手書き文字認識を題材とした機械学習入門

@Metachick_2021, @Tofojisan, @quwrof36941574

I think the brain is essentially a computer and consciousness is like a computer program.

—Stephen Hawking

まえがき

機械学習という言葉は今や多くの人々が一度は耳にしたことがあるでしょう。しかし、その内部の仕組みを理解している人は少なく、一部の人々にとってはまるで魔法のように思えるかもしれません。でも心配は無用です。この授業が終わっているころには機械学習の基本的な概念とその実装について理解しているでしょう。機械学習は、私たちの日常生活の中で既に広く利用されています。例えば、文書生成 AI や画像生成 AI などの人工知能は機械学習の一種である深層学習を利用しています。特に、あなた自身がこれらのサービスを使った経験があるならすでに機械学習の成果を体験していることでしょう。今回は機械学習の基本から始め、ニューラルネットワークと全結合層の理論、そして MNIST 手書き文字認識という具体的な問題を解くための実装について学びます。Python とそのライブラリを使った実装を通じて、機械学習の理論が具体的な形でどのように動作するのかを理解していきましょう。

目次

第 I 部　理論 2

1 **機械学習の基礎** 3
- 1.1 機械学習とは何か . . . 3
- 1.2 機械学習の種類 . . . 5

2 **ニューラルネットワーク** 6
- 2.1 ニューラルネットワークとは何か . . . 6
- 2.2 ニューラルネットワークの構造 . . . 6
- 2.3 パーセプトロン . . . 7
- 2.4 活性化関数 . . . 8

3 **全結合層** 11
- 3.1 全結合層とは何か . . . 11
- 3.2 全結合層の計算の性質 . . . 12

4 **損失関数と最適化** 13
- 4.1 損失関数とは何か . . . 13
- 4.2 最適化アルゴリズム . . . 16
- 4.3 誤差逆伝播法 . . . 19

5 **MNIST 手書き文字認識** 20
- 5.1 MNIST データセットの概要 . . . 20
- 5.2 モデルの構築 . . . 21
- 5.3 モデルの評価・精度 . . . 21

第 II 部　実装 23

6 **機械学習ライブラリ** 23
- 6.1 ライブラリとは何か . . . 23
- 6.2 代表的なライブラリ . . . 23

7 **モデルの作成** 24
- 7.1 関数の紹介 . . . 24
- 7.2 実装 . . . 24

8 **モデルの評価と試用** 26

第 I 部
理論

1　機械学習の基礎

1.1　機械学習とは何か

機械学習 (machine learning) とは、コンピュータが十分に多量なデータを学習することにより、データの構造やパターンを自動的に抽出する技術のことである。このような表現からは機械学習が統計的なデータ解析と類似しているように思えるかもしれないが、実際には大きな違いがある。機械学習においては、データのルールやパターンを人が明示的に定義するのではなく、コンピュータがデータをもとに自動的に学習する。つまり、機械学習はデータの特徴を自ら発見し、その特徴を活用して新しいデータに対して予測や判断を行うことができる。特に、機械学習は統計的な手法よりも柔軟性があり、複雑なデータや非線形な関係性をより効果的に解析することに特化している。

さて、右側の数字は何を示しているだろうか？ もちろん、多くの人が”5” と答えるだろうし、実際に 5 である。しかし、この画像が 28×28 ピクセルという極めて低い解像度で描かれているにもかかわらず我々の脳はすぐさまそれを 5 だと認識することができる。これはいったいなぜだろうか？ 我々の脳がどうやってこれを 5 だと認識するのか、そのメカニズムは何なのだろう？ これは我々が日常生活で無意識に行っている認識のプロセスである。我々の脳は画像の各ピクセルの色や位置、形状などの情報を解析し、それを 5 という数字と関連付ける。さらに、下の 3 枚の画像も見てみよう。

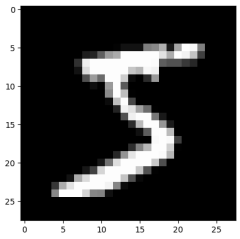
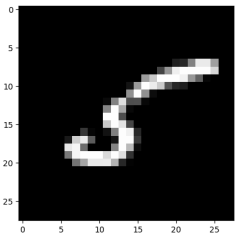
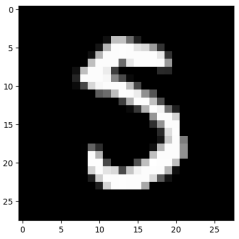


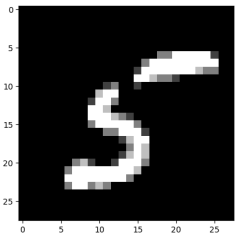
図 1: 手書きの”5”(1)



(a) 手書きの”5”(2)



(b) 手書きの”5”(3)



(c) 手書きの”5”(4)

図 2: 三枚の”5”

これらもまた 5 だと認識するだろう。しかし、これらのピクセルの値は全て明らかに異なる。それなのに、我々はこれら全てを 5 だと認識する。そこにはどんな法則があるのか？ これをどのように言語化したら良いのか？ 実際に、手書きの数字の画像から数字を的中させるプログラムを書くことが如何に難しいがが容易に想像できると思う。これこそが、機械学習が解決しようとしている問題である。機械学習は人間の脳が自然に行っているこのような情報の解析と理解を機械にも可能にしようとする技術のことである。

機械学習とは、**「関数近似器」**である。つまり、ある入力から出力を生成する関数を見つけ出すプロセスと言える。

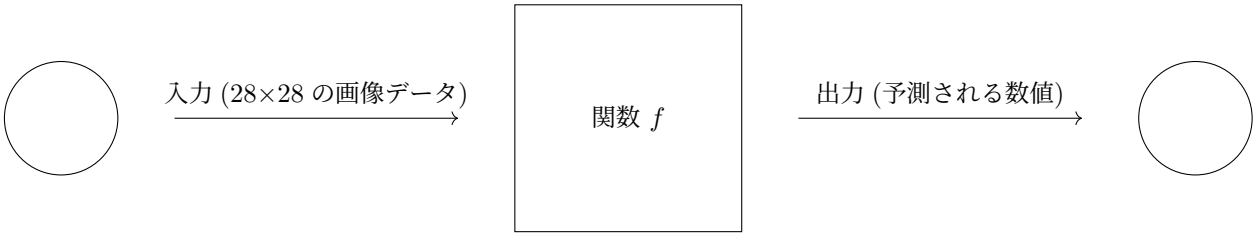


図 3: 最適な f を見つける作業こそが機械学習である。

改めて、手書き文字認識の問題を考えてみよう。今回の目標は、28×28 の手書きの数字の画像を入力として与えられたときに、予測される数字を出力するプログラムを作成することである。入力となる 28×28 の画像データは、各ピクセルにグレースケール値が割り振られた数値の集まりとして解釈できる。そして、この数値の集まりを別の数値（予測される数字）に変換する関数を見つけ出すことが我々の目標である。もちろん、そのような関数を人間が条件などを並べて定義することは非常に難しい。機械学習は、こういった複雑な場面で用いられ、真価を発揮するのである。

■余談：機械学習は万能か？

しかし、今一度機械学習は万能ではないという点は確認しておくべきである。最近の生成系 AI などの目まぐるしい進化を見ると、機械学習は万能であるかのように思えてしまう。だが、当たり前ながら機械学習は万能ではない。

まず、機械学習はデータに大きく依存している。つまり、良質で十分な量のデータがなければ機械学習アルゴリズムは有用な結果を生み出すことができない。さらに、データが偏っていたり誤った情報を含んでいたりするとそれは学習結果にも反映されてしまう。これは「ゴミ入ればゴミ出る」（Garbage In, Garbage Out）の原則と呼ばれている。

次に、機械学習は「黒い箱」の問題を抱えている。機械学習モデルがどのようにして特定の結果を導き出したのかを理解するのが難しい場合がある。機械学習は、あくまで関数近似器であり、それ以上の存在ではないのである。これは特に深層学習などの複雑なモデルに当てはまり、モデルの解釈可能性と透明性に関する問題を引き起こしている。

1.2 機械学習の種類

まずは、機械学習の種類について確認する。機械学習は**人工知能 (Artificial Intelligence)** の一種である。機械学習には主に以下の二つのタイプが存在する。

教師あり学習

教師あり学習とは、アルゴリズムがラベル付きの訓練データから学習する方法のことである。ラベル付きデータとは、各データポイントが特定の出力値（ラベル）と関連付けられているデータのことを指す。つまり、問題と解答をセットで与えて学習させる手法のことである。教師あり学習は主に**分類**と**回帰**との二つに分類される。手書き文字認識も教師あり学習であり、特に分類問題である。

- 分類：出力が離散的なカテゴリーである場合の予測問題である。
  例) メールが「スパム」または「スパムでない」かを予測する問題

- 回帰：出力が連続的な数値である場合の予測問題である。
  例) 天気、気温などの入力からその日の飲み物の売り上げ (数値) を予測する問題。

教師なし学習

教師なし学習とは、アルゴリズムがラベルなしのデータからパターンや構造を見つけ出す方法のことである。つまり、問題だけを与えて学習させる手法のことである。教師あり学習は問題と解答をセットで与えていたのに対し、教師なし学習は問題だけを与えるので主要な用途も異なる。教師なし学習は主に**クラスタリング**で用いられる。

- クラスタリング：データを自然なグループまたはクラスタに分ける問題である。
  例) 商品データベースを分析して、商品を異なるグループ（例えば、人気な商品、不人気な商品など）に分ける問題。

分類とクラスタリングはどちらもデータを特定のグループに分けるという点で似ているが、与えるデータにラベルがあるかどうかという点で異なる。分類は既知のカテゴリーに新しいデータポイントを割り当てることを学習し、クラスタリングはデータの自然なグループを見つけ出すことを目指すものである。

教師あり学習と教師なし学習は機械学習の主要な手法であるが、それだけが全てではない。他にも強化学習や半教師あり学習など、さまざまな手法が存在する。これらは特定の問題解決に有効な手段となることがあるので、興味がある方はこれらの手法についても一度調べてみると良いだろう。

2　ニューラルネットワーク

2.1　ニューラルネットワークとは何か

機械学習にはデータからパターンや関係性を自動的に抽出し、未知のデータに対して予測や分類を行うためのアルゴリズムがいくつか存在する。特に、それらを**モデル (model)** と言う。今回はその中でも、ニューラルネットワークを扱う。**ニューラルネットワーク (neural network)** とは、人間の脳の神経細胞（ニューロン）の動作を模倣した機械学習のモデルの一つである。このモデルは複数の層からなるネットワーク構造を持ち、各層は多数の**ニューロン (neuron)** 或いは**ノード (node)** で構成されている。各ノードは他のノードからの入力を受け取り、それらを加重和として計算し、さらに非線形の活性化関数を適用して出力を生成する。ニューラルネットワークは画像認識、自然言語処理、音声認識など、多くの応用分野で高い性能を発揮している。

2.2　ニューラルネットワークの構造

ニューラルネットワークは以下のような**入力層**、**中間層 (隠れ層)**、**出力層**から成り立っている。特に、中間層の個数が二つ以上のモデルを**ディープラーニング (deep learning)** と呼ぶ。

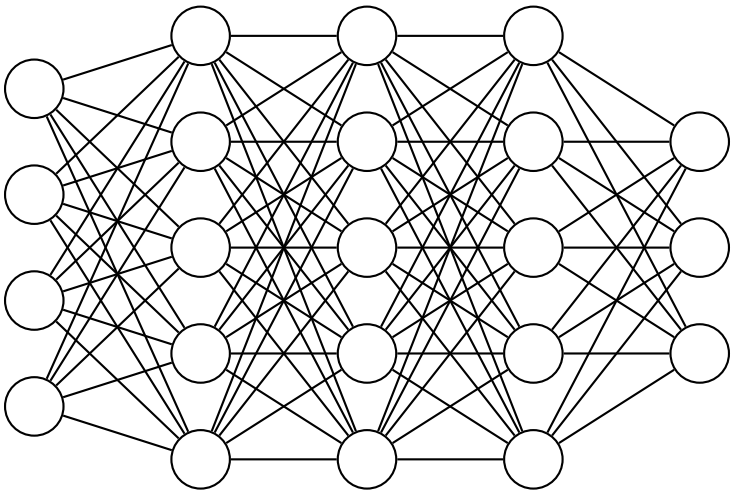


図 4: ニューラルネットワーク

- 入力層：図の一番左側の層にあたる部分で、データの各特徴量がこの層の各ノードに対応する。入力層は、データをネットワークに供給する役割を果たしている。
- 中間層：図の真ん中三つの層にあたる部分で、これらの層は入力層と出力層の間にあり、ネットワークが複雑な特徴を学習するのに貢献する。中間層が多いほど（深いほど)、ネットワークはより複雑な特徴を学習することができる。
- 出力層：図の一番右側の層にあたる部分で、この層のノード数は予測したいクラスの数に等しくなる。出力層は、ネットワークの最終的な予測結果を生成する。

2.3 パーセプトロン

入力層から入力された情報は入力層→中間層→出力層のように伝わっていく。この際に用いられるのがパーセプトロン (正確にはその派生) というアルゴリズムである。**パーセプトロン (perceptron)** とは、複数の入力を受け取り、一つの信号を出力するような情報伝達のアルゴリズムである。ここでは、以下のような二つの入力を持つような単純パーセプトロンを例に考えていく。

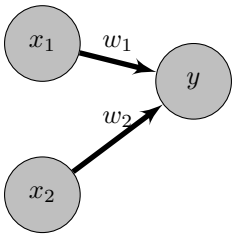


図 5: パーセプトロン

図の円は前頁の図 4 と同様にニューロン (ノード) と呼ばれ、それぞれが数値を持つ。ここでは、左側二つの円を入力とし、x_1, x_2 で表す。同様に、w_1, w_2 は重みを、b はバイアスを、y は出力を表す。このとき、出力 y と入力 x_1, x_2 の関係は以下のように表される。

$$y = \begin{cases} 0 & (w_1x_1 + w_2x_2 + b \leq \theta) \\ 1 & (w_1x_1 + w_2x_2 + b > \theta) \end{cases}$$

ここで、θ は閾値である。この式は、入力の加重和が閾値を超えたら 1 を、そうでなければ 0 を出力することを意味する。このような関数はステップ関数と呼ばれ、パーセプトロンの活性化関数として用いられる。**ステップ関数 (step function)** は、その名の通り階段のようにステップを上がるような形状を持つ関数である。特定の閾値を境に出力が変わるという特性を持つため、パーセプトロンのような二値出力を必要とする場合に適している。具体的には、ステップ関数は入力が閾値を超えたか否かで出力を切り替える。閾値を超えなければ出力は 0（または-1)、閾値を超えれば出力は 1 となる。
パーセプトロンは複数の入力信号に対して個別の重みを割り当てる。これらの重みは各信号の相対的な重要性を調整する役割を果たす。つまり、重みが大きいほど、対応する信号の重要性も高くなる。もちろん、入力が二つである必要はない。一般に n 個の入力では出力は以下のように表される。

$$y = \begin{cases} 0 & (\sum_{i=1}^{n} w_ix_i + b \leq \theta) \\ 1 & (\sum_{i=1}^{n} w_ix_i + b > \theta) \end{cases}$$

パーセプトロンを用いれば、AND や OR、NAND などの簡単な論理回路を作成することができる。例えば、OR 回路を再現するには それぞれの重みを 0.5 に、閾値を 0.3 などにすればよい。なお、XOR は単層のパーセプトロンでは作成することができない。ただし、パーセプトロンを重ねることによって XOR やもっと複雑な仕組みも作成することができる。ニューラルネットワークもこのパーセプトロンを何層にも重ねて、活性化関数を用いることによって構成されている。

2.4 活性化関数

活性化関数 (activation function) とは、ニューロンの出力値を決定する非線形関数のことを指す。パーセプトロンの基本的な形では出力はステップ関数によって決定される。つまり、ある閾値を超えたら 1 を、そうでなければ 0 を出力する。しかし、このステップ関数は出力が 0 か 1 しか取らないため、情報の微妙なニュアンスを伝えることができない。そこで、実際のニューラルネットワークでは、活性化関数として非線形関数、例えばシグモイド関数やハイパボリックタンジェント関数、ReLU（Rectified Linear Unit）などが使用される。活性化関数を導入することで、多層パーセプトロンは複雑な関数を表現する能力を持つようになり、より広範かつ深淵な問題領域に対応可能となる。その結果、画像認識、自然言語処理、音声認識といった様々なタスクで、ニューラルネットワークはその強力な表現力を発揮することができるようになった。

f を活性化関数としたときに、これは以下のように適用される。

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

具体的に活性化関数に求められる条件には以下のようなものが挙げられる。つまり、活性化関数は、以下の条件を満たしていてステップ関数に似ているような” 丁度よい” 関数である。

- 非線形性：非線形性は表現力の源泉であり、これによりネットワークは複雑なパターンを捉えることが可能になる。
- 微分可能性：学習の際に用いられる誤差逆伝播法 (誤差逆伝播法は第 4 節を参照) を適用するためには、活性化関数が微分可能であることが必要である。
- 計算効率：活性化関数はネットワークの各層で頻繁に計算されるため、計算効率が高いことが重要である。
- 出力の範囲：活性化関数の出力範囲が制限されていると、学習過程が安定しやすくなる。

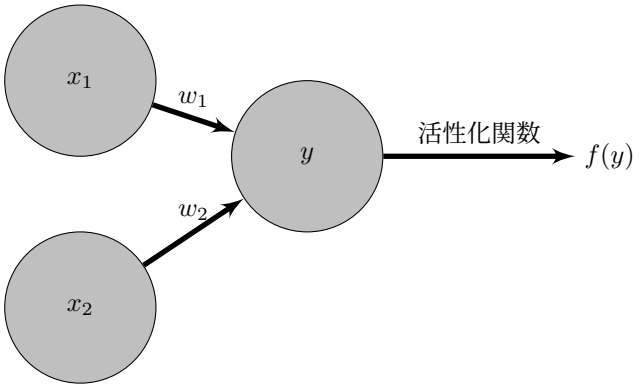


図 6: 活性化関数を通すノード

中間層の主要な活性化関数

以下に、中間層の主要な活性化関数を示す。

- シグモイド関数：シグモイド関数は、入力値を 0 から 1 の範囲に変換するために使用される。

$$f(x) = \frac{1}{1 + e^{-x}}$$

- ハイパボリックタンジェント関数：ハイパボリックタンジェント関数は、入力値を-1 から 1 の範囲に変換するために使用される。

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

- ReLU 関数：ReLU 関数は、入力が 0 以下の場合は 0 を、0 より大きい場合は入力値をそのまま出力する。なお、微分が定義されない原点に関しては、その微分係数を 1 と定義する。

$$f(x) = \begin{cases} 0 & (x \leq 0) \\ x & (x > 0) \end{cases}$$

以下に、ステップ関数と中間層の主要な活性化関数のグラフを示す。

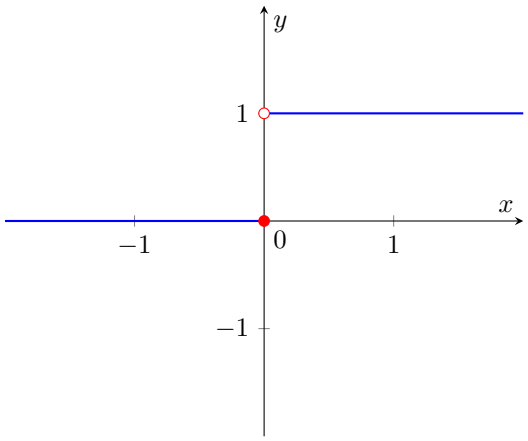


図 7: ステップ関数

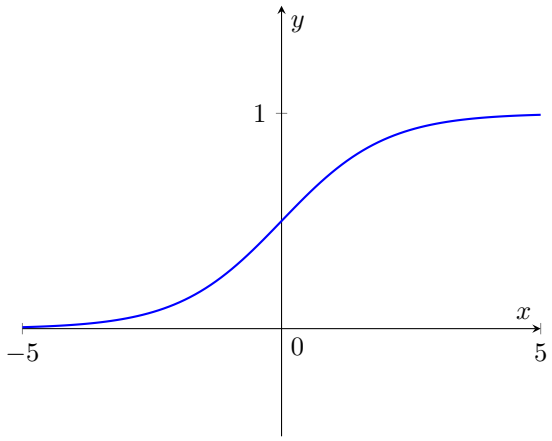


図 8: シグモイド関数

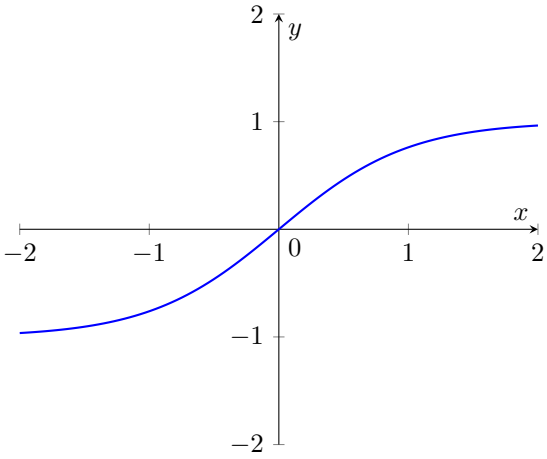


図 9: ハイパボリックタンジェント関数

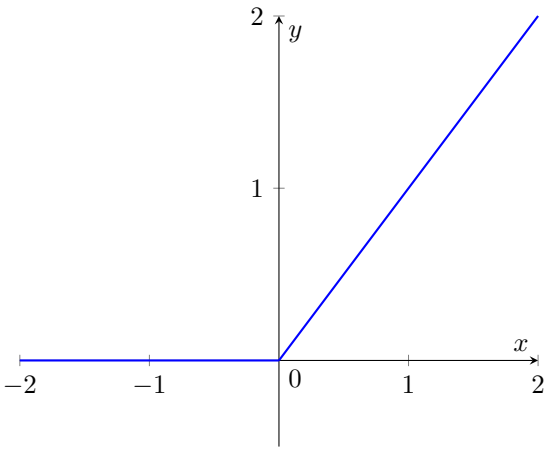


図 10: ReLU 関数

出力層の活性化関数

出力層の活性化関数は中間層の活性化関数と区別して考える。特に、目的のタスクによって出力層の活性化関数は変わってくる。以下に、タスクと出力層の主要な活性化関数の組を示す。

- 回帰問題 + 恒等関数：回帰問題では、数値の情報を漏れなく伝えるために恒等関数を用いる。

$$f(x) = x$$

- 二値分類問題 + シグモイド関数：二値分類問題では、シグモイド関数を用いる。

$$f(x) = \frac{1}{1 + e^{-x}}$$

- 多クラス分類問題 + ソフトマックス関数：多クラス分類問題では、マックス関数と似ている” 丁度よい” 関数を用いる。値域は 0 から 1 である。

$$f(x_k) = \frac{e^{x_k}}{\sum_{i=1}^{n} e^{x_i}}$$

以下に、出力層の主要な活性化関数のグラフを示す。ただし、ソフトマックス関数は二入力であり、一方の入力を 1 に固定している。

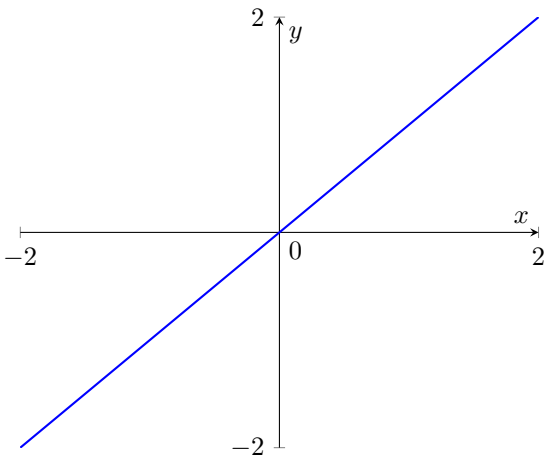


図 11: 恒等関数

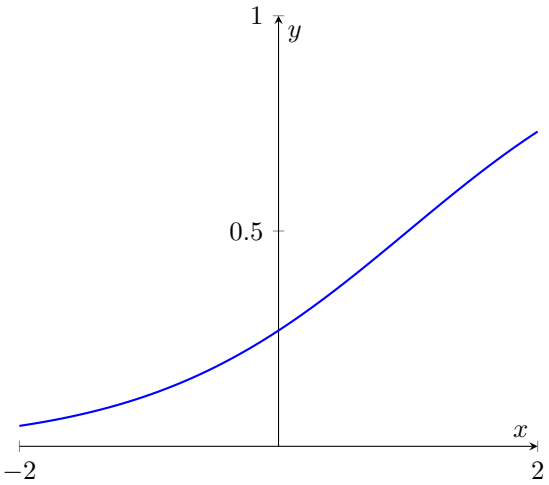


図 12: ソフトマックス関数

3　全結合層

3.1　全結合層とは何か

全結合層 (fully-connected layer 或いは dense layer) とは、全てのニューロンが次の層の全てのニューロンと接続されているような層のことである。つまり、全ての変数を使う一番基本的な層のことである。

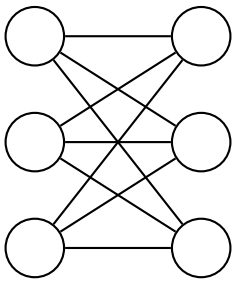


図 13: 全結合層

全結合層の役割は各入力に対して全体的な重み付けを行い、ニューロン間で情報を統合することにある。

- 情報の統合：全結合層はネットワークの前の層からの情報をすべて受け取ることから、それぞれの特徴に重みを付けて結合することが可能となる。これにより、各特徴の影響力を調整することが可能となる。
- 非線形性の導入：全結合層は通常は活性化関数と一緒に使われるので、その影響で出力が非線形になる。このことから、モデルがより複雑なパターンや特徴を学習することが可能となる。
- 出力の生成：ニューラルネットワークの最後の層には、通常は全結合層が用いられる。全結合層は、全体的なパターン認識や最終的な判断を行う役割を担っている。

■余談：ニューラルネットワークの普遍性定理

ニューラルネットワークの理論において、重要な定理が存在する。それは、中間層が一層だけ存在すれば、任意の連続関数を近似することが可能であるというものである。この定理は、普遍性定理あるいは万能近似定理として知られている。

この定理の主要な特徴は、任意の連続関数を表現するために必要な中間層が一つであるという事実にある。これは、ニューラルネットワークの表現力についての深遠な洞察を提供する。この定理は、その単純さと強力さから非常に興味深いものである。

しかしながら、さらに興味深い現象が観察されている。それは、現実の問題においては、中間層の数が増えるほど、ニューラルネットワークの性能が向上するという現象である。これは、理論的な最小限の要件と実際の性能との間に存在する興味深いギャップを示している。この現象の理解は、ニューラルネットワークの設計と最適化において重要な役割を果たしている。

3.2 全結合層の計算の性質

全結合層の重みの個数は、入力ニューロン数を I として、出力ニューロン数を O とすれば、$I \times O$ で表される。したがって、全結合層一層あたりに必要な乗算の回数は IO 回となる。乗算に限らない、計算の回数はバイアスを含めるため、$IO + O$ 回となる

利点

全結合層の計算の性質には以下の利点がある。

- 単純性：計算は直接的であり、具体的な位置情報に依存しないため、どの入力ニューロンもどの出力ニューロンとも接続できる。
- 表現力：すべてのニューロンが互いに接続されているため、ニューラルネットワークの普遍性定理にあるように複雑な関数も表現することができる。

欠点

全結合層の計算の性質には以下の欠点がある。

- 計算コスト：入力ニューロンと出力ニューロンの数が増えると、必要な計算量も急激に増大してしまう。
- メモリ使用量：上記の計算量の増加に伴い、モデルのサイズやメモリの使用量も増加してしまう。これは、特にメモリが制限されているデバイスでの実装に問題を生じる可能性がある。
- 過学習のリスク：高い表現力を持ちすぎるが故に、訓練データには非常に高い性能を示すが、テストデータや実際のデータに対しては低い性能を示す可能性が高まってしまう。

4　損失関数と最適化

4.1　損失関数とは何か

前章まででは情報伝達の流れについて確認してきた。一方、この章では機械学習の中核となる学習過程における動きを詳しく見ていく。学習の本質は、**重みの調整**である。学習の際には、まず目標を明確に設定し、現状がその目標からどれだけ離れているかの指標を決定する必要がある。この指標を**損失関数 (loss function)** と呼ぶ。基本的な損失関数とその性質を確認していく。

絶対平均誤差

絶対平均誤差 (MAE) は $\hat{x_i}$ を予測値、x_i を正解値 (教師データ) とすれば、以下の式で与えられる。

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}|\hat{x_i} - x_i|$$

絶対平均誤差は各データの誤差の絶対値を平均することによって計算される。後述する二乗平均誤差よりも外れ値に寛容である。平均絶対値誤差には以下の性質がある。

- 連続性：連続性を持つため、連続した対象の予測に適している。
- 微分可能性：原点を除いて微分可能であるため、多くの最適化アルゴリズムが適用できる。原点に関しては、便宜上の微分係数を定めてしまえば問題ない。
- 外れ値への寛容性：外れ値に寛容で、後述する二乗平均誤差よりも影響が少ない。
- 非負性：絶対値をとっているため、負の値を取ることはない。
- 次元の一致：元のデータと次元が一致しているため、人間が直感的に誤差を評価できる。

グラフからわかる通り、絶対平均誤差は微分の値が一定になってしまうので細かい学習が難しくなってしまう。

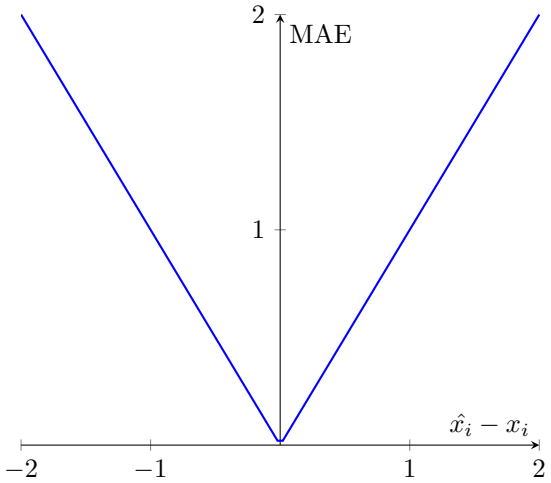


図 14: 絶対平均誤差

二乗平均誤差

二乗平均誤差 (MSE) は $\hat{x_i}$ を予測値、x_i を正解値 (教師データ) とすれば、以下の式で与えられる。

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{x_i} - x_i)^2$$

二乗平均誤差は各データの誤差を二乗して平均することによって計算される。二乗平均誤差は最も一般的な損失関数であり、幅広く使用されている。前述の絶対平均誤差よりも誤差に敏感である。二乗平均誤差には以下の性質がある。

- 連続性：連続性を持つため、連続した対象の予測に適している。
- 微分可能性：微分可能であるため、多くの最適化アルゴリズムが適用できる。
- 外れ値への敏感性：外れ値に敏感で、大きく影響を受ける。
- 非負性：二乗されているため、負の値を取ることはない。

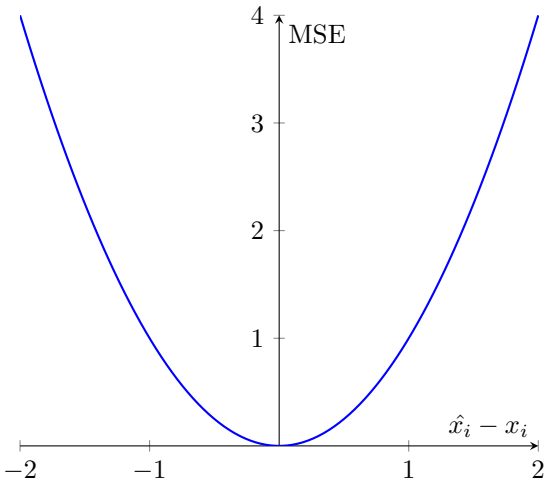


図 15: 二乗平均誤差

二乗平均平方根誤差

二乗平均平方根誤差 (RMSE) は $\hat{x_i}$ を予測値、x_i を正解値 (教師データ) とすれば、以下の式で与えられる。

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{x_i} - x_i)^2}$$

二乗平均平方根誤差は各データの誤差を二乗して平均したものの平方根をとることによって計算される。基本的な性質は二乗平均誤差と変わらないが、二乗平均平方根誤差では誤差に寛容であり、次元が一致している点が異なる。

交差エントロピー誤差

交差エントロピー誤差 (cross entropy) は $\hat{x}_i$ を予測値の確率分布、x_i を正解値の確率分布とすれば、以下の式で与えられる。

$$H = \sum_{i=1}^{n} -x_i \log \hat{x}_i$$

交差エントロピー誤差は二つの確率分布の間に定義される誤差であり、分類問題で使用される。交差エントロピー誤差は正解値と予測値の対数の積の総和をとることによって計算される。今までの誤差関数よりも複雑なので具体例を用いて理解していく。

前提として、一般に、正解値の確率分布は、one-hot ベクトル、つまり、正解のラベルだけが 1、他はすべて 0 になっているベクトルで表現されている。例えば、(inu, neko, usagi) のうちの”inu” の画像であれば、(1,0,0) という具合である。これは、”inu” の確率が 100 %であることを表す。

- モデルの出力が (0.2,0.4,0.4) の場合：$-(1 \times \log 0.2 + 0 \times \log 0.4 + 0 \times \log 0.4) \simeq 1.60$
- モデルの出力が (0.8,0.1,0.1) の場合：$-(1 \times \log 0.8 + 0 \times \log 0.1 + 0 \times \log 0.1) \simeq 0.22$

このように、確率分布同士が近いほど、交差エントロピー誤差は小さくなっていることがわかる。確率分布を扱う場面において、二乗平均誤差を使用する場合は距離が遠くても近くても誤差の傾きが一定となってしまうが、確率の逆数のような形 $(-1/x)$ に傾きが変化するため、実際の確率分布の違いをより適切に捉えることができる。交差エントロピー誤差には以下の性質がある。

- 連続性：交差エントロピーは連続関数であるため、連続的な出力変数の予測に適している。
- 微分可能性：交差エントロピーは微分可能であるため、勾配降下法やその他の最適化アルゴリズムでの使用に適している。
- 外れ値への敏感性：予測が正解から大きく外れると、交差エントロピー誤差は急激に増加する。これは、モデルが確信を持って誤った予測をすると、大きなペナルティが課せられることを意味する。
- 非負性：交差エントロピー誤差は、2 つの確率分布間の「距離」を測定するものであるため、負の値を取ることはない。

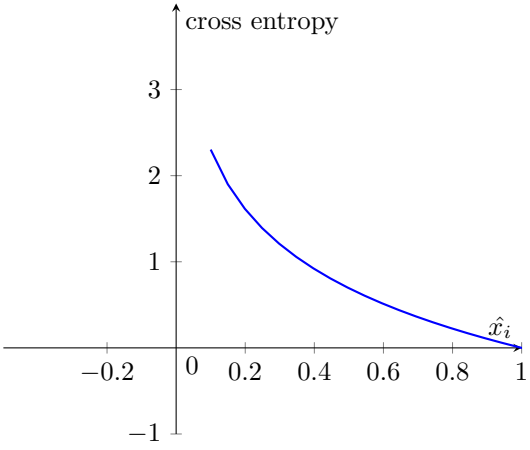


図 16: 交差エントロピー誤差

4.2 最適化アルゴリズム

学習において、適切に重みを調整するために損失関数を最小化する方向に進むことが必要である。この重みの調整プロセスを**最適化 (optimization)** と呼ぶ。最適化の基本的なアイデアは、一般的な関数の最小値を見つける際と同様に、微分を利用することである。微分とは、ある関数の接線や接面を求める操作のことであり、最適化アルゴリズムにおいては、誤差関数を最小にするための関数の調整方法が分かる。この微分を繰り返すことによって、より良い目的とする関数を探していく。代表的な最適化アルゴリズムとその性質を確認していく。

最急降下法

最急降下法 (gradient descent) は最も基本的な最適化アルゴリズムの一つである。最急降下法は以下の手順で行われる。

- 各重みの初期値を設定
- 各重みについて偏微分
- 各重みが損失関数を最小化する方向に移動
- 収束判定 (収束していない場合は偏微分と移動を繰り返す)

まずはじめに、各重みの初期値を設定する。次に、各重みについて偏微分を行い、各変数について損失関数が最小となる方向を特定する。偏微分とは、簡単に言えば各変数について微分することである。損失関数の値が小さくなる方向に移動を繰り返す。この重みの更新一回分を**イテレーション**と言う。これはハイパーパラメータの一つである。そして、収束するまでイテレーションを繰り返し、収束したら終了する。

山を下ることをイメージしてみると分かりやすい。各重みの偏微分は、山の下り方向を特定することに対応していて、各重みが損失関数を最小化する方向に移動することは実際に山を下ることに対応する。

ただし、この方法では、最小値に収束するとは限らず、極小値 (局地的な最小値) に陥って抜けられなくなってしまうこともある。例えば、下図において、初期値が負の場合は極小値に陥ってしまう。

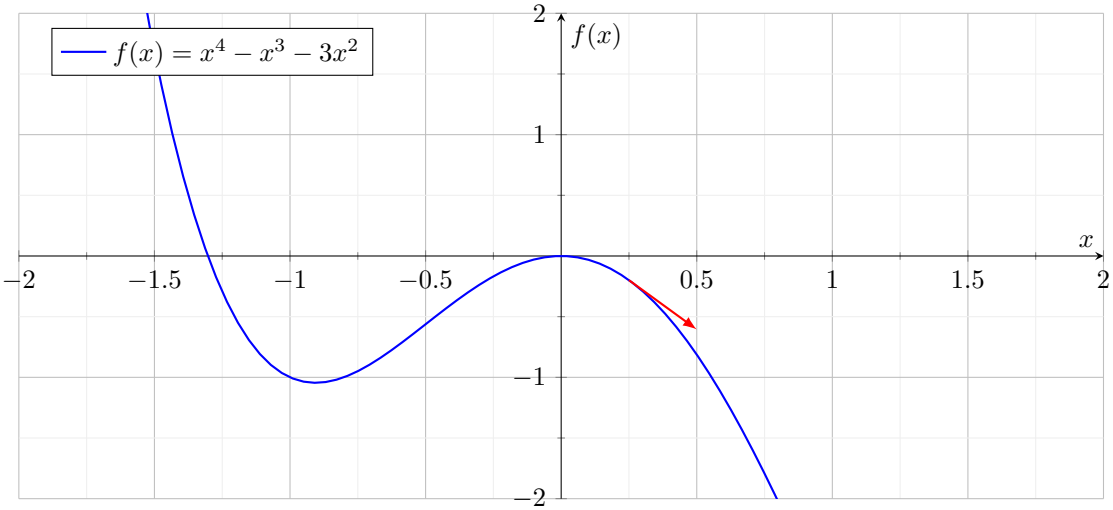


図 17: 最急降下法

確率的勾配降下法

前頁でも述べた通り、最急降下法を用いて最適化を試みると、極小値に陥ってしまうことがある。これに対処するために、ある程度小さい方向へ移動できるが、真っ直ぐに最小値へと向かわないような丁度良い手法があるとよい。そこで、最急降下法にランダム性を用いる。この手法こそが**確率的勾配降下法 (stochastic gradient descent, SGD)** である。確率的勾配降下法は以下の手順で行われる。

- 各重みの初期値を設定
- データセットからランダムに一つの変数を選択
- 選択された変数についてのみ偏微分
- 各重みが損失関数を最小化する方向に移動
- 収束判定 (収束していない場合は新たな変数の選択と偏微分、移動を繰り返す)

確率的勾配降下法は、各イテレーションで全てのデータセットに基づいて勾配を計算するのではなく、ランダムに選ばれた一つのサンプルのデータだけを使用して勾配を計算する。ランダム性を加えると、極小値を回避しやすくなる理由は以下にある。

- 振動的な更新：確率的勾配降下法では、各イテレーションでランダムに選択された 1 つのデータ点に基づいて勾配の更新を行う。この結果、勾配の更新はノイズが多くなり、このノイズが極小値に捉えられることを回避する力として作用する。
- 探索と活用のバランス：ランダム性が加わった方法では、各ステップで異なる方向への「探索」が行われるため、局所的な最小値からの脱出が容易になる。
- 大規模なデータセットの場合の効果：特に大規模なデータセットの場合においては、全データに基づく真の勾配と一部のサンプルに基づく勾配の間には差異が生じ、この差異が極小値周辺でランダムな「振動」をもたらすことで、極小値から脱出する助けとなる。
- 鞍点の回避：高次元の最適化の問題では、極小値よりも鞍点 (ある変数に注目すると極小値だが、別の変数から見ると極大値となるような点) に陥ることがより一般的である。その極小値になっている変数にさえ着目しなければ、脱出することができる。

ただし、ランダム性があるからと言って、必ずしも全ての局所的な最小値や鞍点から脱出するわけではないことに注意せよ。

最急降下法の山を下るアナロジーを考えると、確率的勾配降下法は高度が下がるが、ランダムな方向に足を踏み出すハイカーのようなものだと言える。そのため、一歩一歩は必ずしも最適な方向への移動を意味しないが、全体としては目的地（最小値または極小値）に向かって進むことが期待される。

この方法の利点は、各イテレーションの計算コストが低く、特に大量のデータでの学習が高速になることである。しかし、確率的勾配降下法の欠点として、更新が振動的であり、収束の挙動が不安定であることが知られている。これは確率的勾配降下法が一度の更新で一つのサンプルだけを考慮するためである。そのため、実際の応用では、Momentum や学習率のスケジューリングといったテクニックと組み合わせて用いられることが多い。

勾配降下法の派生アルゴリズム

以下はその他の最適化アルゴリズムである。これらは主に確率的勾配降下法の派生であり、**Adam** に関しては現在も最前線で使用されている。主要なアルゴリズムについて、以下に示す。

- Momentum：Momentum は物理学の概念 (特に、運動量) を取り入れた最適化手法で、重みの更新に前回の更新分を加算することで、勾配が緩やかな方向の動きを加速し、振動を減少させる。これにより、より効率的に最適な点に収束することが期待される。
- Adagrad (Adaptive Gradient Algorithm)：Adagrad は学習率をパラメータごとに適応的に調整する手法である。具体的には、よく更新されるパラメータは小さな学習率を、あまり更新されないパラメータは大きな学習率を持つことになる。
- RMSprop (Root Mean Square Propagation)：RMSprop は Adagrad の発展形と見なすことができ、累積する過去の勾配の二乗の平均を使用して学習率を調整する。Adagrad の学習率の減少を緩和するための手法である。
- Adam (Adaptive Moment Estimation)：Adam は Momentum と RMSprop のアイデアを組み合わせた手法である。動きの加速（Momentum）と学習率の適応的調整（RMSprop）を同時に行うため、多くのタスクで効果的に動作することが実証されている。

勾配降下法以外の最適化アルゴリズム

勾配降下法は損失関数の一階微分（勾配）を用いて最適化を実行する手法である。対照的に、二階微分を用いることで、関数の曲率情報を含めることができ、その結果、効率的な収束や安定した最適化を期待できる方法も開発されている。この二階微分を活用したアプローチにより、最適化の方向性やステップサイズを精度良く調整することができる。このカテゴリには、ニュートン法や準ニュートン法といった手法が挙げられる。通常のニュートン法は、例えば $\sqrt{2}$ のような値を計算する際に $f(x) = 0$ の解を求めることを目指し、それには一階微分が用いられる。一方、最適化におけるニュートン法は、$f'(x) = 0$ を満たす解、すなわち損失関数の最小値または最大値を探求することが目的となり、このために二階微分が必要とされる。だが、ニュートン法を用いる際の欠点として、ヘッセ行列（二階微分の行列）の逆行列計算が必要となり、この計算の複雑さが変数の数の 2 乗もしくは 3 乗に比例するため、大規模な問題への適用は計算コストが高くなることが指摘されている。

4.3 誤差逆伝播法

ニューラルネットワークの微分は、多くの入力やニューロン間の複雑な相互作用が存在すると非常に計算量が多くなる。この計算の効率を向上させるために、**誤差逆伝播法 (backpropagation)** というアルゴリズムが導入される。本節では、この重要なアルゴリズムの理解とその動作原理を解説する。

誤差逆伝播法の主な目的は、ニューラルネットワークの重みやバイアスの勾配を効率的に計算することである。具体的には、i 層目 j 番目の重みに対する損失関数の勾配、$\frac{\partial L}{\partial w_{i,j}}$ を計算したい。ここで、L は損失関数の値を表す。この勾配を計算するためのキーとなるのは、偏微分の連鎖律である。連鎖律は以下の式で表現される。

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u} \cdot \frac{\partial u}{\partial x}$$

ここで、$\frac{\partial y}{\partial u}$ は次の層の誤差に関する現在の層の出力の微分、$\frac{\partial u}{\partial x}$ は現在の層の入力に関する重み付き和の微分を示す。これを繰り返すことで、効率的に微分が計算できる。

具体例を確認していこう。

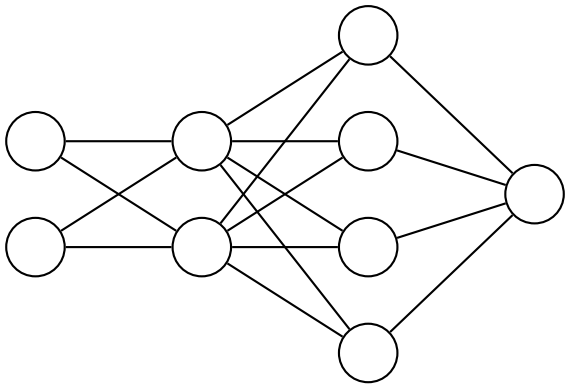


図 18: ニューラルネットワーク

模式的に中間層を 2 層もつニューラルネットワークを用意した。入力層から順に 1,2,3 層と呼び、i 層 j 番目の重みを $w_{i,j}$ とする。ニューロンも同様に入力から順に 0,1,2,3 層と呼び、出力を $O_{i,j}$ とする。また、最も右側のニューロンの出力 $O_{3,1}$ を L とする。ここでは、$\frac{\partial L}{\partial w_{1,1}}$ を計算することを目標とする。$w_{1,1}$ を動かすと、後ろのネットワークが全て変動してしまい、計算が大変になる。よって、誤差逆伝播法を利用する。連鎖律によって、以下のように計算できる。

$$\frac{\partial L}{\partial w_{1,1}} = \sum_{i \leq 4,\ j \leq 2} \frac{\partial L}{\partial O_{2,i}} \frac{\partial O_{2,i}}{\partial O_{1,j}} \frac{\partial O_{1,j}}{\partial w_{1,1}}$$

このように、ニューラルネットワークの出力から入力までの途中の各層における誤差を効率的に計算する。推論のときに用いる順伝播方向とは逆に勾配の情報が伝わっていく。

5　MNIST 手書き文字認識

ここからは、実際に機械学習を実装してその仕組みを学習していく。ここでは、その理論についてを扱い、実装のためのコードや、ライブラリについては第 2 章で扱う。

今回は、MNIST データセットを用いた手書き数字認識を行う。事前に用意されたデータセットを用いて、手書き数字を識別するためのモデルを作成する。模式図は以下のようになる。

図 19: 実際に、機械学習を用いて最適な f を見つけていく。

5.1　MNIST データセットの概要

MNIST（Modified National Institute of Standards and Technology）データセットは、機械学習とディープラーニングの分野で広く利用される手書き数字認識のベンチマークデータセットである。このデータセットには、手書きの 0 から 9 までの数字が 28x28 ピクセルの白黒画像として収録されている。

主な特徴と用途は以下の通りになる。

- **規模**: MNIST データセットには 6 万枚の訓練画像と 1 万枚のテスト画像が含まれている、比較的小規模なデータセット。
- **手書き数字**: 0 から 9 までの手書き数字が含まれており、個々の数字に対する分類タスクに使用される。
- **画像データ**: 画像は 28x28 ピクセルの解像度を持つ白黒画像。この低解像度の特性は、単純なモデルから高度なディープラーニングモデルまで、さまざまなアルゴリズムで扱うことができる。
- **学習と評価**: MNIST データセットは、機械学習アルゴリズムの学習と評価のために、訓練データとテストデータの分割がされている。
- **ベンチマーク**: MNIST は、新しいアルゴリズムやモデルの性能を比較するための標準的なベンチマークとして広く使用されていて、競技会や研究プロジェクトでも頻繁に利用される。

5.2 モデルの構築

今回は、全結合層 3 層からなるモデルを使用する。

- 入力層：784 個のニューロン (画像は $28 \times 28 (= 784)$ のデータであるため)
- 第一層：100 個のニューロン
- 第二層：50 個のニューロン
- 出力層：10 個のニューロン (予測したい数値は 0 〜 9 の 10 種類あるため)

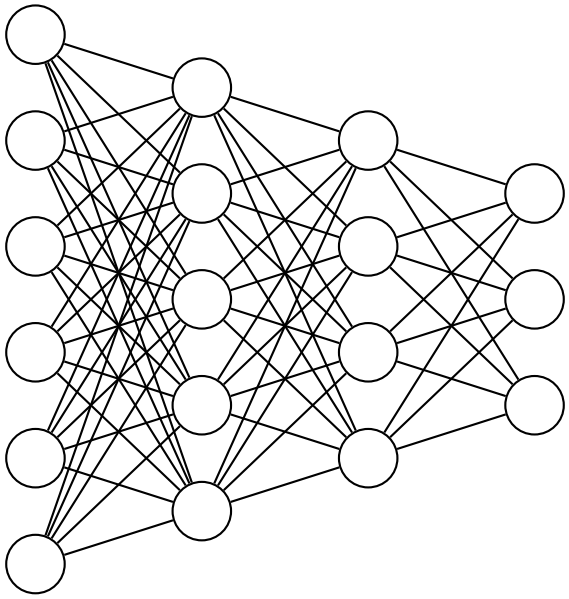


図 20: 構築モデルのイメージ図 (実際のニューロン数はもっと多い)

5.3 モデルの評価・精度

使用する損失関数や最適化アルゴリズムを以下に示す。

- 損失関数：交差エントロピー誤差 ('nn.CrossEntropyLoss()')
- 最適化アルゴリズム：Adam('optim.Adam()')
- 学習率：0.001('lr=0.001')

演習課題

課題 1. 単層のパーセプトロンでは *XOR* を作成することができない。その理由について考察せよ。

課題 2. $f(x) = \frac{1}{1+e^{-2x}}$ と $g(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ について、以下の等式が成り立つことを示せ。ここで、e は定数 、特にネイピア数であることに注意せよ。

$$g(x) = 2f(x) - 1$$

また、値域とグラフの形状に着目することによって、シグモイド関数とハイパボリックタンジェント関数の関係性について考察せよ。

課題* 3. $f(x) = \frac{1}{1+e^{-x}}$ について、以下の等式が成り立つことを示せ。

$$f'(x) = f(x)(1 - f(x))$$

また、誤差逆伝播法の計算効率にどのように寄与するかに注目することによって、シグモイド関数の微分の特性について考察せよ。

課題* 4. 以下のサイトは、ニューラルネットワークの可視化を行える。これを利用してみて、さらに気づいたことがあれば述べよ。

TensorFlow Playground (https://playground.tensorflow.org/)

課題 5. 自分で適当なタスクを設定し、それに適切と考えられる損失関数・最適化アルゴリズムを挙げよ。また、その理由も述べよ。

※課題*に関しては任意とする。

第 II 部
実装

ここからは、配布した Google colaboratory も参照しながら進めていく。ダウンロードは、こちらから行える。(https://drive.google.com/file/d/1cp4LDkTFwL-YmslWVxnCdY-7OPGyUPfA/view?usp=sharing)

6　機械学習ライブラリ

6.1　ライブラリとは何か

機械学習を実装する際、大抵はゼロからコードを書くよりも、事前に定義された関数やクラス、手続きをまとめた「ライブラリ」というコードの集合を利用する。ライブラリは、繁雑なプログラムを効率的に実装するためのツールと言える。以下、機械学習に関して代表的なライブラリのいくつかを紹介する。

6.2　代表的なライブラリ

scikit-learn

Scikit-Learn は 2007 年から存在する機械学習ライブラリで、サポートベクターマシンやランダムフォレストなど、深層学習以外の技術に特化している。データセットの取り扱いやデータの前処理関数などが充実しており、他のライブラリと併用されることも多い。

Pytorch

Pytorch は Facebook・META が開発したライブラリ。かつての Chainer と統合され、特に研究者に好まれる傾向にある。モデル定義や学習ループを自分で設計する必要があるため、柔軟性が求められる場面での利用が多い。推論速度は TensorFlow に比べやや遅いが、ONNX などの高速な実行環境でモデルを実行することも一般的である。

TensorFlow と Keras

TensorFlow は Google が中心となって開発した機械学習ライブラリ。一方、Keras はユーザーフレンドリーな API を提供し、TensorFlow をバックエンドとして利用することで、簡単にモデルを設計・学習することができる。もともとは独立していたが、現在はほぼセットで語られることが多い。Pytorch の Define by Run 方式とは異なる、Define and Run 方式が主流であったが、現在はどちらの方式もサポートされている。TensorFlow は、WEB 技術向けの TensorFlow.js や組み込み、スマートフォン向けの TensorFlowLite も提供しており、実務での利用に適している。

JAX・FLAX

JAX・FLAX は新興のライブラリで、Google が主導して開発している。関数型プログラミングの考え方を採用しており、高速な推論が可能である。

7　モデルの作成

7.1　関数の紹介

ニューラルネットワークの実装を重要な三つの関数を中心に確認していく。Pytorch において、モデルはオブジェクトとして記述される。Pytoch では、モデルのクラスを定義するためには、torch.nn.model を継承する。クラスの定義には、他に、forward、___init___メソッドが必須となる。また、継承元のクラスにはよく使うメソッドとして eval,train というメソッドも用意されている。それぞれの役割を説明する。

7.1.1　___init___

___init___メソッドは、モデルのインスタンスを作るときに呼び出されるコンストラクタである。モデルが持つ内部の状態変数などを生成する。このメソッド内で、必要とする層を用意する。

7.1.2　forward

forward メソッドは、モデルの推論時に呼び出される。このメソッド内で、パラメータとして渡されたデータがどのように層を通るか、データの流れを記述する。

7.1.3　eval,train

これらのメソッドは親クラス内で定義されている。eval は推論のモードに、train は学習のモードに、モデルを設定するためのメソッドである。

7.2　実装

これらを用いてモデルを定義するコードを以下の Listing1〜3 に示す。

Listing 1: モデル定義

```
1  class Net(nn.Module):
2      def __init__(self):
3          super().__init__()
4          self.l01 = nn.Linear(28*28, 100)
5          self.l12 = nn.Linear(100, 50)
6          self.l23 = nn.Linear(50, 10)
7
8      def forward(self, x):
9          x = torch.reshape(x, (-1, 28 * 28))
10         x = F.relu(self.l01(x))
11         x = F.relu(self.l12(x))
12         x = nn.Softmax()(self.l23(x))
13         return x
```

Listing 2: 学習ループ

```
num_epochs = 3

model.train()
for epoch in range(num_epochs):
    for i, data in enumerate(trainloader):
        # inputs: ([batch_size, 1, 28, 28]),   labels: ([batch_size])
        inputs, labels = data
        inputs, labels = inputs.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
```

Listing 3: その他の準備

```
  device = torch.device("cuda")
  #device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
  print(f'device: {device}')

  transform = transforms.Compose([transforms.ToTensor(), transforms.
      Normalize((0.5,), (0.5,))])
  trainset = MNIST(root='./data', train=True, download=True, transform=
      transform)
  trainloader = DataLoader(trainset, batch_size=64, shuffle=True)

  model = Net().to(device)

  criterion = nn.CrossEntropyLoss()
  optimizer = optim.Adam(model.parameters(), lr=0.001)
```

これらはそれぞれ、モデルの定義、学習ループの設定、その他細かい設定 (デバイス、最適化アルゴリズムなど) を行っている。特に、紹介した関数が用いられているのは一つ目と二つ目のコードである。これらのコードの具体的な説明は、colaboratory を参照せよ。

8　モデルの評価と試用

モデルの学習まで完了したので、次は実際にモデルに手書き文字を予測させてその精度を調べる。そのコードを以下の Listing4〜5 に示す。

Listing 4: モデルの評価

```
testset = MNIST(root='./data', train=False, download=True, transform=
    transform)
testloader = DataLoader(testset, batch_size=64, shuffle=False)

model.eval()
with torch.no_grad():
    correct = 0
    total = 0
    for images, labels in testloader:
        images, labels = images.to(device), labels.to(device)

        # torch.Size([batch_size, 10])
        outputs = model(images)

        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)

        correct += (predicted == labels).sum().item()

print(f'Accuracy: {100 * correct / total}%')
```

Listing 5: モデルの試用

```
import matplotlib.pyplot as plt
import numpy as np

num_images = 100

images_set = MNIST('data/', train=False, download=True, transform=
    transform)
images_loader = DataLoader(images_set, batch_size=1, shuffle=True)


model.eval()
```

```
11
12
13  for index, (img, label) in enumerate(images_loader):
14      if index < num_images:
15          img, label = img.to(device), label.to(device)
16
17          pred = model(img)
18          pred = pred.squeeze(0).to("cpu").detach().numpy()
19          pred = np.argmax(pred)
20
21          # パターン1. 全ての正解ラベル，推論ラベル，画像の表示
22          print(f'correct␣label:␣{label}')
23          print(f'predicted␣label:␣{pred}')
24
25          img = torchvision.transforms.functional.to_pil_image(img[0])
26
27          plt.imshow(img, cmap='gray')
28          plt.show()
29
30          # パターン2. 推論結果が誤っているものの正解ラベル，推論ラベル，画
                像の表示
31          if label != pred:
32              print(f'correct␣label:␣{label}')
33              print(f'predicted␣label:␣{pred}')
34
35              img = torchvision.transforms.functional.to_pil_image(img[0])
36              plt.imshow(img, cmap='gray')
37              plt.show()
```

これらはそれぞれ、モデルの評価、モデルの試用である。これらのコードの具体的な説明は、colaboratoryを参照せよ。

演習課題

課題 6. 配布された *Google colaboratory* を実行し、その結果について記録・考察せよ。

課題 7. 配布された *Google colaboratory* のニューラルネットワークの深さを変更して再度実行し、その結果について記録・考察せよ。特に、ニューラルネットワークを深くしていった場合の結果について考察せよ。

課題 8. 配布された *Google colaboratory* のニューラルネットワークの活性化関数を変更して再度実行し、その結果について記録・考察せよ。

課題 9. 配布された *Google colaboratory* のデータセットを変更して再度実行し、その結果について記録・考察せよ。

変更するデータセットの一例として、*FashionMNIST(*ファッション画像版の *MNIST)* が挙げられる。これは *"torchvision.datasets.FashionMNIST"* から得られる。その他、必要なことに関しては各自で調べよ。

課題* 10. $\sin x$ や $x^2 + 5x$ などの関数や、その他のデータセットに関してニューラルネットワークを構築し、実行せよ。その結果について、記録・考察せよ。

※課題*に関しては任意とする。

参考文献

[1] 斎藤 康毅. **ゼロから作る** *Deep Learning1*. オライリージャパン, 2016.
[2] 斎藤 康毅. **ゼロから作る** *Deep Learning2*. オライリージャパン, 2018.