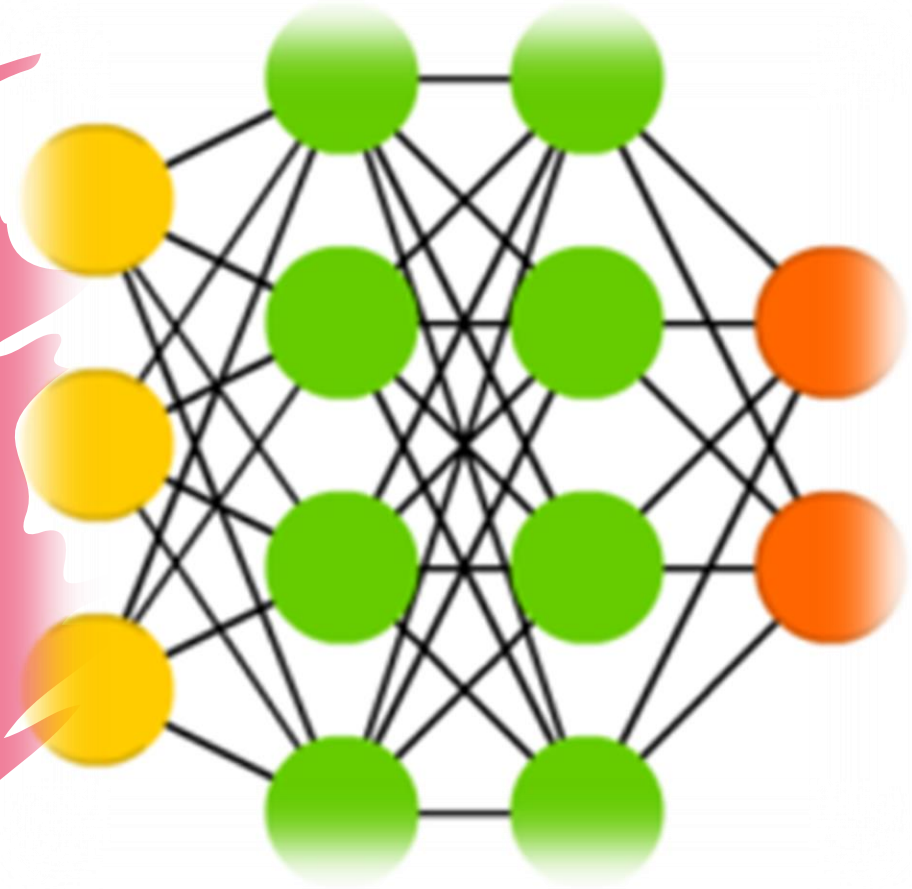
手計算
ニューラルネットワーク
入門　④実装1続き

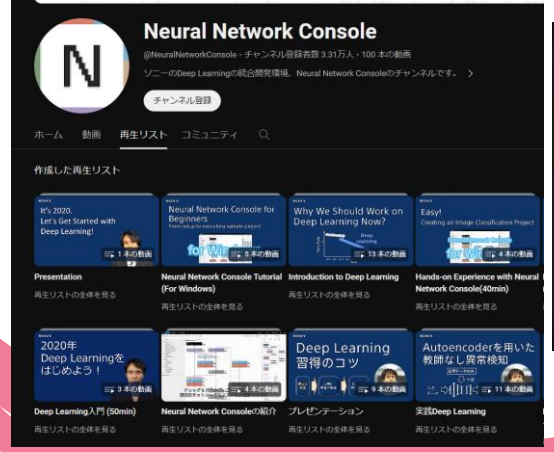


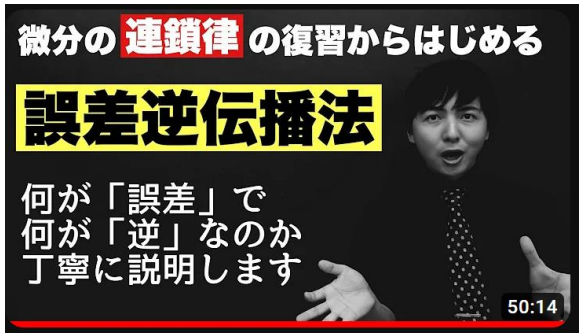
予定

第１回　理論：順伝播
第２回　理論：逆伝播
第３回　実装：実装1
第４回　実験：実装1の続き
第５回　実験：実装2iris,titanic
第６回　実験：実装3mnist

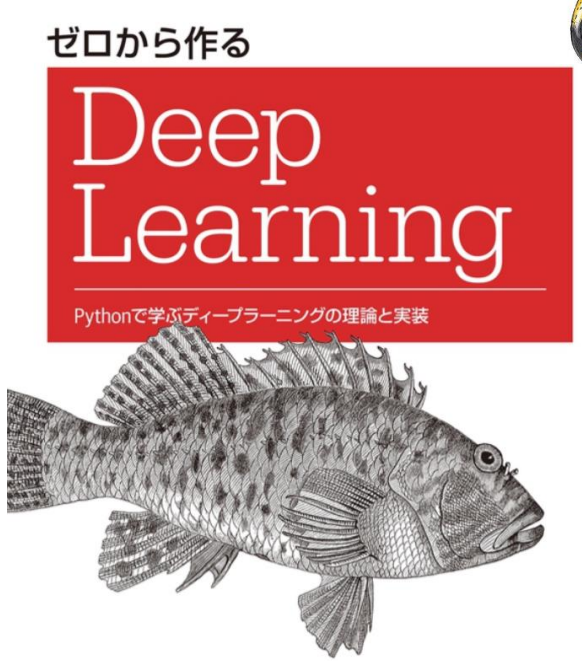
ニューラルネットワーク
を完全に理解したい

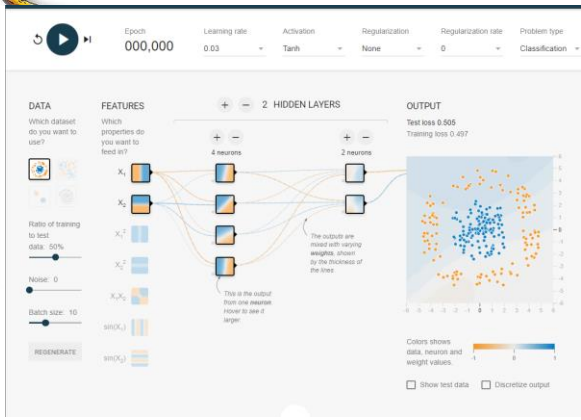
参考文献


Neural Network Console
@NeuralNetworkConsole - チャンネル登録者数 3.31万人 - 100 本の動画
ソニーのDeep Learningの統合開発環境、Neural Network Consoleのチャンネルです。
チャンネル登録
ホーム 動画 再生リスト コミュニティ
作成した再生リスト
Presentation
Neural Network Console Tutorial (For Windows)
Introduction to Deep Learning
Hands-on Experience with Neural Network Console(40min)
2020年 Deep Learningをはじめよう！
Deep Learning 習得のコツ
Autoencoderを用いた教師なし異常検知
Deep Learning入門 (50min)
Neural Network Consoleの紹介
プレゼンテーション
実践Deep Learning

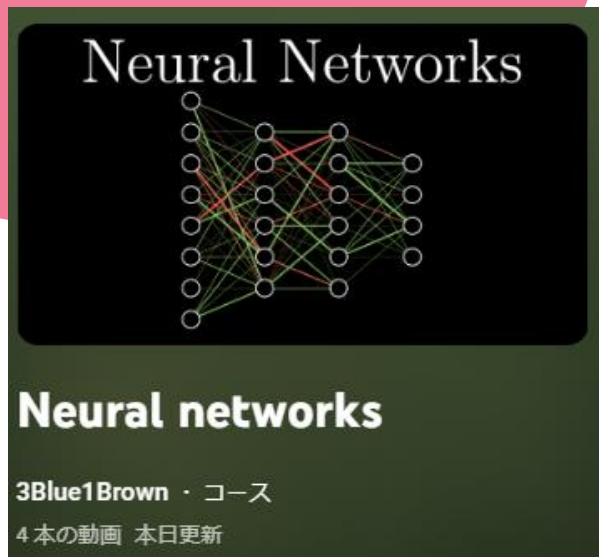

微分の連鎖律の復習からはじめる
誤差逆伝播法
何が「誤差」で
何が「逆」なのか
丁寧に説明します
50:14


O'REILLY
オライリー・ジャパン
第2版
scikit-learn、Keras、TensorFlowによる
実践機械学習
powered by
jupyter
Aurélien Géron 著
下田 倫大 監訳
長尾 高弘 訳


ゼロから作る
Deep Learning
Pythonで学ぶディープラーニングの理論と実装


Epoch
000,000
Learning rate
Activation
Regularization
Regularization rate
Problem type
DATA
FEATURES
2 HIDDEN LAYERS
OUTPUT
Test loss 0.505
Training loss 0.497

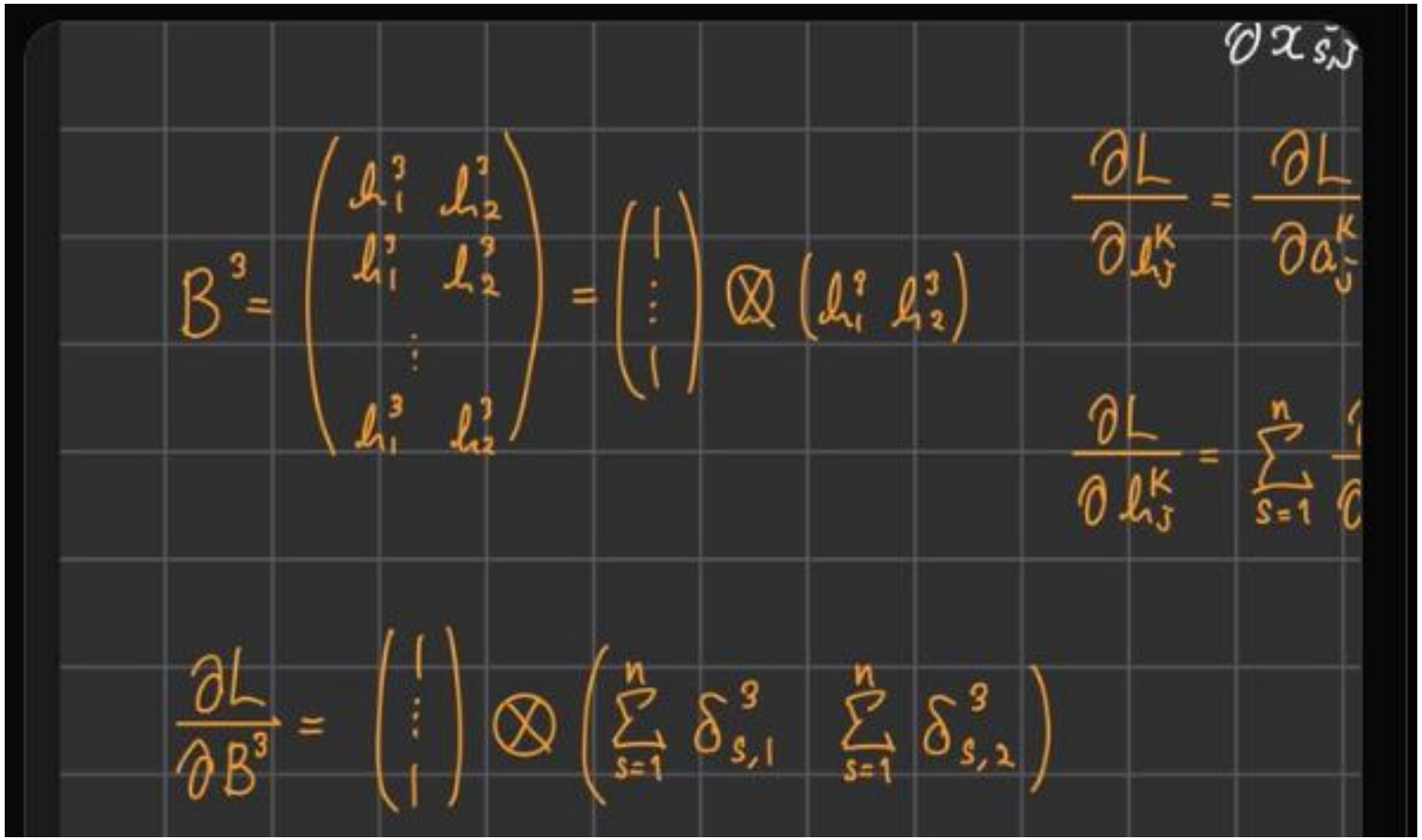

DEEP LEARNING 入門
徹底解説
Part 0
初心者のための
の前準備
ディープラーニング入門：
仕組み理解から実装まで
某処生活_LiveSomewhere
5 本の動画 996 回視聴 最終更新日: 2021/02/28


Neural Networks
Neural networks
3Blue1Brown ・ コース
4 本の動画 本日更新

前回のスライドの訂正

ミニバッチ学習のバイアス

ミニバッチ学習の時のバイアスの微分



クロネッカー積

定義 [編集]

$A = (a_{ij})$ を $m \times n$ 行列、$B = (b_{kl})$ を $p \times q$ 行列とすると、それらのクロネッカー積 $A \otimes B$ は

$$A \otimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix}$$

で与えられる $mp \times nq$ 区分行列である。もっとはっきり成分を示せば、 $A \otimes B$ は

$$\begin{bmatrix} a_{11}b_{11} & a_{11}b_{12} & \cdots & a_{11}b_{1q} & \cdots & \cdots & a_{1n}b_{11} & a_{1n}b_{12} & \cdots & a_{1n}b_{1q} \\ a_{11}b_{21} & a_{11}b_{22} & \cdots & a_{11}b_{2q} & \cdots & \cdots & a_{1n}b_{21} & a_{1n}b_{22} & \cdots & a_{1n}b_{2q} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{11}b_{p1} & a_{11}b_{p2} & \cdots & a_{11}b_{pq} & \cdots & \cdots & a_{1n}b_{p1} & a_{1n}b_{p2} & \cdots & a_{1n}b_{pq} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots & \ddots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{m1}b_{11} & a_{m1}b_{12} & \cdots & a_{m1}b_{1q} & \cdots & \cdots & a_{mn}b_{11} & a_{mn}b_{12} & \cdots & a_{mn}b_{1q} \\ a_{m1}b_{21} & a_{m1}b_{22} & \cdots & a_{m1}b_{2q} & \cdots & \cdots & a_{mn}b_{21} & a_{mn}b_{22} & \cdots & a_{mn}b_{2q} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{m1}b_{p1} & a_{m1}b_{p2} & \cdots & a_{m1}b_{pq} & \cdots & \cdots & a_{mn}b_{p1} & a_{mn}b_{p2} & \cdots & a_{mn}b_{pq} \end{bmatrix}$$

実装
前回の続きから

誤差逆伝播に必要な関数の実装

誤差関数の微分
今回はcross entropy

$$CE = f_L = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m} t_{ij}\log x_{ij}$$

$$\frac{\partial f_L}{\partial x_{pq}} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m} t_{ij}\frac{\partial}{\partial x_{pq}}(\log x_{ij})$$

$$= -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m}\frac{t_{ij}}{x_{ij}}\frac{\partial x_{ij}}{\partial x_{pq}}$$

$$\frac{\partial x_{ij}}{\partial x_{pq}} = \begin{cases} 0(p \neq i) \\ 1(p = i, q = j) \\ -1(p = i, q \neq j) \end{cases}$$

$\frac{\partial f_L}{\partial X}$(n行m列)の行列を返す関数を作成

```
vvd calc_r_cross_entropy(vvd &x, vvd &t) {
    //ここに書く
}
```

誤差逆伝播に必要な関数の実装

```
vvd calc_r_cross_entropy(vvd &x, vvd &t) {
    int n = x.size(), m = x[0].size();
    vvd tmp(n, vd(m, 0));
    for (int s=0; s<n; ++s) {
        for (int j=0; j<m; ++j) {
            for (int k=0; k<m; ++k) {
                if (j == k) tmp[s][j] -= t[s][j] / x[s][j];
                else tmp[s][j] += t[s][k] / (x[s][k]);
            }
            tmp[s][j] /= n;
        }
    }
    return tmp;
}
```

誤差逆伝播に必要な関数の実装

（とあるsインスタンス目に対して）

softmaxの微分

$$\frac{\partial h_K}{\partial A^K} = \left(\begin{pmatrix} \frac{\partial x_{1,1}{}^K}{\partial a_{1,1}{}^K} & \cdots & \frac{\partial x_{1,1}{}^K}{\partial a_{1,3}{}^K} \\ \vdots & \ddots & \vdots \\ \frac{\partial x_{1,3}{}^K}{\partial a_{1,1}{}^K} & \cdots & \frac{\partial x_{1,3}{}^K}{\partial a_{1,3}{}^K} \end{pmatrix} \cdots \right)$$

n個

$$\frac{\partial x_{s,i}{}^K}{\partial a_{s,j}{}^K} = x_{s,i}{}^K \left(1 - x_{s,j}{}^K\right), i = j$$

$$\frac{\partial x_{s,i}{}^K}{\partial a_{s,j}{}^K} = x_{s,i}{}^K \left(0 - x_{s,j}{}^K\right), i \neq j$$

```
//rx_k/ra_j
//m class 分類
//m次正方行列を返す
vvvd calc_r_softmax(vvd &x) {
    //ここに書く
}
```

がmini-batch個あるテンソルを返す

$\frac{\partial h_K}{\partial A^K}$(n層？m行m列)のテンソルを返す関数を作成

誤差逆伝播に必要な関数の実装

```
vvvd calc_r_softmax(vvd &x) {
    int n = x.size(), m = x[0].size();
    vvvd ret(n, vvd(m, vd(m, 0)));
    for (int s=0; s<n; ++s) {
        for (int i=0; i<m; ++i) {
            for (int j=0; j<m; ++j) {
                if (i == j) ret[s][i][j] = x[s][i]*(1 - x[s][j]);
                else ret[s][i][j] = x[s][i]*(0 - x[s][j]);
            }
        }
    }
    return ret;
}
```

誤差逆伝播に必要な関数の実装

ReLUの微分

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

$\frac{\partial h_l}{\partial A^l}$(n行m列)の行列
を返す関数を作成

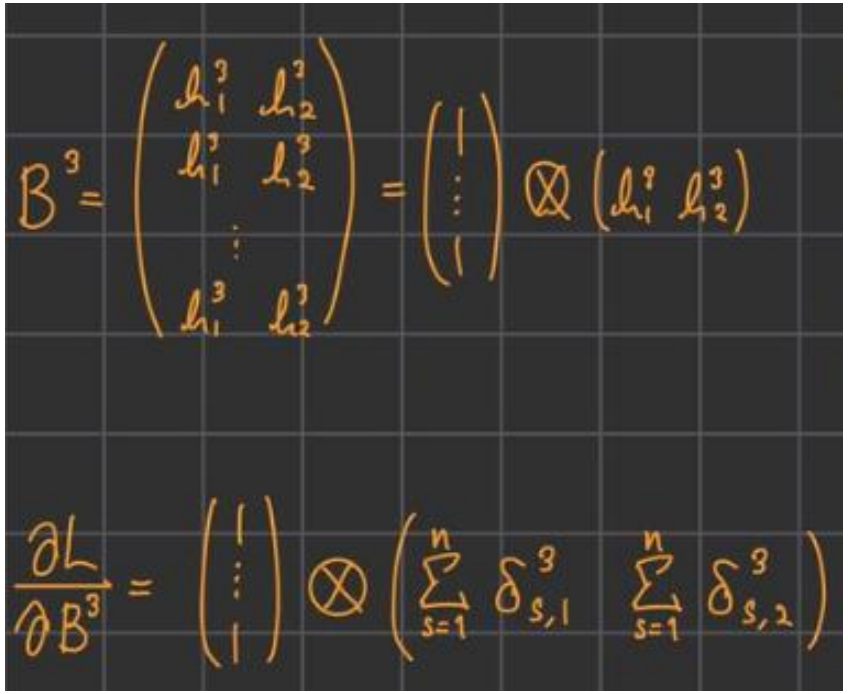
```
vvd calc_r_ReLU (vvd &a) {
    //ここに書く
}
```

誤差逆伝播に必要な関数の実装

```
vvd calc_r_ReLU (vvd &a) {
    int n = a.size(), m = a[0].size();
    vvd tmp(n, vd(m, 0));
    for (int s=0; s<n; ++s) {
        for (int j=0; j<m; ++j) {
            if (a[s][j] >= 0) tmp[s][j] = 1;
        }
    }
    return tmp;
}
```

誤差逆伝播に必要な関数の実装

バイアスの微分



$\frac{\partial L}{\partial B^l}$(n行m列)の行列
を返す関数を作成

```
vvd calc_r_bias (vvd &delta) {
    //ここに書く
}
```

誤差逆伝播に必要な関数の実装

```
vvd calc_r_bias (vvd &delta) {
    int n = delta.size(), m = delta[0].size();
    vvd rb;
    if (n != delta.size() || m != delta[0].size()) cout << "size is not match" <<
    rb.assign(1, vd(m, 0));
    for (int j=0; j<m; ++j) {
        for (int i=0; i<n; ++i) {
            rb[0][j] += delta[i][j];
        }
    }
    rb = expansion_bias(rb, n);
    return rb;
}
```

順伝播の実装

入力 X^0 $(n \times 2)$

$$A^1 = X^0 W^1 + B^1$$
$$X^1 = ReLU(A^1)$$
$$A^2 = X^1 W^2 + B^2$$
$$X^2 = ReLU(A^2)$$
$$A^3 = X^2 W^3 + B^3$$
$$X^3 = SoftMax(A^3)$$
$$L = CrossEntropy(X^3)$$

```
//forward propagation
for (int k=0; k<depth; ++k) {
    //ここに書く
}
```

逆伝播の実装

```
//back propagation
for (int k=depth-1; k>=0; --k) {
    //ここに書く
}
```

誤差逆伝播-公式 まとめ 行列表現

出力層

$$\frac{\partial L}{\partial W^K} = X^{(K-1)T}\Delta^K$$

$$\frac{\partial L}{\partial B^K} = \begin{pmatrix}1\\ \vdots \\ 1\end{pmatrix} \otimes \left(\sum_i {\delta_{i,1}}^K \quad \sum_i {\delta_{i,2}}^K \quad \sum_i {\delta_{i,3}}^K\right)$$

$$\Delta^K = \frac{\partial f_L}{\partial X^K} \odot \frac{\partial h_K}{\partial A^K}$$

中間層

$$\frac{\partial L}{\partial W^l} = X^{(l-1)T}\Delta^l$$

$$\frac{\partial L}{\partial B^l} = \begin{pmatrix}1\\ \vdots \\ 1\end{pmatrix} \otimes \left(\sum_i {\delta_{i,1}}^l \quad \sum_i {\delta_{i,2}}^l \quad \sum_i {\delta_{i,3}}^l\right)$$

$$\Delta^l = \Delta^{l+1} W^{(l+1)T} \odot \frac{\partial h_l}{\partial A^l}$$

訓練のテクニック

バッチ学習：すべての訓練インスタンスを一気に学習に用いる

必ず極小値に収束する

訓練インスタンスが大量だと時間にかかる

訓練インスタンスが数百GBだとそもそもパソコンに収まらない.

訓練のテクニック

ミニバッチ学習：訓練インスタンスの一部を学習に用いる
オンライン学習：訓練インスタンス１つを学習に用いる

訓練インスタンスが大量でも一部をパソコンにロードして
学習して捨てて，またロードすればいい→アウトオブコア学習

極小値の近くで行ったり来たりしちゃうかも
→学習計画

訓練のテクニック

学習計画
学習が進むにつれて，学習率の値を小さくする
いつか学習率が0になるのでパラメータの更新が起きなくなる
そこが極小値付近なら収束が見込める

訓練のテクニック

Momentum

$V \leftarrow \alpha V - \eta \frac{\partial L}{\partial W}$　　V：速度，α：空気抵抗　0.9など

$W \leftarrow W + V$

AdaGrad

学習係数を減衰させる

はじめ大 → 次第に小

適応的 Adaptive

$h \leftarrow h + \frac{\partial L}{\partial W} \odot \frac{\partial L}{\partial W}$

$W \leftarrow W - \eta \frac{1}{\sqrt{h}} \frac{\partial L}{\partial W}$

Adam ≈ Momentum + AdaGrad (2015)

訓練のテクニック

```
int main() {
    vvd x, t;
    double eta = 0.03, attenuation = 0.6;
    int n = 1000;
    int show_interval = 1000;
    int learning_plan = 2000;
    int loop = 9500;//9500
    int batch_size = 100;
```

学習計画
今回は単に適当なタイミングで
公比をかけて小さくする

```
//学習率の更新
if ((i+1) % learning_plan == 0) eta *= attenuation;
```

実装

プログラム動いた？
いい感じのハイパーパラメータを見つけてみよう
手でGrid Searchをする
ちなpython↓

```
from sklearn.model_selection import GridSearchCV

param_grid = [{'weights': ["uniform", "distance"], 'n_neighbors': [3, 4, 5]}]

knn_clf = KNeighborsClassifier()
grid_search = GridSearchCV(knn_clf, param_grid, cv=5, verbose=3)
grid_search.fit(X_train, y_train)
```

実装の確認

train_setも初見のtest_setにも適用したモデルができた
モデル：ハイパーパラメータとパラメータの総称，作ったAIそのもの

テストセットを可視化してみる
mainの最後のこれのコメントアウトを外す

```
//一旦csvに出力したのちpythonで描画してみる
// drawing_by_python(nn, depth);
```

実装の確認

```
void drawing_by_python(vector<layer_t> &nn, int depth) {
    vvd data;
    for (int i=0; i<depth; ++i) {
        nn[i].b = expansion_bias(nn[i].b, 1);
    }
    for (double x=-6; x<=6; x+=1) {
        for (double y=-6; y<=6; y+=1) {
            vvd tmp = {{x, y}};
            //forward propagation
            for (int k=0; k<depth; ++k) {
                if (k == 0) nn[k].a = matrix_add(matrix_multi(tmp, nn[k].w), nn[k].b);
                else nn[k].a = matrix_add(matrix_multi(nn[k-1].x, nn[k].w), nn[k].b);
                // if (k < depth-1) nn[k].x = hm_ReLU(nn[k].a);
                if (k < depth-1) nn[k].x = hm_tanh(nn[k].a);
                else nn[k].x = hm_softmax(nn[k].a);
            }
```

```
            vvd tmp = {{x, y}};
            //forward propagation
            for (int k=0; k<depth; ++k) {
                if (k == 0) nn[k].a = matrix_add(matrix_multi(tmp, nn[k].w), nn[k].b);
                else nn[k].a = matrix_add(matrix_multi(nn[k-1].x, nn[k].w), nn[k].b);
                // if (k < depth-1) nn[k].x = hm_ReLU(nn[k].a);
                if (k < depth-1) nn[k].x = hm_tanh(nn[k].a);
                else nn[k].x = hm_softmax(nn[k].a);
            }
            if (nn[depth-1].x[0][0] > nn[depth-1].x[0][1]) {
                //inside
                tmp[0].push_back(1);
            } else {
                //outside
                tmp[0].push_back(0);
            }
            data.push_back(tmp[0]);
        }
    }
    outputfile(data);
}
```

実装の確認

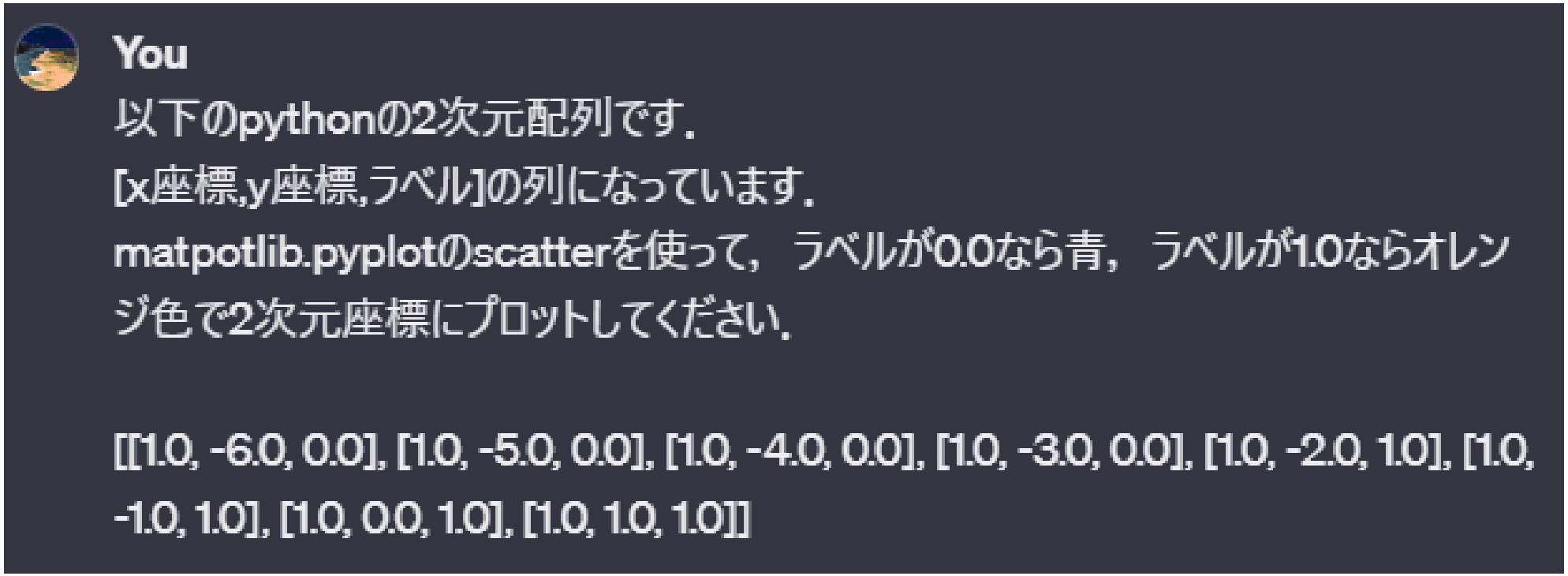
```cpp
//x, y, tを列挙
void outputfile(const vvd &output) {
    int n = output.size(), m = output[0].size();
    string fname = "circle_.csv";
    ofstream outputFile (fname);

    for (int i=0; i<n; ++i) {
        for (int j=0; j<m; ++j) {
            outputFile << output[i][j];
            if (j != m-1) outputFile << ", ";
        }
        outputFile << endl;
    }
}
```

```
55    -2, -4, 0
56    -2, -3, 0
57    -2, -2, 1
58    -2, -1, 1
59    -2, 0, 1
60    -2, 1, 1
61    -2, 2, 1
62    -2, 3, 0
```

circle_.csv

実装の確認



drawing.py　1文字も書いてないけどできた

実装の確認　感動を味わってほしいところ

```
void drawing_by_python(vector<layer_t> &nn, int depth) {
    vvd data;
    for (int i=0; i<depth; ++i) {
        nn[i].b = expansion_bias(nn[i].b, 1);
    }
    for (double x=-6; x<=6; x+=1) {
        for (double y=-6; y<=6; y+=1) {
```

点の差分を1→0.05にする　.cppを実行

実装の確認　感動を味わってほしいところ

```
30    # 散布図をプロット
31    plt.scatter(x_values, y_values, c=colors, s=20)
```

点のサイズs=20→1に変更　.pyを実行

実装の確認　感動を味わってほしい演習

中間層の活性化関数をReLUじゃなくてtanhにする
ACTIVATIONに以下を追記

```
double h_tanh(double x) {
    return (exp(x)-exp(-x)) / (exp(x)+exp(-x));
}
vvd hm_tanh(vvd &x) {
    int n = x.size(), m = x[0].size();
    vvd tmp(n, vd(m));
    for (int i=0; i<n; ++i) {
        for (int j=0; j<m; ++j) {
            tmp[i][j] = h_tanh(x[i][j]);
        }
    }
    return tmp;
}
```

実装の確認　感動を味わってほしい演習

中間層の活性化関数をReLUじゃなくてtanhにする
BACK PROPAGATIONに以下を追記

```
vvd calc_r_tanh(vvd &a) {
    int n = a.size(), m = a[0].size();
    vvd tmp(n, vd(m, 0));
    for (int s=0; s<n; ++s) {
        for (int j=0; j<m; ++j) {
            tmp[s][j] =
        }
    }
    return tmp;
}
```

長い

実装の確認　感動を味わってほしい演習

中間層の活性化関数をReLUじゃなくてtanhにする
MAINの順伝播・逆伝播
tarin_setの順伝播・test_setの順伝播
drawing_by_python関数内の

ReLUの部分を適切にtanhに書き直す
（関数の部分だけで大丈夫だと思う）

C++実行→Python実行

実装の確認　感動を味わってほしい演習

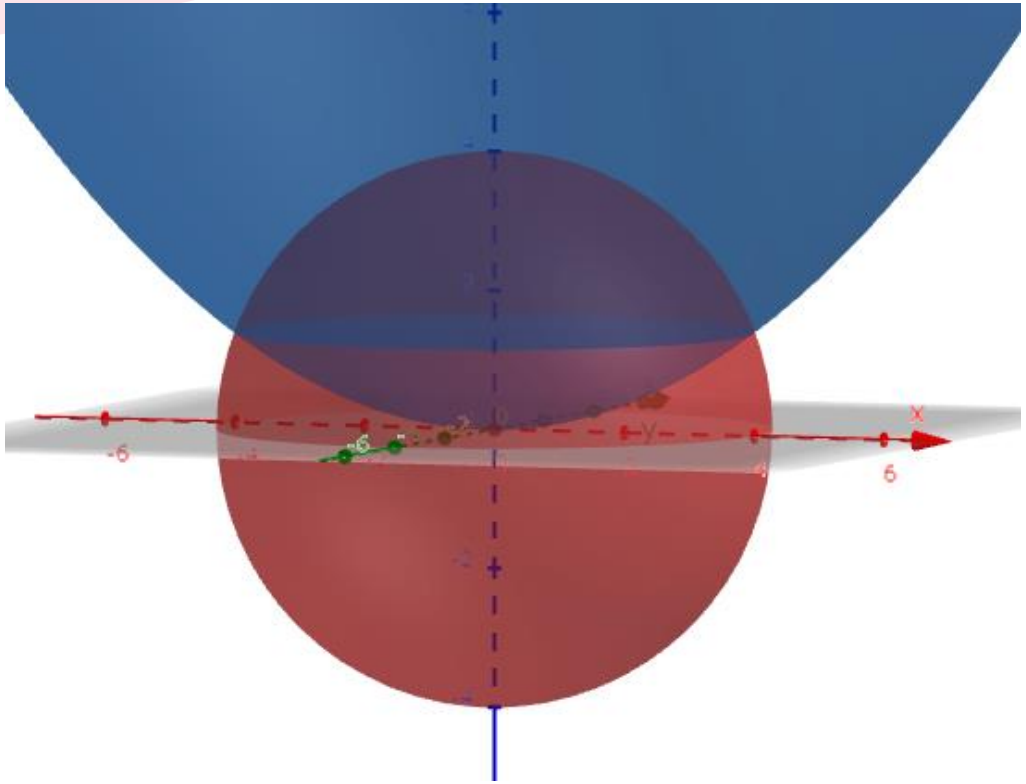
judge_term関数を編集して（すぐ下に書いてある）
xorのような分類器・線形分離可能な分類器が正しく
動作することを確認する

多クラス分類とone-hot表現

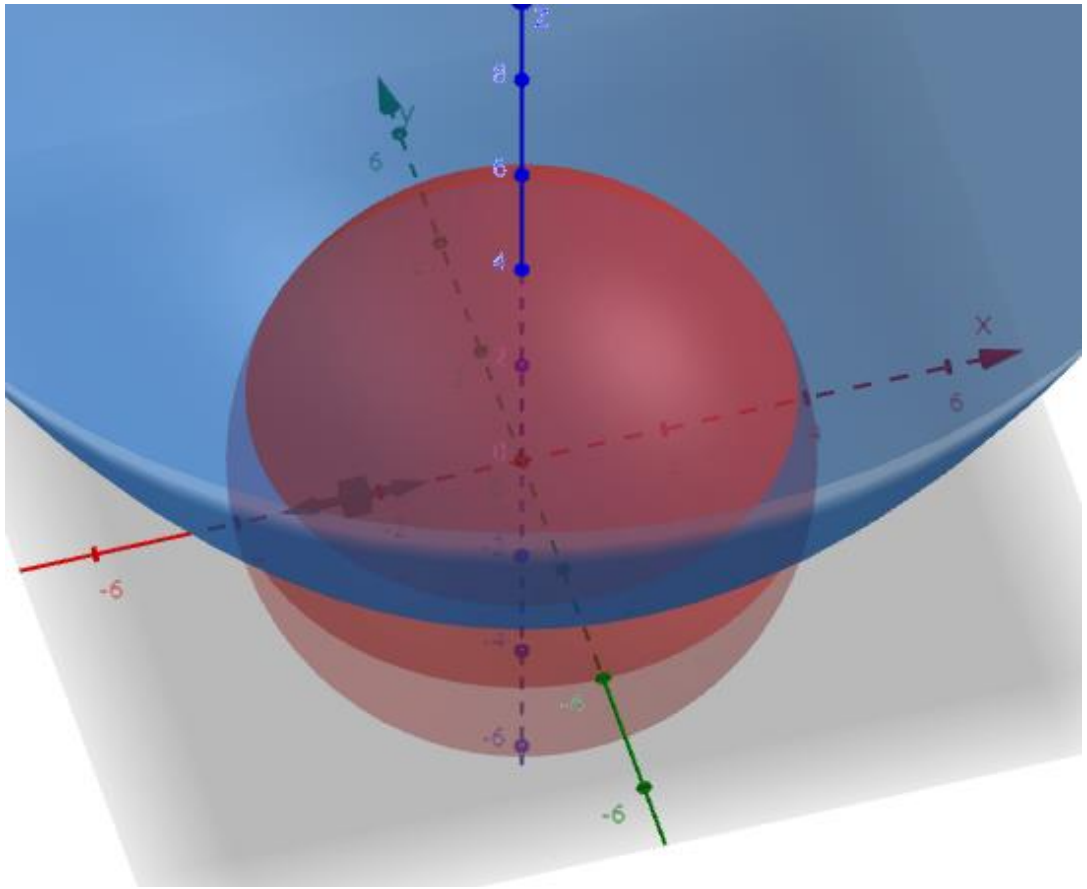
出力層が1ノードでクラスの番号（整数）を予測するようにすると
class2をclass1と誤認識した場合の誤差と
class3をclass1と誤認識した場合の誤差が
異なる.
→対等なクラス分類でも"近さ"的な要らない概念が出てきちゃう

class1:{1, 0, 0}
class2:{0, 1, 0}
class3:{0, 0, 1}
としてcross entropyをすればクラス同士に何の順序も生まれない

宿題 4class分類

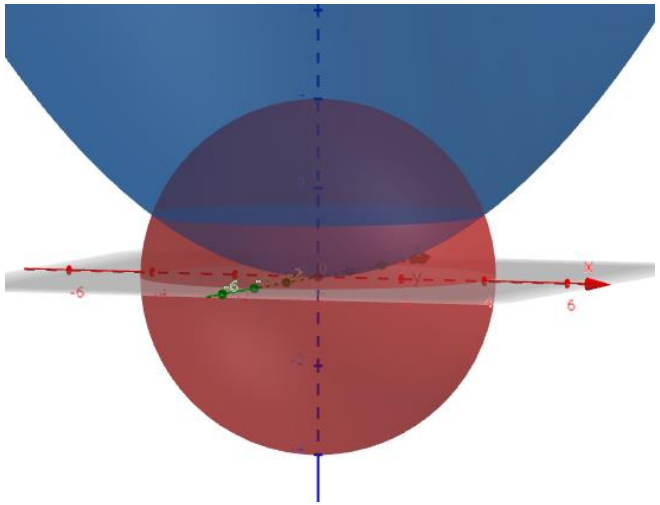


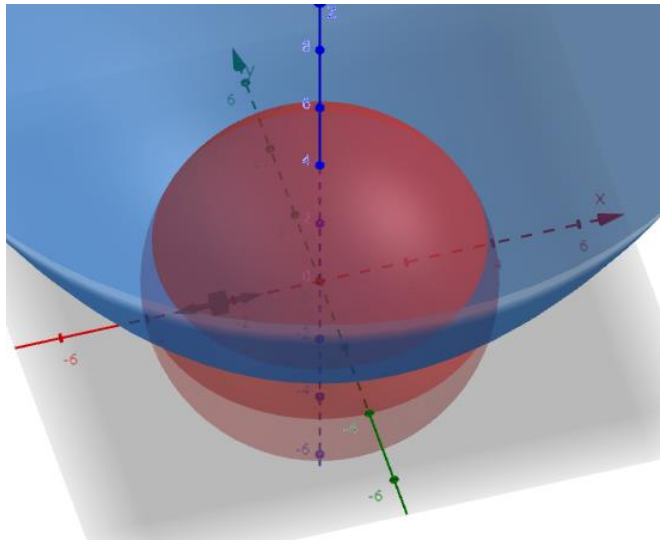
$$x^2 + y^2 + z^2 = 16$$



$$z = \frac{1}{10}(x^2 + y^2)$$

宿題 4class分類





インスタンス：特徴量(x,y,z)を持つ
すべてのインスタンスは
|x|<6, |y|<6, |z|<6の立方体の内側

class1:球の中かつ放物面の上(z>放物面)
class2:球の中かつ放物面の下
class3:球の外かつ放物面の上
class4:球の外かつ放物面の下