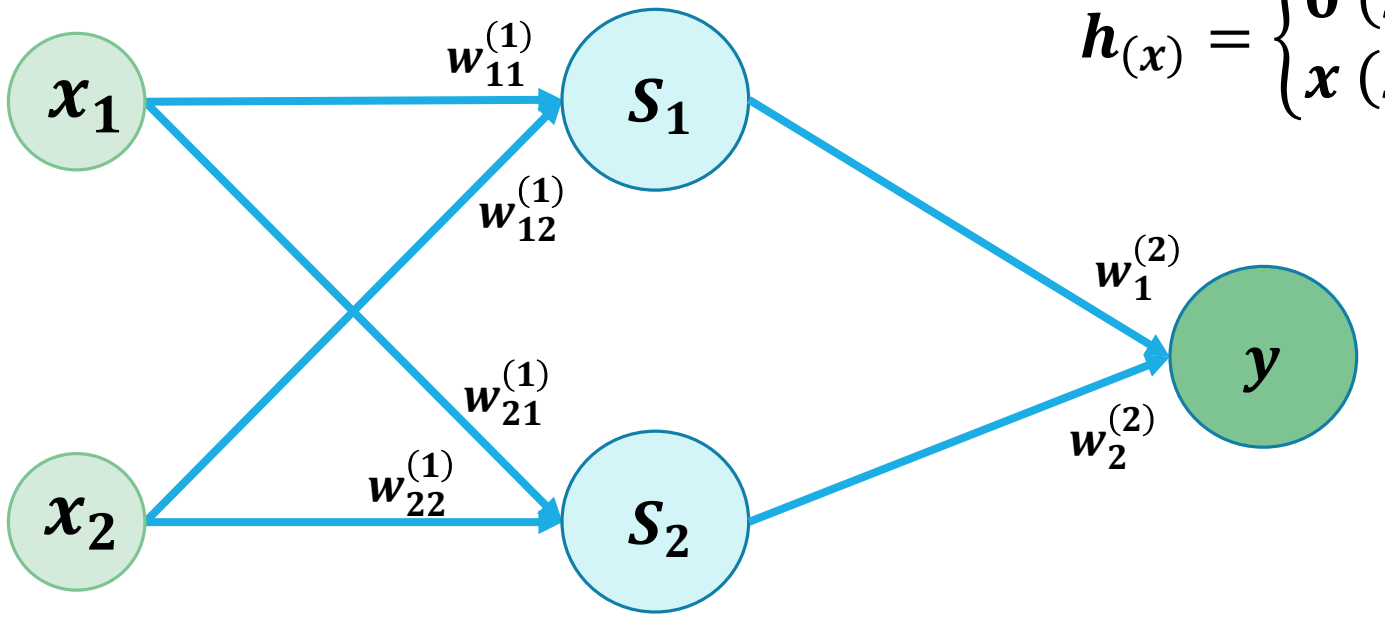
# 深層学習入門

## ＃5 ニューラルネットワークの実装 I

たくのろじぃ / takunology

# 2層ニューラルネットワークの実装

- 2入力1出力のニューラルネットワークの実装（下図）
- 重み（入力値）を乱数で初期化する
- 各層ではReLU関数の活性化関数を用い、ここではバイアスは考慮しない
- 学習なしのフォワード構成




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

# 浮動小数点数の乱数生成(C#)

- C#には乱数生成クラスがある
- 乱数の生成範囲は整数型かつ 0 以上でのみ指定可能 → Next (min, max) メソッド
- 少数、負の数を考慮した乱数生成にはちょっとした工夫が必要
- -2.0 ~ +2.0 までの乱数を得る例を考える

1. ランダムクラスのインスタンス生成
   Random rnd = new Random( );
2. メソッドから値を取得
   var value = rnd.NextDouble( );

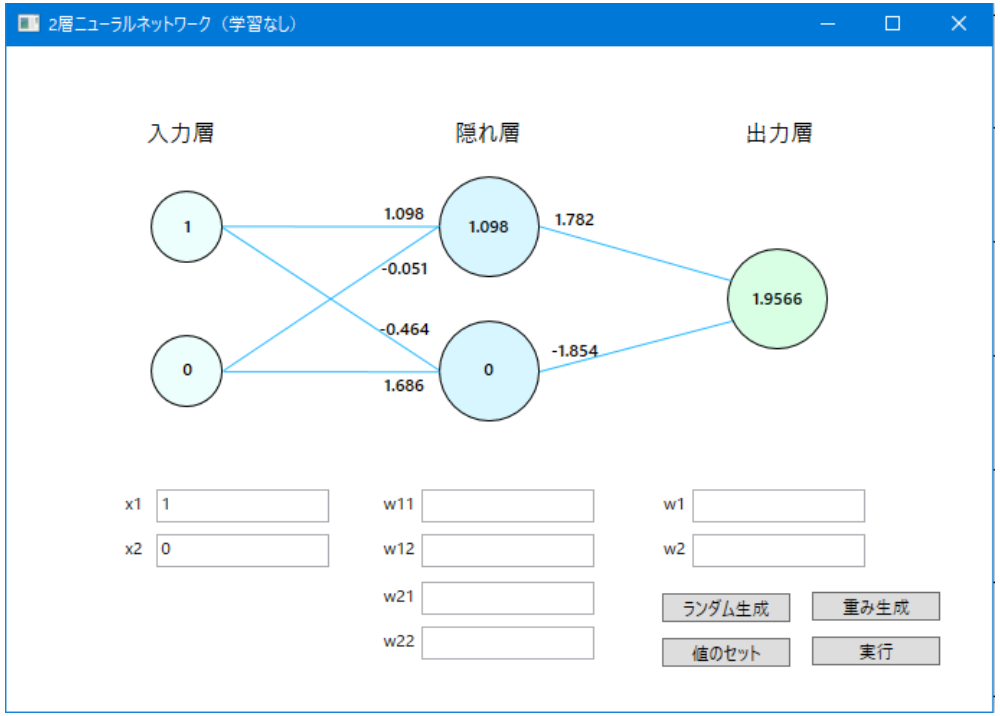
これでは 0 ~ 1.0 の範囲でしか
乱数を取得できない

重み付けの値は負の値もある

1. ランダムクラスのインスタンス生成
   Random rnd = new Random( );
2. メソッドから値を取得（整数）
   var tmp = rnd.Next(0, 4000 );
3. 基準は 2000 を 0 にしたいので -2000 する
   var value = tmp – 2000;
4. 整数を小数に変換する
   value = value / 1000;

# ハンズオン ＃1

- 2層ニューラルネットワークを作る
- プラットフォームは C# (WPFアプリケーション)
- XAMLにてニューロンの図を描いて視覚的に動きをみる



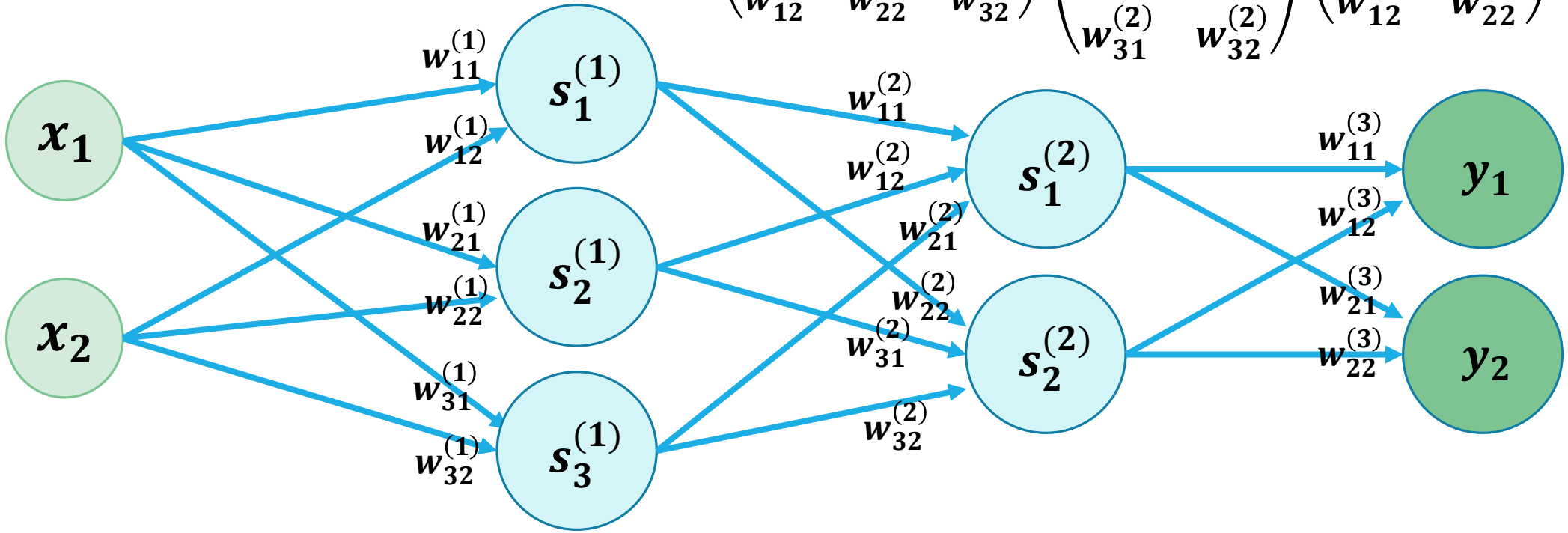

**２層ニューラルネットワークのプログラムはGithubにアップしています**

https://github.com/takunology/DeepLearning/tree/master/2NN_NotLearning

# 3層ニューラルネットワークの実装

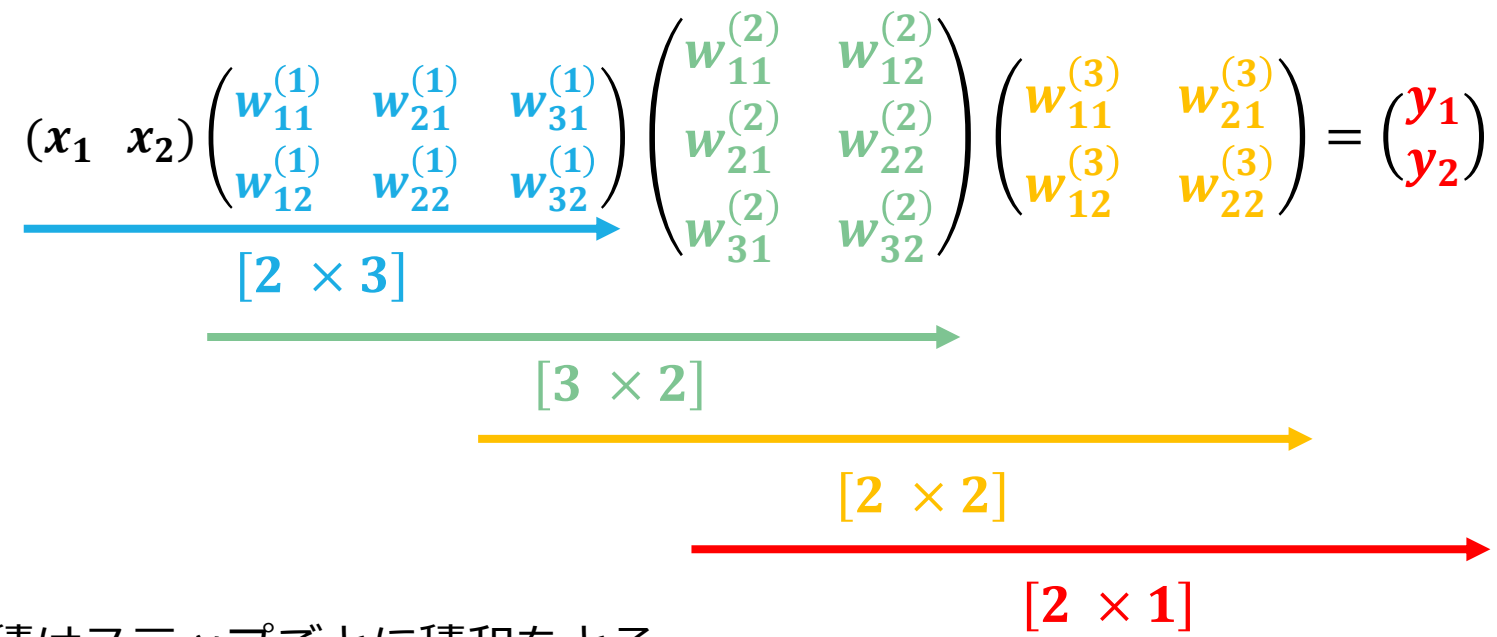
- 2入力2出力の３層ニューラルネットワークの実装 （下図）
- ここでは、各層でシグモイド関数の活性化関数を用いる
- 計算は行列の積を用いる → $[1 \times 2]\,[2 \times n]\,[n \times 1][1 \times 2]$
- 学習無しのフォワード構成

$$(x_1 \quad x_2)\begin{pmatrix} w_{11}^{(1)} & w_{21}^{(1)} & w_{31}^{(1)} \\ w_{12}^{(1)} & w_{22}^{(1)} & w_{32}^{(1)} \end{pmatrix}\begin{pmatrix} w_{11}^{(2)} & w_{12}^{(2)} \\ w_{21}^{(2)} & w_{22}^{(2)} \\ w_{31}^{(2)} & w_{32}^{(2)} \end{pmatrix}\begin{pmatrix} w_{11}^{(3)} & w_{21}^{(3)} \\ w_{12}^{(3)} & w_{22}^{(3)} \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$$



# 行列の積（複数）

- 行列A, Bの積を求める場合、Aの列とBの行の数が一致している必要がある
- 例えば[3 × 2]型の行列は、[2 × $n$]型の行列とのみ積を計算できる




行列の積はステップごとに積和をとる。
入力は１行２列　→　出力は２つ

# 行列の積（ロジック）

- 入力データ：行列$A$の行と列 $(Ax, Ay)$ 、行列$B$の行と列 $(Bx, By)$
- 行列$A$と$B$の各要素を for 文で回す
- 共通部分の $Ay$ と $By$ は 1つの変数でまとめて for 文で回す
- 行列$C$は2層目の入力パラメータとして再利用、重みの行列も同様

**行列 $A$ の定義 ( $B$ も同様)**

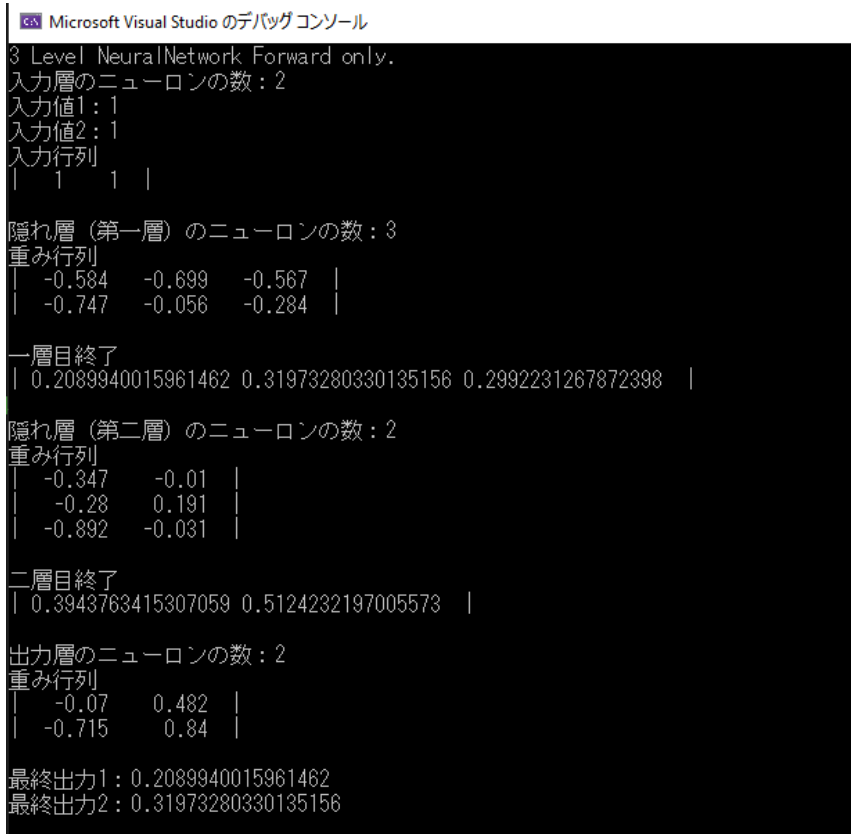
```
int Ax, Ay; //行数と列数
double[,] matrixA = new double[Ax, Ay]; //行列
for (int x = 0; x < Ax; x++) //行数
{
    for (int y = 0; y < Ay; y++) //列数
    {
        // 行列要素の値を入れていく
        matrixA[x, y] = 要素の値;
    }
}
```

**行列の積**

```
double[,] matrixC = new double[Ax, By]; //行列(解)
for (int x = 0; x < Ax; x++) // 行列Aの行成分
{
    for (int y = 0; y < By; y++) // 行列Bの列成分
    {
        for (int k = 0; k < Ay; k++) //共通要素数 k
        {
            //行列AとBの積
            matrixC[x, y] += (matrixA[x, k] * matrixB[k, y]);
        }
    }
}
```

# ハンズオン ＃2

- 3層ニューラルネットワークを作る
- プラットフォームは C# (.NET Core 3.0 コンソールアプリケーション)
- XAMLにてニューロンの図を描いて視覚的に動きをみる




**3層ニューラルネットワークのプログラムは**

**Githubにアップしています**

https://github.com/takunology/DeepLearning/tree/master/3NN_NotLearningCLI

# 入力層と出力層

- 入力層と出力層のニューロンはいくつあっても良い
- 例えば「手書き数字認識」は数百の入力ニューロン、10 の出力ニューロンがある
- 最終的には出力結果を「確率」として表示
- シグモイド関数ならば正の値で取り出せる