レポート課題

自己符号化器の特性評価

2018年5月18日

S152114　宮地雄也

1　実験目的

ニューラルネットワークによる生成モデルの基礎となる自己符号化器の原理および特性について理解する．

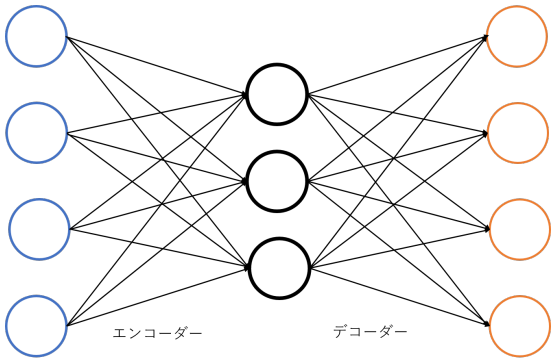
2　実験原理

2.1　自己符号化器（Autoencoder，AE）

自己符号化器とは入力に対して同じものを出力するように学習するニューラルネットワークである．教師なし学習に分類される．

2.1.1　自己符号化器の構成要素とそれぞれの役割

Auto Encoder には大きく分けて，エンコーダー部分とデコーダー部分に分かれる．エンコーダー部分でその特徴量を学習しながらその次元を圧縮し，それを基にデコーダー部分では復元を行う．そのイメージを以下に示す．



2.1.2　主な応用先

自己符号化器は次元削減や情報検索の分野で応用されてきた．

次元削減では自己符号化器で学習したパラメータを深層学習の初期パラメータとして使うことで多次元のデータも効率よく学習ができ，次元を圧縮してもカテゴリ分類がうまくされていた．低次元表現は分類問題で効果を発揮し，より効率よくタスクを分類できる．また消費するメモリーと実行時間も短く出来る．

情報検索とは入力データから似た項目を検索するタスクである．入力から低次元のバイナリーを作成することで検索を効率化することが出来る．

2.1.3　学習を成功させるための条件

学習を成功させるために今回はミニバッチ法を用いる．ミニバッチ法とは正則化の方法のひとつで，学習時にデータをためてから一気に更新していく手法である．

2.1.4　主成分分析との関連

主成分分析 (Principle Component Analysis) とは，学習データ $x_i = (x_{i1}, ..., x_{id})$ の分散が最大になる方向への線形変換を求める手法である．

2.2　雑音除去自己符号化器（Denoising Autoencoder，DAE）

雑音除去自己符号化器は破損したデータ点を入力として受け取りもとの破損していないデータ点を復元するよう訓 練された自己複合化器である．以下に DAE の訓練手順を示す．

1. 入力データに一部ノイズを加えたデータを AE を行う
2. 生成したデータとノイズをつける前のデータの誤差で逆伝播を行う
3. 繰り返すことでノイズを除去するよう訓練していく

このようにして学習した自己符号化器はノイズを除去するように復元するのでノイズを含んだ学習データから元の綺麗な画像を作り出すことができる

3　実験方法

本実験にて確認したいことは以下の 2 つである。

- 画像がきちんとデノイジングされるかどうか
- 自己符号化器で学習したパラメータを使うと画像認識にどのような効果があるか

この二点について実験を行い、その効果を確認する．上記のことを確認するために実験は次のように行う．

学習条件

- 学習データ：Cifer10
- 学習データ数：100 枚
- 分類テストデータ数：1000 枚
- 学習回数：500 回
- パラメーターの初期化方法：ガウス分布でランダムに初期化
- パラメータの更新方法：Adam アルゴリズムで更新
- ノイズの入れ方：画像の 3 割から 5 割をランダムで画素値を 0 にしてノイズとする

今回，実験で利用したネットワーク構成は以下のとおり．

表 1: オートエンコーダー ARA の構成

1	アフィン層	入力チャネル数:3，入力サイズ:32 × 32，出力チャネル数:1，出力サイズ:500
2	検出層	ReLU
3	Affine 層	入力ノード数:500，出力チャネル数:3, 出力ノード数:32 × 32

表 2: オートエンコーダー ARAA の構成

1	アフィン層	入力チャネル数:3，入力サイズ:32 × 32，出力チャネル数:1，出力サイズ:1000
2	検出層	ReLU
3	Affine 層	入力ノード数:500， 出力ノード数:512
4	Affine 層	入力ノード数:512，出力チャネル数:3, 出力ノード数:32 × 32

表 3: オートエンコーダー CRPCRPAA の構成

1	畳み込み層	入力チャネル数:3，入力サイズ:32 × 32，出力チャネル数:10，出力サイズ:28 × 28，カーネルサイズ:5 × 5
2	検出層	ReLU
3	プーリング層	入力サイズ:28 × 28，出力サイズ:14 × 14，ブロックサイズ:2 × 2，集約演算:最大値
4	畳み込み層	入力チャネル数:10，入力サイズ:14 × 14，出力チャネル数:10，出力サイズ:10 × 10，カーネルサイズ:5 × 5
5	検出層	ReLU
6	プーリング層	入力サイズ:10 × 10，出力サイズ:5 × 5，ブロックサイズ:2 × 2，集約演算:最大値
7	Affine 層	入力ノード数:250，出力ノード数:500
8	Affine 層	入力ノード数:500，出力チャネル数:3, 出力ノード数:32 × 32

表 4: オートエンコーダー CRPCRPAAA の構成

1	畳み込み層	入力チャネル数:3，入力サイズ:32 × 32，出力チャネル数:10，出力サイズ:28 × 28，カーネルサイズ:5 × 5
2	検出層	ReLU
3	プーリング層	入力サイズ:28 × 28，出力サイズ:14 × 14，ブロックサイズ:2 × 2，集約演算:最大値
4	畳み込み層	入力チャネル数:10，入力サイズ:14 × 14，出力チャネル数:10，出力サイズ:10 × 10，カーネルサイズ:5 × 5
5	検出層	ReLU
6	プーリング層	入力サイズ:10 × 10，出力サイズ:5 × 5，ブロックサイズ:2 × 2，集約演算:最大値
7	Affine 層	入力ノード数:250，出力ノード数:500
8	Affine 層	入力ノード数:500，出力ノード数:512
9	Affine 層	入力ノード数:512，出力チャネル数:3, 出力ノード数:32 × 32

表 5: オートエンコーダー CCRPACCRPAA の構成

1	畳み込み層	入力チャネル数:3，入力サイズ:32 × 32，出力チャネル数:10，出力サイズ:29 × 29，カーネルサイズ:4 × 4
2	畳み込み層	入力チャネル数:10，入力サイズ:29 × 29，出力チャネル数:10，出力サイズ:26 × 26，カーネルサイズ:4 × 4
3	検出層	ReLU
4	プーリング層	入力サイズ:26 × 26，出力サイズ:13 × 13，ブロックサイズ:2 × 2，集約演算:最大値
5	畳み込み層	入力チャネル数:10，入力サイズ:13 × 13，出力チャネル数:10，出力サイズ:10 × 10，カーネルサイズ:3 × 3
6	畳み込み層	入力チャネル数:10，入力サイズ:111 × 11，出力チャネル数:10，出力サイズ:10 × 10，カーネルサイズ:2 × 2
7	検出層	ReLU
8	プーリング層	入力サイズ:10 × 10，出力サイズ:5 × 5，ブロックサイズ:2 × 2，集約演算:最大値
9	Affine 層	入力ノード数:250， 出力ノード数:500
10	Affine 層	入力ノード数:500，出力チャネル数:3, 出力ノード数:32 × 32

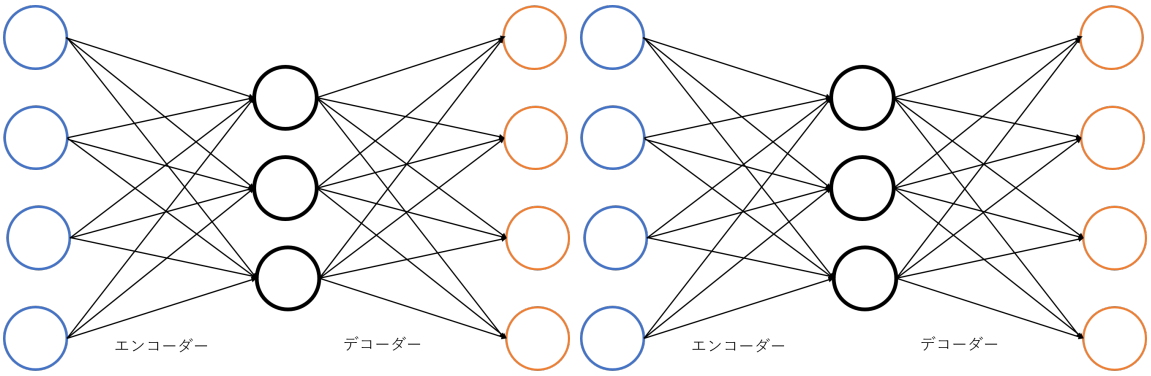
表 6: オートエンコーダー CCRPACCRPAAA の構成

1	畳み込み層	入力チャネル数:3，入力サイズ:32 × 32，出力チャネル数:10，出力サイズ:29 × 29，カーネルサイズ:4 × 4
2	畳み込み層	入力チャネル数:10，入力サイズ:29 × 29，出力チャネル数:10，出力サイズ:26 × 26，カーネルサイズ:4 × 4
3	検出層	ReLU
4	プーリング層	入力サイズ:26 × 26，出力サイズ:13 × 13，ブロックサイズ:2 × 2，集約演算:最大値
5	畳み込み層	入力チャネル数:10，入力サイズ:13 × 13，出力チャネル数:10，出力サイズ:10 × 10，カーネルサイズ:3 × 3
6	畳み込み層	入力チャネル数:10，入力サイズ:111 × 11，出力チャネル数:10，出力サイズ:10 × 10，カーネルサイズ:2 × 2
7	検出層	ReLU
8	プーリング層	入力サイズ:10 × 10，出力サイズ:5 × 5，ブロックサイズ:2 × 2，集約演算:最大値
9	Affine 層	入力ノード数:250， 出力ノード数:500
10	Affine 層	入力ノード数:500， 出力ノード数:512
11	Affine 層	入力ノード数:512，出力チャネル数:3, 出力ノード数:32 × 32

4　実験結果

以下のような結果が得られた.

・AE



[1] 正解率（実験 1)　　[2] 平均二乗誤差（実験 1)

図 1: CNN(Adam で更新)

・DAE

・画像分類

5　考察

今回、自己符号化器を作成し、それを用いた画像識別を行った。さまざまな構成で試したが、()が最も良い結果を生んだ。ARA、CRPCRPA ともに学習回数を重ねるとノイズなしでは高い再現性を実現した。しかしながらその学習にはとても時間がかかる。また自己符号化器を使った事前学習での画像識別は前回やった MNIST データで学習する時よりも初期の正解率が高く、平均二乗誤差も低い数値から始まり、結果として早く終息する傾向があるように思う。これは自己符号化で早くからパラメータの数値が良い値を示し、収束の加速につながったと思う。しかし同じ構成のネットワークで学習をしてみるとノイズなしの方がより良い結果を示しているように思う。これはノイズをつけた方がノイズにフィットするように学習が進むためだと思われる。

デノイジングに関してはこちらも学習回数というよりネットワーク構成が大きく影響することがわかる。雑音除去はないものを周りのものを使って構成するという機能なのでかけてしまった歴史的な壁画や、これを文章に置き換え穴あき問題を解かせるなどに使えそうだなと作成しながら構想した。

今回はエンコーダーには畳み込み層を用いているが、デコーダ部分には畳み込みを持ちいらずアフィン層で拡張したこの構成が結果にどのような影響をあたえているかわからないので deconvolution を用いた実装をしてみたいと思う。

コーディングしていく中でうまくネットワークを作るにはトライアンドエラーが必要不可けつだと感じた。おそらく元とする学習データでもその構成は変わると思われる。なのでコツは改変しや

すいようクラスやオブジェクトで各モジュールをパッケージ化していくことだと思われる。より柔軟性の高いコーディングを目指していきたい。

図 2: 学習データ

図 3: 学習データの復号結果（実験条件との対応がとれるようにすること）: 学習回数=200 回．平均二乗誤差=7.1715