アジャイルワーク2

第3-4週（フルカラースキャナのデータ取得品質向上）

2025年10月3日

情報変革科学部　情報工学科

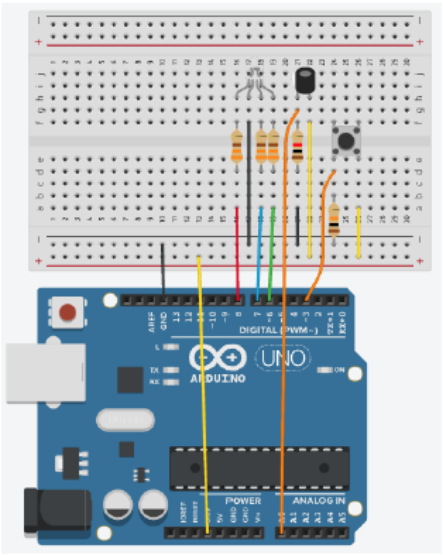
前川　仁孝（yoshitaka.maekawa@p.chibakoudai.jp）

大西　隆之（onishi.takayuki@p.chibakoudai.jp）

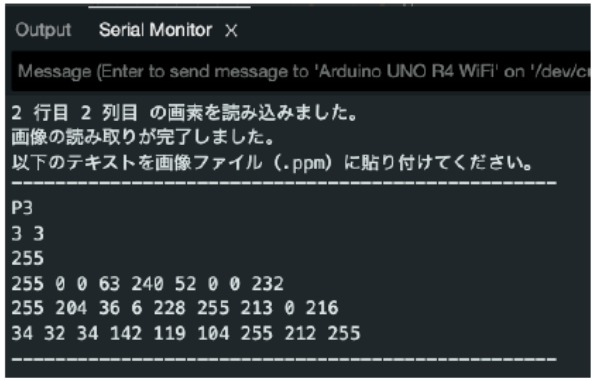
今回課題の概要

読み取る画像

自作スキャナで読み取り



読み取り結果の出力



ArduinoIDEのシリアルモニタに出力

画像として保存

印刷

正解データ

① 誤差が小さいほど良い

②読み取り時間が短いほど良い

読み取りデータ

前回までの実施内容

- （１）電子回路作成＋Arduinoサンプルプログラムによる、1ボタン読み取り動作～PPM画像出力までの実装
- （２）Arduinoサンプルプログラムの拡張による、2ボタン（最大最小値の反映、読み取動作～PPM画像出力までの実装
- 前回、多くの方が（２）まで到達したようです
（前回（１）まで実施した方は、本日（２）にチャレンジしてください）

本日～次回までの実施内容

- 読み込み品質の測定
  - 2ボタンプログラムによる、カラーチャートの読み取り
  - PPM画像出力
  - 平均二乗誤差（MSE）の計算
- 読み込み品質のソフトウェア的な改善
  - ニューラルネットワークによる学習環境の構築
  - Arduinoプログラムへの推論プログラム組み込み

読み込み品質の向上にむけて

- ソフトウェア的な改良を行う前に、スキャン時に色の情報をなるべく正確に読み取ることが大前提
  - 環境光、LEDからの直接光の影響を少なくする工作的なくふうは有益となる可能性あり
  - ただ、「周りを黒い紙で囲う」「LEDとフォトダイオードの間を仕切る」などの簡易な工作では、あまりよい結果にはならない可能性あり
  - 工作をしない場合も、少なくともカラーチャートの紙をしっかりとLEDとダイオードに近接させて測定することは効果的

読み込み品質の測定方法

- a) 事前に、黒と白で最小値・最大値を読み込み
- b) 前回配布したカラーチャート（3x3）を読み取り、PPM形式の画像ファイルを作成
- c) 平均二乗誤差（MSE）測定ソフトで、読み取り結果の品質を測定（ソフトはmanabaにアップロード済み）

平均二乗誤差（MSE）とは

- 2枚の画像の同じ位置どうしのピクセル輝度差の2乗平均値

$$MSE = \frac{1}{m\,n} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i,j) - K(i,j)]^2$$

https://qiita.com/yoya/items/510043d836c9f2f0fe2f

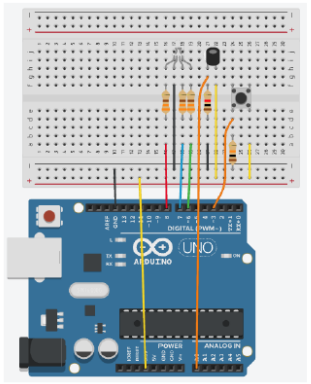
  - 今回の画像は3x3ピクセルなので、m=n=3

  - 今回はRGB値なので、ソフトではR, G, Bそれぞれのピクセル輝度差の2乗を加算して、3で割っている

    - mse += (rDiff * rDiff + gDiff * gDiff + bDiff * bDiff) / 3;

読み取り結果（PPMファイル）の例

読み取り



RGBの読み取り時間ずれを防ぐため、#define RGBFlashDelay の値を小さく（10など）にすることを推奨

シリアル出力をコピー＆ペーストしてPPMファイルを作成するには、VS Codeのテキストエディタなどを使用

MSEの計算方法

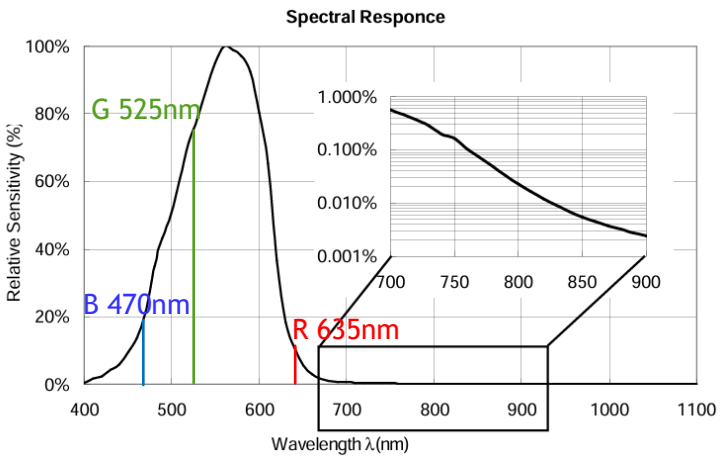
- 評価ツール（mse_calculator.html）をブラウザで開く
- 評価ツール に画像をアップロード
  - 基準画像：reference_image.ppm
  - アップロード画像：自分で作成したPPM画像
- MSEの値を記録する

- 例として前ページの結果を評価すると
  MSEは2930.11（これはかなり悪い値）

自身で測定したMSE値は、計画書・報告書（レポート）作成のために記録しておくこと

読み込み品質のソフトウェア的改良

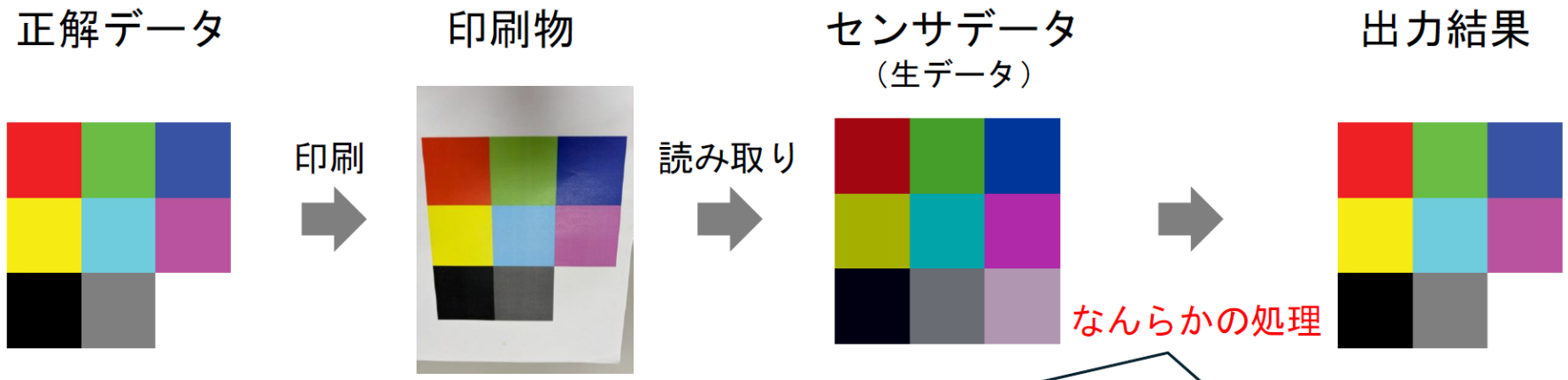
- 色が正確に読み取れない原因
  - 元ファイルの色情報と、印刷物の色との不一致
  - RGBの読み取り感度の差
  - フォトダイオードの読み取り特性の非線形性
    - ただし、本実験で用いるフォトダイオードの特性はほぼ線形
  - …
- 読み取り後に、Arduino側で修正を行う方法はあるか？



NJL7302L-F3　データシート

https://www.nisshinbo-microdevices.co.jp/ja/products/ambient-light-sensor/spec/?product=njl7302l-f3

読み込み品質のソフトウェア的改良



なんらかのルールに基づいて変換？

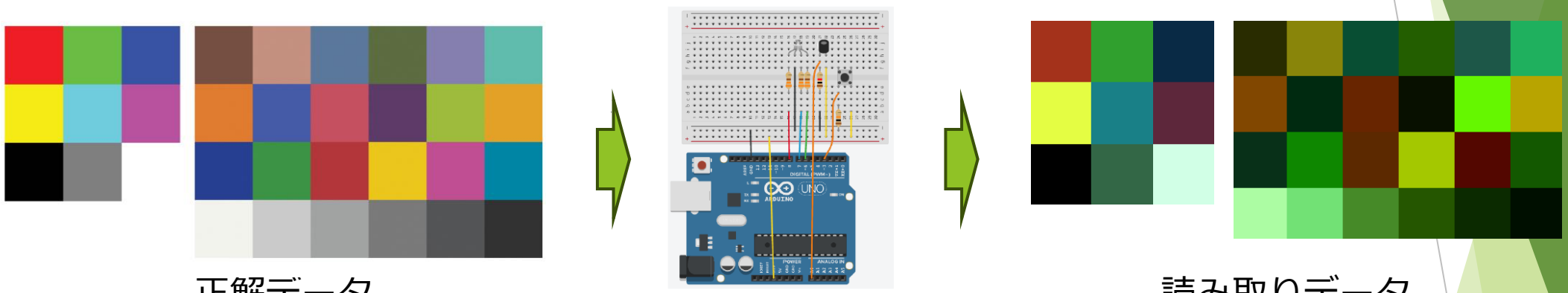
センサ(生)データ		変換後のデータ
(219, 10, 23)	→	(237, 30, 36)
(101, 167, 54)	→	(105, 189, 69)
(17, 93, 171)	→	(57, 83, 164)
⋮		⋮

統計処理や機械学習を使う？

- ルールベース？
- 線形回帰？
- ニューラルネットワーク？

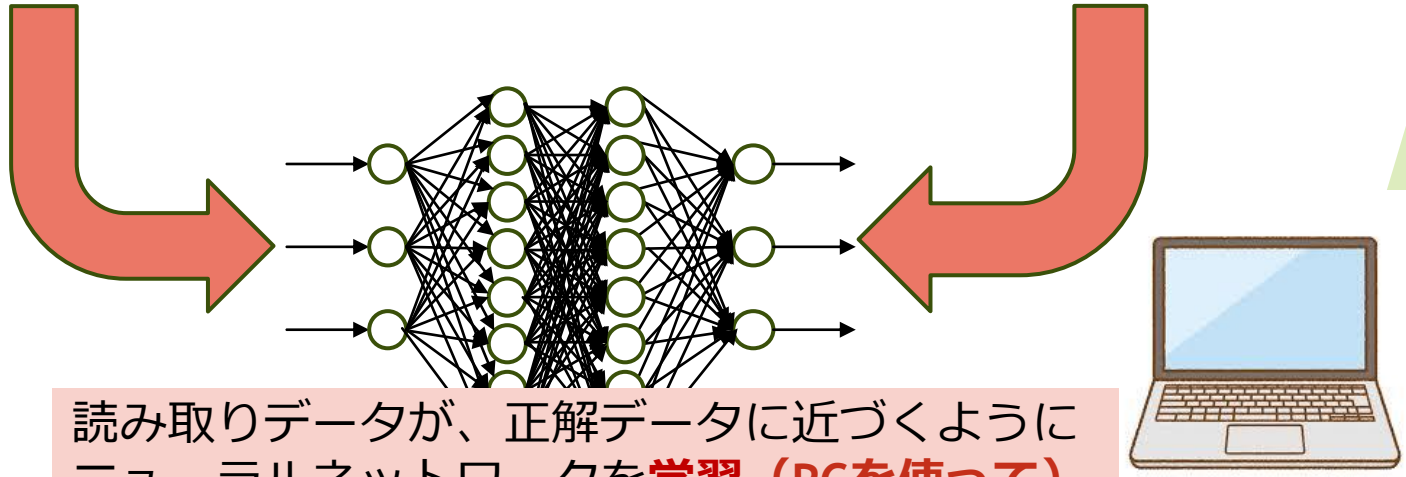
ニューラルネットワークによる学習・推論（学習フェーズ）

読み取り



正解データ
（学習に使えるように色数を増やしたもの）

読み取りデータ

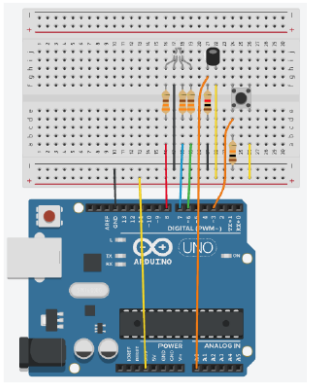


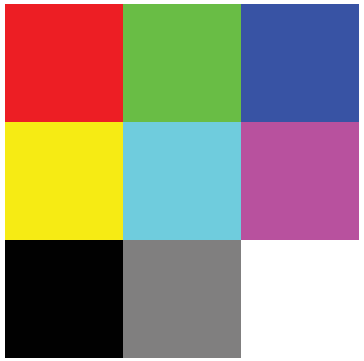
読み取りデータが、正解データに近づくように
ニューラルネットワークを**学習（PCを使って）**

ニューラルネットワークによる学習・推論（推論フェーズ）

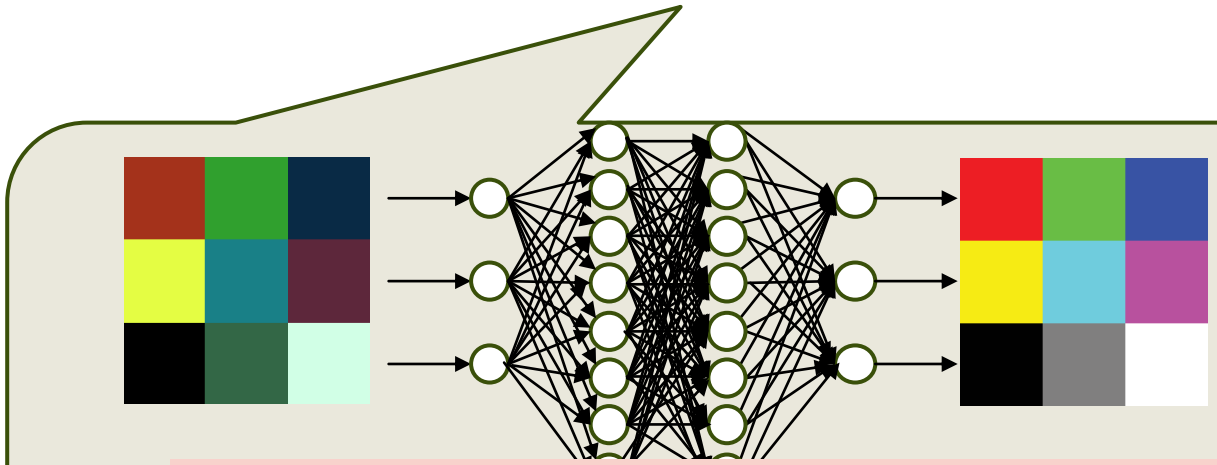
読み取り

入力データ





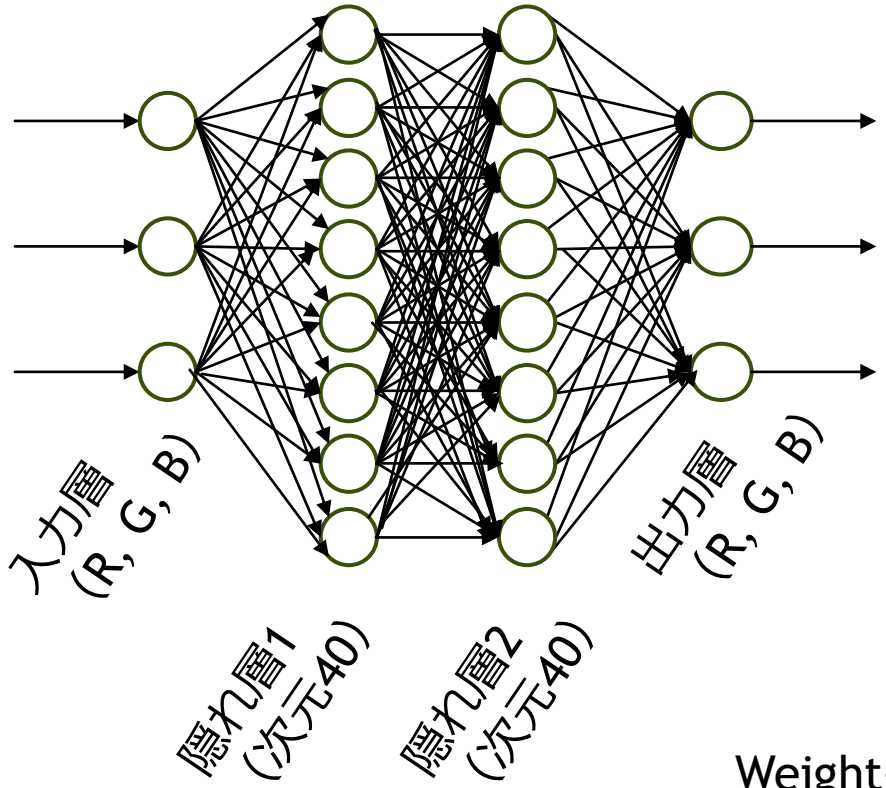
出力データ



読み取りデータが、入力データに近づくように
ニューラルネットワークで**推論（Arduinoの中で）**

使用するニューラルネットワーク

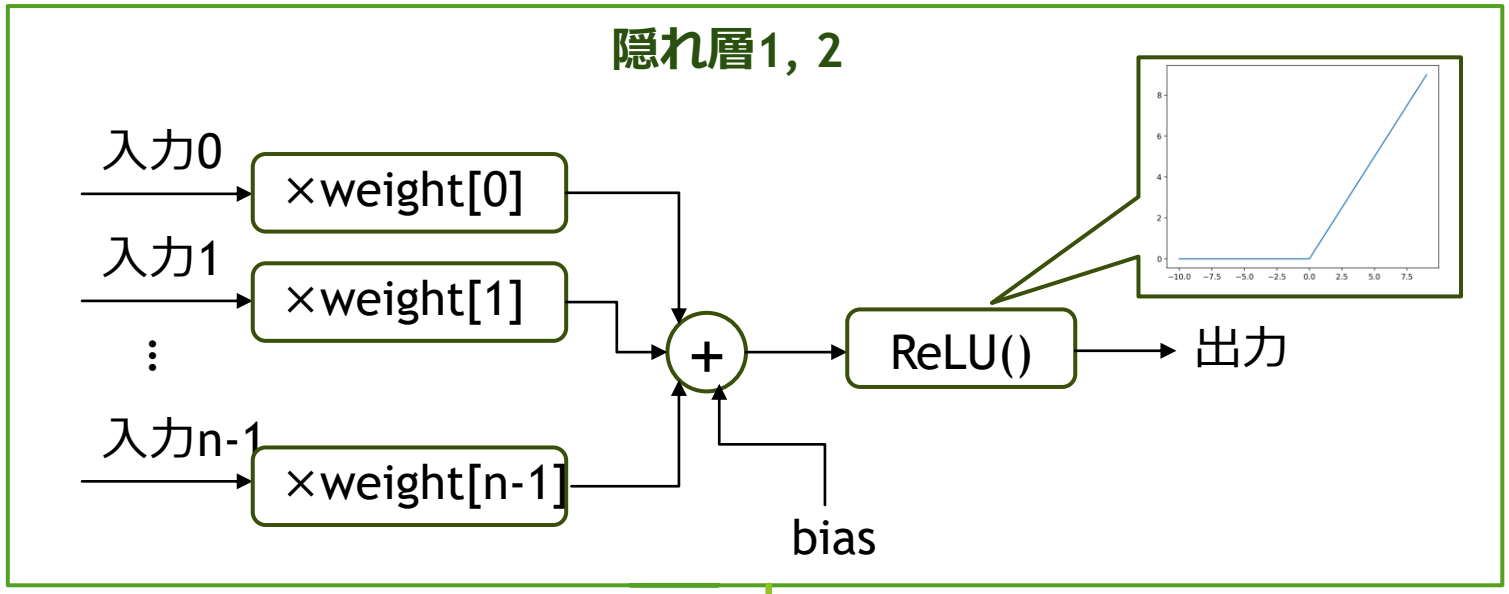
- 全結合型、4層のニューラルネットワークを使用
- 2層ある隠れ層は、それぞれ次元40（40個のニューロンが全結合）



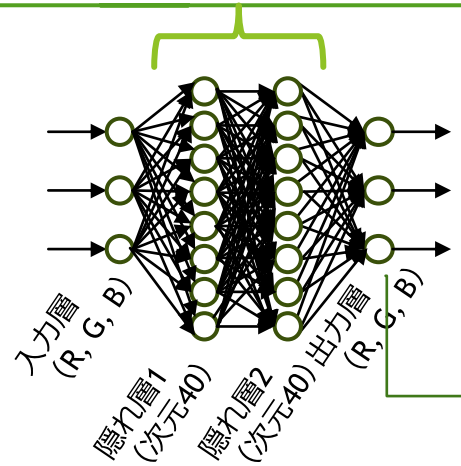
Weight+bias　ReLu()

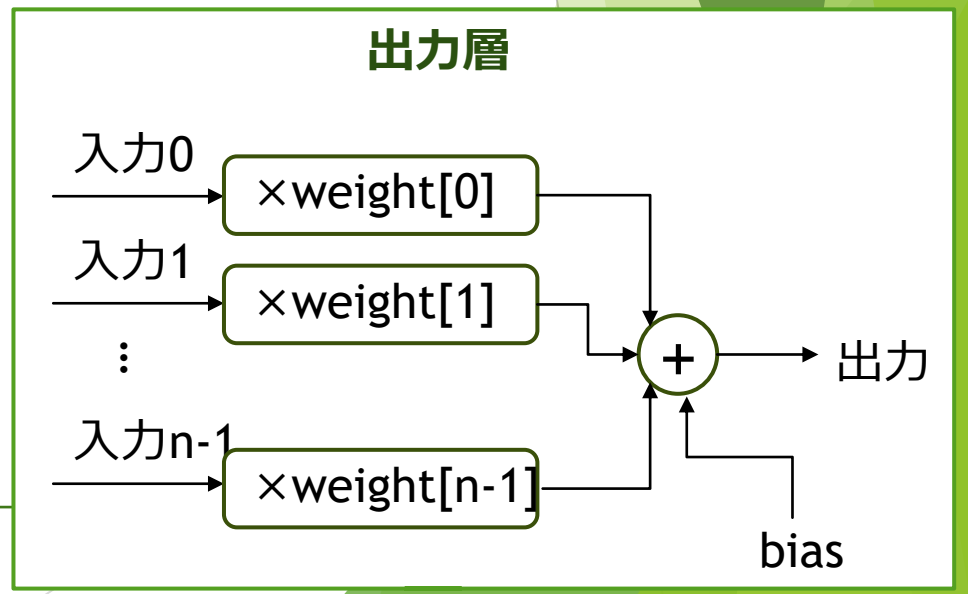
使用するニューラルネットワーク

- 各ニューロンで行われる推論演算は、以下の通り



https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/





読み込み用カラーチャート（その2）

- 一人一枚配布します
- 足りない場合は、manabaからダウンロードして印刷すること

学籍番号　　氏名

学習データの取得

- 新たなカラーチャートデータを読み込む
  - 3x3ブロック
  - 6x4ブロック（プログラムの指定は6x6として、下2行は白とする）

- 読み込み結果を、RGBそれぞれ0～255から0.0～1.0にスケールする
  - 方法は任意

学習データが非常に少ない＝1つ1つのデータの正確性がとても重要
PPMファイルをチェックして、想定外の色ずれや取得ミスがないこと
を必ず確認

学習環境の確認

- Visual Studio Code for Macのインストール
- Python環境の設定

- 上記の環境構築ができていない人は、setup_python.pdf（manabaにアップロード済み）を見て環境構築すること

データの学習

- train_L1_normalization.ipynbの # 推論データ　の項に、先ほど取得した学習データを記載

- 実行すると、model_parameters.h が出力される

```
# 推論データ
X =  np.array([[0.77, 0.10, 0.06],
               [0.22, 0.86, 0.17],
               [0.95, 0.19, 0.31],
               [0.95, 1.00, 0.24],
               [0.29, 0.97, 0.75],
               [0.39, 0.15, 0.24],
               [0.00, 0.00, 0.00],
               [0.12, 0.25, 0.15],
               [1.00, 1.00, 1.00],

               [0.11, 0.04, 0.00],
               [0.41, 0.41, 0.19],
               [0.10, 0.29, 0.27],
               [0.05, 0.15, 0.03],
               [0.19, 0.35, 0.35],
               [0.25, 0.85, 0.44],

               [0.55, 0.29, 0.05],
               [0.05, 0.18, 0.30],
               [0.43, 0.13, 0.11],
               [0.05, 0.01, 0.08],
               [0.34, 0.80, 0.14],
               [0.66, 0.60, 0.12],

               [0.05, 0.08, 0.21],
               [0.08, 0.39, 0.06],
               [0.28, 0.00, 0.00],
               [0.63, 0.84, 0.11],
               [0.35, 0.10, 0.18],
               [0.06, 0.39, 0.34],

               [0.90, 1.00, 0.87],
               [0.56, 0.94, 0.55],
               [0.29, 0.49, 0.28],
               [0.12, 0.19, 0.11],
               [0.03, 0.05, 0.03],
               [0.00, 0.00, 0.00]], dtype=np.float32)
```

左側3x3のRGBデータを順に列挙

右側6x4のRGBデータを順に列挙

model_parameters.h の内容

```
float weight_1[] = {.....};                    //  隠れ層1のweight（3×40=120個）

float bias_1[] = {....};                       //  隠れ層1のbias（40個）

float weight_2[] = {.....................};    //  隠れ層2のweight（40×40=1600個）

float bias_2[] = {...}                         //  隠れ層2のbias（40個）

float weight_3[] = {.....};                    //  出力層の weight（40×3=120個）

float bias_3[] = {....};                       //  出力層のbias（3個）
```

Arduinoプログラムの改良

改良前のスケッチは、計画書・報告書（レポート）作成のための追実験用に別途保存しておくこと

- 学習結果をもとに、読み取り値の修正が可能なようにプログラムを追加する
- 配布したプログラムには、修正ポイントが記されている
- ただし、predict(); 関数は自身で作成しなければならない

```
// ニューラルネットで読み込み値を修正
  // predict(RGBInput, RGBOutput);

  // predict();が存在しない初期は、値を単純に同値をコピー
  RGBOutput[0] = RGBInput[0];
  RGBOutput[1] = RGBInput[1];
  RGBOutput[2] = RGBInput[2];
```

```
// ニューラルネットで読み込み値を修正
 predict(RGBInput, RGBOutput);

  // predict();が存在しない初期は、値を単純に同値をコピー
  // RGBOutput[0] = RGBInput[0];
  // RGBOutput[1] = RGBInput[1];
  // RGBOutput[2] = RGBInput[2];
```

プログラムの変更が必要なポイント

- プログラム先頭で model_parameters.h の読み込み、および全結合ニューラルネットワークのパラメータを定義する必要がありそう

```
// 実装例
#include "model_parameters.h"
#define INPUT_SIZE 3
#define HIDDEN_SIZE1 40
#define HIDDEN_SIZE2 40
#define OUTPUT_SIZE 3
```

- ReLU()関数は定義する必要がありそう

```
// 実装例
float relu(float x) {
    return (x > 0) ? x : 0;
}
```

predict();関数の書き方

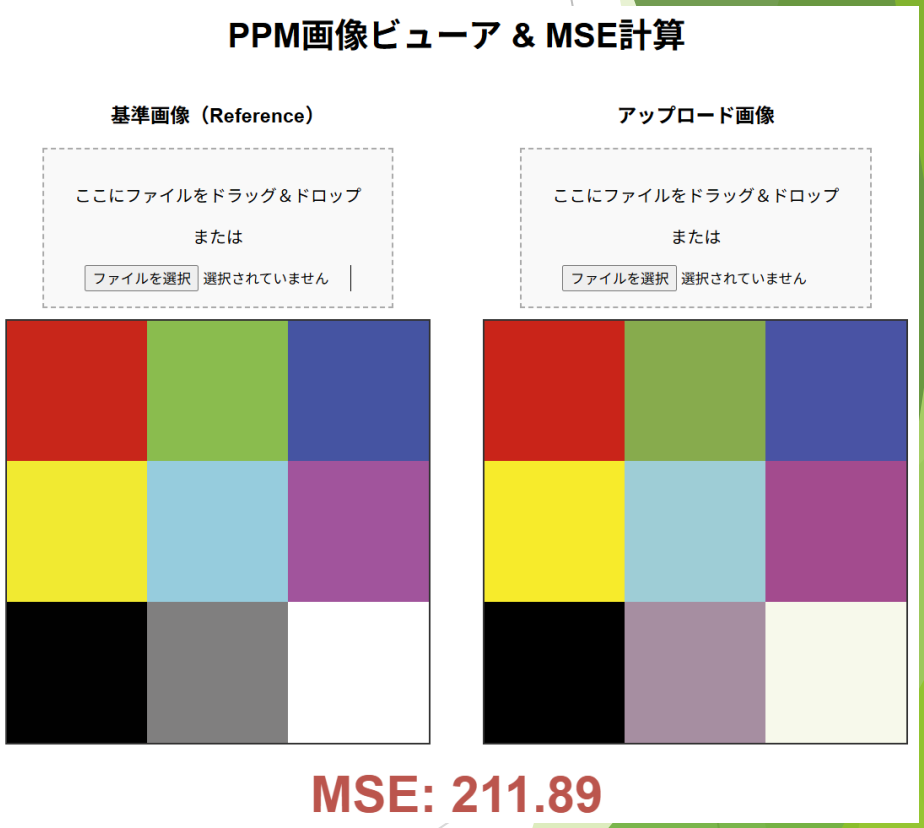
```
// 実装例
void predict(float input[INPUT_SIZE], float output[OUTPUT_SIZE]) {
    float layer1[HIDDEN_SIZE1] = {0};
    float layer2[HIDDEN_SIZE2] = {0};

    // **Layer 1 計算**
    for (int i = 0; i < HIDDEN_SIZE1; i++) {
        layer1[i] = bias_1[i];
        for (int j = 0; j < INPUT_SIZE; j++) {
            layer1[i] += weight_1[i * INPUT_SIZE + j] * input[j];
        }
        layer1[i] = relu(layer1[i]);  // ReLU
    }

    // **Layer 2 計算**

    // **Layer 3（出力層）計算**

}
```

Layer2と3の処理は、上記を参考に自分で書いてみること
Layer3（出力層）にはrelu();の処理が不要であることに注意

学習結果の例

MSE: 3271.96

学習後



MSE: 211.89

動作したら

- 読み取り画像（3×3）のMSEが改善するかを試すこと
- その他の改良ポイントがないか考え、反映したらMSEがさらに改善しているかを試すこと