付 録C　ニューラルネットワークによる機械学習

3.1　目的

本節では、ニューラルネットワーク（Neural Network; NN) を用いた機械学習の基礎を学習する。この枠組みでは、手書き文字認識など多くのタスクが、NN に含まれるパラメータに関する特殊な構造を有した最適化問題として定式化される。そこで、3.2 節においてこれらの最適化問題を効率よく解く手法である確率勾配法や誤差逆伝播法などの考え方とその実装方法を概説する。さらに、3.3 節以降では、Tensorflow と呼ばれるフレームワークを用いてアルゴリズムを実装する。ただし、独力でこれらのコードが書けるようになることは目指さず、準備されたサンプルコードを利用して、動作を理解することを目標とする。

3.2　数学的準備

3.2.1　ニューラルネットワーク

まずは図 C.1 で表現される関数を考えよう。それぞれのノード（円）は活性化関数と呼ばれる非線形関数の入力と出力の関係

$$h_1^{(1)} = f^{(1)}(u_1^{(1)}) \tag{C.1}$$

などを表現している。こうしたノードの縦方向の集まりのことを層と呼ぶ。層と層をつなぐエッジ（矢印）は重み $w_{ij}^{(l)}$ をもつ線型結合

$$u_j^{(l)} = \sum_i w_{ij}^{(l)} h_i^{(l-1)}$$

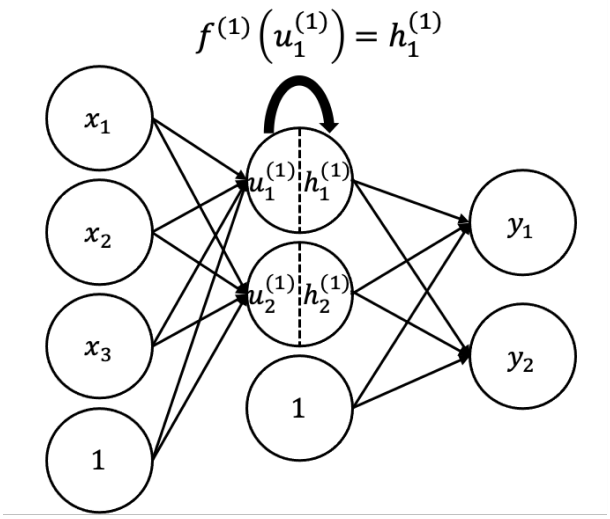

図 C.1: 3 層のニューラルネットワーク

をあらわす。NN はこうした構造をもつ関数であり、深層学習とは層の数が非常に多い NN を用いた機械学習の手法や、その周辺の研究領域のことを指す。

より一般的に L 層からなる NN を考え、第 l 層のノードの個数とそのノードを d_l, $\boldsymbol{u}^{(l)} = (u_1^{(l)}, \ldots, u_{d_l}^{(l)})^T$ と表記する。まず、入力層と呼ばれる第 1 層（input layer）は活性化関数を持たず、入力ベクトル $\boldsymbol{x} = (x_j)$ をそのまま出力する。

$$\boldsymbol{u}^{(1)} = \boldsymbol{x} \tag{C.2}$$

つぎに中間層 (internal layer) や隠れ層 (hidden layer) と呼ばれる第 l 層（$1 < l < L$）における演算処理は、パラメータ $w_{ij}^{(l)}$ と $b_j^{(l)}$ をまとめて $\boldsymbol{W}^{(l)}, \boldsymbol{b}^{(l)}$ と表記すると、

$$\boldsymbol{u}^{(l)} = \boldsymbol{W}^{(l)}\boldsymbol{h}^{(l-1)} + \boldsymbol{b}^{(l)}, \quad \boldsymbol{h}^{(l)} = f^{(l)}\left(\boldsymbol{u}^{(l)}\right) \tag{C.3}$$

と書ける。活性化関数 $f^{(l)}$ はノードごとに変えても構わないが、一般的には層ごとに共通の関数を用いる。最後の第 L 層が出力層に相当し、出力 $\hat{\boldsymbol{y}}$ は次で与えられる。

$$\hat{\boldsymbol{y}} = f^{(L)}(\boldsymbol{u}^{(L)}) \tag{C.4}$$

このように、入力が与えられたとき、NN の各層を順番に計算していき、出力まで計算を行うアー

キテクチャを順伝播 (feed forward; FF)NN と呼ばれる。

注意 3.2.1 隠れ層の活性化関数をタスクに応じて定める一般的な基準は未だ存在しない。現場では経験則や試行錯誤に頼ることが多いが、役に立つ活性化関数の選び方がいくつも知られている。例えば、学習をスムーズに進行させるには活性化関数として、正規化線形関数 (rectified linear unit; ReLU)

$$f(u) = \max\{0, u\} = \begin{cases} u & (u \geq 0) \\ 0 & (u < 0) \end{cases}$$

を用いると良いことがわかっている。活性化関数を持つノードを ReLU とも呼ぶが、関数自体も略して ReLU と呼ぶ。

3.2.2 関数回帰と最適化

設計する NN に課すタスクに応じてその構造を決定し、重み係数などのパラメータ集合 $\boldsymbol{w}$ に関する最適化問題を設定しなければならない。代表例として、訓練データ $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i\in\mathcal{B}}$, $\mathcal{B} = \{1, 2, \ldots\}$ に対して、$\hat{\boldsymbol{y}}(\boldsymbol{x}_i) \approx \boldsymbol{y}_i$ が成り立つ関数 $\hat{\boldsymbol{y}}$ を求める関数回帰問題を考える。この問題に対して、$\boldsymbol{w}$ によりパラメトライズされる関数の集合 $\{\hat{\boldsymbol{y}}(\boldsymbol{x}; \boldsymbol{w})\}_{\boldsymbol{w}}$ から求める関数を探すことにしよう。このとき、ある $\boldsymbol{w}$ を用いたモデルの予測値 $\hat{\boldsymbol{y}}(\boldsymbol{x}_i; \boldsymbol{w})$ と実際のデータ $\boldsymbol{y}_i$ ができるだけ近くなるように、平均二乗誤差

$$L(\boldsymbol{w}) = \frac{1}{|\mathcal{B}|} \sum_{i\in\mathcal{B}} \mathcal{L}(\hat{\boldsymbol{y}}(\boldsymbol{x}_i; \boldsymbol{w}), \boldsymbol{y}_i), \tag{C.5}$$

$$\mathcal{L}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \|\hat{\boldsymbol{y}} - \boldsymbol{y}\|^2 \tag{C.6}$$

に対して、

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} L(\boldsymbol{w}) \tag{C.7}$$

とすることは妥当であろう。これは特別な場合として、アフィン関数

$$\hat{\boldsymbol{y}}(\boldsymbol{x}; \boldsymbol{w}) = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}, \boldsymbol{W} \in \mathbb{R}^{n_y \times n_x}, \boldsymbol{b} \in \mathbb{R}^{n_y}$$

によりフィッティングを行う線形回帰を含む。以降では、線形回帰は困難なデータにも適用することを目指し、前節で導入した NN を関数集合として採用する。このとき、$\boldsymbol{w} = (\boldsymbol{W}^{(2)}, \boldsymbol{b}^{(2)}, \ldots, \boldsymbol{W}^{(L)}, \boldsymbol{b}^{(L)})$ であり、層の数や活性化関数などは適切に選択する必要がある。

他の代表的なタスクとして、データを 2 つのクラスに分類する 2 値分類がある。これは、各 $\boldsymbol{x}_i$ に対して $y_i \in \{0, 1\}$ が与えられていると考えれば良い。求めたい関数は $\{0, 1\}$ に値をとる関数であるため、出力層がシグモイド関数

$$\sigma(x) = 1/(1 + e^{-x}) \tag{C.8}$$

である NN が標準的に用いられ、$\hat{y}(\boldsymbol{x}; \boldsymbol{w}) < 1/2$ であるとき所属クラスを 0、それ以外の場合は 1 と判定する。損失関数としては、

$$\mathcal{L}(\hat{y}, y) = -(y \log \hat{y} + (1 - y) \log(1 - \hat{y})) \tag{C.9}$$

を採用すれば $y_i = 1$ であれば $\hat{y}(\boldsymbol{x}_i, \boldsymbol{w}) \approx 1$、$y_i = 0$ であれば $\hat{y}(\boldsymbol{x}_i, \boldsymbol{w}) \approx 0$ となることが期待できる。

3.2.3 勾配降下法による学習

NN の学習は損失関数 $L(\boldsymbol{w})$ の最小化として定式化されていた。NN における損失関数は一般に非常に複雑な非凸関数であるため、大域的極小値 (global minimum) 以外にも、膨大な数の局所的極小値 (local minima) を持つ。実は、深層学習では真の最小値を見つけずとも、損失関数の良い極小値さえ見つければ十分であることが予想されている。そこで以下では方程式

$$\frac{\partial L(\boldsymbol{w})}{\partial \boldsymbol{w}} = 0$$

を満たす停留点を求める。さまざまな場面で用いられるニュートン法は、ヘシアン逆行列の評価にかかる計算コストが大きいため、深層学習ではほとんど用いられない。以下では、(C.7) が

- 決定変数 $\boldsymbol{w}$ が NN という特殊な関数に含まれるパラメータである、
- 損失関数 (C.5) が各データ i に対する誤差 $\mathcal{L}(\hat{\boldsymbol{y}}(\boldsymbol{x}_i; \boldsymbol{w}), \boldsymbol{y}_i)$ の和である

という特徴を利用した効率的な最適化手法を紹介する。以下では記述を簡単化し、中心となる考え方に注目するために $d_l = 1,\ \boldsymbol{b}^{(l)} = 0$ とするが、一般の場合も同様の議論が成立する。

誤差逆伝播法　パラメータ $\boldsymbol{w}$ に関する損失関数 $\mathcal{L}(\hat{\boldsymbol{y}}(\boldsymbol{x};\boldsymbol{w}))$ の $\boldsymbol{w}$ に関する勾配を計算する。ここで、NN の特別な構造を用いると、素朴に $\boldsymbol{w}$ の各要素に関する勾配を独立に計算するよりも大幅に計算量を削減することができる。この手法は誤差逆伝播法 (back propagation) と呼ばれ、1986 年にラメルハートらによって発表され、NN の第 2 次ブームを巻き起こした。

まず、微分の連鎖律と $\boldsymbol{h}^{(l-1)}$ が $\boldsymbol{W}^{(l)}$ に依存しないという性質を用いると、

$$\frac{\partial \mathcal{L}}{\partial w_{ij}^{(l)}} = \frac{\partial \mathcal{L}}{\partial u_i^{(l)}} \frac{\partial u_i^{(l)}}{\partial w_{ij}^{(l)}} = \delta_i^{(l)} h_j^{(l-1)} \tag{C.10}$$

が $l \leq L$ に対して成り立つ。ここで、$\delta_i^{(l)} = \partial \mathcal{L}/\partial u_i^{(l)}$ とした。一方で、$l < L$ に対して

$$\begin{aligned}\delta_i^{(l)} &= \sum_{k=1}^{d_{l+1}} \frac{\partial \mathcal{L}}{\partial u_k^{(l+1)}} \frac{\partial u_k^{(l+1)}}{\partial u_i^{(l)}} \\ &= \sum_{k=1}^{d_{l+1}} \delta_k^{(l+1)} w_{ki}^{(l+1)} f'(u_i^{(l)}) \end{aligned} \tag{C.11}$$

および $l = L$ に対して

$$\delta_i^{(L)} = \frac{\partial \mathcal{L}}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial u_i^{(L)}} = \frac{\partial \mathcal{L}}{\partial \hat{y}_i} f'(u_i^{(L)}) \tag{C.12}$$

が成り立つ。

したがって、与えられた $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ および $\boldsymbol{w}$ に対して、具体的に $\mathcal{L}(\boldsymbol{y}(\boldsymbol{x}_i;\boldsymbol{w}), \boldsymbol{y}_i)$ の $\boldsymbol{w}$ に関する勾配を求めるには、

1. 順方向に (C.2),(C.3),(C.4) により $\boldsymbol{h}^{(l)}, l < L$ を求める、
2. (C.12) により、$\delta^{(L)}$ を求める、
3. (C.11) により $\delta^{(l)}, l < L$ を逆方向に求める、
4. (C.10) により $\partial \mathcal{L}, \boldsymbol{y}_i)/\partial w_{ij}^{(l)}$ を求める、

という手順をたどればよい。この手順は L によらないが、層が深くなるに連れて勾配消失問題や勾配爆発問題などが起こることが知られている。パラメータの事前学習や活性化関数の工夫などにより、こうした問題を避ける手法の開発が盛んにおこなわれている。

課題 20 スカラー変数 $x,\ w_2,\ w_3$ に対して

$$\begin{aligned} &u^{(1)} = x,\ h^{(l)} = f(u^{(l)}), \\ &u^{(l)} = w^{(l)} h^{(l-1)},\ \hat{y} = f^{(3)}(u^{(3)}) \end{aligned}$$

を定義し，$f(u) = u^2,\ \mathcal{L}(\hat{y}) = \hat{y}^2$ とする．このとき，$u^{(l)},\ h^{(l)},\ \hat{y}, \mathcal{L}$ を x, w_2, w_3 の関数としてあらわし、$\frac{\partial \mathcal{L}}{\partial w_l},\ l = 2, 3$ を求めよ。またこの例において

$$\delta^{(3)} = (2\hat{y})(2u^{(3)}),\ \delta^{(2)} = \delta^{(3)} w^{(3)} (2u^{(2)})$$

を用いて誤差逆伝播法の構造を説明せよ。

この課題において、たとえば $(x, w^{(2)}, w^{(3)}) = (1, 2, 3)$ における勾配を求める場合、$u,\ h,\ \delta$ は簡単な代入操作を繰り返すことで計算することができ、勾配もそれらの積で表現される。一方で、$\mathcal{L}$ およびその偏微分の解析表現を求め、値を代入することも可能であるが効率が悪い。また、変数が増えると、誤差逆伝播法においては使い回せる値が増えるため、その差はさらに顕著となる。

課題 21 活性化関数にシグモイド関数は勾配消失問題を起こすため、現在使われない。一方で、ReLU は勾配消失を起こす可能性が低いことが知られている。このことについて説明せよ。

確率的勾配降下法　勾配降下の過程において、局所的極小値にはまる (図 3.2.3 左) 状況をできるだけ回避するために確率的要素を取り入れる。各更新ステップ t で、ミニバッチと呼ばれる一部の訓練サンプル $\mathcal{B}^{(t)}$ のみを用いる方法をミニバッチ学習と呼ぶ（$\mathcal{B}^{(t)} = \mathcal{B}$ の場合をバッチ学習と呼ぶ)。通常ミニバッチは学習前にランダムに作成しておく。そしてステップ t では、ミニバッチ上で平均した誤差関数

$$L^{(t)}(\boldsymbol{w}) = \frac{1}{|\mathcal{B}^{(t)}|} \sum_{i \in \mathcal{B}^{(t)}} \mathcal{L}(\hat{\boldsymbol{y}}(\boldsymbol{x}_i;\boldsymbol{w}), \boldsymbol{y}_i) \tag{C.13}$$

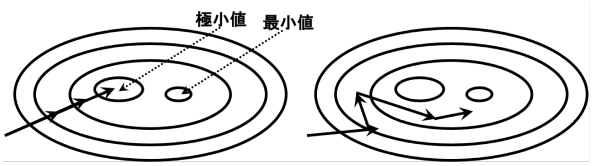

図 C.2: ミニバッチで極小値を避けるイメージ。左は極小値にはまってしまうが、ミニバッチによって極小値を避けて最小値に向かうことができる。

の勾配にもとづいて $\boldsymbol{w}$ を更新する。特に、各時刻のミニバッチに 1 つの訓練しか含まない $|\mathcal{B}^{(t)}| = 1$ という場合をオンライン学習や確率的勾配降下法と呼ぶ。ミニバッチ学習ではミニバッチ選択のランダム性により、時刻ごとに損失関数 $L^{(t)}(\boldsymbol{w})$ もランダムに変化し、望ましくない臨界点にはまり込む可能性がぐっと小さくなる (図 3.2.3)。なお、各 i に対する $\mathcal{L}(\hat{\boldsymbol{y}}(\boldsymbol{x}_i;\boldsymbol{w}),\boldsymbol{y}_i)$ の勾配計算は容易に並列化できるため、並列計算環境がある場合には、ある程度のサイズのミニバッチを利用する方が良い。

ミニバッチ $\mathcal{B}^{(t)}$ 内の訓練データに対して計算された $\mathcal{L}(\hat{\boldsymbol{y}}(\boldsymbol{x}_i;\boldsymbol{w}),\boldsymbol{y}_i)$ を用いて、勾配

$$\Delta \boldsymbol{W}^{(t,l)} = -\frac{1}{|\mathcal{B}^{(t)}|}\sum_{i\in\mathcal{B}^{(t)}}\frac{\mathcal{L}(\hat{\boldsymbol{y}}(\boldsymbol{x}_i;\boldsymbol{w}),\boldsymbol{y}_i)}{\partial \boldsymbol{W}^{(l)}}$$

を計算し

$$\boldsymbol{W}^{(t+1,l)} = \boldsymbol{W}^{(t,l)} + \eta^{(t)}\Delta \boldsymbol{W}^{(t,l)}$$

と重みパラメータを更新する。行列での微分は、∇ によるベクトルの成分ごとでの微分を行ったように行列の成分ごとの微分を表している．ベクトルステップサイズ $\eta^{(t)}$ の選び方や更新アルゴリズムによっても挙動は大きく変化するため、様々な手法が提案されている。

3.3 計算機実装

本節では、具体的に前節の最適化アルゴリズムを計算機上で実装する。

3.3.1 Google Colaboratory

Google Colaboratory (略称: Colab) はインターネット環境と Google のアカウントさえ持っていれば、環境構築なしでブラウザから Jupyther という Python のインターフェイスを使うことができるサービスである。深層学習の際には不可欠となる GPU も含めて無料で使うことができる。ブラウザに Google アカウントでログインした後に、以下のサンプルファイルを開く。

`https://bit.ly/39JsDg8`

サンプルファイルの中に、実装方法や本節以降の実装例も含まれている。

本節では MNIST (Mixed National Institute of Standards and Technology database) を使った画像認識を演習する。

注意 3.3.1 Colaboratory を立ち上げると、ノートブックを指定することが求められるので、「アップロード」のタブを選択する。Colaboratory が使用するノートブックはログインしている Google Drive 上にある。また、右上の「共有」のボタンを押すことにより、ノートブックの共有設定を変えられる。その上で、ノートブックが表示されているブラウザ上の URL を伝えることにより、教員や TA にノートブックを見せることができる。Colab でノートブックを実行する際は、実行環境として GPU や TPU が接続された環境を利用することができ、本演習では、GPU を接続した環境での環境での利用を想定している (デフォルトでは CPU の設定になっている)。

3.3.2 レポートの作成

Colab 内では、コード形式とテキスト形式で入力ができる。特に、テキスト形式ではマークダウン記法ができるので、レポートとして提出する際は第 1 章を参考にしてほしい。Google Drive 上のノートブックをパソコンにダウンロードするには、Colab のファイルメニューで「.ipynb をダウンロード」を選択し、提出することができる。

Colab による課題は、サンプルファイルを参照すること。

注意 3.3.2 本演習では、最適化の違いまでは解説していないが、実装で用いる TensorFlow にはいくつか Optimizer が登録されている。AdamOptimizer と FtrlOptimizer は、比較的安定して精度の高い値を与えることが知られているので、本演習では基本的には AdamOptimizer を用いることとする。

3.3.3 NN による画像認識

手書き文字認識や画像認識は典型的な多クラス分類である。K クラスの分類は、各 $\boldsymbol{x}_i$ に対して $y_i \in \{0, 1, \ldots, K\}$ が与えられていると考えれば良い。ここでは、$\hat{y}(\boldsymbol{x}; \boldsymbol{w})$ の出力は K 次元確率ベクトル (要素が非負かつ総和が 1) であるとして、その k 番目の要素が最大であるとき、所属クラスを k と判定する。出力層の活性化関数にソフトマックス関数（$\mathbb{R}^K \to \mathbb{R}^K$）

$$\mathrm{softmax}_i(x_1, \ldots, x_K) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}},\ i = 1, \ldots, K$$

を用いれば、このような NN が設計できる。

課題 22 損失関数を “MSE” にしてクロスエントロピーとの結果を比較せよ。

課題 23 次の関数

$$\begin{aligned} f(x_1, x_2) =& 10 - 10\exp(-0.2(x_1^2 + x_2^2)) \\ & - \exp((\cos(x_1) + \cos(x_2))/2) \end{aligned}$$

に対して回帰問題を考える。n 個のデータ (x_{i1}, x_{i2}), $i = 1, \ldots, n$ からこの関数上の値 $y_i = f(x_{i1}, x_{i2}) + \varepsilon$ を生成し、データセット $\{(x_{i1}, x_{i2}, y_i)\}_{i=1}^n$ をもとに，NN による関数 $f_{NN}(x_1, x_2)$ を学習し、$(x_1, x_2, f_{NN}(x_1, x_2))$ を 3 次元プロットとして図示せよ。

課題 24 TF や keras のフレームワークを使わずに、numpy を用いて、勾配法と SGD での実装せよ。 データは MNIST を使い、計算時間や損失の挙動について報告せよ。

課題 25 ノードの数を $10, 10^2, 10^3, 10^4, 10^5$ の 5 つで比較せよ。

課題 26 多層 NN の層数、ノード、バッチの設定において、エポック 100 としてエポックを増やしたときにの学習の様子を報告せよ。

課題 27 Optimizer クラスに 6 つの Optimizer（GradientDescent、Adadelta、Adagrad、Adam、Ftrl、RMSProp）を使用することで比較せよ。

課題 28 MNIST と CIFAR-10 それぞれで畳み込み層を入れたかどうかでどれほどの精度に違いがあるかを方向せよ。付録を参照のこと。

3.4 次元削減による特徴量抽出

前節までは入出力データ $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i \in \mathcal{B}}$ の関数回帰問題を念頭に、一般的に教師あり学習と呼ばれる問題に対する NN の議論を進めてきた。本節では s、（正解ラベルのついていない）データ集合 $\{\boldsymbol{x}_i\}_{i \in \mathcal{B}}$ の特徴量を次元削減により抽出する教師なし学習を考える。

3.4.1 オートエンコーダ

上記の目的のために、$\boldsymbol{y}_i = \boldsymbol{x}_i$ として前節と同様の関数回帰をおこなう。このように、$\hat{y}(\boldsymbol{x}_i) \approx \boldsymbol{x}_i$ を達成し、図 3.4.1 のように中間層のノード数が少ない「くびれた」構造をもつ NN をオートエンコーダ (Autoencoder; AE、自己符号化器) と呼ぶ。ここで、中間層の出力を潜在変数、入力から潜在変数への関数をエンコーダ、潜在変数から出力までの関数をデコーダと呼ぶ。与えられたデータの集合は、低次元の潜在変数をデコーダに入力することで再現できるという意味で、潜在変数をある種の特徴量と捉えることができる。

3.4.2 手書き文字の特徴量抽出

MNIST から 0 と 1 の画像をそれぞれ 100 個ずつ取り出し、エンコーダ、潜在変数、デコーダ、

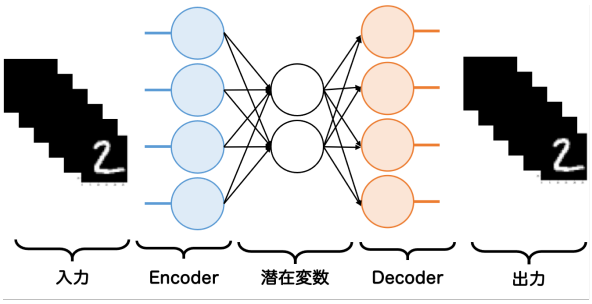

図 C.3: オートエンコーダのアーキテクチャ

出力の順に結合し、学習を行なった結果が図 3.5 である。学習を行った結果エンコーダにより特徴が圧縮され、デコーダによりもとのデータが復元できるのであれば潜在変数には圧縮された特徴が得られていることがわかる。

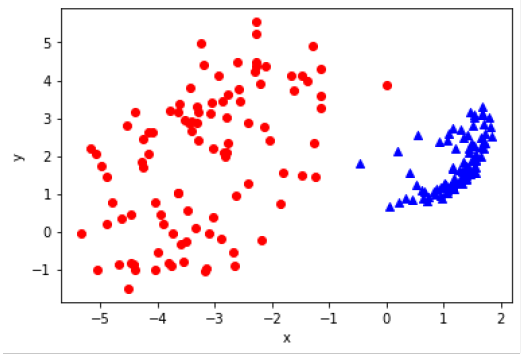

図 C.4: MNIST の 0 と 1 のそれぞれ 100 個のデータを AE で 2 次元に圧縮した散布図

課題 29 0, 1, 2 の 3 つの画像をそれぞれ 100 枚ずつ準備し、中間層で 2 次元まで圧縮し、散布図を表示せよ。

3.5 生成モデル学習

前節の手書き文字の特徴量抽出の例において、図 3.5 の赤（青）の特徴量をデコーダに入力すると 0（1）に「近い」画像が出力される。一方で、これらの特徴量の分布に関しては、$\{\boldsymbol{x}_i\}_i$ をエンコーダに入力することで知ることができるものの、学習時には特に考慮していない。そこで、本節ではこうした元データや特徴量の分布を陽に考慮した生成モデル学習と呼ばれる手法を概観する。

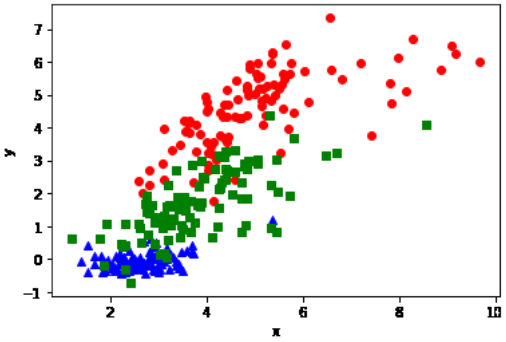

図 C.5: MNIST の 0 と 1 と 2 のそれぞれ 100 個のデータを AE で 2 次元に圧縮した散布図

3.5.1 変分オートエンコーダ

本節では、図 3.5.1 のような変分オートエンコーダ (Variational autoencorder; VAE) を紹介する。VAE での目的は潜在変数の分布が正規分布に従う（$\boldsymbol{z} \sim \mathcal{N}(0, I)$）ようなエンコーダ・デコーダを構成することである。準備として、確率密度関数 $q(x), p(x)$ に対してカルバック・ライブラー (Kullback-Leibler; KL) ダイバージェンスは次のように定義される。

$$D_{KL}(q||p) = \int q(x) \log(q(x)/p(x)) dx.$$

KL ダイバージェンスは 2 つの確率分布がどの程度似ているかを表す尺度である。エンコーダはデータ $\boldsymbol{x}$ に対して

$$\boldsymbol{z}(\boldsymbol{x}; \phi) = \mu(\boldsymbol{x}; \phi) + \sigma(\boldsymbol{x}; \phi)\varepsilon, \ \varepsilon \sim \mathcal{N}(0, I) \tag{C.14}$$

により、確率的に $\boldsymbol{z}$ を割り当てるとする。

デコーダの出力として生成されるデータの周辺尤度 $p_\theta(\boldsymbol{x}) = \int p(\boldsymbol{z}) p_\theta(\boldsymbol{x}|\boldsymbol{z}) d\boldsymbol{z}$ が与えられたデータの分布に一致するように、

$$\sum_i \log p_\theta(\boldsymbol{x}_i) \tag{C.15}$$

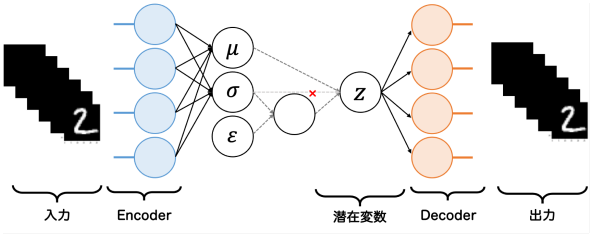

図 C.6: 変分オートエンコーダのアーキテクチャ

を最大化する θ が望ましい。ここで、推定困難な $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ の推定のために、その近似として $p_\phi(\boldsymbol{z}|\boldsymbol{x})$ を導入する。$\log p_\theta(\boldsymbol{x})$ に対して

$$\begin{aligned}\log p_\theta(\boldsymbol{x}) \geq & - D_{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\boldsymbol{z})) \\ & + \int q_\phi(\boldsymbol{z}|\boldsymbol{x}) \log p_\theta(\boldsymbol{x}|\boldsymbol{z}) d\boldsymbol{z} \\ & (= L_{ELBO}(\boldsymbol{x};\theta,\phi)) \qquad \text{(C.16)}\end{aligned}$$

が任意の $\phi, \theta, \boldsymbol{x}$ に対して成り立つ。これを利用して、(C.15) の最大化の代わりに ELBO（Evidence Lower Bound）と呼ばれる L_{ELBO} の和

$$\sum_{i \in \mathcal{B}} L_{ELBO}(\boldsymbol{x}_i;\theta,\phi) \qquad \text{(C.17)}$$

の最大化を考える。第 1 項は解析的に求めることができる。例えば、潜在変数が 1 次元で $f_\theta(z)$ に $N(0,1)$、$q_\phi(z|x)$ に $N(\mu,\sigma^2)$ を仮定すると、$D_{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\boldsymbol{z})) = (\sigma^2 + \mu^2 - \log\sigma^2 - 1)/2$ となる。一方で、再構成誤差に対応する第 2 項は積分計算が必要となるので、バッチ内のサンプルを用いてモンテカルロ法で近似される。以上の議論により、前節と同様の手法により θ, ϕ の最適化をおこなうことができる。学習後は、$\boldsymbol{z} \sim N(0,1)$ にしたがって、$\boldsymbol{z}$ のサンプルを生成し、それをデコーダに入力することでデータを生成できる。これが生成モデルと呼ばれる所以である。

課題 30 (C.16) を示せ。

課題 31 CIFAR-10 に対して VAE を用いて画像を生成せよ。

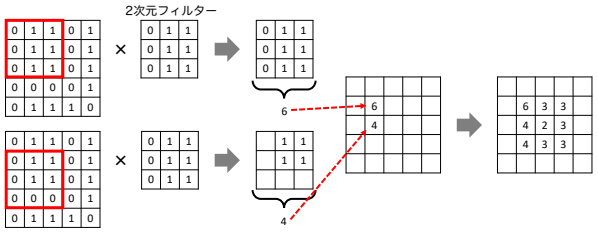

図 C.7: 5×5 の画像に対して 2 次元フィルターを適用した場合

(付録) 畳み込みニューラルネットワーク

画像ソフトにおけるぼかしやエッジ抽出などに応用されている画像処理手法にフィルターがある。畳み込みニューラルネットワーク (CNN) は図 3.5.1 のように画像入力データに 2 次元フィルターを適用して畳み込み演算を行うことで、元の画像の情報を保持しつつ、サイズの小さな画像に変換する手法であり、多くの場面で性能を向上させることができる。CNN の性能を引き上げるための様々な手法が考案されているが、最も効果があるとされているのは畳み込み層や全結合層の間に挿入するプーリング層（最大プーリングや平均プーリング）である。図 3.5.1 からもわかるように、フィルターを適用した後は画像のサイズが小さくなり、連続して適用していくと画像が小さくなりすぎてしまう。このような場合の対策として、あらかじめ元の画像の周りを 0 で埋めておくことで元のサイズを維持するゼロパディングなどがある。

参考文献

[1] 瀧雅人，機械学習スタートアップシリーズ これならわかる深層学習入門，講談社，2017.

[2] Jakub Langr，Vladimir Bok，大和田茂，実践 GAN 敵対的生成ネットワークによる深層学習，マイナビ出版，2020.

[3] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning (Adaptive Computation and

Machine Learning series) , The MIT Press, 2016.

[4] 中井悦司，TensorFlow で学ぶディープラーニング入門: 畳み込みニューラルネットワーク徹底解説，マイナビ出版，2019.

[中山、加嶋]