0 から作る DeepLearning

花崎 諒

2020 年 1 月 22 日

概要

オライリージャパンより出版された「0 から作る DeepLearning」を自己流で要約しました、。

1　はじめに

本文書は茨城大学工学部の同好会「IBAAI」の勉強会を進めるにあたり、作成するに至った。

2　パーセプトロン

1. パーセプトロンとは
   ここで言うパーセプトロンとは「人口ニューロン」「単純パーセプトロン」と呼ばれるものを指す。パーセプトロンはニューロン、ノードからなり、また、ノードには重み w が存在する。以下に 2 入力のパーセプトロンの例を示す。

   パーセプトロンは、複数の信号を入力として受け取り、1 つの信号を出力する。本書では入力は x, 出力は y で表されている。情報が入力された際、信号は 0 または 1 の値を取り、それぞれ以下のようになる。

出力値	信号の状態
0	信号を流さない
1	信号を流す

2. バイアス
   バイアスはパーセプトロンの発火のしやすさを表す。例えば、バイアスが-20 なら各入力の重み付き和が 20 を上回れば発火し、そうでなければ発火しないようになっている。以下にバイアス b を加味した式を示す。

$$y = \begin{cases} 1 & (x \leq 1 \text{ のとき}) \\ 0 & (x > 1 \text{ のとき}) \end{cases} \tag{1}$$

   また、バイアスを加えたパーセプトロンの図を以下に示す。バイアスを加えたず p 42

3. 多層パーセプトロンパーセプトロンは単層では、線形な形でしか領域を分割できない (AND, NAND,OR)。しかし、多層にすることによって、非線形な形でも領域を分割することができる (XOR)。

3　ニューラルネットワーク (NN)

1. ニューラルネットワーク (NN) とは

   前述のパーセプトロンの計算では、重みを用いていた。しかし、パーセプトロンの重みを決定する作業は人間が行わなければならない。これは不便である。一方、NN は適切な重みをデータから自動で学習できるという性質を持っている。NN は基本的にはパーセプトロンと似た形をとるが、入力層、中間層 (隠れ層)、出力層の 3 つの層からなる。

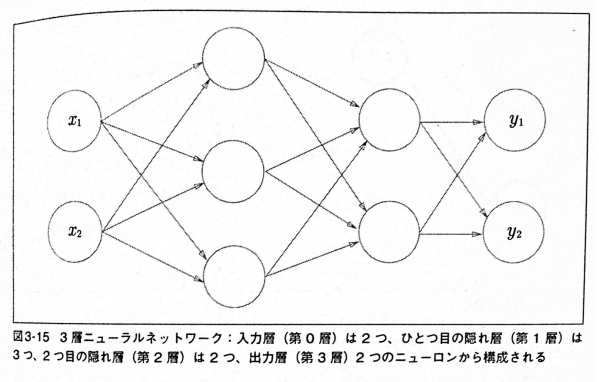

図 1　ニューラルネットワーク (0 から作る DeepLearning より抜粋)

2. 活性化関数

   活性化関数は閾値を境にして入力が切り替わる関数で、「ステップ関数」や「シグモイド関数」などが存在する。パーセプトロンはステップ関数を活性化関数として使用しており、本書では $h(x)$ と表現している。活性化関数も含めてパーセプトロンを式で表現すると以下のようになる。

$$a = b + w_1 x_1 + w_2 x_2 \tag{2}$$

$$y = h(a) \tag{3}$$

   NN ではシグモイド関数や ReLU 関数などの活性化関数を用いる。

   - シグモイド関数

$$h\left(x\right) = \frac{1}{1 + exp\left(-x\right)} \tag{4}$$

   - ReLu 関数

$$h(x) = \begin{cases} x\ (x > 0) \\ 0\ (x \leq 0) \end{cases} \tag{5}$$

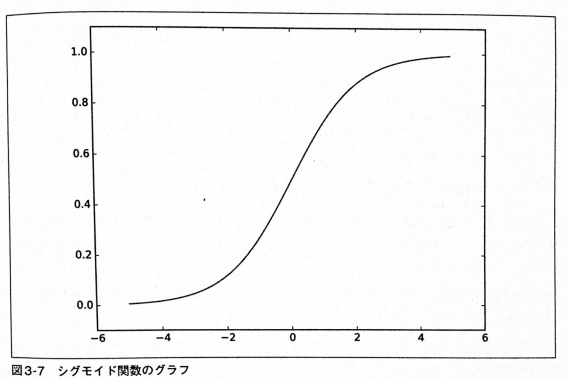

図 2　シグモイド関数 (0 から作る DeepLearning より抜粋)

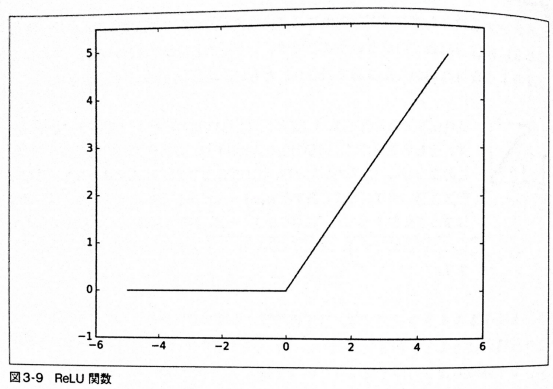

図 3　ReLu 関数 (0 から作る DeepLearning より抜粋)

3. 各層における信号の伝達

NN での入力層から出力層までの信号の伝達の様子を示す。x は入力層の値、a は中間層の値、w は重み、y は出力層の値となっている。さらに、上付き文字、下付き文字は以下のような意味とする。

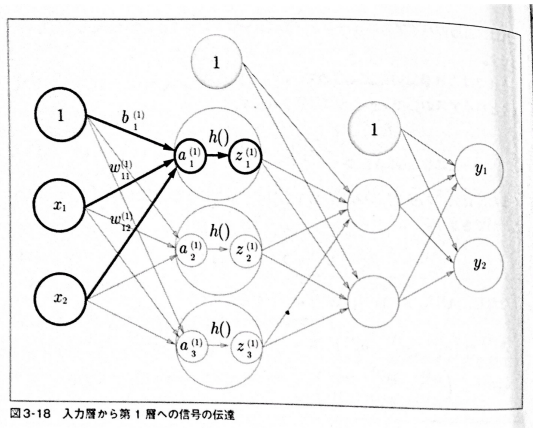

図 4　入力層から 1 層目への信号の出力 (0 から作る DeepLearning より抜粋)

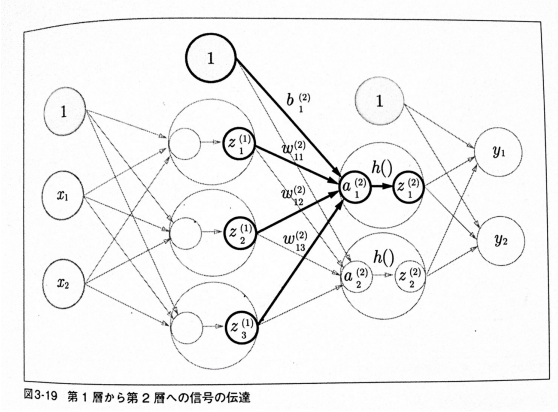

図 5　1 層目から 2 層目への信号の出力 (0 から作る DeepLearning より抜粋)

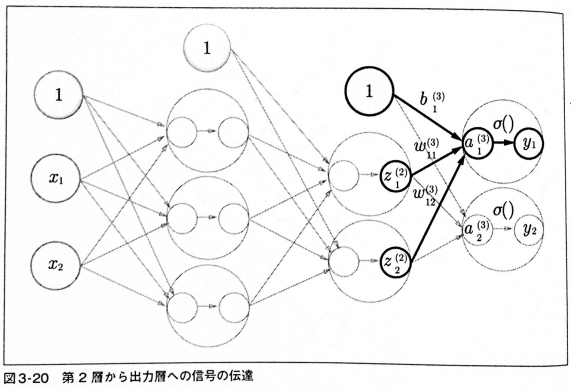

図 6　2 層目から出力層への信号の出力 (0 から作る DeepLearning より抜粋)

4. 出力層の設計

NN は分類問題と回帰問題の両方に用いることができる。回帰問題に用いるときは恒等関数、分類問題ではソフトマックス関数を用いる。

- 恒等関数

  恒等関数は入力をそのまま出力する、すなわち入ってきたものをそのまま出力する関数である。
  よって、恒等関数によって変換されるプロセスは、以下の図のような形になる。

- ソフトマックス関数

  ソフトマックス関数は以下の式となる。

$$y_k = \frac{exp\,(a_k)}{\sum_{i=1}^{n} exp\,(a_i)} \tag{6}$$

  ソフトマックス関数で変換されるプロセスは以下の図ような形になる。図で示されたように、ソフトマックス関数の出力はすべての入力信号からの結びつきがある。つまり、出力の各ニューロンがすべての入力信号から影響を受けることになる。

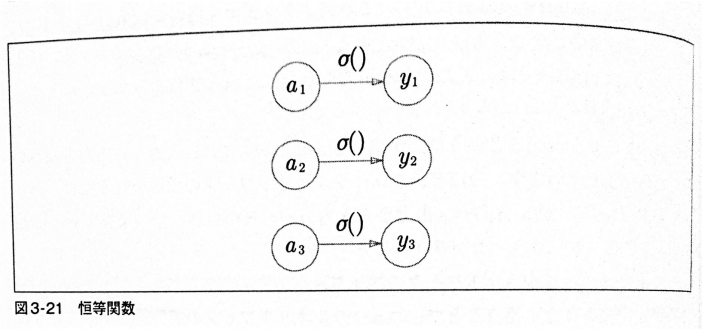

図 7　恒等関数 (0 から作る DeepLearning より抜粋)

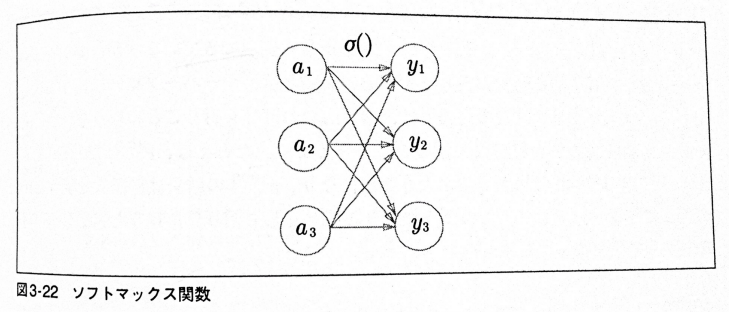

図 8　ソフトマックス関数 (0 から作る DeepLearning より抜粋)

4　NN の学習

1. NN の学習、その利点とは

   NN の学習とは、訓練データから最適な重みパラメータの値を自動で獲得することを指す。ディープラーニングでは、人間の手をこれまでの機械学習以上に介さず学習を行うことができる。また「認識する対象物がどういったものなのか」などのような問題の詳細とは関係なく、与えられたデータをひたすら学習するので、どの問題も同じ流れで解くことができるという利点がある。

2. 訓練データとテストデータ

   機械学習の問題では、訓練データとテストデータの 2 つに分けて学習や実験を行う。訓練データを用いて学習を行い、テストデータでその学習したモデルの性能を評価する。このような方法をとるのは、我々が評価したいものがモデルの汎化能力であるためである。汎化能力とは、訓練データに含まれていないデータに対する能力である。汎化能力を高めるためには、様々なデータセットを学習させる必要がある。

   学習したモデルがある一つのデータセットに過度に対応してしまった状態を過学習という。

3. 損失関数

- 2 乗和誤差

  2 乗和誤差は以下のように表される。

$$E = \frac{1}{2}\sum_{k}(y_k - t_k)^2 \tag{7}$$

  ここで、y_k はニューラルネットワークの出力、t_k は教師データ、k はデータの次元数を表している。

- 交差エントロピー誤差

  交差エントロピー誤差は以下のように表される。

$$E = -\sum_{k} t_k \, log \, y_k \tag{8}$$

  ここで、log は底が e の自然対数を表す。y_k はニューラルネットワークの出力、t_k は正解ラベル、k はデータの次元数を表している。また、t_k は正解ラベルとなるインデックスだけが 1 で、その他は 0 であるとする。すなわち、正解ラベルが 1 に対応する出力の自然対数を計算すればよい。

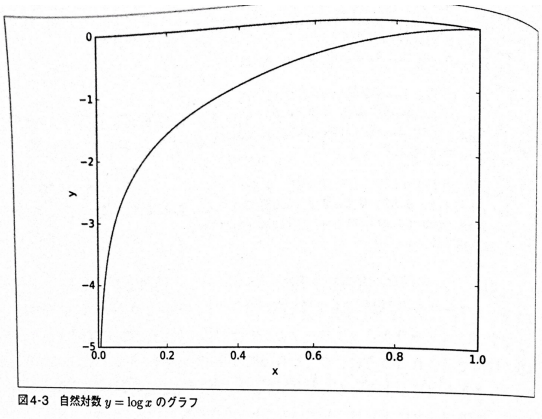

図 9　$y = logx$ のグラフ (0 から作る DeepLearning より抜粋)

4. ミニバッチ学習

機械学習の学習は、訓練データに対する損失関数を求め、その値をできるだけ小さくするようなパラメータを探し出すという形で進んでいく。そのため、損失関数はすべての訓練データを対象として求めなければならない。しかし、データの数が数千万、数億となった時、すべてのデータを対象にしていては非常に時間がかかってしまう。そこで、データの中からランダムに一部を選び出し、その一部を全体の「近似」として学習に利用する方法がとられている。この選び出されたデータセットを「ミニバッチ」といい、そのミニバッチを用いる学習を「ミニバッチ学習」という。

5. なぜ損失関数を設定するのか

   なぜ損失関数などというものを導入するのか。目的は認識精度の高い NN を作成することなのだから、認識精度を指標にすればよいではないか、という意見もあるだろう。しかし、損失関数の導入には理由がある。

   これに対する答えは、NN の学習における微分の役割に注目すると解決する。次節に詳しい説明を記述するが、NN の学習では、最適なパラメータ (重みとバイアス) を探索する際、損失関数の値ができる限り小さくなるようなパラメータを探索する。ここで、パラメータの部分 (正確には勾配) を計算し、その微分の値を手掛かりにパラメータの値を徐々に更新していく。

   例えば、ある NN が存在したときに、ある一つの重みパラメータに注目するとする。このとき、一つの重みパラメータの損失関数に対する微分は、その重みパラメータの値を少しだけ変化させたときに、損失関数がどのように変化するか、ということを表す。もし、その微分値が負であれば、その重みパラメータを正の方向へ変化させることで、損失関数を減少させることができる。逆に微分値が正だった場合、その重みパラメータを負の方向へ変化させることで、損失関数を減少させることができる。

   しかし、微分値が 0 になってしまうと、重みパラメータをどちらに動かしても損失関数の値が変わらないため、その重みパラメータの更新がストップしてしまう。

   ここで、認識精度を指標とした場合について考える。ある NN が存在し、現在 100 枚ある訓練データの中で、32 枚を正しく認識できているとする。すなわち認識率は 32% である。もし認識精度を指標とした場合、重みパラメータを少し変化させただけでは認識精度は 32% のまま変化しない。つまり、パラメータの少しの調整では、認識精度は改善されず一定のままなのである。もし認識精度が改善されたとしても、その値は 32.0123…% のような連続な値ではなく、33% や 34% のような離散的な値になってしまう。一方損失関数を指標とした場合、現在の損失関数の値は 0.92534…% のような値によってあらわされる。そしてパラメータの値を少し変化させると、それに反応して損失関数も 0.93232…% のように連続的に変化するのである。

   認識精度はパラメータの微小な変化には反応せず、反応があるにしてもその値は不連続にいきなり変化する。これは、活性化関数のステップ関数と同様の形である。つまり、認識精度を指標にすると、パラメータの微分値がほとんどのところで 0 になってしまうである。

   また、活性化関数においても同様のことが言える。例えば活性化関数にステップ関数を用いると、ステップ関数の微分値が 0 になってしまうため、たとえ損失関数を指標に用いたとしても、パラメータの微小な変化は抹殺されてしまい、損失巻子の値は何の変化も示さなくなってしまう。なので、連続的な値を取り、微分値が 0 を示さないシグモイド関数を活性化関数として用いるのである。

   p97

6. 勾配

   勾配は、数式のすべての変数を偏微分してベクトルとしてまとめたものを指す。

5 誤差逆伝播法

1.
2.

3.
4.

6　学習のテクニック

1. パラメータの更新

   NN の学習の目的は、損失関数の値をできるだけ小さくするパラメータを見つけること。つまりは最適なパラメータを見つける問題、最適化問題といえる。

   これまでに本書では、パラメータの勾配 (微分) を利用して勾配方向にパラメータを更新するというステップを何度も繰り返すことによって最適なパラメータを導く手法、確率的勾配降下法 (SGD) という手法で最適化を行ってきた。

   しかし、状況に応じてこの SGD よりも適した手法が存在するため、本章では SGD についてやそのほかの手法について紹介する。

2. SGD

   SGD は以下のような式で表すことができる。p167

   ここで、更新する重みパラメータを W、W に関する損失関数の勾配を 微分 とする。 n は学習係数を表し、実際には 0.01 や 0.001 などの小さな値を設定しておく。また、式中の←は右辺の値で左辺の値を更新するということを示す。この式に表されているように、SGD は勾配方向へある一定の距離だけ進むという単純な手法である。

   しかし、先に触れたように、SGD では非効率な問題もある。例として以下のような関数の最小値を求める問題を考える。

   p168

   この関数は、以下の図のような形状、等高線をしている。

   p168

   この関数の勾配を図で表すと次のようになる。

   p169

   この勾配は y 軸方向には大きく、x 軸方向には非常に小さくなっている。言い換えれば、y 軸方向は急な傾斜になっているが、x 軸方向は緩やかな傾斜となっているということである。また、ここでの注意点は、式の最小値の場所は $(x, y) = (0, 0)$ であるが、図で表される勾配は、多くの場所で $(0, 0)$ の方向を指し示さないということである。

   この関数に対して SGD を適用する。初期値は $(x, y) = (-7.0, 2.0)$ として開始する。結果は次の図のようなジグザグな遷移となり、かなり非効率な結果となった。つまり、SGD は関数の形状が等方向でないと (伸びた形の関数だと) 非効率な経路で探索を進めることになってしまう。そこで、SGD のように単に勾配方向に進んで探索を進めるよりもスマートな方法が必要となる。なお、SGD の非効率な探索経路の根本的な原因は、勾配の方向が孫頼の最小値ではない方向を指していることに起因する。

3. Momentum

   Momentum とは運動量という意味。この手法は以下の式で表される。

p170

前の SGD と同じく、W は更新する重みパラメータ、微分は W に関する損失関数の勾配、n は学習係数を表す。ここで、新たに登場した v という変数は、物理で言うところの速度を示す。式では物体が勾配方向に力を受け、その力によって物体の速度が加算されるという物理法則を表している。この Momentum のイメージは次の図のようになっている。

p171

また、式には $\alpha\ v$ という項が存在するが、この項は物体が何も力を受けない場合に徐々に減速するための役割を担う (α には 0.9 などの値を設定する)。物理では、地面の抵抗や空気抵抗に相当する。

Momentum を使うと、式 6.2 は次の図のように最適化されていく。図の通り、更新経路は滑らかになっており、ジグザグ度合いが軽減されている。これは、x 方向に受ける力はとても小さいが、常に同じ方向の力を受けるため、同じ方向へ一定して加速することになるためである。逆に、y 軸方向へ受ける力は大きいが、正と負の力を交互にうけるため、それらが互いに打ち消し合い、y 軸方向の速度は安定しない。これらのことにより、SGD の場合と比べて x 軸方向に早く近づくことができ、ジグザグの動きを軽減することができたのである。

4. AdaGrad

ニューラルネットワークの学習では、学習係数 n の値が重要になる。学習係数が小さすぎると学習に時間がかかりすぎてしまい、逆に大きすぎると発散して正しい学習が行えなくなってしまう。

この学習に関するテクニックとして、学習係数を減衰させる、すなわち初めは大きな学習係数で、次第にその値を小さくしていくという方法もある。

学習係数を徐々に小さくしていくというアイデアは、パラメータ全体の学習係数の値を一括して下げることに相当する。これをさらに発展させたのが AdaGrad という手法である。AdaGrad は、一つ一つのパラメータに対して、「オーダーメイド」の値を設定する。

AdaGrad は、パラメータの要素ごとに適応的に学習係数を調整強いながら学習を行う手法である (AdaGrad の Ada は適応的を意味する Adaptive に由来する)。AdaGrad の更新方法は以下の式で表される。

p173

前の SGD と同じく、W は更新する重みパラメータ、微分は W に関する損失関数の勾配、 n は学習係数を表す。ここで新たに登場した変数 h は、式で示されるように、これまで経験した勾配の値を 2 乗和として保持する (式の〇は行列の要素ごとの掛け算を意味する)。そして、パラメータ更新の際に、1/ルート h を乗算することで、学習のスケールを調整する。これは、パラメータの要素の中で大きく更新された要素は、学習係数が小さくなることを意味する。つまり、大きく更新されたパラメータの学習係数は次第に小さくなる、という学習係数の減衰をパラメータごとに行うことができるのである。

AdaGrad を使うと、式は以下の図のように最適化されていく。

図を見てみると、最小値に向かって効率的に動いているのが分かる。y 軸方向へは勾配が大きいため、最初は大きく動くが、その大きな動きに比例して、更新のステップが小さくなるように調整が行われている。そのため、y 軸方向への更新度合いは弱められていき、ジグザグの動きが軽減される。

5.

6.

7.

7 畳み込み NN

1.
2.
3.
4.

8 DeepLearning

1.
2.
3.
4.