修　士　学　位　論　文

Stacked Denoising Autoencoderを利用した語義曖昧性の教師なし領域適応

平成28年度

茨城大学大学院理工学研究科

情報工学専攻

河野 和平

平成 28 年度 茨城大学大学院理工学研究科情報工学専攻　修士学位論文

Stacked Denoising Autoencoder を利用した 語義曖昧性の教師なし領域適応

著者：河野和平 (15NM707G)
指導教員：新納浩幸教授

論文要旨

本論文では深層学習 (Deep Learning) のモデルのひとつである Stacked Denoising Autoencoder(SdA) を利用し，語義曖昧性解消 (Word Sense Disambiguation，WSD) の領域適応の問題解決を図る．

WSD は文章中に出現する単語の語義を一意に識別するタスクである．一般に WSD は分類問題として処理され，教師有り学習などで語義を識別する．自然言語処理のタスクにおける分類問題において機械学習を用いる際，訓練データとテストデータは同じ領域のコーパスから得られたデータであることが前提であった．しかし，WSD のタスクにおいて，これらは異なる領域から得られる場合がある．このような場合，ある領域 (ソース領域) のコーパス S から得た訓練データによって学習した分類器では，それとは別の領域 (ターゲット領域) のコーパス T から得たテストデータを精度よく識別することが困難である．そこで，ソース領域の訓練データで学習した分類器をターゲット領域のテストデータに適用できるようにチューニングすることを領域適応と呼ぶ．

WSD の問題点として，異なる領域から得られた訓練データとテストデータでは同じ単語に関するデータであっても，その素性に大きな差が生じてしまうことが挙げられる．これは，異なる領域の文章はその話題や内容に差があり，共起語が大きく異なることによる影響であると考えられる．そこで，深層学習によって当該問題の緩和を試みる．深層学習はニューラルネットワーク (Neural Network，NN) を用いて，データの抽象的な表現を獲得する手法である．ターゲット領域に頻出しない共起語からなる文章で分類器を学習する場合も，深層学習によって獲得した抽象的なデータによって学習することで，上述した問題を緩和できると考えた．本論文では複数存在する深層学習モデルのうち，SdA を利用した．

従来の研究から，WSD の領域適応に対して単一の手法を適用した場合，対象単語とソース領域，ターゲット領域の組み合わせによって，識別精度が改善されるケースと悪化してしまうケースが存在し，平均して識別精度が変わらない，もしくは悪化してしまうことが知られている．そこで本論文では，ソース領域とターゲット領域の素性の類似度を定義し，対象単語ごとにこれを測定して類似度が極端に低い場合には，SdA を適用せず，そのままのデータを SVM によって識別する方法を取った．これは，類似度が低い場合，SdA が適切な素性を学習できない可能性が高いと考えたことによるものである．

実験では，現代日本語書き言葉均衡コーパス (BCCWJ) の Yahoo!知恵袋 (OC)，書籍 (PB) 及び新聞 (PN) のデータのうち 16 単語を対象に，OC，PB，PN をそれぞれ各領域として組み合わせ 6 通りの領域適応による比較を行った．結果，素性の類似度を導入した SdA は比較手法の中で平均して最も良い結果を示した．

Master's Thesis in Scholastic 2016, Major in Computer and Information Sciences, Graduate School of Science and Engineering, Ibaraki University

Unsupervised Domain Adaptation for Word Sense Disambiguation using Stacked Denoising Autoencoder

Author : Kazuhei Kouno(15NM707G)
Adviser : Prof. Hiroyuki Shinnou

ABSTRACT

In this paper, we propose an unsupervised domain adaptation for Word Sense Disambiguation(WSD) using Stacked Denoising Autoencoder(SdA).

WSD is the task of identifying the sense of a target word in a sentence. In general, supervised learning, such as Support Vector Machine(SVM), can be used for this task because of the fact that this approach is highly accurate.

In the task of natural language processing, it is premised that the training and test data come from same domain. However, in many cases it is not so often in WSD. If the training and test data come from different domains, the accuracy of this approach is lowered. This problem is called a domain adaptation. It is considered that this problem occurs due to the difference between the distributions of features in training and test data. This problem occurs because there is a difference in topics and contents. Therefore, we use SdA of Deep Learning and solve this problem.

SdA is an unsupervised learning method of obtaining the abstract feature set of input data using Neural Network. The abstract feature set absorbs the difference of domains, and thus SdA can solve a problem of domain adaptation.

However, SdA does not always cope with any problems of domain adaptation. Especially, difficulty of domain adaptation for WSD depends on the combination of source domain, a target domain and a target word. As a result, any method of domain adaptation for WSD has adverse effect for a part of the problem. Therefore, we defined the similarity between two domains, and judge whether we use SdA or not through this similarity. This approach avoids an adverse effect of SdA because we thought that SdA can not learn the proper feature when the similarity of features between two domains is extremely small.

In our experiment, we have used three domains: Yahoo! Answers (OC), Books (PB), and newspaper (PN) from the Balanced Corpus of Contemporary Written Japanese, along with 16 selected ambiguous words. Domain adaptation has the following six transitions: (1) PB → OC,(2) OC → PB, (3) OC → PN, (4) PN → OC, (5)PB → PN and (6) PN → PB. In comparison with baseline, our method has got higher average accuracies for all combinations of two domains. Furthermore, we have obtained better results against conventional domain adaptation methods.

目 次

表 目 次

図 目 次

第1章　序論

1.1　概要

本論文では深層学習 (Deep Learning) のモデルのひとつである Stacked Denoising Autoencoder(SdA) を利用し，語義曖昧性解消 (Word Sense Disambiguation，WSD) の領域適応の問題解決を図る．

WSD は文章中に出現する単語の語義を一意に識別するタスクである．一般に WSD は分類問題として処理され，教師有り学習などで語義を識別する．自然言語処理のタスクにおける分類問題において機械学習を用いる際，訓練データとテストデータは同じ領域のコーパスから得られたデータであることが前提であった．しかし，WSD のタスクにおいて，これらは異なる領域から得られる場合がある．このような場合，ある領域 (ソース領域) のコーパス S から得た訓練データによって学習した分類器では，それとは別の領域 (ターゲット領域) のコーパス T から得たテストデータを精度よく識別することが困難である．そこで，ソース領域の訓練データで学習した分類器をターゲット領域のテストデータに適用できるようにチューニングすることを領域適応 [1] と呼び，本研究室においても関連研究が多数行われている [2][3].

WSD の問題点として，異なる領域から得られた訓練データとテストデータでは同じ単語に関するデータであっても，その素性に大きな差が生じてしまうことが挙げられる．これは，異なる領域の文章はその話題や内容に差があり，共起語が大きく異なることによる影響であると考えられる．そこで本論文では，深層学習によって当該問題の緩和を試みる．深層学習はニューラルネットワーク (Neural Network，NN) を用いて，データの抽象的な表現を獲得する手法である．近年活発に研究されており，画像認識や音声認識の分野でよい結果得られている [4]．ターゲット領域に頻出しない共起語からなる文章で分類器を学習する場合も，深層学習によって獲得した抽象的なデータによって学習することで，上述した問題を緩和できると考えた．本論文では複数存在する深層学習モデルのうち，SdA[5] を利用した．

従来の研究から，WSD の領域適応に対して単一の手法を適用した場合，対象単語とソース領域，ターゲット領域の組み合わせによって，識別精度が改善されるケースと悪化してしまうケースが存在し，平均して識別精度が変わらないもしくは悪化してしまうことが知られている．そこで本論文では，ソース領域とターゲット領域の素性の類似度を定義し，対象単語ごとにこれを測定して類似度が極端に低い場合には，SdA を適用せず，そのままのデータを SVM によって識別する方法を取った．これは，類似度が低い場合，SdA が適切な素性を学習できない可能性が高いと考えたことによるものである．

実験では，現代日本語書き言葉均衡コーパス (BCCWJ) の Yahoo!知恵袋 (OC)，書籍

(PB) 及び新聞 (PN) のデータのうち 16 単語を対象とした．OC，PB，PN をそれぞれ各領域とし，組み合わせ 6 通り (OC → PB，OC → PN，PB → OC，PB → PN，PN → OC，PN → PB) の領域適応を行った．比較手法として，そのままのデータを SVM によって識別する手法と従来研究である uLSIF と SCL を使用した手法，通常の SdA を利用した手法と類似度を導入した SdA による手法の 5 種類を利用した．実験の結果，素性の類似度を導入した SdA は比較手法の中で平均して最も良い結果を示した [6].

1.2 本論文の構成

本論文では，はじめに理論とその手法について紹介する．第 2 章で語義曖昧性解消 (Word Sense Disambiguation，WSD) について，第 3 章では領域適応 (Domain Adaptation) の概要を述べる．第 4 章で深層学習 (Deep Learning) について概説する．

そして，WSD の領域適応に対して深層学習を利用する手法について述べ，その実験と結果の考察を行う．第 5 章で WSD の領域適応に対して SdA を利用する手法の提案をし，第 6 章では素性の類似度を導入して SdA を適用するかどうかの選択を行う手法について述べる．第 7 章で SVM を利用した比較実験の内容と結果を示し，第 8 章ではその実験結果について考察する．最後に第 9 章で結論と今後の課題について述べる．

第2章　語義曖昧性解消

2.1　概要

単語には一般に複数の意味が存在する．語義曖昧性解消 (Word Sense Disambiguation, WSD) は，文章中に出現するこのような単語の語義を一意に識別するタスクである．

例えば，単語「のむ」という単語には少なくとも表 2.1 のような 3 つの意味が存在する．

表 2.1: 「のむ」の語義

単語	語義
のむ	(1) 飲食物を口から体内に送り込む． (2) 受け入れる．妥協する． (3) 外に出さないで抑える．堪える．

ここで,「のむ」に関する次のような例文が与えられたとする．

1. 息をのむ.

2. お茶をのむ.

例文 1 における「のむ」の語義は，表 2.1 のうち 3 番目の「外に出さないで抑える．堪える.」に該当する．一方，例文 2 で用いられている「のむ」の語義は 1 番目の「飲食物を口から体内に送り込む.」である．このようにして，与えられた文章中に出現する単語の語義を一意に識別することを語義曖昧性解消と呼ぶ.

2.2　一般的な手法

一般に，WSD の識別には教師有り学習，半教師有り学習，教師無し学習などが利用される．教師有り学習とは，事例であるデータに対して正解である語義をラベルとして付加したデータ (ラベル付きデータ) を用いて分類器の学習を行う手法である．一方，教師無し学習ではこの正解ラベルのないデータ (ラベル無しデータ) を利用して分類器の学習を行う．また，半教師有り学習はラベル付きデータとラベル無しデータの混在したデータを利用して学習を行う手法である．代表的な例として，サポートベクターマシンは，ラベル付きの訓練データを用いた教師有り学習によって分類器を学習し，その分類器を用いてテストデータの分類を行う．

学習にはコーパスと呼ばれる大量に集めた文書データを利用する．WSD のタスクの場合，このコーパスの一部のデータに語義や構文構造，品詞などの付加情報を含む自然言語処理に特化した注釈付きコーパスが存在する．一般にはこのようなコーパスのデータを利用して学習を行う．

次節では教師有り学習の代表的な例として挙げた SVM について概説する．

2.3 Support Vector Machine

サポートベクターマシン (Support Vector Machine，SVM)[7] は線形二値分類器で WSD のような分類問題によく使用される．通常は分類するクラスが「正クラス」と「負クラス」の 2 種類である場合に用いられ，正クラスに属する事例を正例，負クラスに属する事例を負例と呼ぶ．

訓練データ D が以下の式で与えられるとする．

$$D = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(|D|)}, y^{(|D|)})\} \tag{2.1}$$

$|D|$ は訓練データ D の事例数を指し，$x^{(n)}$ 及び $y^{(n)}$ はそれぞれ n 番目の事例の素性ベクトルとクラスラベルを表す．ここで，クラスラベル $y^{(n)}$ は正例なら+1，負例なら-1 を持つ．分離平面の方向ベクトルを w，切片を b とすれば，SVM は以下の式で表される．

$$f(x) = w \cdot x - b \tag{2.2}$$

SVM は未知のデータ x^{uk} が与えられたとき，$f(x^{uk}) \geq 0$ なら正例，$f(x^{uk}) < 0$ なら負例に分類する．

マージンの最大化

与えられた訓練データを分類する分類平面は無数に存在する．ここでは，そのような場合に SVM が良い分類平面を学習するための基本的な手法について解説する．

例えば図 2.1 のように分布する訓練データが与えられたとする．

ここで，異なる記号は異なるクラスの事例を表す．分類平面は $w \cdot x = b$ を満たすような点 x の集合である．このようなデータを分類する平面は無数に存在する．例えば，図 2.2 に破線で示すような 3 つの分類平面が考えられる．

SVM では無数に存在する分類平面のうち，「正クラスと負クラスどちらのクラスからもなるべく遠い位置にある分類平面」を選択する．これをマージンの最大化と呼ぶ．分類平面のマージンは最も近い訓練事例への距離として定義される．

分類平面の最も近くにある正例を x_+，x_+ と分類平面を結ぶ垂線と分類平面の交わる点を x_* とすると，正例と分類平面とのマージンは $|x_+ - x_*|$ と表せる．

ここで，以下の式が成り立つ．

$$w \cdot (x_+ - x_*) = |w||x_+ - x_*| \tag{2.3}$$

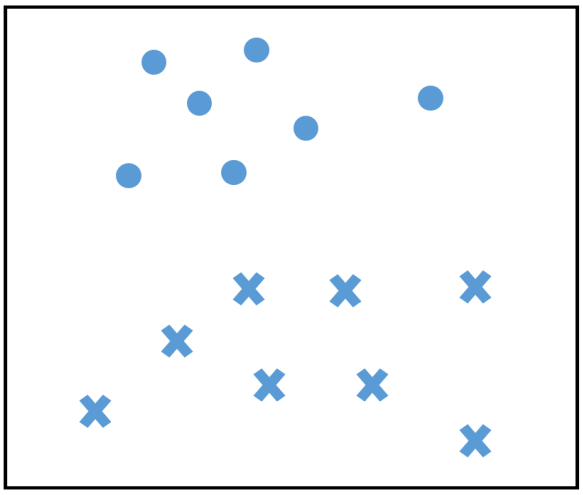

図 2.1: 訓練データの分布例

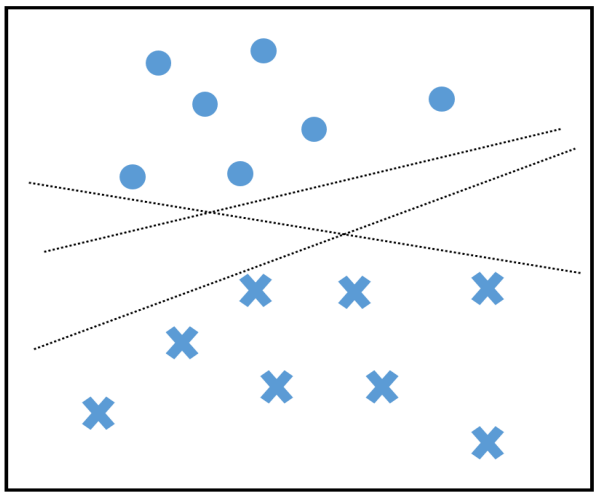

図 2.2: 分離平面の例

分離平面 $w \cdot x = b$ は任意の定数を掛けても不変なので，$w \cdot x - b = 1$ とできる．従って，

$$\begin{aligned} w \cdot (x_+ - x_*) &= w \cdot x_+ - w \cdot x_* \\ &= (b+1) - b \\ &= 1 \end{aligned} \tag{2.4}$$

となる．式 2.3 によれば，式 2.4 の左辺は $|w||x_+ - x_*|$ に等しいので，以下の式が成り立つ．

$$|w||x_+ - x_*| = 1 \tag{2.5}$$

以上より，

$$|x_+ - x_*| = \frac{1}{|w|} \tag{2.6}$$

である．つまり，SVM におけるマージンの最大化とは $1/|w|$ を最大化する方向ベクトル $|w|$ を求める問題である．ただし，一般的には計算簡単化のため，w^2 を最小化する問題として解かれる．

第3章　領域適応

従来，自然言語処理のタスクにおいて機械学習を用いる場合，前提として訓練データとテストデータは同じ領域のコーパスから得られたデータである必要があった．しかし，WSD のタスクにおいて訓練データとテストデータは異なる領域から得られていることも多い．例えば，訓練データとして書籍から得た文章を利用して分類器を学習し，新聞から得た文章中の単語の語義を識別したいような場合である．このような場合，書籍から得たデータで学習した分類器では，新聞から得たデータを精度良く識別することは困難である．

そこで，ある領域 S の訓練データによって学習した分類器を別の領域 T のデータに合うようにチューニングすることを領域適応 (Domain Adaptation) と呼ぶ．ここで，訓練データを獲得した領域 S をソース領域，テストデータを獲得した領域 T をターゲット領域と呼ぶ．例として，新聞から得た訓練データで分類器を学習し雑誌から得たテストデータに適用する流れを図 3.1 に示す．

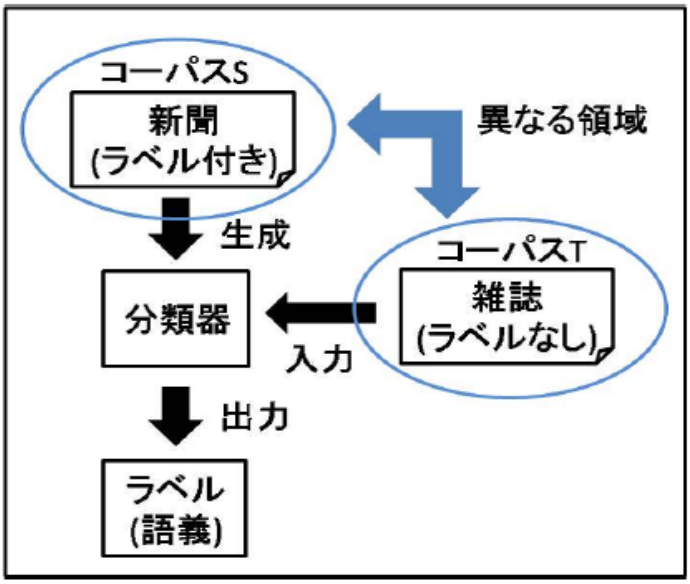

図 3.1: Domain Adaptation

第4章　深層学習

深層学習 (Deep Learning) は，ニューラルネットワーク (Neural Network，NN) を用いた教師無し学習法で，NN を多層に積み重ね学習することにより，抽象的なデータ表現を獲得する手法である．近年活発に研究されており，画像認識や音声認識の分野で有意な結果が得られている [4]．深層学習のモデルには確率論的モデルである RBM(Restricted Boltzmann Machines) や決定論的モデルの SdA(Stacked Denoising Autoencoder)，RBM の積み重ねからなる DBN(Deep Belief Nets) などが存在する．本論文では，これらのモデルのうち，SdA を利用し，WSD の領域適応の問題解決を図る．SdA は dA(Denoising Autoencoder) と呼ばれる学習モデルを複数回実施するこで学習を積み重ねるモデルである．本章では SdA 及びその基盤となる Autoencoder，Denoising Autoencoder について概説する．

4.1　Autoencoder

Autoencoder(AE) は図 4.1 のような 3 層構造からなり，出力が入力を再現するようなモデルの学習を行う．

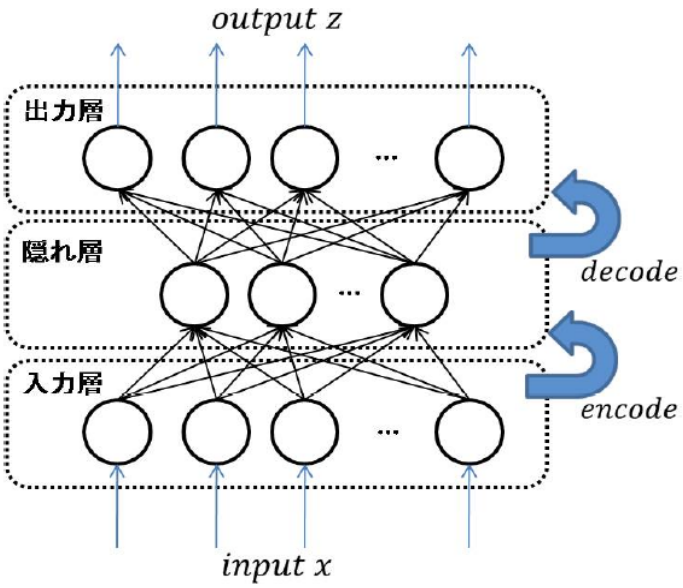

図 4.1: Autoencoder

まずはじめに，入力層にデータ x を与える．このデータに対して符号化 (encode) を

行い，隠れ層 y を得る．次に，隠れ層 y を複合化 (decode) し，出力層 z を獲得する．一般に，入力層と隠れ層のノード数は同じであり，隠れ層はこれよりも少ないノード数を設定する．AE は出力層 z と入力データ x の差ができるだけ小さくなるような符号化器，複合化器の学習を行う．このような過程で得られた隠れ層 y は，入力データよりも少ないノード数であるが，複合化器によって入力層 (≒出力層) を復元することができる．従って，隠れ層 y は入力データ x をより抽象的に表現していると捉えることができる．

4.2 Denoising Autoencoder

Denoising Autoencoder(dA) は図 4.2 のように AE の入力データ x に対して確率的にノイズを付加する．その後，AE と同様に入力データ x と出力層 z の差ができるだけ小さくなるようなモデルを学習する．ここで，出力層 z との差を比較する入力データ x はノイズを付加する前のデータであり，入力層 x' とは異なることに注意されたい．従って，dA による学習では初めに付与したノイズを除去するようなモデルを学習する．

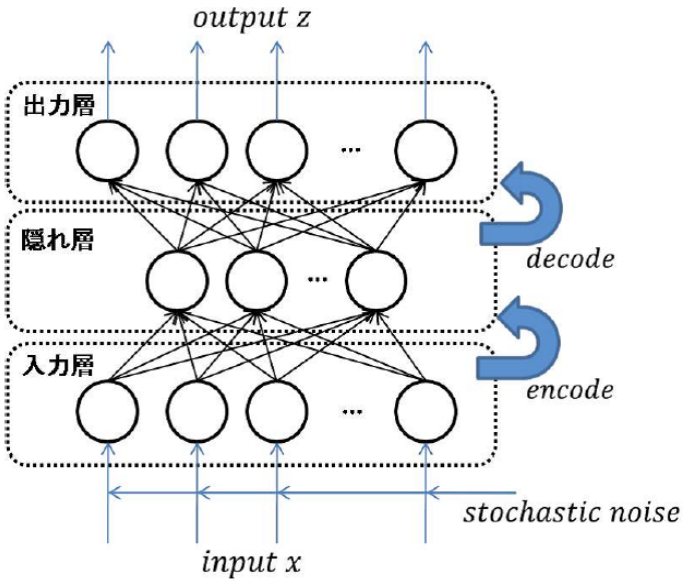

図 4.2: Denoising Autoencoder

入力層 x' から隠れ層 y，隠れ層 y から出力層 z への写像は次のように表される．

$$y = sigmoid(Wx' + b) \tag{4.1}$$
$$z = sigmoid(W^T y + b') \tag{4.2}$$

ここで，b，b' はバイアス，W は重み行列を示し，W^T はその転置行列である．また，$sigmoid()$ はシグモイド関数を表す．確率的勾配降下法 (Stochastic Gradient Descent, SGD) により，dA では次の式で表される平均二乗誤差が最小となる W，b，b' を求める．

$$E_N = \|z - x\|^2 \tag{4.3}$$

このような方法で学習した符号化器を通して獲得した隠れ層のデータ y は入力データ x の抽象的な表現となる．

4.3 パラメータの更新式

dA では平均二乗誤差が最小となるパラメータ W，b，b' を求める．ここでは，確率的勾配降下法による各パラメータの更新式を導出する．

前節で述べた通り，入力層 x' から隠れ層 y，隠れ層 y から出力層 z への写像はそれぞれ次の用に表される．

$$\begin{aligned} y &= sigmoid(Wx' + b) \\ z &= sigmoid(W^T y + b') \end{aligned}$$

また，二乗誤差は次の式で表される．

$$E(x, z) = (z - x)^2$$

ここで,

$$\begin{aligned} h_1(x') &= Wx' + b \\ h_2(y) &= W^T y + b' \end{aligned}$$

と置けば,

$$\begin{aligned} y &= sigmoid(h_1) \\ z &= sigmoid(h_2) \end{aligned}$$

のように表せる．E の各パラメータにおける勾配は,

$$
\begin{aligned}
\frac{\partial E}{\partial W} &= \frac{\partial E}{\partial h_1}\frac{\partial h_1}{\partial W} + \frac{\partial E}{\partial h_2}\frac{\partial h_2}{\partial W} \\
\frac{\partial E}{\partial b} &= \frac{\partial E}{\partial h_1}\frac{\partial h_1}{\partial b} + \frac{\partial E}{\partial h_2}\frac{\partial h_2}{\partial b} \\
&= \frac{\partial E}{\partial h_1}\frac{\partial h_1}{\partial b} \\
&= \frac{\partial E}{\partial h_1} \\
\frac{\partial E}{\partial b'} &= \frac{\partial E}{\partial h_1}\frac{\partial h_1}{\partial b'} + \frac{\partial E}{\partial h_2}\frac{\partial h_2}{\partial b'} \\
&= \frac{\partial E}{\partial h_2}\frac{\partial h_2}{\partial b'} \\
&= \frac{\partial E}{\partial h_2}
\end{aligned}
$$

である．シグモイド関数の微分公式を用いれば，

$$
\begin{aligned}
\frac{\partial E}{\partial h_1} &= \frac{\partial E}{\partial y}\frac{\partial y}{\partial h_1} \\
&= \frac{\partial E}{\partial y} * y * (1-y) \\
\frac{\partial E}{\partial h_2} &= \frac{\partial E}{\partial z}\frac{\partial z}{\partial h_2} \\
&= 2(z-x) * z * (1-z)
\end{aligned}
$$

となる．従って，各パラメータにおける勾配は

$$
\begin{aligned}
\frac{\partial E}{\partial W} &= (W(2(z-x) * z * (1-z)) * y * (1-y))x'^T \\
&\quad + ((2(z-x) * z * (1-z)) * y^T)^T
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial E}{\partial E} &= W(2(z-x) * z * (1-z)) * y * (1-y) \\
\frac{\partial E}{\partial b'} &= 2(z-x) * z * (1-z)
\end{aligned}
$$

であり，各パラメータの更新式は以下のようになる．

$$
\begin{aligned}
W_{t+1} &= W_t - \frac{\epsilon}{N}\sum_{n=1}^{N}\frac{\partial E}{\partial W_t} \\
b_{t+1} &= b_t - \frac{\epsilon}{N}\sum_{n=1}^{N}\frac{\partial E}{\partial b_t} \\
b'_{t+1} &= b'_t - \frac{\epsilon}{N}\sum_{n=1}^{N}\frac{\partial E}{\partial b'_t}
\end{aligned}
$$

4.4 Stacked Denoising Autoencoder

Stacked Denoising Autoencoder(SdA) は，dA による学習を複数回施行し，学習を積み重ねるモデルである．まず，1 つ目の dA によって学習を行う．次に，得られた隠れ層を入力として再度 2 つ目の dA によって学習する．さらに，2 つ目の dA の隠れ層を入力とする 3 つ目の dA による学習を行う．これを繰り返し行うことで，dA による学習を積み重ね，生のデータからデータの抽象的な表現を段階的に獲得する．従って，dA によって得られた出力層は誤差関数を求めるだけに使用し，SdA ではこの過程で得られた隠れ層を利用する．

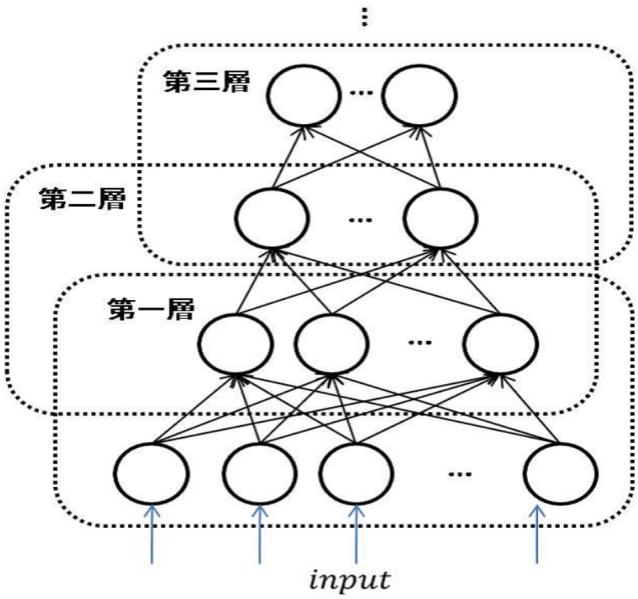

図 4.3: Stacked Denoising Autoencoder

第5章　SdAを利用した語義曖昧性解消の教師なし領域適応

WSDにおいて教師有り学習などで識別を行う場合，注釈付きコーパスから素性と呼ばれる特徴量を抽出し，これを用いて学習を行う．素性は対象単語と共起する単語の有無やその単語の品詞などを数値化するのが一般的である．WSDのタスクにおいては訓練データとテストデータが異なる領域から得られているようなケースが存在し，このような場合，識別精度の低下が危惧されることは第3章で既に述べた．これは，異なる領域から得た文章はその話題や内容に差があり，共起語が大きく異なるため，その素性に大きな差が生じることによる影響であると考えられる．例として第2章で挙げた「のむ」という単語について考える．この単語が新聞から得た文章に出現した場合，次のような文が考えられる．

1. 条件をのんだ.

2. 契約をのんだ.

一方，この単語を得たデータがブログや書籍などであった場合，次のような文が頻出する．

1. 水をのむ.

2. 薬をのませた.

3. 息をのんだ..

新聞から得たデータにおける素性は「条件」や「契約」といった単語の特徴がよく表れる．一方，ブログや書籍などから得たデータの素性は「水」や「薬」,「息」などといった単語の特徴が反映される．このようにして，例えば新聞から得たデータの素性で分類器を学習し，ブログや書籍などの文章中に出現する単語を識別しようとした場合，分類器には「水」や「薬」,「息」などの単語に関する情報がほとんどなく，識別が困難になってしまう．

本論文では，深層学習を利用してこの問の解決を図る．第4章で述べたように，深層学習は与えられたデータから抽象的な表現を獲得する機械学習手法である．深層学習によって，データの抽象的な表現をうまく獲得することができれば,「水」や「契約」といった具体的な単語に関する素性ではなく,「のむ」という対象単語の全体像を表現した素性になり，上述した問題を緩和させることができる．そこで，SdAを用いて獲得したデータの抽象的な表現を入力データに付加することで，当該問題の解決を図る．図5.1にSdAを利用したWSDの教師無し領域適応の流れを示す.

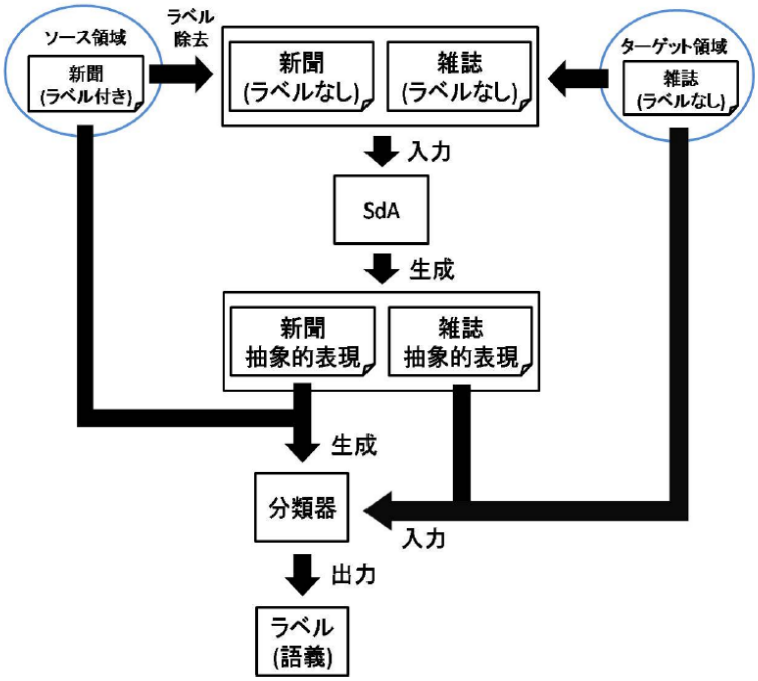

図 5.1: SdA を利用した WSD の教師無し領域適応

具体的には，SdA を利用して n_x 次元の入力データ x から n_{abst} 次元の抽象的表現 x_{abst} を抽出する．そして，入力データ x 及び抽象的表現 x_{abst} を結合した $n_x + n_{abst}$ 次元のデータを得る．これを Support Vector Machines(SVM) によって識別する．SdA による抽象的表現 x_{abst} の獲得及び入力データ x との結合の流れを図 5.2 に示す．

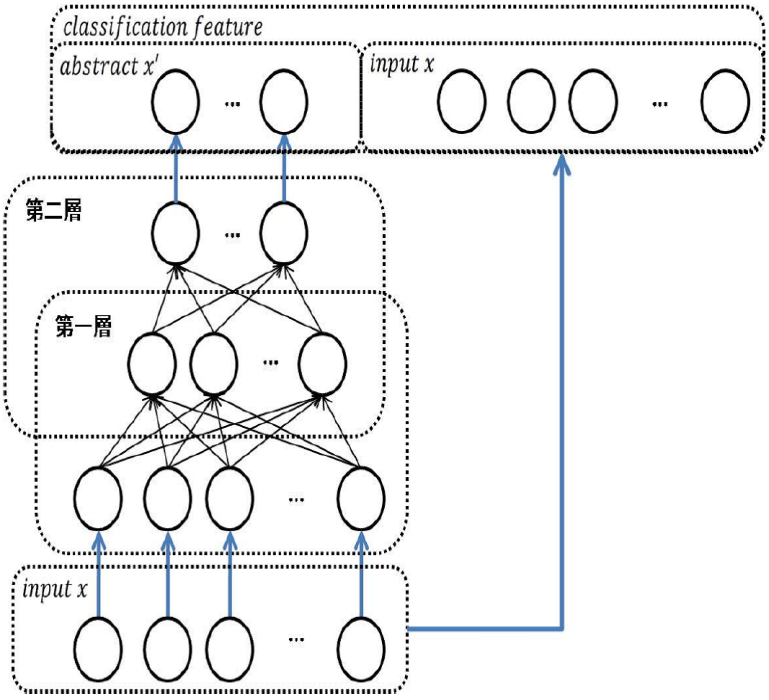

図 5.2: 識別素性獲得の流れ

第6章　素性の類似度

従来の研究から，WSD の領域適応に対して，単一の手法を適用した場合，対象単語とソース領域，ターゲット領域の組み合わせによって，識別精度が改善されるケースと悪化してしまうケースが存在し，平均して識別精度に変化がない，あるいは悪化してしまうことが知られている．本論文の第 5 章で提案した SdA による手法もこの手法単一で適用した場合，平均して識別精度が改善されないことが考えられる．

通常，dA における隠れ層のノード数は入力層や出力層と比較して少ないノード数を設定する．従って，SdA は入力データの次元を縮約する形で抽象的な素性を獲得していく．訓練データとテストデータの共通する素性が極端に少ないような場合，SdA が適切なモデルを学習するには大量のデータが必要である．しかし，一般的に WSD のタスクにおいては訓練データとテストデータはそれほど多くないく，2 領域の共通する素性が極端に少ない場合には，SdA が誤ったモデルを学習してしまう可能性が高い．結果的に，SdA を適用することで識別精度が低下してしまうことが考えられる．

そこで本論文では，ソース領域とターゲット領域の素性の類似度を定義し，対象単語ごとにこれを測定して，類似度が極端に低い場合には SdA を適用せず，そのままのデータを SVM によって識別する方法を取る．

一般的に類似度はコサイン類似度や相互情報量が尺度として用いられることが多いが，本論文では素性の類似度を以下の式で単純に表す．

$$P_f \quad = \quad \frac{\vec{T} \cdot \vec{S}}{n}$$

ここで，$\vec{S}$，$\vec{T}$ は訓練データ X_S とテストデータ X_T で各素性が出現するかどうかの生起ベクトルを表す．例えば，k 個の d 次元文章データからなる訓練データ X_S の素性の生起ベクトル $\vec{S}$ は d 次元のベクトルとなる．また一般的に，WSD のタスクで利用されるデータは訓練データとテストデータの次元数が共通であり，これを n で表す．このような式で表される類似度 P_f の値が，閾値 T よりも小さい場合には，訓練データとテストデータの素性の類似度が極端に低いとみなし，SdA を適用せずに，そのままのデータを SVM によって識別する．類似度 P_f が閾値 T よりも大きければ，SdA が適切なモデルを学習できると判断し，SdA による抽象的な素性を付加したデータを SVM によって識別する．

第7章　実験

7.1　実験概要

WSD の領域適応のタスクに SdA のモデルを用いた深層学習を利用することの有効性を検証する．実験データとして，現代日本語書き言葉均衡コーパス (Balanced Corpus of Contemporary Written Japanese，BCCWJ)[8] における Yahoo!知恵袋 (OC)，書籍 (PB) 及び新聞 (PN) のデータのうち，多義語 16 単語を利用する．SemEval-2 の日本語 WSD タスクではこれらの領域のコーパスの一部に語義タグを付けたデータを公開しており，そのデータを利用する [9]．表 7.1 に各対象単語の語義数及び各領域における語義数とデータ数を示す.

表 7.1: Target words

単語	dictionary of senses	OC freq of word	OC of senses	PB freq of word	PB of senses	PN freq of word	PN of senses
言う	3	666	2	1114	2	363	2
入れる	3	73	2	56	3	32	2
書く	2	99	2	62	2	27	2
聞く	3	124	2	123	2	52	2
子供	2	77	2	93	2	29	2
時間	4	53	2	74	2	59	2
自分	2	128	2	398	2	71	2
出る	3	131	3	152	3	89	3
取る	8	61	7	81	7	43	7
場合	2	126	2	137	2	73	2
入る	3	68	4	118	4	65	3
前	3	105	3	160	2	106	4
見る	6	262	5	273	6	87	3
持つ	4	62	4	153	3	59	3
やる	5	117	3	156	4	27	2
ゆく	2	219	2	133	2	27	2
平均	3.44	148.19	2.94	199.56	3.00	75.56	2.69

本実験では，領域適応は 3 つの領域 (OC，PB，PN) の組み合わせ 6 通り (OC → PB,

OC → PN，PB → OC，PB → PN，PN → OC，PN → PB) を行い，比較対象として以下に示す 5 つの手法を比較する．

1. baseline：そのままの入力データを SVM によって分類
2. unconstrained least squares importance fitting(uLSIF)[10]
3. Structural correspondence learning(SCL)[11]
4. SdA(提案手法 1)
5. 素性の類似度を導入した SdA(提案手法 2)

比較方法は，マクロ平均を採用する．マクロ平均とは，N 回のテストを実施した場合に i 回目のテストが n_i 回のテストで成り立つ場合，テストが正解した回数を x_i 回とすると，以下の式で表される値を平均正解率とするものである．

$$\frac{1}{N}\sum_{i=1}^{N}\frac{x_i}{n_i} \tag{7.1}$$

初めに対象単語とソース領域，ターゲット領域の組み合わせごとに正解率を算出し，対象単語ごとにその平均を取って平均正解率を求める．そして，単語ごとの平均正解率の平均を算出し，これを各手法の精度とする．

本実験で使用するデータの素性は表 7.2 に示す 8 種類の素性からなる．ただし，w は対象単語を表し，対象単語の 1 単語前の単語を w_{-1}，1 単語後の単語を w_1 のように表す．

表 7.2: 文の素性

素性	内容
(e0)	w の表記
(e1)	w の品詞
(e2)	w_{-1} の表記
(e3)	w_{-1} の品詞
(e4)	w_1 の表記
(e5)	w_1 の品詞
(e6)	w_{-3} から w_3 に出現する独立後の表記
(e7)	(e6) の分類語彙表の値

7.2 SdA

SdA の学習には，DeepLearning ライブラリとして公開されている Pylearn2[1]を利用する．SdA における dA の学習回数は 2 回とし，初めに学習を行う dA を dA_1，二番

[1]http://deeplearning.net/software/pylearn2/

目に学習を行う dA を dA_2 とする．入力データの次元数を N_x 次元とした場合の dA_1，dA_2 における入力層，隠れ層，出力層のノード数は表 7.3，表 7.4 のようである．ただし，計算結果に少数が含まれる場合には，小数点以下第 1 位の四捨五入による端数処理を行う．

表 7.3: dA_1 の各層のノード数

層	ノード数
入力層	N_x
隠れ層	$2/3 * N_x$
出力層	N_x

表 7.4: dA_2 の各層のノード数

層	ノード数
入力層	$2/3 * N_x$
隠れ層	$2/3 * N_x$
出力層	$2/3 * N_x$

従って，SdA の各層のノード数は表 7.5 のようになる．例えば入力データの次元数を 5 とした場合の SdA 図を 7.1 に示す.

表 7.5: SdA の各層のノード数

層	ノード数
入力層	N_x
第一層	$2/3 * N_x$
第二層	$2/3 * N_x$

素性の類似度を導入する SdA による手法では，素性の類似度 P_f の値が，閾値 T よりも大きい場合に SdA を適用するが，本実験ではこの閾値 T を 0.2 に設定した．

それぞれの手法で獲得したデータをサポートベクターマシン (Support Vector Machines，SVM) によって識別する．SVM による識別を行うデータは SdA によって獲得した素性 (縮約素性) と入力データの素性 (入力素性) を結合したものを利用する．ただし，そのまま 2 つの素性を結合した場合，縮約素性は入力素性と比べて次元数が少ないことは明らかであり，入力素性の情報がより強く結果に表れてしまう．そこで，入力素性と縮約素性の重みを等しくするために，それぞれの素性を正規化したものを結合する．SVM による識別には libsvm[2]を用いる．

[2]http://www.csie.ntu.edu.tw/?cjlin/libsvm/

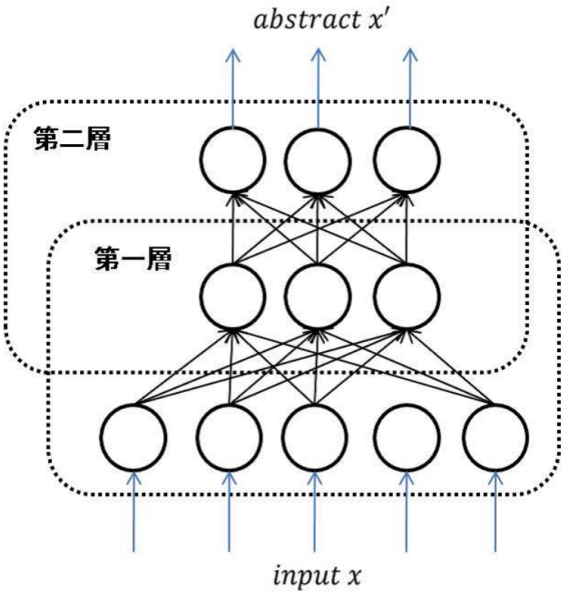

図 7.1: SdA(input Dim = 5)

7.3 実験結果

それぞれの手法で識別した結果を表 7.6 に示す.

表 7.6: 実験結果 (%)

領域適応	baseline	uLSIF	SCL	SdA	SdA using similarity
OC → PB	71.33	71.34	71.34	71.09	71.43
OC → PN	68.81	68.98	69.24	68.18	68.81
PB → OC	70.10	70.45	70.18	71.01	70.93
PB → PN	76.76	76.99	76.65	77.33	77.02
PN → OC	69.09	69.05	68.94	67.49	69.24
PN → PB	74.55	74.50	73.47	75.37	74.59
平均	71.77	71.89	71.64	71.74	72.00

baseline と比較して，uLSIF では 4 つの領域適応 (OC → PB，OC → PN，PB → OC，PB → PN) で，SCL では 3 つの領域適応 (OC → PB，OC → PN，PB → OC) で識別精度が改善された．また，SdA でも同様に 3 つの領域適応 (PB → OC，PB → PN，PN → PB) で識別精度が改善されている．一方で，そのほかの領域適応においては識別精度が低下してしまっている．素性の類似度を導入した SdA では，5 つの領域適応で識別精度が改善され，もう 1 つの領域適応においても baseline と同水準の精度を得た．結果，平均して最も良い精度を示した．

第8章　考察

本論文では，WSD の領域適応のタスクの主問題がソース領域とターゲット領域のデータにおける素性の相違にあると考え，深層学習モデルのうち，SdA を利用し，識別精度の改善を試みた．SdA の核であるモデルの dA は入力層や出力層と比べて少ないノード数であるにもかかわらず，入力データを復元することができるという観点から，隠れ層は入力データを抽象的に表現していると言える．従って，これをうまくデータに付加することができれば，ソース領域とターゲット領域の素性の相違を緩和できると考えた．

実験では，uLSIF や SCL といった過去に提案された手法と比較し，SdA が同程度の精度を示すことを確認した．しかしながら，SdA 単体ではデータによって識別精度が改善されなかったケースが存在する．ここでは，その主要因について考察する．

8.1　素性の類似度の導入

表 7.6 に示した通り，uLSIF と SCL，SdA の 3 つの手法では，ソース領域とターゲット領域の組み合わせによって識別精度が改善されている場合とそうでない場合が存在する．これは，第 6 章で述べた通り，WSD の領域適応のタスクにおいてはソース領域とターゲット領域，単語の組み合わせによって適した手法があり，単一の手法ではすべてのデータで精度が改善されず，平均してそれほど良い結果が得られていないと考えられる．そこで，実験では第 6 章で提案した素性の類似度を導入した SdA(以下，提案手法と呼ぶ) の精度も比較した．結果，素性の類似度を導入した SdA では，baseline と比較して 5 つの領域適応で識別精度が改善され，もう 1 つの領域適応においても baseline と同水準の精度を得た．

しかしながら，SdA をそのまま適用した方法で basline よりも精度が改善された 3 つの領域適応 (PB → OC，PB → PN，PN → PB) では，提案手法は SdA よりも精度が落ちていることがわかる．これは，提案手法が類似度が低い場合に SdA を適用しないということを行っており，提案手法は SdA よりも baseline に近い結果を出しているためである．図 8.1 に示すように提案手法は素性の類似度によって SVM(baseline) をするか，SdA を適用するかを選択しているため，2 手法の中間的な結果を出力していると言える．その結果，SdA で精度が改善された場合に提案手法では SdA よりも精度が落ちてしまっていると考えられる．

このような場合に提案手法が SdA と同程度の精度を得ることができれば更なる識別精度改善が見込める．

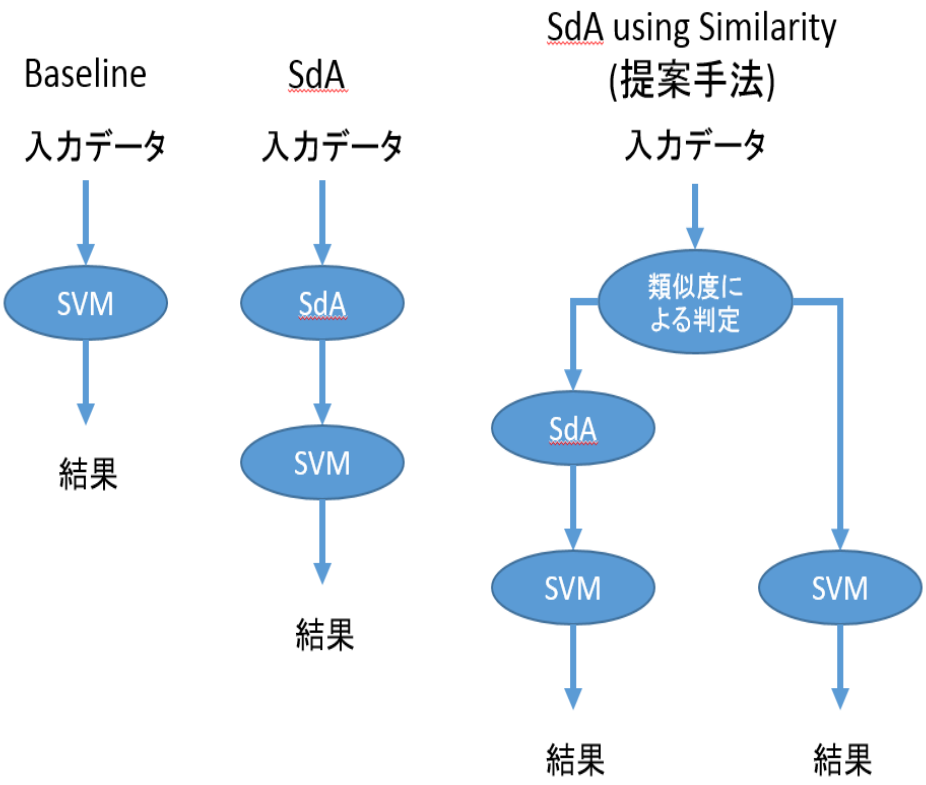

図 8.1: 各手法の流れの比較

8.2 パラメータ

各パラメータを変更することで，提案手法の精度を向上させることができると考えられる．SdA のパラメータは主に dA の学習回数と dA の隠れ層のノード数が存在する．また，第 7 章で述べたように，本実験では SdA によって獲得したデータを入力データに付加する形で利用しているが，この時，それぞれを正規化したものを結合している．正規化することで，2 つのデータが結果に与える重みを均等にしているがこの重みもパラメータとして捉えることができる．

dA の隠れ層のノード数を小さくした場合，隠れ層は入力データをより抽象的に表現したデータとなる．従って，素性の類似度による判定の後，類似度が低い場合にはこのノード数を大きくすることで提案手法は baseline により近い結果を得ることができる．類似度が高い場合にはノード数を小さくすることで，より抽象的に変換し，より良い精度が得られると考えられる．類似度が低い場合に SdA を適用しないのではなく，ノード数を適宜変更することでより柔軟に対応でき，精度が改善されると考えられる．

dA の学習回数についても学習回数を増やせば，データをより抽象的に変換していると捉えることができる．また，抽象的なデータと入力データの結合時の重みに関しても同様のことが言える．従って，この 2 つのパラメータに関しても隠れ層のノード数と同様にそのパラメータを適宜変更することが望ましい．

8.3　手法の選択

前節では，類似度によって各パラメータを変更することで精度が改善されると考えたが，パラメータを変更したところで，SdA がデータに適した形になるとは限らない．既に何度も述べたように，ソース領域，ターゲット領域と対象単語の組み合わせによって手法が適している場合とそうでない場合が存在する．これは，表 7.6 の実験結果にも表れている．ここで，uLSIF，SCL，SdA の 3 つの手法だけで比較した場合でも，6 つの領域適応すべてで，いずれかの手法が baseline よりも良い結果を示していることがわかる．すなわち，素性の類似度，あるいは他の何らかの尺度によって適用する手法自体を選択することができれば，より多くのデータで精度が改善され，平均して良い結果が得られると考えられる．

第9章　結論

本論文ではWSDの領域適応のタスクにおいてSdAを用いてデータの抽象的な表現を抽出し，素性に加えることで識別精度の改善を試みた．

領域適応の主問題として同じ単語に関するデータであっても，ソース領域とターゲット領域の素性には大きな差があり，ソース領域のデータで学習した分類器ではターゲット領域のデータをうまく識別できず，識別精度が低下してしまうことが挙げられる．SdAによってデータの抽象的な表現を獲得することができれば，この2領域間の素性の相違を緩和できると考えた．また，ソース領域とターゲット領域，対象単語の組み合わせによって手法が適している場合とそうでない場合が存在すると考えられるため，素性の類似度と呼ばれる概念を導入し，SdAがデータに適しているかどうかを判定し，SdAを適用するかどうかの選択をさせる方法を提案した．

実験ではBCCWJの3つの領域のデータのうち16個の多義語を対象に，baseline(SVM)，uLSIF，SCL，SdA，素性の類似度を導入したSdAの5つの手法で比較した．その結果，SdAはuLSIFやSCLと同程度の精度を得ることが分かった．しかし，SdAはbaselineと比較してデータによって精度が改善されるケースとそうでないケースが存在した．素性の類似度を導入することでSdAはそのまますべてのデータに手法を適用するよりも平均して良い結果となり，baselineと比較してすべての領域適応で同精度以上の結果を示した．さらに，素性の類似度を導入したSdAは5つの手法の中でも平均して最も良い結果を得た．

SdAをすべてのデータに適用した場合に精度が改善された領域の組において，素性の類似度を導入した場合の結果が導入しない場合の結果よりも精度が落ちていることに関して，導入しない手法と同程度の精度を得るように手法を改善することが今後の課題である．

謝辞

本研究を進めるにあたり，多くのご指導，ご協力を頂いた指導教員の新納浩幸教授に感謝致します．また，日常の議論を通じて多くの知識や示唆を頂いた佐々木稔講師，古宮嘉那子講師と新納研究室の皆様に感謝します．

参考文献

[1] Anders Søgaard. ”Semi-Supervised Learning and Domain Adaptation in Natural Language Processing”. Morgan & Claypool, 2013.

[2] 新納浩幸，佐々木稔．”共変量シフト下の学習による語義曖昧性解消の教師なし領域適応”．自然言語処理，Vol.21，No.5，pp.1011-1035，2014.

[3] 新納浩幸，佐々木稔．”k 近傍法とトピックモデルを利用した語義曖昧性解消の領域適応”．自然言語処理，Vol.20，No.5，pp.707-726，2013.

[4] Quoc Le，Marc’Aurelio Ranzato，Rajat Monga，Matthieu Devin，Kai Chen，Greg Corrado，Jeff Dean，and Andrew Ng．”Building high-level features using large scale unsupervised learning”．In ICML-2012，2012.

[5] 河野和平，新納浩幸，佐々木稔，古宮嘉那子．”Stacked Denoising Autoencoder を利用した語義曖昧性解消の領域適応”．言語処理学会第 21 回年次大会，pp.896-899，2015.

[6] Kazuhei Kouno, Hiroyuki Shinnou, Minoru Sasaki and Kanako Komiya, ”Unsupervised Domain Adaptation for Word Sense Disambiguation using Stacked Denoising Autoencoder”.PACLIC-29, pp.224-231,2015.

[7] 高村大也,”言語処理のための機械学習入門”．コロナ社，2010.

[8] Kikuo Maekawa. ”Design of a Balanced Corpus of Contemporary Written Japanese”．In LKR-2007，pp.55-58，2007.

[9] Manabu Okumura，Kiyoaki Shirai，Kanako Komiya，Hikaru Yokono．”SemEval-2010 Task: Japanese WSD”．5th International Workshop on Semantic Evaluation,pp.69-74，2010.

[10] Takafumi Kanamori，Shohei Hido，and Masashi Sugiyama．”A Least-Squares Approach to Direct Importance Estimation”．The Journal of Machine Learning Research，10:1391-1445，2009.

[11] John Blitzer，Ryan McDonald，and Fernando Pereira．”Domain Adaptation with Structural Correspondence Learning”．In EMNLP-2006，pp.120-128.

ソースリスト

```
1 """
2 adult dataset wrapper for pylearn2
3 """
4
5 import csv
6 import numpy as np
7 import os
8
9 from pylearn2.datasets.dense_design_matrix import DenseDesignMatrix
10 from pylearn2.utils import serial
11 from pylearn2.utils.string_utils import preprocess
12
13 class AdultDataset( DenseDesignMatrix ):
14
15 def __init__(self,
16 path = 'train.csv',
17 one_hot = False,
18 with_labels = True,
19 start = None,
20 stop = None,
21 preprocessor = None,
22 fit_preprocessor = False,
23 fit_test_preprocessor = False):
24 """
25 which_set: A string specifying which portion of the dataset
26 to load. Valid values are 'train' or 'public_test'
27 base_path: The directory containing the .csv files from kaggle.com.
28 This directory should be writable; if the .csv files haven't
29 already been converted to npy, this class will convert them
30 to save memory the next time they are loaded.
31 fit_preprocessor: True if the preprocessor is allowed to fit the
32 data.
33 fit_test_preprocessor: If we construct a test set based on this
34 dataset, should it be allowed to fit the test set?
35 """
36 self.no_classes = 2
37
38 self.test_args = locals()
39 self.test_args['which_set'] = 'test'
40 self.test_args['fit_preprocessor'] = fit_test_preprocessor
41 del self.test_args['start']
42 del self.test_args['stop']
43 del self.test_args['self']
44
45 path = preprocess(path)
46 X, y = self._load_data( path, with_labels )
47
48
49 if start is not None:
50 assert which_set != 'test'
51 assert isinstance(start, int)
52 assert isinstance(stop, int)
53 assert start >= 0
54 assert start < stop
55 assert stop <= X.shape[0]
56 X = X[start:stop, :]
57 if y is not None:
58 y = y[start:stop, :]
59
60
61 super(AdultDataset, self).__init__(X=X, y=y)
62
63 if preprocessor:
64 preprocessor.apply(self, can_fit=fit_preprocessor)
65
66 def _load_data(self, path, expect_labels):
67
68 assert path.endswith('.dat')
69
70 libsvmformat = open(path, 'r')
71
72 l = []
73
74 for line in libsvmformat:
75 s = line.split()
76 m = []
77 for mtos in s:
```

```
78 tmp = mtos.split(':')
79 m.append(tmp)
80
81 l.append(m)
82
83 numpy_label = np.empty(len(l),dtype=str)
84 dim = 0
85
86 for i in range(len(l)):
87 if dim < int(l[i][-1][0]):
88 dim = int(l[i][-1][0])
89
90
91 numpy_feature = np.zeros((len(l),dim))
92
93 for i in range(len(l)):
94 numpy_label[i] = str(l[i][0][0])
95
96 for i in range(len(l)):
97 for j in range(len(l[i])-1):
98 numpy_feature[i][int(l[i][j+1][0])-1] = l[i][j+1][1]
99
100 data = np.zeros((len(numpy_label),dim+1))
101
102 for i in range(len(numpy_label)):
103 data[i][0] = int(0)
104 for j in range(dim):
105 data[i][j+1] = int(numpy_feature[i][j])
106
107 if expect_labels:
108 y = numpy_label
109 X = data[:,1:]
110
111 one_hot = np.zeros((y.shape[0], self.no_classes ),dtype='float32')
112 for i in xrange( y.shape[0] ):
113 label = y[i]
114 if label == 1:
115 one_hot[i,1] = 1.
116 else:
117 one_hot[i,0] = 1.
118
119 y = one_hot
120 else:
121 X = data
122 y = None
123
124 return X, y
```

ソースコード 1 学習用 libsvm 形式のデータの読み込み部分 (pylearn2)

```
#-*- coding: utf-8 -*-
# kouno

import cPickle as pickle
import sys
import numpy as np
import os

filename = 'train_v.dat'
pickle_file = 'dae_l1.pkl'
pickle_file2 = 'dae_l2.pkl'

fp = open(pickle_file)
par = pickle.load(fp)
pv = par.get_param_values()

fp2 = open(pickle_file2)
par2 = pickle.load(fp2)
pv2 = par2.get_param_values()

libsvmformat = open(filename, 'r')

l = []

for line in libsvmformat:
s = line.split()
m = []
for mtos in s:
tmp = mtos.split(':')
m.append(tmp)

l.append(m)

numpy_label = np.empty(len(l),dtype=str)
dim = 0

for i in range(len(l)):
if dim < int(l[i][-1][0]):
dim = int(l[i][-1][0])

numpy_feature = np.zeros((len(l),dim))

for i in range(len(l)):
#print "label = ",l[i][0][0]
numpy_label[i] = str(l[i][0][0])

for i in range(len(l)):
for j in range(len(l[i])-1):
numpy_feature[i][int(l[i][j+1][0])-1] = l[i][j+1][1]

data = np.zeros((len(numpy_label),dim+1))

for i in range(len(numpy_label)):
#data[i][0] = int(numpy_label[i])
data[i][0] = int(0)
for j in range(dim):
data[i][j+1] = int(numpy_feature[i][j])

c = np.dot(numpy_feature,pv[2])

e = np.dot(c,pv2[2])

norm = numpy_feature
for i in norm:
i /= np.linalg.norm(i)

norm2 = e
for i in norm2:
i /= np.linalg.norm(i)

print np.shape(norm)
print np.shape(norm2)

x_new = np.c_[norm,norm2]
x_new_dl_only = norm2

label_oc = open('label-oc', 'r')
label_pb = open('label-pb', 'r')

class_oc = []
for line in label_oc:
oc = line.split()
class_oc.append(oc[-1])

class_pb = []
for line in label_pb:
pb = line.split()
class_pb.append(pb[-1])

f_oc = open('OCdl.dat', 'w')
f_pb = open('PBdl.dat', 'w')

```

```
93 f_oc_only = open('OCdl_only.dat','w')
94 f_pb_only = open('PBdl_only.dat','w')
95
96 oc = open('OC.dat','w')
97 pb = open('PB.dat','w')
98
99 print len(class_oc)
100
101 for i in range(len(class_oc)):
102 f_oc.write(class_oc[i][-1])
103 f_oc_only.write(class_oc[i][-1])
104 oc.write(class_oc[i][-1])
105 for j in range(len(x_new[i])):
106 if x_new[i][j] != 0:
107 f_oc.write('  '+str(j+1)+':'+str(x_new[i][j]))
108 for j in range(len(x_new_dl_only[i])):
109 if x_new_dl_only[i][j] != 0:
110 f_oc_only.write('  '+str(j+1)+':'+str(x_new_dl_only[i][j]))
111 for j in range(len(numpy_feature[i])):
112 if numpy_feature[i][j] != 0:
113 oc.write('  '+str(j+1)+':'+str(numpy_feature[i][j]))
114 f_oc.write('\n')
115 f_oc_only.write('\n')
116 oc.write('\n')
117
118 for i in range(len(class_oc),len(class_oc)+len(class_pb)):
119 f_pb.write(class_pb[i-len(class_oc)][-1])
120 f_pb_only.write(class_pb[i-len(class_oc)][-1])
121 pb.write(class_pb[i-len(class_oc)][-1])
122 for j in range(len(x_new[i])):
123 if x_new[i][j] != 0:
124 f_pb.write('  '+str(j+1)+':'+str(x_new[i][j]))
125 for j in range(len(x_new_dl_only[i])):
126 if x_new_dl_only[i][j] != 0:
127 f_pb_only.write('  '+str(j+1)+':'+str(x_new_dl_only[i][j]))
128 for j in range(len(numpy_feature[i])):
129 if numpy_feature[i][j] != 0:
130 pb.write('  '+str(j+1)+':'+str(numpy_feature[i][j]))
131 f_pb.write('\n')
132 f_pb_only.write('\n')
133 pb.write('\n')
134
135 f_oc.close()
136 f_pb.close()
137 f_oc_only.close()
138 f_pb_only.close()
139 oc.close()
140 pb.close()
```

ソースコード 2 識別素性 (入力素性+縮約素性) を生成するプログラム