平成 30 年度茨城大学工学部情報工学科
卒業研究論文

Variational AutoEncoder を利用した感情分析の領域適応

平成 31 年 2 月 4 日提出

茨城大学 工学部情報工学科
15T4032G 芝山 直希
指導教員： 新納 浩幸 教授

平成 30 年度茨城大学工学部情報工学科卒業研究

Variational AutoEncoder を利用した感情分析の領域適応

氏名：15T4032G 芝山 直希

指導教員：新納 浩幸 教授

論文要旨

本論文では Variational AutoEncoder (以下、VAE）を次元縮約器として利用し、文書データをソース領域とターゲット領域の共通の空間に埋め込む。その埋め込みベクトルを追加の素性として利用することで、感情分析の領域適応を試みる。

感情分析とはレビュー文書（例えば映画のレビュー）が肯定的なものか、否定的なものかを判定するタスクである。これは文書分類の一種であり、教師あり学習を用いて解決できる。しかし判定先の文書が学習データの領域とは異なる領域の文書（例えば書籍のレビュー）であった場合に、教師あり学習で得られた分類器の精度が下がってしまう。これが領域適応の問題である。

領域適応の問題に対する有効な手法の一つとして、データをソース領域とターゲット領域の共通の空間に埋め込むことが行われている。共通の空間で学習と識別を行えば、領域適応の問題は起こらない。問題はどのようにして共通の空間に埋め込むかである。簡易な方法として、ソース領域の空間とターゲット領域の空間を合わせた空間を考え、主成分分析（特異値分解）を用いて次元を縮約することが考えられる。一方、近年、深層生成モデルとして VAE が提案された [3] [2]。VAE の Encoder はデータに対してその潜在変数を求めるが、その潜在変数はデータを次元縮約したベクトルと見なすこともできる。このため VAE から構築できる潜在変数を、ソース領域とターゲット領域の共通の空間に埋め込まれたベクトルとして扱える。本論文では上記の考えに基づき、VAE を次元縮約器として利用し、感情分析の領域適応に応用する。具体的にはソース領域とターゲット領域の文書ベクトル x を VAE の Encoder により z に次元縮約する。学習と識別には x の代わりに x と z を連結した $[x; z]$ を用いる。

実験では次元縮約の手法として特異値分解 (SVD) を用いた物と比較し、感情分析の領域適応に対する VAE の利用可能性について検討した。その結果 VAE は有効ではないという結論を得たが、実験に問題点がないか再確認したところ、z の領域依存度が高くなる様 VAE を訓練していたことが分かった。

目次

第 1 章

はじめに

1.1 概要

本論文では Variational AutoEncoder (以下、VAE）を次元縮約器として利用し、文書データをソース領域とターゲット領域の共通の空間に埋め込む。その埋め込みベクトルを追加の素性として利用することで、感情分析の領域適応を試みる [8]。

感情分析とはレビュー文書（例えば映画のレビュー）が肯定的なものか、否定的なものかを判定するタスクである。これは文書分類の一種であり、教師あり学習を用いて解決できる。しかし判定先の文書が学習データの領域とは異なる領域の文書（例えば書籍のレビュー）であった場合に、教師あり学習で得られた分類器の精度が下がってしまう。これが領域適応の問題である。

領域適応の問題に対する有効な手法の一つとして、データをソース領域とターゲット領域の共通の空間に埋め込むことが行われている。共通の空間で学習と識別を行えば、領域適応の問題は起こらない。問題はどのようにして共通の空間に埋め込むかである。簡易な方法として、ソース領域の空間とターゲット領域の空間を合わせた空間を考え、主成分分析（特異値分解）を用いて次元を縮約することが考えられる。一方、近年、深層生成モデルとして VAE が提案された [3] [2]。VAE の Encoder はデータに対してその潜在変数を求めるが、その潜在変数はデータを次元縮約したベクトルと見なすこともできる。このため VAE から構築できる潜在変数を、ソース領域とターゲット領域の共通の空間に埋め込まれたベクトルとして扱える。本論文では上記の考えに基づき、VAE を次元縮約器として利用し、感情分析の領域適応に応用する。具体的にはソース領域とターゲット領域の文書ベクトル x を VAE の Encoder により z に次元縮約する。学習と識別には x の代わりに x と z を連結した $[x; z]$ を用いる。

実験では次元縮約の手法として特異値分解 (SVD) を用いた物と比較し、感情分析の領域適応に対する VAE の利用可能性について検討した。その結果 VAE は有効ではないという結論を得たが、実験に問題点がないか再確認したところ、z の領域依存度が高くなる様 VAE を訓練していたことが分かった。

1.2 構成

2 章では機械学習で問題を解決するためのモデルを作成する 2 種の手法に軽く触れた上で、本論文の解決目標である感情分析及び領域適応について説明する。3 章では VAE の基礎知識及び次元縮約について触れ、VAE を利用した領域適応の手法を提案する。4 章では実験の詳細及び実験結果について言及する。5 章では 4 章で得られた実験結果について考察する。

第 2 章

機械学習概要と感情分析・領域適応

機械学習で解決可能な問題において、用意するモデルの訓練手法は 2 つに大別することができる。まずこの 2 つの概要を示し、その後本論文のタスクとなる感情分析の領域適応について説明する。

2.1　教師あり学習・教師なし学習

図 2.1 に示すように、学習時に使用するデータ (訓練データ) にラベルなどの「正解」が付与されている手法を教師あり学習という。この手法では「正解」を参考にして訓練データが属する領域のデータを高い精度で予測出来るようにモデルを最適化する。主にデータの分類や過去のデータを用いた予測などはこの訓練手法で解決できる。

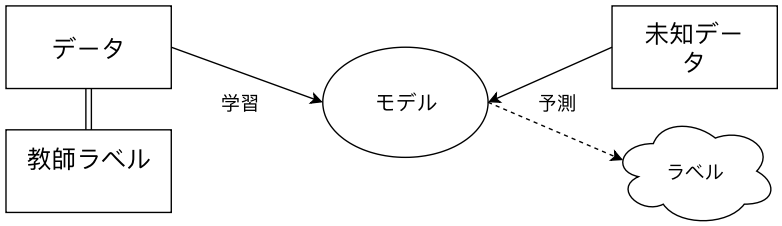

図 2.1　教師あり学習の概要

一方、訓練データにラベルなどの「正解」が付与されていないものを教師なし学習という。任意の基準をもとにモデルやその出力を最適化していく。本論文の要である Variational AutoEncoder では入力データそのものを基準とし、最終的な出力が入力に近い値になるようモデルを最適化する。

2.2　感情分析

感情分析とはレビュー文書の内容が肯定的か、否定的かを判別するタスクである。レビュー文書の例として、Webis-CLS10 [6] の日本語データに収録されている否定的文書

データの一つを紹介する。

> 前作「自分の小さな「箱」から脱出する方法」の内容とあまり変わらないです。前作とは違った発見とや、気付きを求めている方にとっては少し物足りないかもしれません。前作を何回も読んだ人にとっては復習みたいな感じになるのではないでしょうか。前作が画期的でハッとさせられただけに、、そんな期待を持っている人には物足りないと思います。

このタスクは文書分類の一種であり、教師あり学習を用いて解決可能である。このタスクの解決を目標とするデータセットは、各データが二値 (または多値) に分類・ラベル付与されている。本論文では、このタスク上で発生する教師あり学習の解決目標の一つである、領域適応の問題の解決を試みる。

2.3 領域適応

教師あり学習において教師データは有限であり、すべての領域に対応したモデルの学習は、計算資源及び教師データの収集・作成に要する労力を考慮すると非常に困難であるのは明らかである。しかし、識別対象となるデータと同じ領域に属する教師データが存在しない場合、識別精度は低下する。図 2.2 にも挙げているように、本のレビュー文書にのみ最適化された分類器では、家電製品のレビュー文書を上手く分類できないのである。これを領域適応の問題といい、タスクとしての領域適応はこの問題の解決を目的としている。

2.3.1 主な手法

ターゲット領域のラベル付きデータを用いない教師なしの領域適応の手法としては、一般に素性ベースの手法が用いられる [4]。素性ベースの手法の基本的な考え方はソース領域とターゲット領域の共通空間にそれぞれのデータをマッピングし、共通空間上で学習と識別を行うというものである。共通空間の求め方としては、単純には、ソース領域とターゲット領域を合わせた空間を特異値分解などで次元縮約を行えばよい。古典的手法である SCL [1] や近年提案されている CORAL [7] なども、この考え方に基づいている。本論文ではこの次元縮約する部分に VAE を利用したものと捉えられる。次元縮約したベクトルをそのまま利用することもできるが、SCL のように基のベクトルに連結して利用する方が、精度が高くなるため、本論文でもそのような使い方を行う。

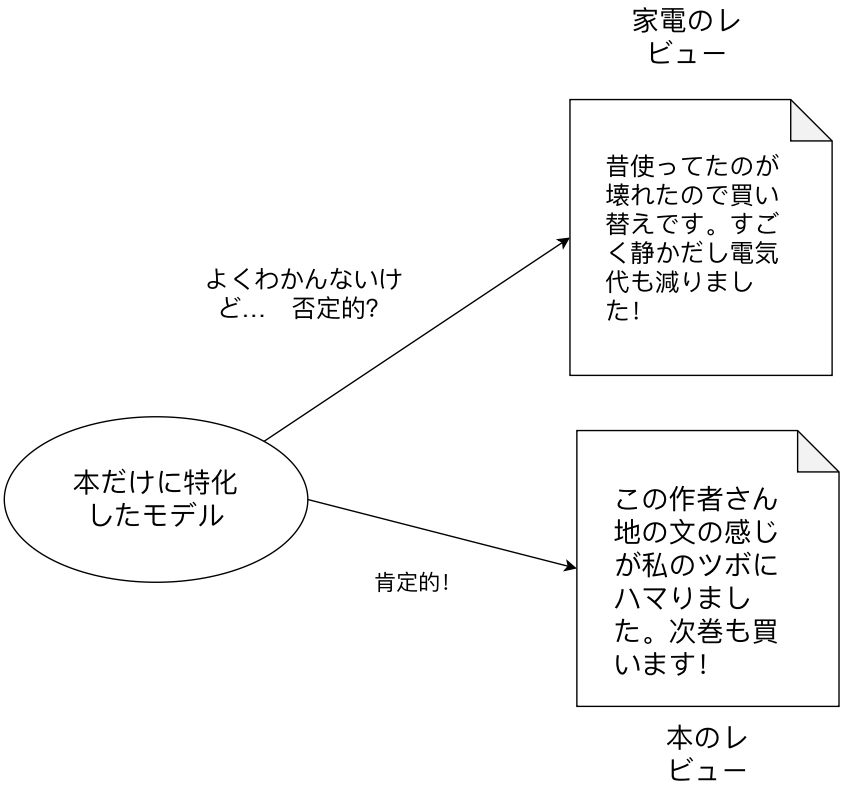

図 2.2　具体例：領域適応の問題

第 3 章

VAE と次元縮約

3.1 Variational AutoEncoder

VAE は [3] [2] にて提案された教師なし生成モデルであり、データ x を生成する確率モデル $p(x)$ を構築するのが目的である。基本的には $p(x)$ の対数尤度を最大化するモデルを求めればよい。

3.1.1 VAE の概要

理想的な VAE の基本的構造を図 3.1 に示す。VAE ではデータ x からその潜在変数 z

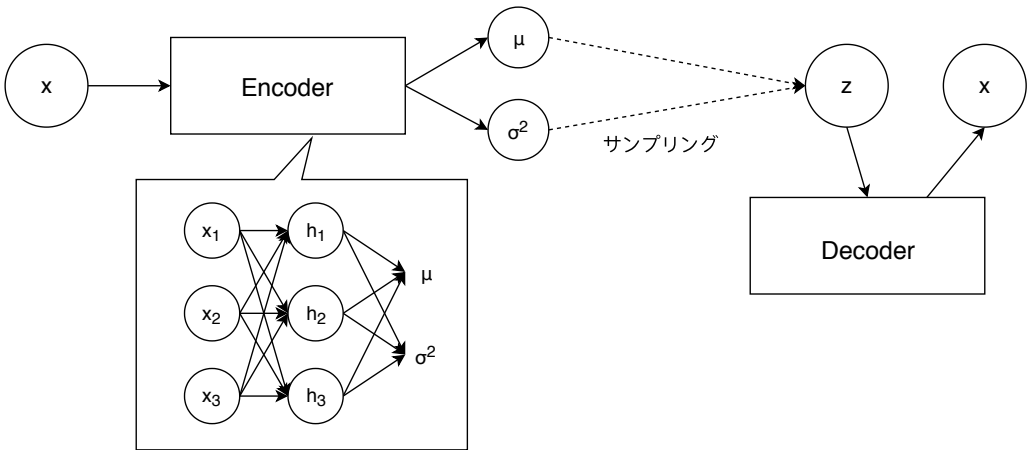

図 3.1 Variational AutoEncoder の基本構造

への Encoder と z から x を生成する Decoder を学習している。Encoder 及び Decoder は基本的なニューラルネットワークの構造のひとつである MLP[*1]を用いて表現される。Encoder はデータ x から x の属する確率分布の平均及び分散を出力し、得られた分布か

[*1] 多層パーセプトロン。入力ベクトルの各次元の値の重み付き和を出力の次元数分計算し、出力ベクトルの各要素の値とする構造である。

ら潜在変数 z をサンプリングする。Decoder には z が入力され、データの復号を試みる。

図 3.1 の様に $x = (x_1, x_2, x_3)$ であり隠れ層が 3 次元である場合、隠れ層となるユニットが持つ値及び Encoder の出力は次の様に計算される。ただし e は Encoder のパラメータである。

$$h_1 = e_{111}x_1 + e_{112}x_2 + e_{113}x_3 \tag{3.1}$$
$$h_2 = e_{121}x_1 + e_{122}x_2 + e_{123}x_3 \tag{3.2}$$
$$h_3 = e_{131}x_1 + e_{132}x_2 + e_{133}x_3 \tag{3.3}$$

その後 μ, σ^2 が計算される。z が 2 次元であるとき、次式の様に算出される。

$$\mu_1 = e_{211}h_1 + e_{212}h_2 + e_{213}h_3 \tag{3.4}$$
$$\mu_2 = e_{221}h_1 + e_{222}h_2 + e_{223}h_3 \tag{3.5}$$
$$\sigma_1^2 = e_{311}h_1 + e_{312}h_2 + e_{313}h_3 \tag{3.6}$$
$$\sigma_2^2 = e_{321}h_1 + e_{322}h_2 + e_{323}h_3 \tag{3.7}$$

VAE では、これらを正規分布の平均・分散であると仮定して z の各要素をサンプリングするのが一般的である。最後に Decoder に z を入力し Encoder と同様の計算を行い、x を得る。以上が理想的な VAE の演算の流れである。

本論文では VAE を次元縮約器として利用する。z を x の次元縮約したベクトルとして考える。

3.1.2 尤度の計算

データ x を生成する確率 $p(x)$ 及びその対数尤度は、潜在変数 z と z の確率分布 $q(z)$ を用いて次の様に表すことができる。

$$p(x) = \int p(x|z)p(z)dz \tag{3.8}$$
$$\log p(x) = \log \int p(x|z)p(z)dz \tag{3.9}$$
$$= \log \int q(z)\frac{p(x|z)p(z)}{q(z)}dz \tag{3.10}$$

Jensen の不等式[*2]を用いると式 3.10 は以下の様に変形できる。

$$\log p(x) \geq \int q(z) \log \frac{p(x|z)p(z)}{q(z)} dz \tag{3.11}$$

$$\geq \int q(z) \{\log \frac{p(z)}{q(z)} + \log p(x|z)\} dz \tag{3.12}$$

$$\geq \int q(z) \log p(x|z) dz - \int q(z) \log \frac{q(z)}{p(z)} \tag{3.13}$$

$$\geq -D_{KL}(q(z)||p(z)) + \int q(z) \log p(x|z) dz \tag{3.14}$$

VAE は $p(x)$ の対数尤度を最大化する代わりに、その下限である式 3.14 の右辺を最大化する。つまり式 3.14 の右辺の符号を反転させた式 3.15 を最小化することで $p(x)$ のパラメータを求める。

$$D_{KL}(q(z)||p(z)) - \int q(z) \log p(x|z) dz \tag{3.15}$$

式 3.15 で z は x の潜在変数であり、q は z の分布である。q に制限はなく、正規分布に設定すれば、式 3.15 の第 1 項は解析的パラメータの式に変形される。また式 3.15 の第 2 項は Reparametrization trick を用いることで、パラメータのサンプリングの形で表現できる。

3.1.3 Reparametrization trick

VAE は Encoder から出力される平均 μ 及び分散 σ^2 をもとに z をサンプリングするが、そのままサンプリングを行うとパラメータ更新時に VAE のパラメータで z を微分出来ず、ニューラルネットワークの学習アルゴリズムである誤差逆伝播法を Encoder に適用出来ない。この状態では Encoder のパラメータを更新することが出来ない。

そこで、図 3.2 に示されているように、以下の様に z を導出しサンプリングの代わりとすることで、z をパラメータで微分することができるようになる。ただし q は正規分布に設定されているものとする。

$$\begin{aligned} z &= \mu + \epsilon\sigma \\ \epsilon &\sim N(0,1) \end{aligned} \tag{3.16}$$

この計算方式を用いることで Encoder にも誤差逆伝播法を適用できる様になり、パラメータの更新が可能となる。

Reparametrization trick を用いると、VAE は図 3.2 のようなネットワーク構造となる。

[*2] $p(x) > 0, \int p(x)dx = 1$ を満たし、$p(x)$ 及び $y(x)$ が実数上での可積分関数であるならば、

$$\int_{-\infty}^{\infty} f(y(x))p(x)dx \geq f\left(\int_{-\infty}^{\infty} y(x)p(x)dx\right)$$

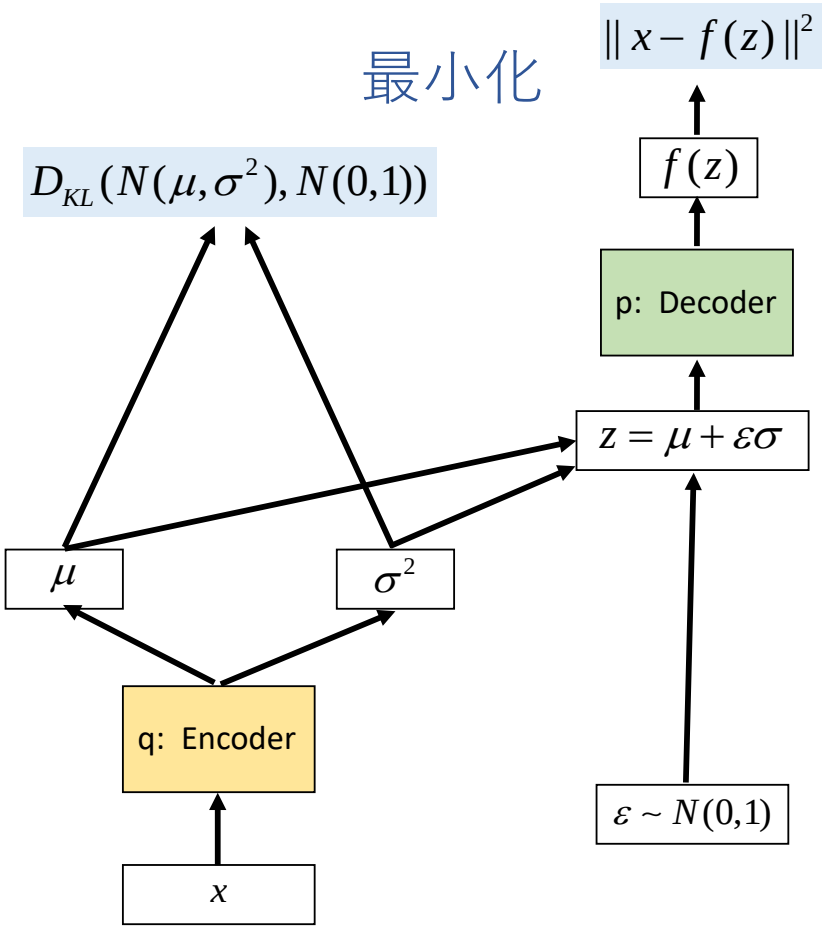

図 3.2　VAE のネットワーク図

3.2　VAE による次元縮約

次元縮約の標準的手法は特異値分解である。これは行列分解を利用した低ランク近似であり、次元縮約は線形変換により行われるため、その能力には自ずと限界がある。そのため非線形な変換法がいくつか提案されている。本論文で行う VAE による次元縮約もその 1 つと捉えられる。

3.2.1　次元縮約

次元縮約とは精度向上の為にデータの次元数を削減する操作である。多くの情報を持つデータは次元数が大きくなりやすいが、全ての情報が目的達成のために必要だとは限らない。また、次元数の大きいデータを扱う場合より多くの計算資源が必要となる。主にこの 2 点の解消を目的とした操作である為、重要な情報の抽出・計算資源の節約が次元縮約の主目的となる。

本論文では次元数が小さい共通空間上に、複数の領域にある各データをマッピングする為に次元縮約を行う。

3.2.2 提案手法

ここでは以下の 3 ステップにより訓練およびテストのデータを変換する（図 3.3 参照）。

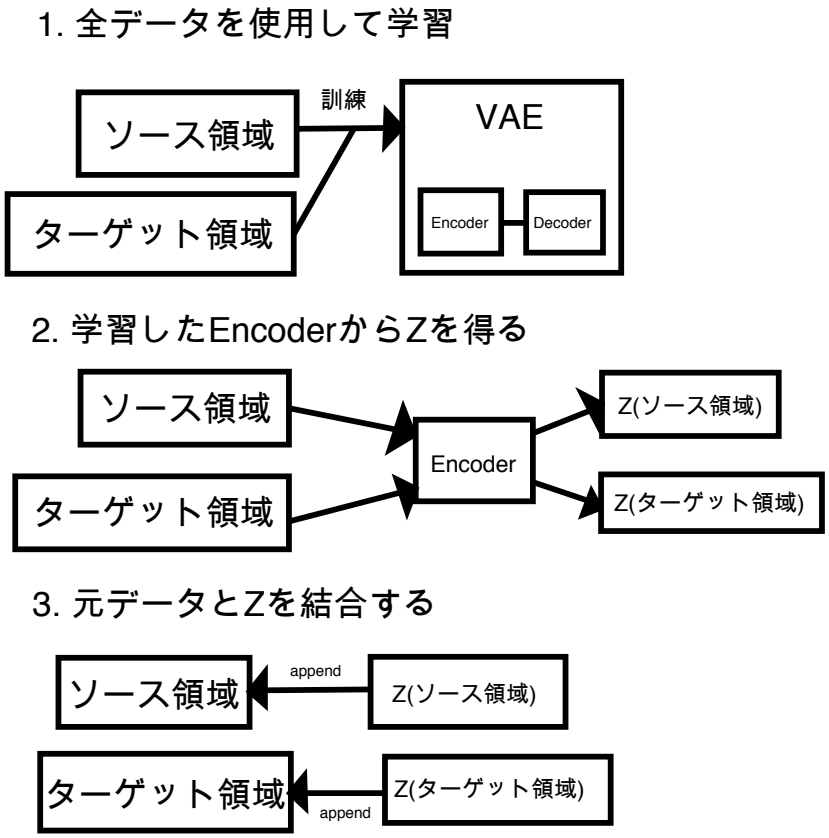

図 3.3　提案手法概要

1. ターゲット領域とソース領域の全データを用いて VAE の学習を行う。
2. 各領域のデータ x に対して VAE の Encoder を用いて、その次元縮約ベクトル z を得る。
3. x に z を連結したデータ $[x; z]$ を作成する。

以降の分類器の学習、テストデータの識別は、上記の $[x; z]$ を用いて行う。

この一連の工程が領域適応手法として機能するのは、学習完了後の VAE が出力するデータの学習領域に対する非依存性に基づいている。Encoder の出力から得られるデー

タは、元となるデータがターゲット領域またはソース領域に属する限り領域に依存しない。故に、対応するデータに出力から得られるデータを付加することが領域適応として動作する。

第 4 章

実験

4.1　使用ライブラリ

実験に用いたプログラミング言語は、機械学習向けライブラリが多数存在する Python[*1]である。本実験で使用したライブラリの概要を示す。

4.1.1　NumPy ・ SciPy

NumPy は多次元固定長配列クラスの他、大規模な数学関数ライブラリなどを有する Python の数値計算ライブラリである。NumPy の公式サイトの URL を以下に示す。

```
http://www.numpy.org
```

NumPy が提供する配列クラス ndarray を入力として受け付ける関数を持つライブラリは多く、Python での数値計算において欠かせないライブラリとなっている。

SciPy は NumPy を基礎とした科学計算用ライブラリである。このライブラリを用いて出来ることは多く、SciPy が提供する疎行列型配列は本実験における計算資源量の削減に大きく貢献した。SciPy の公式サイトの URL を以下に示す。

```
http://scipy.org
```

両ライブラリの論文執筆時の最新バージョンは、NumPy が 1.16.1、SciPy が 1.2.0 である。

4.1.2　Scikit-learn

Scikit-learn は多種多様な手法を実装している機械学習ライブラリである。機械学習アルゴリズムだけでなくデータの前処理や最適パラメータ (モデル) の探索など、機械学習

[*1] バージョン 2.x とバージョン 3.x 間には大きな変更点がある。本実験ではバージョン 3 系を使用した

に欠かせない様々な機能を有する。本実験ではデータの前処理に利用した他、評価用の分類器はこのライブラリに実装されているものを用いた。Scikit-learn の公式サイトの URL を以下に示す。

```
https://scikit-learn.org/stable/
```

論文執筆時の最新バージョンは 0.20.2 である。

4.1.3 Chainer

Chainer は Python が利用可能である数多くの深層学習向けライブラリの一つである。パラメータ更新時に必要となるデータ構造をプログラム実装時に動的に生成する特徴を有する。オープンソースライブラリであるため、ソースコードと共にサンプルプログラムが多数公開されている。本実験に用いた VAE はこのライブラリのサンプルプログラムを用いている。Chainer の公式サイトの URL を以下に示す。

```
http://chainer.org
```

4.2 使用データ

実験には Webis-CLS-10 [6] の日本語データ[*2]を使用した。このデータセットは books, DVD, music の 3 つの領域を持ち、それぞれ Bag of Words 形式で記録された文章データの集合である。各データは positive, negative の二値に分類・ラベル付与された Amazon カスタマーレビューであり、訓練用、テスト用、その他 (Unlabeled) の 3 つのファイルに分けられている。

4.3 実験方法

4.3.1 前処理

まず全領域の全データを用いて Bag of Words 形式から配列上の index に対応させる変換表を作成した。良い結果を得るために、作成工程において Unlabeled ファイルを最後に読み込むようにした。Unlabeled ファイルの情報を含まない変換表も別途作成した。

その後変換表を元に Python のライブラリの一つである SciPy の疎行列型の配列に変換して保存した。この際訓練用、テスト用のデータは 2 つの変換表から次元数の小さい配列と大きい配列にそれぞれ変換し保存した。以後 Unlabeled データに対応した変換表

[*2] [6] の本来のタスクはクロスリンガル感情分析であるが、各言語のデータは「対象言語での感情分析をタスクとするデータセット」として流用できる。

を用いて作成したデータを Large データ、対応していない変換表を用いて作成したデータを Small データとする。

4.3.2 提案手法の適用

実験を簡略化する為、全ての Large データを用いて VAE を学習した。VAE は Chainer のサンプルプログラムのモデルを利用した。学習時のハイパーパラメータを表 1 に示す。

表 4.1 Variational AutoEncoder 学習時のモデルのパラメーター

epoch 数	100
隠れ層の次元数	100
縮約後ベクトルの次元数	100
ミニバッチサイズ	400

その後、学習したモデルの Encoder を用いて訓練・テスト用の Large データの次元縮約を行なった。そして、各 Small データに対応する縮約された Large データを付加した。付加が完了したデータを実験データとする。また、SVD を用いて同様の次元縮約及び負荷を行った。SVD による縮約データが付加されたデータを SVD データとする。

4.4 計測方法

訓練用実験データを用いて各領域の分類器を訓練した。分類器には Scikit-learn [5] にて実装されている SVM を用いた。訓練時のハイパーパラメータを表 2 に示す。

表 4.2 SVM 学習時のモデルのパラメーター

使用モデル	SVC
事前処理	MinMaxScaler
C	1,10,100,1000 GridSearchCV を用いて最適なものを選択
備考	表記のないパラメータはモデルの初期値を使用

その後、各分類器に各領域のテスト用実験データを分類させ、精度を記録した。また、SVD データに対して同様の操作・記録を行った。

4.5　実験結果

実験の結果を表 4.3 に示す。表 4.3 における None は領域適応の手法を用いず、ソース領域の訓練データのみから分類器を学習し、その分類器を用いてターゲット領域のテストデータを識別した結果である。SVD は 特異値分解を利用して次元縮尺する領域適応の手法、VAE は VAE を利用して次元縮尺する領域適応の手法（提案手法）を表す。

表 4.3　実験結果（正解率 %）

領域適応	None	SVD	VAE
B → D	**77.75**	77.65	76.65
B → M	72.50	**75.20**	74.80
D → B	**77.65**	76.90	77.00
D → M	76.40	77.65	**77.75**
M → B	76.65	**76.80**	75.90
M → D	**80.00**	79.45	78.70
平均	76.83	77.28	76.77

第 5 章

考察

提案手法を用いる場合よりも SVD を用いる方が領域適応時の平均精度が 0.51% 高いという結果が得られた。この結果から提案手法及び実験の問題点を考察していく。

まず考えられる実験時の問題点が、VAE の訓練エポック数が適切でなかった可能性である。当実験と同条件下で訓練した VAE の損失は次のように遷移した。

図 5.1 から学習進行度に比例して損失のばらつきが大きくなっている印象を受けた。また、全体を通して損失が激増する頻度が高い。これは特定領域への適応度が高い状態のモデルに適応度の低い領域のデータが入力される現象が多発していると見ることが出来る。この点から、訓練スクリプトに問題があり、訓練時の入力順に領域依存の偏りが発生していると推測した。

訓練スクリプトは大規模なデータに対応可能にするために SciPy 疎行列型で読み込んだ全データを結合することなく、プログラム引数の順にミニバッチ学習させる方式を用いている。引数は訓練・テストデータ、Unlabeld データ (領域順は book, DVD, music) の順となっていた。読み込む全てのデータを単一の配列に統合し、ランダムな順番でミニバッチ学習させればより領域依存度の低い出力が得られるだろう。

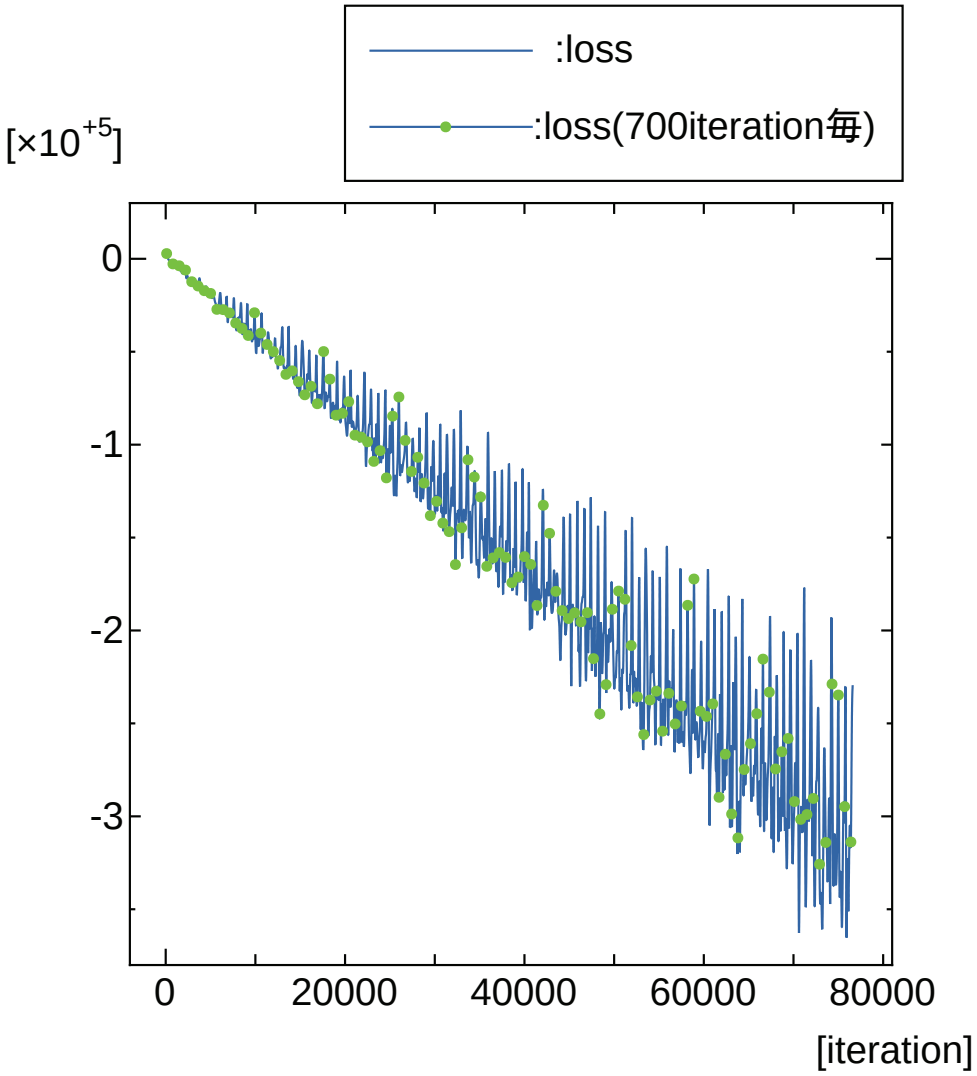

図 5.1　VAE の損失の推移

第 6 章

結論

本論文では感情分析の領域適応に対して、Variational AutoEncoder の Encoder を利用した領域適応の手法を提案した。ソース領域とターゲット領域のデータを用いて訓練した Variational AutoEncoder の Encoder で両データを次元縮約し、得られたデータを対応するデータに付加することでターゲット領域に適応する。実験では、提案手法と SVD の 2 つの手法を SVM での識別精度を用いて比較した。その結果は、SVD を使用した方が有効度がわずかに高いというものだった。

実験結果及び実験に使用したプログラムを再度確認・推敲したところ、提案手法のモデル訓練の過程で、領域に依存しない出力が得られるような訓練ができていなかった可能性が存在することが分かった。まずは提案手法の訓練過程の問題点を解消して出力の領域依存度を低下させ、VAE の利用可能性を再度検証していきたい。

謝辞

本論文の作成にあたり、たくさんのご指導ご協力を頂いた指導教員の新納浩幸教授をはじめとした、自然言語処理研究室の皆様に心より感謝申し上げます。

参考文献

[1] John Blitzer, Ryan McDonald, and Fernando Pereira. Domain adaptation with structural correspondence learning. In *EMNLP-2006*, pages 120–128, 2006.

[2] Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

[3] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[4] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *Knowledge and Data Engineering, IEEE Transactions on*, 22(10):1345–1359, 2010.

[5] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[6] Peter Prettenhofer and Benno Stein. Cross-Language Text Classification using Structural Correspondence Learning. In *48th Annual Meeting of the Association of Computational Linguistics (ACL 10)*, pages 1118–1127. Association for Computational Linguistics, July 2010.

[7] Baochen Sun, Jiashi Feng, and Kate Saenko. Return of Frustratingly Easy Domain Adaptation. *AAAI*, 2016.

[8] 古宮嘉那子 芝山直希, 新納浩幸. Variational AutoEncoder を利用した感情分析の領域適応. In 言語処理学会第 *25* 回年次大会 *(NLP2019)*, 2019.

付録 A

ソースリスト

A.1　VAE の訓練

プログラム A.1　train_vae.py

```
#!/usr/bin/env python3
"""Chainer example: train a VAE on scipy.sparse
"""
import argparse
import chainer
from chainer import functions as F
from chainer import datasets
from chainer.backends import cuda
import numpy as np
import scipy as sp
import os

import net


def log(folderpath, text, mode="w"):
    if not os.path.isdir(folderpath):
        os.mkdir(folderpath)
    with open(folderpath+"/log.csv",mode) as f:
        f.write(text+"\n")

def main():
    parser = argparse.ArgumentParser(description='Train for
        zipping datasets with VAE (Using Chainer Example model)')
    parser.add_argument('--initmodel', '-m', default='',
                        help='Initialize the model from given file
                            ')
    parser.add_argument('--gpu', '-g', default=-1, type=int,
```

```
                        help='GPU ID (negative value indicates CPU
                            )')
    parser.add_argument('--out', '-o', default='result',
                        help='Directory to output the result')
    parser.add_argument('--epoch', '-e', default=100, type=int,
                        help='number of epochs to learn')
    parser.add_argument('--dimz', '-z', default=200, type=int,
                        help='dimention of encoded vector')
    parser.add_argument('--minibatch_size', '-b', type=int,
        default=100,
                        help='minibatch size')
    parser.add_argument('--print_interval', '-p', type=int,
        default=100,
                        help='print ang log interval')
    parser.add_argument('--n_data_per_file','-nd',type=int,
        default=1000,help='num of zipped data per 1 file')
    parser.add_argument('inputs', type=str, nargs='+',
                        help='source files(scipy Matrix)')
    parser.add_argument('--dimhlayer', '-hl', default=200, type=
        int, help='dimention of hidden layer')
    args = parser.parse_args()

    print('GPU: {}'.format(args.gpu))
    print('# dim z: {}'.format(args.dimz))
    print('# dim hidden-layer: {}'.format(args.dimhlayer))
    print('# minibatch-size: {}'.format(args.minibatch_size))
    print('# epoch: {}'.format(args.epoch))
    print('')

    #Load local data to make dataset
    sets = []
    for src in args.inputs:
        print(src)
        data = sp.sparse.load_npz(src)
        sets.append(data.tocsr().astype(np.float32))
    # Prepare VAE model, defined in net.py
    model = net.VAE(sets[0].shape[1], args.dimz, args.dimhlayer)

    # Setup an optimizer
    optimizer = chainer.optimizers.Adam()
    optimizer.setup(model)

    # Initialize
    if args.initmodel:
        chainer.serializers.load_npz(args.initmodel, model)
```

```
if args.gpu >= 0:
    model.to_gpu(args.gpu)
lossfunc=model.get_loss_func()
iterate=0
bs = args.minibatch_size
pr = args.print_interval
print("epoch\t\titeration\t\tloss\t\trec_loss")
log(args.out,"epoch,iteration,loss,rec_loss",mode="w")
for epoch in range(args.epoch):
    for d in sets:
        indexes = np.random.permutation(d.shape[0])
        for i in range(0,d.shape[0],bs):
            if args.gpu >= 0:
                x = chainer.Variable(cuda.to_gpu(d[i:(i+bs) if
                    (i+bs) < d.shape[0] else d.shape[0]].
                    toarray()))
            else:
                x = chainer.Variable(d[i:(i+bs) if (i+bs) < d.
                    shape[0] else d.shape[0]].toarray())
            model.cleargrads()
            loss = lossfunc(x)
            loss.backward()
            optimizer.update()
            iterate += 1
            if iterate % pr == 0:
                print(str(epoch+1)+"\t\t"+str(iterate)+"\t\t"+
                    str(model.loss.data)+"\t\t"+str(model.
                    rec_loss.data))
                log(args.out,str(epoch+1)+","+str(iterate)+","
                    +str(model.loss.data)+","+str(model.
                    rec_loss.data),mode="a")
# Save Model and zipped dataset
chainer.serializers.save_npz(args.out+"/model.npz",model)
print("model saved")
model.to_cpu()
nd = args.n_data_per_file
with chainer.using_config("train",False):
    for src,c in zip(sets,range(len(sets))):
        for i in range(0,src.shape[0],nd):
            mu, ln_var = model.encode(src[i:(i+nd) if (i+nd) <
                src.shape[0] else src.shape[0]].toarray())
            zipped = F.gaussian(mu, ln_var)
            zipped = sp.sparse.coo_matrix(zipped.data)
```

```
                sp.sparse.save_npz(args.out+"/zipped_"+str(c)+"_"+
                    str(i),zipped)
    print("Zipping_completed")

if __name__ == '__main__':
    main()
```

A.2 Variational AutoEncoder

プログラム A.2 net.py

```
import six

import chainer
import chainer.functions as F
from chainer.functions import gaussian_kl_divergence
import chainer.links as L


class VAE(chainer.Chain):
    """Variational AutoEncoder"""

    def __init__(self, n_in, n_latent, n_h, c=1.0):
        super(VAE, self).__init__()
        with self.init_scope():
            # encoder
            self.le1 = L.Linear(n_in, n_h)
            self.le2_mu = L.Linear(n_h, n_latent)
            self.le2_ln_var = L.Linear(n_h, n_latent)
            # decoder
            self.ld1 = L.Linear(n_latent, n_h)
            self.ld2 = L.Linear(n_h, n_in)
            self.C = c
    def __call__(self, x, sigmoid=True):
        """AutoEncoder"""
        return self.decode(self.encode(x)[0], sigmoid)

    def encode(self, x):
        h0 = self.le1(x)
        h1 = F.tanh(h0)
        mu = self.le2_mu(h1)
        ln_var = self.le2_ln_var(h1)  # log(sigma**2)
        return mu, ln_var
```

```
    def decode(self, z, sigmoid=True):
        h0 = self.ld1(z)
        h1 = F.tanh(h0)
        h2 = self.ld2(h1)
        if sigmoid:
            return F.sigmoid(h2)
        else:
            return h2

    def get_loss_func(self, k=1):
        """Get loss function of VAE.

        The loss value is equal to ELBO (Evidence Lower Bound)
        multiplied by -1.

        Args:
            C (int, self.c): Usually this is 1.0. Can be changed
                to control the
                second term of ELBO bound, which works as
                    regularization.
            k (int): Number of Monte Carlo samples used in encoded
                vector.
        """
        C = self.C
        def lf(x):
            mu, ln_var = self.encode(x)
            batchsize = len(mu.data)
            # reconstruction loss
            rec_loss = 0
            for l in six.moves.range(k):
                z = F.gaussian(mu, ln_var)
                rec_loss += F.bernoulli_nll(x, self.decode(z,
                    sigmoid=False)) \
                    / (k * batchsize)
            self.rec_loss = rec_loss
            self.loss = self.rec_loss + \
                C * gaussian_kl_divergence(mu, ln_var) / batchsize
            chainer.report(
                {'rec_loss': rec_loss, 'loss': self.loss},
                    observer=self)
            return self.loss
        return lf
```

A.3　実験に使用した分類器

プログラム A.3　SVC.py

```
#!/usr/bin/env python3
# coding: utf-8

import numpy as np
import scipy
from sklearn.svm import SVC
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import GridSearchCV
import argparse

ap = argparse.ArgumentParser("Train SVC with labeled data")
ap.add_argument("train_data",type=str,help="Training datasets")
ap.add_argument("train_label",type=str,help="Labels for train_data
    ")
ap.add_argument("test_data",type=str,help="Test datasets")
ap.add_argument("test_label",type=str,help="Labels for test_data")
ap.add_argument("-c","-C",type=float,default=1.0,help="SVM's
    parameter C")
ap.add_argument("--scaling","-s",action="store_true",help="MinMax
    Scaling before train")
ap.add_argument("--grid_C","-gc",type=float,default=None,nargs="+"
    ,help="list of parameter C(gridsearch mode)")
ap.add_argument("--gamma", "-g", type=float, default=None, help="
    SVM's parameter gamma")
args = ap.parse_args()

X_train = scipy.sparse.load_npz(args.train_data)
y_train = np.load(args.train_label)
X_test = scipy.sparse.load_npz(args.test_data)
y_test = np.load(args.test_label)

if args.gamma:
    g = args.gamma
else:
    g = 'auto'

if args.scaling:
    scaler = MinMaxScaler()
    if scipy.sparse.isspmatrix_csr(X_train):
        scaler.fit(X_train)
```

```
    else:
        scaler.fit(X_train.tocsr())
    X_train = scipy.sparse.csr_matrix(scaler.transform(X_train))
    X_test = scipy.sparse.csr_matrix(scaler.transform(X_test))
    print("Scaling finished")
if args.grid_C:
    C = args.grid_C
    pg = {"C":C}
    b_clf = GridSearchCV(SVC(),pg,n_jobs=2)
else:
    b_clf = SVC(C=args.c,gamma=g)
print("Training begins")
if not scipy.sparse.isspmatrix_csr(X_train):
    b_clf.fit(X_train.tocsr(),y_train)
else:
    b_clf.fit(X_train,y_train)
print("Training finished")
print("Evalating")
bestparams = b_clf.best_params_
if not scipy.sparse.isspmatrix_csr(X_train):
    train_accrate = b_clf.score(X_train.tocsr(), y_train)
    test_accrate = b_clf.score(X_test.tocsr(), y_test)
else:
    train_accrate = b_clf.score(X_train, y_train)
    test_accrate = b_clf.score(X_test, y_test)
print("training score:"+str(train_accrate))
print("best params:"+str(bestparams))
print("test score:"+str(test_accrate))

print("Evalation ends")
with open("result.txt","w") as r:
    r.write("training score:"+str(train_accrate)+"\n")
    r.write("best params:"+str(bestparams)+"\n")
    r.write("test score:"+str(test_accrate)+"\n")
```