にはなっていないが、Denoising Autoencoder の代わりとして、Contractive Autoencoder[79][78] の実装も提供されている。これは、Denoising Autoencoder の亜種だが、誤差測定時に適切なペナルティ項を与えることで、学習される表現の robustness がさらに向上するというものである。

4.4.5 機械学習ライブラリ Pylearn2

pylearn2 は、python で記述された、Deep Learning など機械学習アルゴリズムを使うためのライブラリである [37]。モントリオール大学の LISA Lab. の方がメインとなって、github 上で現在も開発が続けられている。LISA Lab. を中心とする開発者によって、ソースコードの開発がほぼ毎日行われている。以下、pylearn2 の長所を記す。

高精度アルゴリズム Maxout Network の実装

Pylearn2 には、3 章にて述べた、Maxout Network という Deep Learning のアルゴリズムの一種が実装されている。このアルゴリズムは、画像認識タスクにおいて非常に高い精度を実現しており、基準 1 を満たす上で都合が良い。

Theano の利用

pylearn2 のプログラムは、前述した Theano を全面的に利用して記述されている。Theano を用いたことで、可読性が大きく上昇している。また、何らかの理由で、pylearn2 の部分的拡張が必要になったときでも、[過去のスライドを基に加筆]

高度な拡張性と可読性

pylearn2 のソースコードは、拡張性や可読性を非常に強く意識して書かれている。基本的な使い方を記したチュートリアルの Web ページが存在し、主なソースファイルには、詳細なドキュメントも記述されている。また、モジュール化が丁寧なので、自分で書く部分が少なくて済む。例えば、データセットを変更する場合は、Dataset クラスのみを書き下せばよく、他の部分に対してコードを書く必要がほとんどない。図 4.4 に、pylearn2 において機械学習のアルゴリズムがどのようにモジュール化されているかを示す。

また、プログラムを改造する場合でも、数値パラメータを変更するだけであれば、設定ファイルのみの変更で完結させることができる。この設定ファイルは、YAML(YAML Ain't a Markup Language) 形式に少し独自拡張を加えた形式になっている。

プログラム本体と、数値の設定ファイルが分離されていることで、プログラミングが不得手な人でも、比較的簡単に設定を変更することができる。

pylearn2 の主なパッケージ

現在の pylearn2 のソースコードは、使われているものと非推奨のものが混在している。これは、pylearn2 に定められた API 変更のルールのためである。pylearn2 では、「変数やクラスを消去する半年前に、ライブラリ上で告知しなければならない」という管理方針を定めている。これは、pylearn2

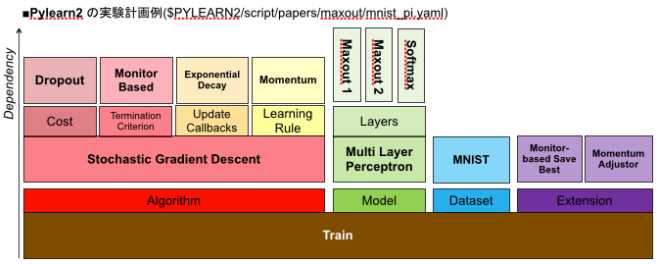

図 4.4　Pylearn2 の実験計画例

の安定性に寄与しているが、一方で、ソースコードリーディングの際には使われていないパッケージがノイズになることも少なくない。ここでは、pylearn2 で現在使われている主なパッケージについて紹介する。

- **costs** 最終レイヤーで誤差を計算するときに使われる、損失関数 (loss function) を集めたパッケージである。主に SGD によって利用される。Dropout、Lp ペナルティ、クロスエントロピー関数、Negative Log Likelihood 関数などがここに含まれている。また、DBN や Autoencoder のように、独自の損失関数が必要な場合も、ここで定義されている。
- **datasets** MNIST や CIFAR10, SVHN などのデータセットがクラス化されている。新しいデータセットに対して Deep Learning を使いたい場合は、ここに定義されている Dataset クラスや DenseDesignMatrix クラスのサブクラスを作成することになる。
- **distributions** ガウス分布などの分布関数が集められている。
- **gui** 主に重みフィルタを可視化するためのクラスが用意されている。
- **linear** Theano の線形変換クラスを作り直したもので、Convolutional Layer もここに含まれる。
- **models** Autoencoder、RBM などの学習モデルが定義されている。
- **sandbox** cuda-convnet のラッパクラスが定義されている。
- **scripts** 実際に学習を行わせたり、学習結果を集計するときに便利なスクリプトが集められている。

4.5　ハードウェアの構成

Deep Learning に限った話ではないが、機械学習のタスクは長時間の計算を必要とする場合が多く、プログラムの実行時間が 1 日以上かかることも稀ではない。普段使用している PC が使えなくなると困る場合は、機械学習を動かしておくための専用サーバを用意することが望ましい。

GPU は必ずしも搭載していなくとも良いのだが、実践上 Theano や CUDA の性能を最大限に生

かすためには、GPU を搭載しなければならない。また、CUDA を用いて GPU マシン専用に書かれたアルゴリズムを動作させるためには、GPU が必須となる。GPU は、NVIDIA 製で、CUDA に対応していなければならない。ただし、グラフィック出力の機能はなくても構わない。2013 年 1 月現在、NVIDIA の Web ページに掲載されている GPU に関しては、全て CUDA に対応しているため、店舗で新品の GPU を購入する場合は基本的に問題は生じないと思われる。

NVIDIA の GPU の中で、どれを選ぶかも問題になる。NVIDIA のホームページでは、GPU のうち Tesla というシリーズが GPGPU に特化しており、非常に高精度/高速な演算を行うことができる、と述べられている。しかし、非常に高価な上、Tesla シリーズは一般的なグラフィックカードではなく、サーバ全体の購入を基本としている。販売員の方によれば、Tesla は主に商用の大規模データや、ミスの許されない長時間科学計算などに用いることを想定して作られている。今回の目的は、Web 工学の一般的タスクにおける Deep Learning 技術を確立するという、いわば実験的な用途であり、Tesla の利用はオーバースペックだと思われる。

NVIDIA の GPU は、Tesla を除くと、Quadro と GeForce という 2 つのシリーズに分かれている。店舗の販売員の方に詳しい話を伺ったところ、Quadro はコンピュータグラフィックの出力機能に注力したシリーズであり、GeForce はより計算性能重視という傾向をもっている。つまり、Deep Learning の計算など GPGPU に用いる場合、同じ価格帯ならば、GeForce シリーズの GPU を用いた方が、費用対効果が大きくなると考えられる。

また、販売員の方によれば、GeForce シリーズの中でも Titan という機種は、Tesla 用の部品の中で品質チェックに漏れてしまったものを流用しており、Tesla とほぼ同様の、非常に高い性能を発揮することが出来る、とのことだった。しかし、今回利用する中で最も計算量が大きい Maxout Network について、元の発表論文によれば、GeForce シリーズの GTX 580 という、少し世代が前の GPU が用いられている。Deep Learning を実行させる上で、GPU が存在すること自体は非常に重要だが、必ずしも最新スペックの GPU が必要というわけではないことがわかる。

今回の構成では、知の構造化センターの中山浩太郎先生のアドバイスもあり、Titan ではなく、同じ GeForce シリーズの GTX 760 という GPU を搭載することにした。

第 5 章

Web 工学への深層学習技術の応用

この章では、第 4 章で得られた Deep Learning 用の構成を用いて、実際に機械学習のタスクを実行する。分類精度、実行時間、消費メモリ量を調べることにより、Deep Learning を Web 工学の問題に適用する際の参考とする。

5.1 利用したデータセット

実験の前提知識として、この節では、実験に使用したデータセットについて記す。機械学習の分野では、分類精度のベンチマークを取るために、様々なデータセットが提供/提案されてきている。同じデータセットに対して、様々な分類モデルやアルゴリズムを用いて分類実験を行うことにより、どの手法が優れているのか比較することが出来る。

ここでは、画像認識のデータセットを用いて、Deep Learning プログラムのベンチマークを行う。画像データを用いる理由は、1 つには、画像認識が Deep Learning が最も高い分類性能を実現している分野だからである。加えて、画像データや画像から抽出された素性は、可視化が比較的容易なことが多い。可視化することで、学習過程を目で見て確かめることが出来るため、アルゴリズムの分析を行いやすい、という利点がある。

5.1.1 MNIST

MNIST database[*1] とは、手書き数字を画像分析によって認識するベンチマークタスクである。このデータセットは、National Institute of Standards and Technology(NIST) が提供する手書き数字のデータにサイズの規格化処理を加え、数字の書き手毎に整理したものである。データはサイズ 28x28 ピクセルの白黒画像 70000 件より構成される [57]。それぞれの画像は、0〜9 のいずれかの数字 1 文字に該当している。画像データを認識プログラムに入力して、どの数字に該当するか判別させることで、画像識別の精度を競うことになる。図 5.1 に、MNIST の画像データの一部を示す ([57]

[*1] http://yann.lecun.com/exdb/mnist/

図 5.1　MNIST データの例

より引用した)。

MNIST の 70000 件のデータは、60000 件のモデル訓練用データと、10000 件の精度測定用データに分かれている。まず 60000 件のデータを使って識別モデルに学習を行わせた後、未知の 10000 件のテストデータを実際に識別してみることで、精度を測定する。テストデータを識別している最中に、更なる学習を行うことは許されない。分類器は、過去に学習に使ったデータだけ良く識別できても意味がなく、むしろ未知のデータこそ精度良く分類出来なければならない。モデルが過去のデータに特化してしまうと、かえって未知のデータに対する識別精度が低下することもある。この低下現象は過学習 (over fitting) と呼ばれ、機械学習における落とし穴の一つとなっている。MNIST に限らず、機械学習用のデータセットにて、訓練データとテストデータをあらかじめ分けておくことは一般的であり、学習アルゴリズムが過学習を防げているかどうか、判定するために有効な方法の 1 つとして知られている。

MNIST は、様々な画像認識アルゴリズムの作者によって使用されている。MNIST の配布 Web サイトには、MNIST を利用した論文のリストが、分類誤差によるランキング形式で掲載されている。表 5.1 に、MNIST データセットにおける各アルゴリズムの分類誤差を、ランキング形式の Web サイト[*2]より再構成して示す。

5.1.2　CIFAR10

CIFAR10[*3]は、写真を画像分析によって識別するタスクである [51]。入力データは 60000 件のカラー画像で、どの画像も、飛行機、自動車、鳥、猫、鹿、犬、蛙、馬、船、トラックの 10 種類のクラスのどれかに属しており、各クラス均等に 6000 枚ずつの画像が割り振られている。画像データは、Google や Flickr などによる Web 検索で集められた画像を、人手でラベリングして作られている。

[*2] http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html
[*3] http://www.cs.toronto.edu/ kriz/cifar.html

表 5.1　MNIST の分類誤差ランキング

手法	発表学会/雑誌/年
DropConnect [93]	ICML 2013
Multi-column Deep Neural Networks [18]	CVPR 2012
Deep Big Simple Neural Nets [17]	Neural Computation 2010
Energy-Based Model [75]	NIPS 2006
Convolutional Neural Networks [86]	Document Analysis and Recognition 2003
Maxout Networks [38]	ICML 2013
COSFIRE filters [2]	PAMI 2013
Multi-Stage Architecture [45]	ICCV 2009
Deformation Models [47]	PAMI 2007
A trainable feature extractor [55]	Journal Pattern Recognition 2007
Invariant Support Vector Machines [24]	Machine Learning 2002
Sparse Coding [53]	TNN 2008
Unsupervised learning of invariant feature hierarchies [76]	CVPR 2007
shape contexts [6]	PAMI 2002
Receptive Field Learning [46]	CVPR 2012
Sparse Activity, Sparse Connectivity [90]	JMLR 2013
Convolutional Deep Belief Networks [59]	ICML 2009
Deep Encoder Network [66]	2009
Deep Boltzmann Machines [83]	AISTATS 2009
Deep Belief Networks [23]	2008
Convolutional Neural Networks [86]	2003
neural networks [40]	2006
Deep learning via semi-supervised embedding [94]	2008

また、サイズを 32x32 ピクセルに縮小されている。

MNIST と同じように、60000 件のデータは、50000 の訓練データと 10000 件のテストデータに分割されている。まず、分類器にこれらの訓練データを読み取らせて、学習を行わせる。その後、テストデータのうちいくつを正しいクラスに分類することができるのか、分類精度を競うことになる。

CIFAR10 についても、データの例を 5.2 に、分類誤差のランキング Web サイト[*4]より、再構成した表を 5.2 に示す。

[*4] http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html

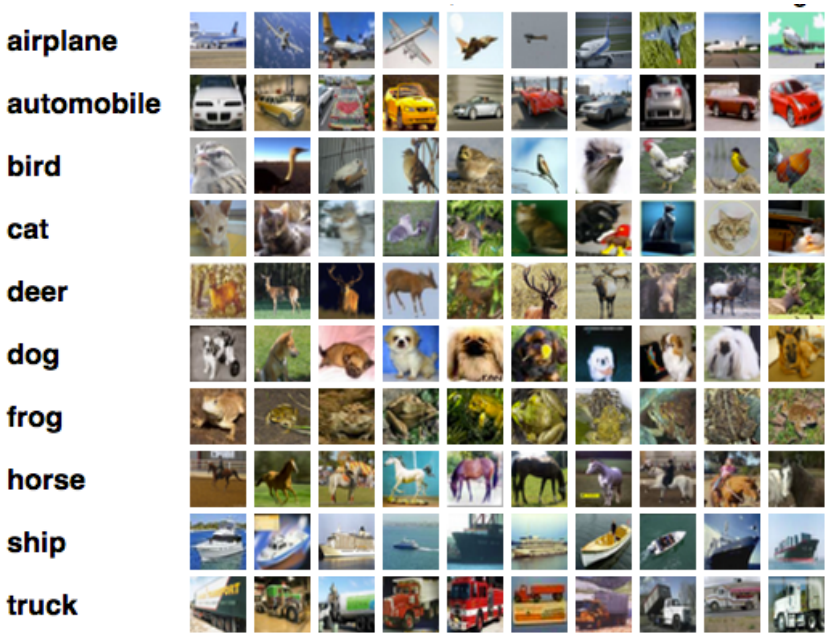

図 5.2　CIFAR10 データの例

手法	発表学会/雑誌	分類誤差
DropConnect [93]	ICML 2013	9.32 %
Maxout Networks [38]	ICML 2013	9.35 %
Bayesian Optimization [88]	NIPS 2012	9.50 %
Deep Convolutional Neural Networks [52]	NIPS 2012	11.00 %
Multi-Column Deep Neural Networks [18]	CVPR 2012	11.21 %
Deep Convolutional Neural Networks [96]	arXiv 2013	15.13 %
Dropout [41]	arXiv 2012	15.60 %
Sum-Product Networks [34]	NIPS 2012	16.04 %
Single-Layer Networks [21]	AISTATS 2011	20.4 %

表 5.2　CIFAR10 分類誤差のランキング

5.2　使用したライブラリ

この章の実験には、pylearn2 という Deep Learning のライブラリを用いた。

5.3　使用したハードウェア

この章の実験では、GPU を搭載したサーバマシンを用いた。表 5.3 に、今回の実験で使用したマシンの主な使用パーツ及び性能を列挙した。また、Maxout Network の実装ソースコードには、実験

	GPU	CUDA core	CPU	クロック数	メモリ容量
今回	GTX760	1152	Intel CPU Core i7 4770S	3.1 GHz	32GB
論文	GTX580	512	Intel Xeon CPU E5620	2.40GHz	(記載無し)

表 5.3　ハードウェア性能の比較

表 5.4　Maxout Network による MNIST(2 次元) 分類の結果

手法	データセット	元論文の誤差	実験誤差	増加分
Maxout Network	MNIST(2 次元)	0.45%	0.51%	+0.06%

マシンのパーツ性能を書いたファイルが付属している[*5]。このファイルの記述内容も、比較のために列挙した。今回の実験に使ったマシンの方が、ハードウェア性能的には優れていることがわかる。

5.4　ベンチマーク実験の詳細

前節で挙げた画像分類のベンチマークセットに対し、Deep Learning の様々なアルゴリズムによって分類を行わせ、その性能を測定した。

5.4.1　Maxout Network による、MNIST の分類タスク (2 次元データとして扱う)

Maxout Network を分類器として用い、MNIST の分類タスクを行わせた。モデル構造としては、Convolutional Layer を 2 層重ねた上に、全てのニューロンが単純に接続された Maxout Layer を 1 層付け加え、最後の層にてロジスティック回帰を行って分類結果を出した。実験を行った結果、分類誤差は 0.51% となった。これは、元の論文が主張している分類誤差より、僅かに悪い結果となった。結果を、表 5.4 に示す。

5.4.2　Maxout Network による、MNIST の分類タスク (1 次元データとして扱う)

前項と同じタスクを、「MNIST のデータは、2 次元の画像データではなく、1 次元のベクトルデータである」という条件の基で行わせた。つまり、2 次元データに対する Convolutional Layer 技術を敢えて使わない状態で、どれだけの精度を Maxout Network が実現できるのか、実験した。この条件でも Maxout Network が良い性能を出すことが出来れば、1 次元のデータ、例えば言語データや音声データにおいても、Maxout Network が応用できる可能性が高くなる。
また、何らかの原因でランダム性が発生していないか確かめるため、この実験は同じ条件で 10 回行った。乱数のシードは、元論文の実験が行われたときと同じ値で固定した。

[*5] pylearn2/scripts/papers/maxout/notes

表 5.5　Maxout Network による MNIST(1 次元) 分類の結果

手法	データセット	元論文の誤差	実験誤差	増加分
Maxout Network	MNIST(1 次元)	0.94%	1.16%	+0.12%

表 5.6　Maxout Network による MNIST(1 次元) 分類の実験詳細

	平均実行時間 (秒)	平均最大使用メモリ (MB)
pretraining	2532.8	779.3
finetuning	756.7	792.5
全体	3289.5	792.5

実験の結果、誤差は 10 回とも共通で、1.16% となった。Maxout の元の論文やはり少し低い精度になってしまった (表 5.5)。なお、平均実行時間は 55 分、最大メモリ使用量は 793MB だった (表 5.6)。

5.4.3　Maxout Network による、CIFAR10 の分類タスク

データセットを変更し、MNIST ではなく、CIFAR10 を Maxout Network を用いて分類させようと試みた。データは 2 次元画像として扱った。つまり、Convolutional Layer の使用は可能とした。レイヤー構造は、Convolutional Layer 3 層 - Maxout Layer1 層 - 識別用 Softmax Layer1 層とした。

しかし、この実験を実際に行ったところ、3 日間実行を続けたところで、残りエポック数と計算速度の比較より、全体の実行が完了するまでに 10 日以上かかることが判明した。これは Web 工学における応用に使う学習アルゴリズムとして、試行錯誤しながら使うものとしては現実的でない実行時間である。また、実験スケジュールの都合もあり、実験の中断を余儀なくされてしまった。

第 6 章

考察と提言

この章では、5 章の結果に対する考察を行うと共に、考察結果を踏まえ、Deep Learning の実問題における効率的な応用方法について、提言を行う。

6.1 全体のまとめ

pylearn2 における Maxout Network は、元論文に記されている精度こそ再現できなかったものの、MNIST の分類タスクにおいて State of the Art に近い精度を実現することが出来た。Maxout Network を利用することで、少なくとも画像認識のタスクにおいて精度を向上させることが出来ると考えられる。

6.2 分類誤差が大きくなってしまう原因の考察

今回の実験では、MNIST の 1 次元版、2 次元版共に、元の論文や pylearn2 ソースコードの付属テキストに書いてあった分類誤差よりも、大きい誤差しか再現できなかった。また、この誤差が大きくなってしまう問題は、複数回の実験を行っても、解決しなかった。

分類精度が悪くなってしまった理由として、当初、図 6.1 に挙げた 3 つの原因が考えられた。しかし、複数回実行しても全く同じ結果が出たため、乱数のシードを変更しない限り、内部的には全く同じ演算が成されていると考えられる。よって、「重みのランダム初期化」、つまり「ニューラルネットワークの接続の重みがランダムに決まっており、たまたま分類に不利な重みからスタートしたため、元の論文よりも悪い結果が出てしまった」という仮説は否定される。残る可能性は、「ソースコードのバージョン」と「ハードウェア構成の違い」であるが、GPU を始めとする各パーツの性能は、元論文の実験時に使われたものよりも、今回の実験で使ったものの方が高いため、「ハードウェアの性能」に原因を求めるのも難しい。残る可能性は「ソースコードのバージョン」である。これは、pylearn2 が依存しているライブラリの、numpy や scipy、Theano また pylearn2 自体などがアップデートされたことにより、内部的に細かい計算方法が変化し、分類誤差の再現性が下がってしまった、という

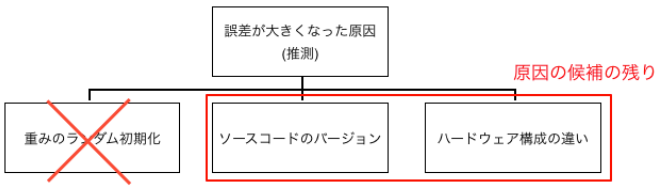

図 6.1　Maxout Network による MNIST 分類実験で、誤差が論文より大きくなった原因

仮説である。

実験の再現性については、Maxout Network の元実験を行った方も、

> All of pylearn2's dependencies (theano / scipy / numpy / etc.) make reproducibility very difficult. As such, we are not currently going to make any effort to ensure that all results are widely reproducible on a variety of platforms. Instead, we will record the platform on which they were verified to be correct.

と記している (pylearn2 のソースコード[*1]より引用)。彼らの認識でも、まずソースコードやライブラリのバージョンが重要であり、またハードウェアの性能も影響すると考えられていることが読み取れる。

6.3　CIFAR10 の実行時間が非常に長くなってしまう問題について

今回の実験では、Maxout による CIFAR10 の分類を、実行時間があまりにも長くかかるため、断念してしまった。実行時間をどうすれば短縮できるのか、特にハードウェアを変えず、分類精度も犠牲にしないで、出来れば長くとも 1 日程度の実行時間で収束させるための方法がないか、検討を重ねる必要がある。その後の実験により、モデルの学習中に統計情報を収集させる部分で時間がかかってしまっている可能性があるとわかっている。精度を変えずに時間短縮をするための、大きなポイントの 1 つとして、追求を続ける必要がある。

ハードウェアを買わず、既存のハードウェアをやりくりして性能を上げる方法として、複数台マシンによる分散処理が挙げられる。論文の中には、数日規模の計算を行っているものもあり、分散処理を行わせる価値は高い。例えば、[52] では、複数台の GPU マシンを用いて Convolutional Net の学習を行わせている。このとき、GPU 間で計算結果を通信する部分がボトルネックになると予想されるが、この研究ではニューロンを GPU ごとに分担させ、一部のレイヤーでのみ同期通信を行うことで、GPU 間通信の回数を抑えて対処している。分散処理の弱点は通信自体にかかる時間だけではなく、同期部分において最も遅いマシンがボトルネックになってしまうという点もある。例えば GPU

[*1] `pylearn2/scripts/papers/maxout/notes`

マシンと CPU マシンが混交している場合、CPU マシンの性能と同じ速度しか出なくなってしまうことが挙げられるが、この問題は、ある程度の回数まで同期の無視を認めることで解決できる [43]。

6.4 深層学習の利用法に関する提言

Maxout Network が良い精度を実現できたこと、改造のしやすさ、GPU 利用の簡便さなどを考え合わせると、現時点では、pylearn2 を通して Deep Learning を利用するのが良いと考えられる。Deep Learning を利用する上で、最も求められる高い識別性能は、Maxout Network を利用することで担保できる。Maxout Network は、画像に対しては Convolutional Layer と併用することにより、ほぼ state of the art と遜色ない精度を実現することができる。また、対象が画像ではなく、1 次元の広がりしかもたないデータの場合、Convolutional Layer を使うことは出来ないが、現時点では最も高い識別精度を出すことが出来る。この識別性能は、画像データだけでなく、他のデータを扱う際にも有用だと思われる。また、任意の連続関数を近似できるため、かつての単層パーセプトロンのように、理論的に学習できない関数が出現するということもない。Deep Learning に識別精度を求める場合、Maxout Network を使うのが最も有効な方法の 1 つである。

Maxout の既存実装としては、論文の著者自らが提供しているライブラリの、pylearn2 を使うのが最も確実である。この実装を使えば、論文にて述べられている分類精度にかなり近い精度を、実際に実現することができる。厳密には、論文の分類精度よりも、若干性能は劣化してしまうが、著者が実験を行ったときのハードウェアと、ソースコードのバージョンを揃えずとも、かなり近い精度を再現できることも、また事実であり、実用上はあまり問題にならないと考える。

Deep Learning の大きなネックの 1 つとして、その長い実行時間が挙げられる。この問題は、実行時に GPU を利用することによってかなり緩和できるが、CUDA を用いた専用のプログラムを書く必要があり、難易度が比較して高くなってしまう。この難点は、計算アルゴリズムを書くときに、数値計算/数式処理ライブラリである Theano を利用することで、かなり楽になる。Theano で書いたプログラムは、GPU 上でも利用でき、CPU のみのマシンでも、実行速度は下がるが問題なく動かすことができる。また、科学技術計算プログラムの記述においてメジャーな python で書かれており、C 言語のプログラムに比べて記述が容易であるにも関わらず、事前に CUDA のコードを生成&コンパイルする仕組みにより、C 言語で CUDA のコードを全て記述した場合と代わらない速度で動かすことができる。

pylearn2 は Theano の機能をフルに活用して記述されており、また Space クラスや画像による可視化など、Theano に不足している部分にもフォローがされている。pylearn2 を使えば、Theano の恩恵をそのまま受けることができる。この点でも、pylearn2 は望ましいと考えられる。

pylearn2 は、規模の比較的大きいライブラリではあるが、ソースコードは再利用性を強く意識して

記述されており、モジュール化が明確である。libsvm のような、テキストファイルにデータを書くだけで利用できるような仕組みは残念ながら用意されていないが、例えば入力データを差し替えるだけならば、Dataset クラスの内容を把握して書くだけで済む。また、アルゴリズムにて使われているハイパーパラメータを調整するだけなら、ソースコードを書く必要すらなく、設定用の yaml ファイルを変更するだけで調整を済ませることができる。この「利用する部分だけ把握すれば良い」という性質は、モジュール化の大きな利点であり、そのまま pylearn2 の大きな利点でもある。

第 7 章

おわりに

7.1 研究の成果

この研究では、Deep Learning を Web 工学の問題に適用するにあたって、その特徴である高い精度を落とすことなく、出来るだけ簡便に応用するための方法論とノウハウを調査した。pylearn2 に実装されている、Maxout Network を用いることで、論文に書かれている精度をほぼ再現することが出来た。また、GPU を使うと大幅な高速化が見込めることがわかった。

7.2 今後の課題

画像分類タスクにおいて有効な方法が、Web 工学のデータでも必ず有効かどうかは、未知数な部分がある。例えば、Convolutional Network は、2 次元の画像データに対しては非常に高い効果を挙げるが、そのまま文章データに応用することは出来ない。pylearn2 で言えば、Dataset クラスを作るだけで識別がうまくいくのか、文章専用の Model を構成する必要があるのか、確かめていく必要がある。

また、精度を上げるためには、どのようにハイパーパラメータを調整すれば良いのか、あるいは精度を多少犠牲にしてでも、比較的短い実行時間で良い結果を得たい時、どのような調整を施せばよいのかは、まだわかっておらず、今後の大きな課題の一つである。

現在の Deep Learning が比較的得意とする画像データ処理について、単純な認識タスクで大量の訓練データが用意されている条件ならば、Convolutional Layer による特徴抽出が非常に良い性能を示す。しかし、問題設定を少し変えて、「シンボルを 1 つだけ見せて、他のシンボルの中から同じものを選ばせる (one-shot classification)」「シンボルを 1 つだけ見せて、同じシンボルを機械と人間に書かせ、どちらが上手く出来るか競う (one-shot generation)」といった、データ数が非常に少ない条件下では、まだ人間が手間をかけて作った素性を使った方が性能が高いことがある [54]。画像をただ認識するだけでなく、良い素性の作り方を少ないデータから学習できるようになれば、Epoch 数を減らし学習時間を短縮することにもつながり、応用の幅が広がると思われる。

Deep Learning を試す上で、大きなネックとなるのが、CUDA を用いた GPU 専用のソースコードの存在である。GPU を搭載していなかったり、使えるメモリが少ない状況下でも、Deep Learning のコードを効率良く動かすことが出来れば、Deep Learning の利便性はますます増加するだろう。

現在の Deep Learning では、画像のフィルタで何が学習されているのかを、部分的に可視化することはできる。しかし、文書解析において、どのような表現を学習したのかを、人間に理解できる形でみることは難しい。言い換えれば、学習によって、分類器がどこに注目するようになったのか、文構造や、感情、文体などに対応するニューロンが存在しているのか、といった情報が、人間に理解できる形になっていない。この部分をどうやって可視化して、人間に理解できる状態にするか、そこからどのような知見を得られるのか、あるいはそもそも人間には理解出来る知識として取り出せるのかどうか、といったことを調べることにより、表現学習としての Deep Learning の側面を、さらに活かすことができると思われる。例えば、Deep Learning にょって学習された、データの着目点や抽象化のポイントを、人間が真似することによって、人間の方が機械から知識を習得し、さらに発展させることも考えられる。

謝辞

(全て完了してから改めて書きます)

東京大学工学部システム創成学科

知能社会システムコース

松尾研究室 4 年 黒滝 紘生

平成 26 年 2 月日

参考文献

[1] Lada A Adamic and Eytan Adar. Friends and neighbors on the web. *Social networks*, Vol. 25, No. 3, pp. 211–230, 2003.

[2] G. Azzopardi and N. Petkov. Trainable cosfire filters for keypoint detection and pattern recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, Vol. 35, No. 2, pp. 490–503, 2013.

[3] Joel S Bader, Amitabha Chaudhuri, Jonathan M Rothberg, and John Chant. Gaining confidence in high-throughput protein interaction networks. *Nature biotechnology*, Vol. 22, No. 1, pp. 78–85, 2003.

[4] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer vision and image understanding*, Vol. 110, No. 3, pp. 346–359, 2008.

[5] Richard Bellman. *Adaptive control processes: a guided tour*, Vol. 4. Princeton university press Princeton, 1961.

[6] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *IEEE Trans. Pattern Anal. Mach. Intell.*, Vol. 24, No. 4, pp. 509–522, April 2002.

[7] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. *CoRR*, Vol. abs/1206.5533, , 2012.

[8] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. pp. 153–160, 2007.

[9] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. Oral Presentation.

[10] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, Vol. 1. springer New York, 2006.

[11] Hervé Bourlard and Yves Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological cybernetics*, Vol. 59, No. 4-5, pp. 291–294, 1988.

[12] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, Vol. 30, No. 1, pp. 107–117, 1998.

[13] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on Machine learning*, pp. 89–96. ACM, 2005.

[14] Christopher JC Burges. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, Vol. 2, No. 2, pp. 121–167, 1998.

[15] Nicolò Cesa-Bianchi, Claudio Gentile, and Giovanni Zappella. A gang of bandits. In C.J.C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pp. 737–745. NIPS, NIPS, 2013.

[16] Soravit Changpinyo, Kuan Liu, and Fei Sha. Similarity component analysis. In *Advances in Neural Information Processing Systems*, pp. 1511–1519. NIPS, 2013.

[17] Dan Claudiu Ciresan, Ueli Meier, Luca Maria Gambardella, and Jürgen Schmidhuber. Deep, big, simple neural nets for handwritten digit recognition. *Neural Computation*, Vol. 22, No. 12, pp. 3207–3220, 2010.

[18] Dan Claudiu Ciresan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *arXiv:1202.2745v1 [cs.CV]*, 2012.

[19] Aaron Clauset, Cristopher Moore, and Mark EJ Newman. Hierarchical structure and the prediction of missing links in networks. *Nature*, Vol. 453, No. 7191, pp. 98–101, 2008.

[20] Aaron Clauset, Mark EJ Newman, and Cristopher Moore. Finding community structure in very large networks. *Physical review E*, Vol. 70, No. 6, p. 066111, 2004.

[21] Adam Coates, Andrew Y Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *International Conference on Artificial Intelligence and Statistics*, pp. 215–223, 2011.

[22] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, Vol. 20, No. 3, pp. 273–297, 1995.

[23] George Dahl and Kit La Touche. Cs81: Learning words with deep belief networks. 2008.

[24] Dennis Decoste and Bernhard Schölkopf. Training invariant support vector machines. *Mach. Learn.*, Vol. 46, No. 1-3, pp. 161–190, March 2002.

[25] Jia Deng, Wei Dong, R. Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255. CVPR, 2009.

[26] Aasa Feragen, Niklas Kasenburg, Jens Petersen, Marleen de Bruijne, and Karsten Borgwardt. Scalable kernels for graphs with continuous attributes. In *Advances in Neural Information Processing Systems*, pp. 216–224. NIPS, 2013.

[27] William M Fisher, George R Doddington, and Kathleen M Goudie-Marshall. The darpa speech recognition research database: specifications and status. In *Proc. DARPA Workshop on speech recognition*, pp. 93–99, 1986.

[28] Andrea Frome, Greg S Corrado, Jon Shlens, Samy Bengio, Jeff Dean, Tomas Mikolov, et al. Devise: A deep visual-semantic embedding model. In *Advances in Neural Information Processing Systems*, pp. 2121–2129. NIPS, 2013.

[29] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, Vol. 36, No. 4, pp. 193–202, 1980.

[30] Kunihiko Fukushima, Sei Miyake, and Takayuki Ito. Neocognitron: A neural network model for a mechanism of visual pattern recognition. *Systems, Man and Cybernetics, IEEE Transactions on*, No. 5, pp. 826–834, 1983.

[31] Ken-Ichi Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, Vol. 2, No. 3, pp. 183 – 192, 1989.

[32] Michael Garland, Scott Le Grand, John Nickolls, Joshua Anderson, Jim Hardwick, Scott Morton, Everett Phillips, Yao Zhang, and Vasily Volkov. Parallel computing experiences with cuda. *Micro, IEEE*, Vol. 28, No. 4, pp. 13–27, 2008.

[33] Thomas Gärtner, Peter Flach, and Stefan Wrobel. On graph kernels: Hardness results and efficient alternatives. In *Learning Theory and Kernel Machines*, pp. 129–143. Springer, 2003.

[34] Robert Gens and Pedro Domingos. Discriminative learning of sum-product networks. In *Advances in Neural Information Processing Systems*, pp. 3248–3256, 2012.

[35] Namrata Godbole, Manja Srinivasaiah, and Steven Skiena. Large-scale sentiment analysis for news and blogs. *ICWSM*, Vol. 7, , 2007.

[36] David Goldberg, David Nichols, Brian M. Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *Commun. ACM*, Vol. 35, No. 12, pp. 61–70, December 1992.

[37] Ian J. Goodfellow, David Warde-Farley, Pascal Lamblin, Vincent Dumoulin, Mehdi Mirza, Razvan Pascanu, James Bergstra, Frédéric Bastien, and Yoshua Bengio. Pylearn2: a ma-

chine learning research library. *arXiv preprint arXiv:1308.4214*, 2013.

[38] Ian J Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. *arXiv preprint arXiv:1302.4389*, 2013.

[39] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, Vol. 18, No. 7, pp. 1527–1554, 2006.

[40] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, Vol. 313, No. 5786, pp. 504–507, 2006.

[41] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[42] Geoffrey E Hinton and Richard S Zemel. Autoencoders, minimum description length, and helmholtz free energy. *Advances in neural information processing systems*, pp. 3–3, 1994.

[43] Qirong Ho, James Cipar, Henggang Cui, Seunghak Lee, Jin Kyu Kim, Phillip B Gibbons, Garth A Gibson, Greg Ganger, and Eric Xing. More effective distributed ml via a stale synchronous parallel parameter server. In *Advances in Neural Information Processing Systems*, pp. 1223–1231. NIPS, 2013.

[44] Paul Jaccard. *Lois de distribution florale dans la zone alpine*. Corbaz, 1902.

[45] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In *Computer Vision, 2009 IEEE 12th International Conference on*, pp. 2146–2153, 2009.

[46] Yangqing Jia, Chang Huang, and T. Darrell. Beyond spatial pyramids: Receptive field learning for pooled image features. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pp. 3370–3377, 2012.

[47] Daniel Keysers, Thomas Deselaers, Christian Gollan, and Hermann Ney. Deformation models for image recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, Vol. 29, No. 8, pp. 1422–1435, August 2007.

[48] Andreas Klöckner, Nicolas Pinto, Yunsup Lee, B. Catanzaro, Paul Ivanov, and Ahmed Fasih. PyCUDA and PyOpenCL: A Scripting-Based Approach to GPU Run-Time Code Generation. *Parallel Computing*, Vol. 38, No. 3, pp. 157–174, 2012.

[49] Andreas Klöckner, Nicolas Pinto, Yunsup Lee, Bryan Catanzaro, Paul Ivanov, Ahmed Fasih, AD Sarma, D Nanongkai, G Pandurangan, P Tetali, et al. Pycuda: Gpu run-time code generation for high-performance computing. *Arxiv preprint arXiv*, Vol. 911, , 2009.

[50] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, Vol. 42, No. 8, pp. 30–37, August 2009.

[51] Alex Krizhevsky and G Hinton. Learning multiple layers of features from tiny images. Master's thesis, 4 2009.

[52] Alex Krizhevsky, Ilya Sutskever, and Geoff Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pp. 1106–1114, 2012.

[53] K. Labusch, E. Barth, and T. Martinetz. Simple method for high-performance digit recognition based on sparse coding. *Trans. Neur. Netw.*, Vol. 19, No. 11, pp. 1985–1989, November 2008.

[54] Brenden M Lake, Ruslan Salakhutdinov, and Josh Tenenbaum. One-shot learning by inverting a compositional causal process. In *Advances in Neural Information Processing Systems*, pp. 2526–2534. NIPS, 2013.

[55] Fabien Lauer, Ching Y. Suen, and Gérard Bloch. A trainable feature extractor for handwritten digit recognition. *Pattern Recogn.*, Vol. 40, No. 6, pp. 1816–1824, June 2007.

[56] Quoc V Le, Marc'Aurelio Ranzato, Rajat Monga, Matthieu Devin, Kai Chen, Greg S Corrado, Jeff Dean, and Andrew Y Ng. Building high-level features building high-level features using large scale unsupervised learning. ICML, 2012.

[57] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, Vol. 86, No. 11, pp. 2278–2324, 1998.

[58] Honglak Lee, Chaitanya Ekanadham, and Andrew Ng. Sparse deep belief net model for visual area v2. In *Advances in neural information processing systems*, pp. 873–880, 2007.

[59] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pp. 609–616, New York, NY, USA, 2009. ACM.

[60] David Liben-Nowell and Jon Kleinberg. The link-prediction problem for social networks. *Journal of the American society for information science and technology*, Vol. 58, No. 7, pp. 1019–1031, 2007.

[61] David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, Vol. 2, pp. 1150–1157. Ieee, 1999.

[62] Thomas Mensink, Jakob Verbeek, Florent Perronnin, and Gabriela Csurka. Metric learning for large scale image classification: Generalizing to new classes at near-zero cost. In *Computer Vision–ECCV 2012*, pp. 488–501. Springer, 2012.

[63] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, Vol. abs/1301.3781, , 2013.

[64] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pp. 3111–3119. NIPS, 2013.

[65] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of NAACL-HLT*, pp. 746–751, 2013.

[66] Martin Renqiang Min, David A. Stanley, Zineng Yuan, Anthony J. Bonner, and Zhaolei Zhang. Large-margin knn classification using a deep encoder network. *CoRR*, Vol. abs/0906.1814, , 2009.

[67] Marvin Lee Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge Mass., expanded ed. edition, 1988.

[68] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. NIPS, 2013.

[69] Hideki Nakayama. Efficient discriminative convolution using fisher weight map. BMVC, 2013.

[70] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: bringing order to the web. 1999.

[71] Mark Palatucci, Dean Pomerleau, Geoffrey E Hinton, and Tom M Mitchell. Zero-shot learning with semantic output codes. In *Advances in neural information processing systems*, pp. 1410–1418. NIPS, 2009.

[72] Bo Pang and Lillian Lee. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of the 42nd annual meeting on Association for Computational Linguistics*, p. 271. Association for Computational Linguistics, 2004.

[73] Bo Pang and Lillian Lee. Opinion mining and sentiment analysis. *Foundations and trends in information retrieval*, Vol. 2, No. 1-2, pp. 1–135, 2008.

[74] Florent Perronnin and Christopher Dance. Fisher kernels on visual vocabularies for im-

age categorization. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pp. 1–8. IEEE, 2007.

[75] Christopher Poultney, Christopher Poultney, Sumit Chopra, and Yann" Lecun. Efficient learning of sparse representations with an energy-based model. *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS (NIPS 2006*, 2006.

[76] M. Ranzato, Fu Jie Huang, Y.-L. Boureau, and Y. LeCun. Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, pp. 1–8, 2007.

[77] Paul Resnick and Hal R. Varian. Recommender systems. *Commun. ACM*, Vol. 40, No. 3, pp. 56–58, March 1997.

[78] Salah Rifai, Xavier Muller, Xavier Glorot, Grégoire Mesnil, Yoshua Bengio, and Pascal Vincent. Learning invariant features through local space contraction. *arXiv preprint arXiv:1104.4153*, 2011.

[79] Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 833–840, 2011.

[80] Marcus Rohrbach, Michael Stark, and Bernt Schiele. Evaluating knowledge transfer and zero-shot learning in a large-scale setting. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pp. 1641–1648. IEEE, 2011.

[81] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, Vol. 65, No. 6, p. 386, 1958.

[82] David E Rumelhart, Geoffrey E Hintont, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, Vol. 323, No. 6088, pp. 533–536, 1986.

[83] Ruslan Salakhutdinov and Geoffrey E Hinton. Deep boltzmann machines. In *International Conference on Artificial Intelligence and Statistics*, pp. 448–455, 2009.

[84] Holger Schwenk and Maurice Milgram. Transformation invariant autoassociation with application to handwritten character recognition. *Advances in neural information processing systems*, Vol. 7, pp. 991–998, 1995.

[85] Thomas Serre, Lior Wolf, Stanley Bileschi, Maximilian Riesenhuber, and Tomaso Poggio. Robust object recognition with cortex-like mechanisms. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, Vol. 29, No. 3, pp. 411–426, 2007.

[86] Patrice Y. Simard, Dave Steinkraus, and John C. Platt. Best practices for convolutional

neural networks applied to visual document analysis. In *Proceedings of the Seventh International Conference on Document Analysis and Recognition - Volume 2*, ICDAR '03, pp. 958–, Washington, DC, USA, 2003. IEEE Computer Society.

[87] Paul Smolensky. Information processing in dynamical systems: Foundations of harmony theory. 1986.

[88] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *arXiv preprint arXiv:1206.2944*, 2012.

[89] Richard Socher, Alex Perelygin, Jean Y Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2013.

[90] Markus Thom and Günther Palm. Sparse activity and sparse connectivity in supervised learning. *J. Mach. Learn. Res.*, Vol. 14, No. 1, pp. 1091–1143, April 2013.

[91] Pascal Vincent. A connection between score matching and denoising autoencoders. *Neural computation*, Vol. 23, No. 7, pp. 1661–1674, 2011.

[92] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. pp. 1096–1103, 2008.

[93] Li Wan, Matthew Zeiler, Sixin Zhang, Yann L. Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In Sanjoy Dasgupta and David Mcallester, editors, *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, Vol. 28, pp. 1058–1066. JMLR Workshop and Conference Proceedings, May 2013.

[94] Jason Weston, Frédéric Ratle, Hossein Mobahi, and Ronan Collobert. Deep learning via semi-supervised embedding. In *Neural Networks: Tricks of the Trade*, pp. 639–655. Springer, 2012.

[95] Theresa Wilson, Janyce Wiebe, and Paul Hoffmann. Recognizing contextual polarity in phrase-level sentiment analysis. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pp. 347–354. Association for Computational Linguistics, 2005.

[96] Matthew D Zeiler and Rob Fergus. Stochastic pooling for regularization of deep convolutional neural networks. *arXiv preprint arXiv:1301.3557*, 2013.

[97] 徳永健伸. 情報検索と言語処理 (言語と計算). 東京大学出版会, 11 1999.

[98] 川人光男. 脳の計算理論. 産業図書, 1998.