深層学習入門

#8 損失関数とミニバッチ学習法

たくのろじぃ / takunology

損失関数

- ラベル（教師データ）と出力の誤差を求めるための関数
- 値が大きいほどニューラルネットワークの性能が悪い
- 誤差が小さくなるようにしてパラメータを調整していく → 学習フェーズ

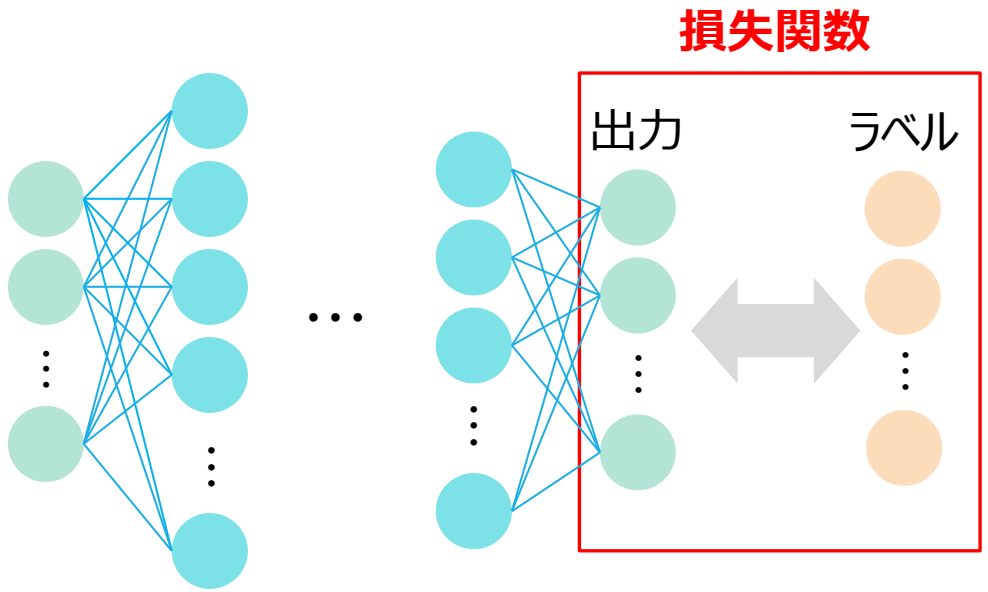
二乗和誤差

交差エントロピー誤差

損失計算

ミニバッチ学習

訓練データを
無作為抽出



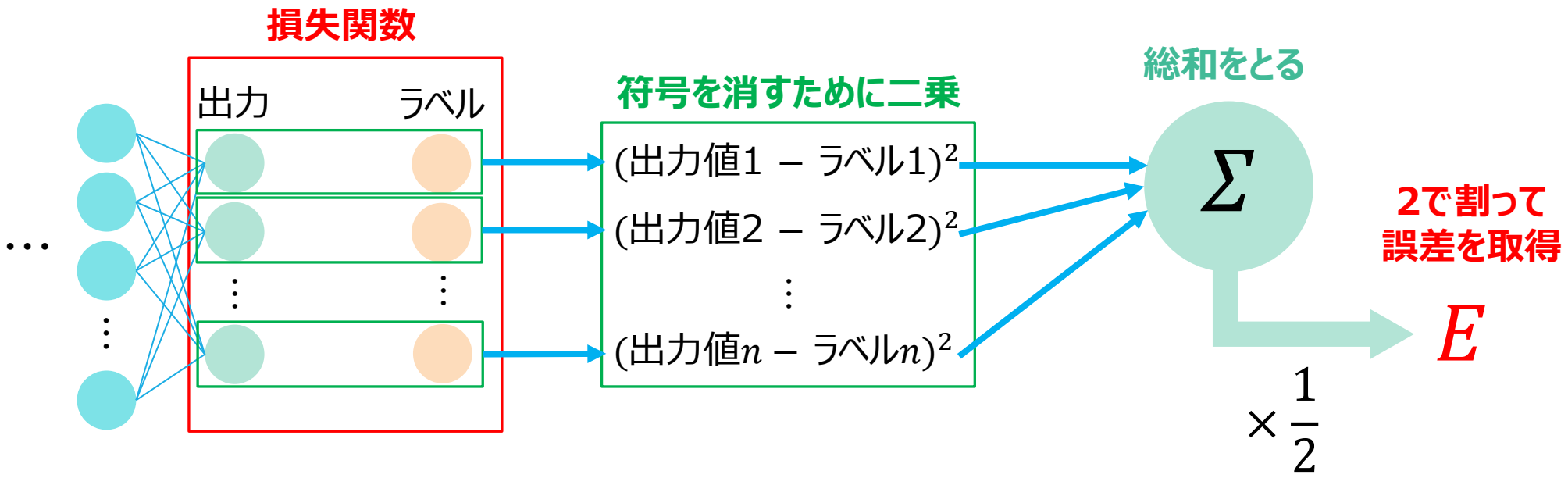
二乗和誤差

教師データは正しい要素につき確率100% なので 1 とみなす

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

y_k：出力層からの出力 k番目

t_k：教師データの値（確率＝1）



ハンズオン ＃1 二乗和誤差の実装

- .NET Core 3.0 コンソールアプリケーションで実装
- 作成した3層ニューラルネットワークの出力層の後につながるように書く
- 出力層の数にあわせて、任意の要素 k を教師データの正解（＝1）とする

2乗計算は Math.Pow() メソッドを
使って実装可能

```
static int Answer = 0; // 正解の要素を選択するための変数
static double[] AnsLabel = new double[0];

public CheckLabel()
{
UG

    for(int i = 0; i < OutputMatrix.GetLength(1); i++)
    {
        Answer = i; //一通りの正解ラベルで比較
        AnsLabel = new double[OutputMatrix.GetLength(1)]; //ラベルの要素 -> 正解の要素は1で初期化
        AnsLabel[Answer] = 1; //正解ラベルは確率100% (1)とみなす
        MSE();
    }
```

```
//二乗和誤差 Mean Squared Error
public void MSE()
{
    double sum = 0;
    for (int i = 0; i < OutputMatrix.GetLength(1); i++)
    {
        // (出力値 - ラベル)^2 を sum に加算していく (総和)
        sum += Math.Pow(OutputMatrix[0, i] - AnsLabel[i], 2);
    }
    ErrorValue = sum / 2; //全体を2で割る
    Console.WriteLine($"正解ラベル : {Answer} => 損失度：{ErrorValue}");
}
```

二乗和誤差の結果を見る

- 正解ラベルとその要素の出力データを併せてみる
- 正解を選ぶ確率が高いほど誤差が小さい → 0 ～ 1 の値で評価

```
----- 出力層の出力 -----
|0.015513037576213104 0.05547968887412087 0.0029159594863410144 0.9161704473407734 0.00992086672255146 |

出力層の確率合計(DEBUG)：0.9999999999999998

正解ラベル ： 0 => 損失度：0.9058838950284147
正解ラベル ： 1 => 損失度：0.8659172437305069
正解ラベル ： 2 => 損失度：0.9184809731182868
正解ラベル ： 3 => 損失度：0.005226485263854458
正解ラベル ： 4 => 損失度：0.9114760658820764
```

正解が 0 のとき、これを 1.55%の確率で選んだ場合の誤差は 0.905 ほど

正解が 3 のとき、これを 91.61%の確率で選んだ場合の誤差は 0.005 ほど

誤差が大きい → 正解を選ぶ確率が低い → ニューラルネットワークの性能が悪い

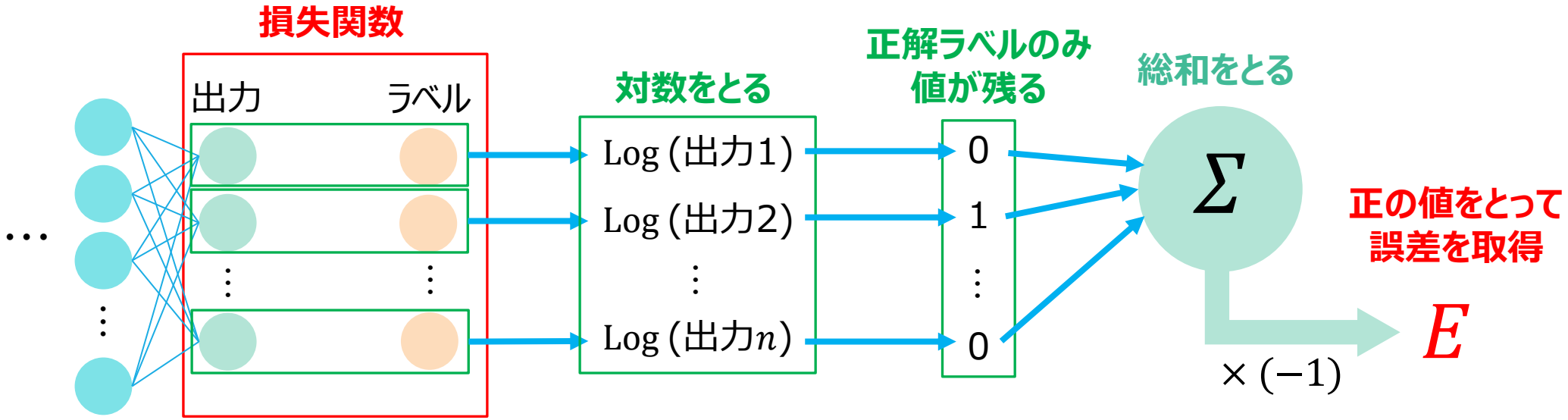
交差エントロピー誤差

Log の底は自然対数

$$E = -\sum_k t_k \log y_k$$

y_k：出力層からの出力 k番目

t_k：教師データの値（確率＝1）



ハンズオン ＃2 交差エントロピー誤差の実装

- .NET Core 3.0 コンソールアプリケーションで実装
- 作成した3層ニューラルネットワークの出力層の後につながるように書く
- 出力層の数にあわせて、任意の要素 k を教師データの正解（＝1）とする

自然対数の計算は Math.Log () メソッドを使って実装可能

```
//交差エントロピー誤差 Cross Entropy Error
public void CEE()
{
    double sum = 0;
    for(int i = 0; i < OutputMatrix.GetLength(1); i++)
    {
        sum += AnsLabel[i] * Math.Log(OutputMatrix[0, i]);
    }
    ErrorValue = sum * (-1);
    Console.WriteLine($"正解ラベル : {Answer} => 損失度：{ErrorValue}");
}
```

損失関数2つ実装したソースコード

https://github.com/takunology/DeepLearning/tree/master/NeuralNetwork/3NN_FO_SM_MSE

交差エントロピー誤差の結果を見る

- 正解ラベルとその要素の出力データを併せてみる
- 正解を選ぶ確率が高いほど誤差が小さい → 二乗和誤差と同じだが変化が大きく出る

```
----- 出力層の出力 -----
|0.08203981804810548 0.048850305090435 9 0.016447809450665107 0.03205109243885066 0.8206109749719429 |

出力層の確率合計(DEBUG) : 1

正解ラベル : 0 => 損失度 : 2.500550563624085
正解ラベル : 1 => 損失度 : 3.0189946550586844
正解ラベル : 2 => 損失度 : 4.107562974731188
正解ラベル : 3 => 損失度 : 3.440424010740795
正解ラベル : 4 => 損失度 : 0.19770612473745355
```

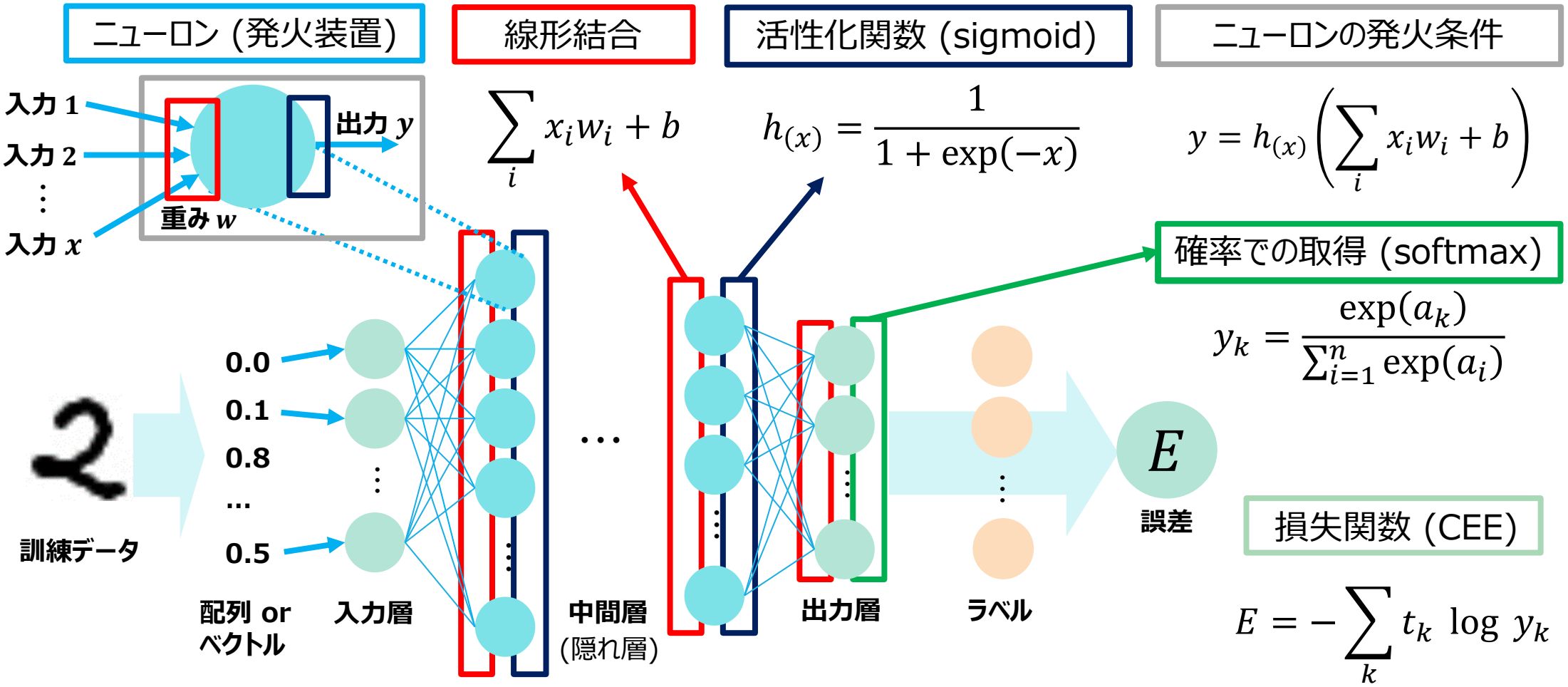
正解が 4 のとき、これを 1.644%の確率で選んだ場合の誤差は 4.107 ほど

正解が 3 のとき、これを 82.06%の確率で選んだ場合の誤差は 0.197 ほど

誤差が大きい → 正解を選ぶ確率が低い → ニューラルネットワークの性能が悪い

これまでの内容を確認

- 学習なしのフォワード方向ニューラルネットワーク



ミニバッチ学習法

- 訓練データ n 個に対する誤差の和を求め、正規化することで一定の誤差を得られる
- 例えばデータが1万個あった場合はランダムに100個ごとに取得して学習
- ここからは交差エントロピー誤差による手法を用いる

$$E = -\frac{1}{N}\sum_{n}\sum_{k} t_{nk} \log y_{nk}$$

$\boldsymbol{N}$ ： 訓練データの総数

$\boldsymbol{t_{nk}}$ ： 教師データにおける
n 個目のデータの k 番目の値

$\boldsymbol{y_{nk}}$ ： ニューラルネットワークの出力における 〃

正規化するための N

交差エントロピー誤差の式

それをデータ数 n 個分評価する

ランダム抽出は重みパラメータの時に使用した
Randクラスを用いる

ハンズオン ＃3 ミニバッチ学習法

- 訓練データ（手書き数字）を100個用意する （1万個用意して100個抽出したいけど大変なので）
- 手書き数字全てを配列に変換してプロジェクトに追加
- ランダムに入力データを10個選択し、それをミニバッチ学習で動かす
- 作った画像は輝度データ(.txt など)にしておく


```
訓練データ 37.txt を読み込みました。
8回目

----- 出力層の出力 -----
|0.09710385064426041 0.051246747488633455 0.148

正解ラベル : 2 => 誤差 : 1.910004140779327

訓練データ 15.txt を読み込みました。
9回目

----- 出力層の出力 -----
|0.213257502549318 0.020507889130350187 0.18221

正解ラベル : 2 => 誤差 : 1.7025900003336734

訓練データ 32.txt を読み込みました。
10回目

----- 出力層の出力 -----
|0.07328871376023677 0.19801870204989558 0.0187

正解ラベル : 2 => 誤差 : 3.975941225044258


誤差合計 : 27.329325417533298
平均誤差 : 2.7329325417533297
```