誤差逆伝播法と
3年生向け補足

2017年6月2日

今回の内容

- 誤差逆伝播法解説。
- 輪読 (3-3)。
- 3年生向けにDeep Learningの補足。

誤差逆伝播法

- 誤差逆伝播法（バックプロパゲーション）について、解説。
  https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/
  上記のページを参照している。
- また、このページのGitHubのリンクから、プログラム(neural-network.py)をダウンロードする。Pythonで実装されたプログラムがある。

Backpropagation in Python

You can play around with a Python script that I wrote that implements the backpropagation algorithm in this Github repo.

誤差逆伝播法（バックプロパゲーション）

- 誤差逆伝播法（BP）は多層パーセプトロンの学習に使われる学習アルゴリズム。ある学習データが与えられたとき、多層パーセプトロンの出力が学習データと一致するように各層の間の結合荷重を修正するという学習法である。
- 多層パーセプトロンは誤差逆伝播法によって教師あり学習をおこない、パターン識別や関数の近似などに用いられる。
- 多層パーセプトロン：ユニットが複数の層を構成するように並び、入力から出力への一方向へのみ信号の伝達がおこなわれるネットワーク。

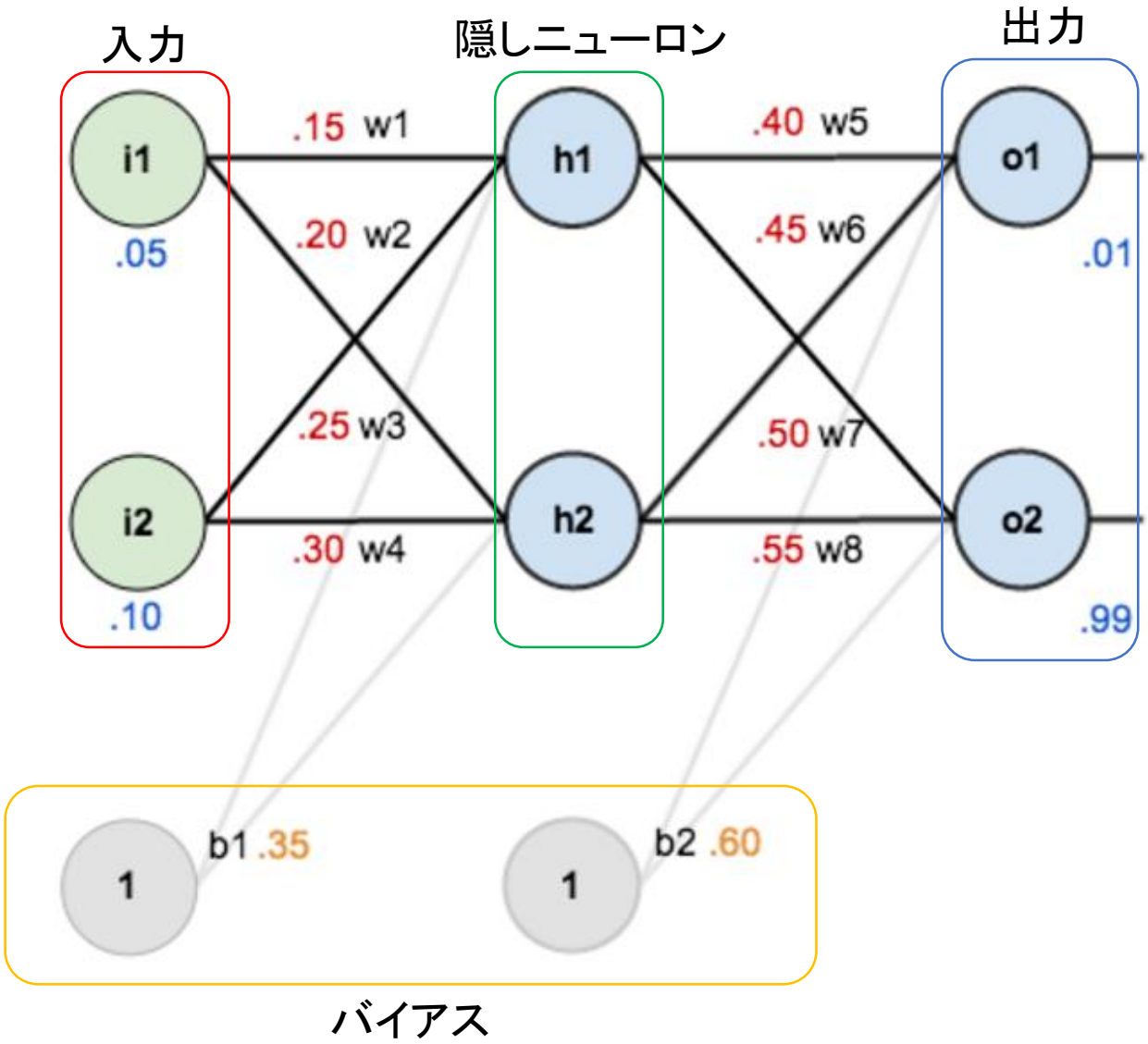
http://www.brain.kyutech.ac.jp/~furukawa/data/bp.html

誤差逆伝播法のアルゴリズム

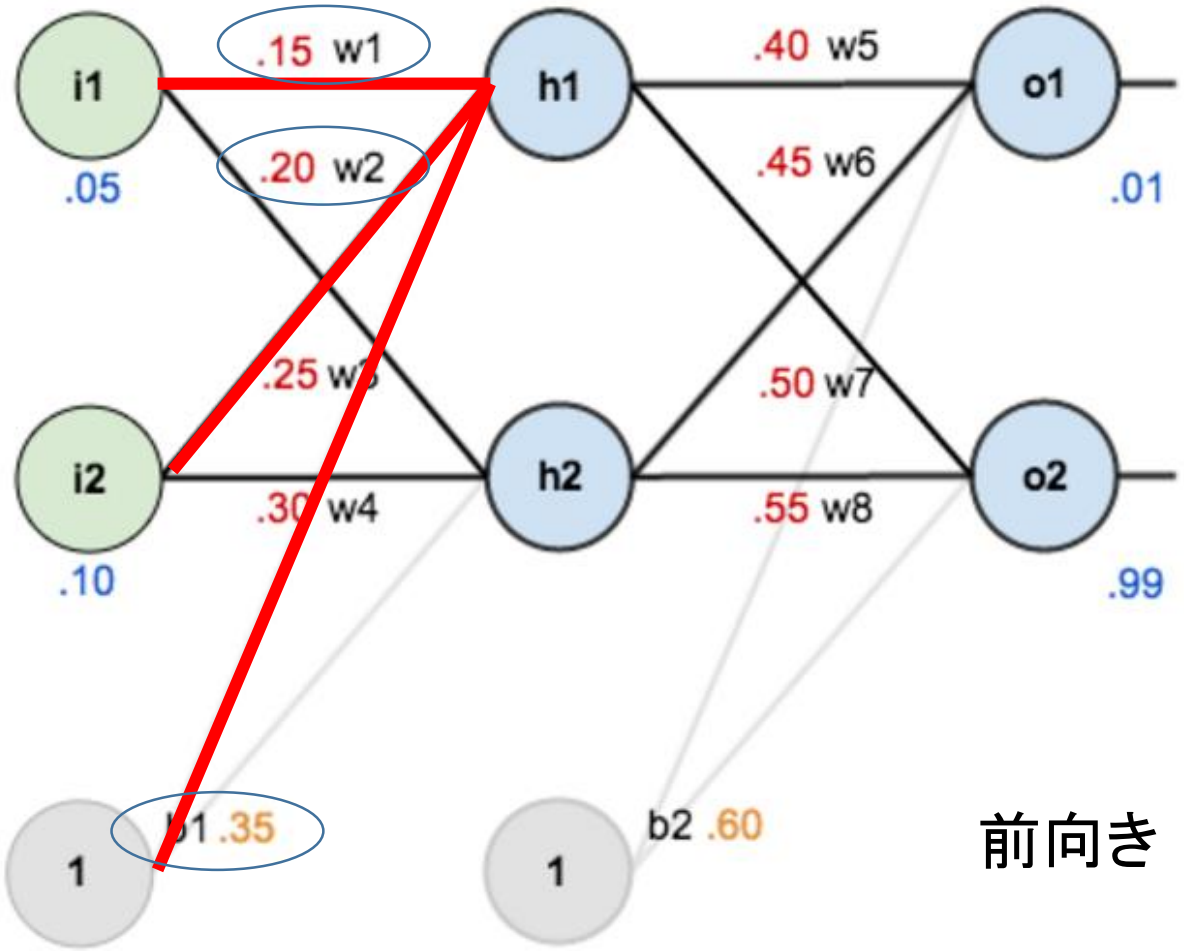
1. ニューラルネットワークに学習のためのサンプルを与える。
2. ネットワークの出力とそのサンプルの最適解を比較する。各出力ニューロンについて誤差を計算する。
3. 個々のニューロンの期待される出力値と倍率(scaling factor)、要求された出力と実際の出力の差を計算する。これを局所誤差と言う。
4. 各ニューロンの重みを局所誤差が小さくなるよう調整する。
5. より大きな重みで接続された前段のニューロンに対して、局所誤差の責任があると判定する。
6. そのように判定された前段のニューロンのさらに前段のニューロン群について同様の処理を行う。

https://ja.wikipedia.org/wiki/%E3%83%90%E3%83%83%E3%82%AF%E3%83%97%E3%83%AD%E3%83%91%E3%82%B2%E3%83%BC%E3%82%B7%E3%83%A7%E3%83%B3

誤差逆伝播法


入力
隠しニューロン
出力
i1
i2
h1
h2
o1
o2
.05
.10
.01
.99
.15 w1
.20 w2
.25 w3
.30 w4
.40 w5
.45 w6
.50 w7
.55 w8
1
1
b1 .35
b2 .60
バイアス

誤差逆伝播法（h_1導出）



前向き

誤差逆伝播法(h_1)

- 各隠れ層のニューロンへの全ての入力を使い(net_{h1})、活性化関数(シグモイド関数)に入れる。

$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

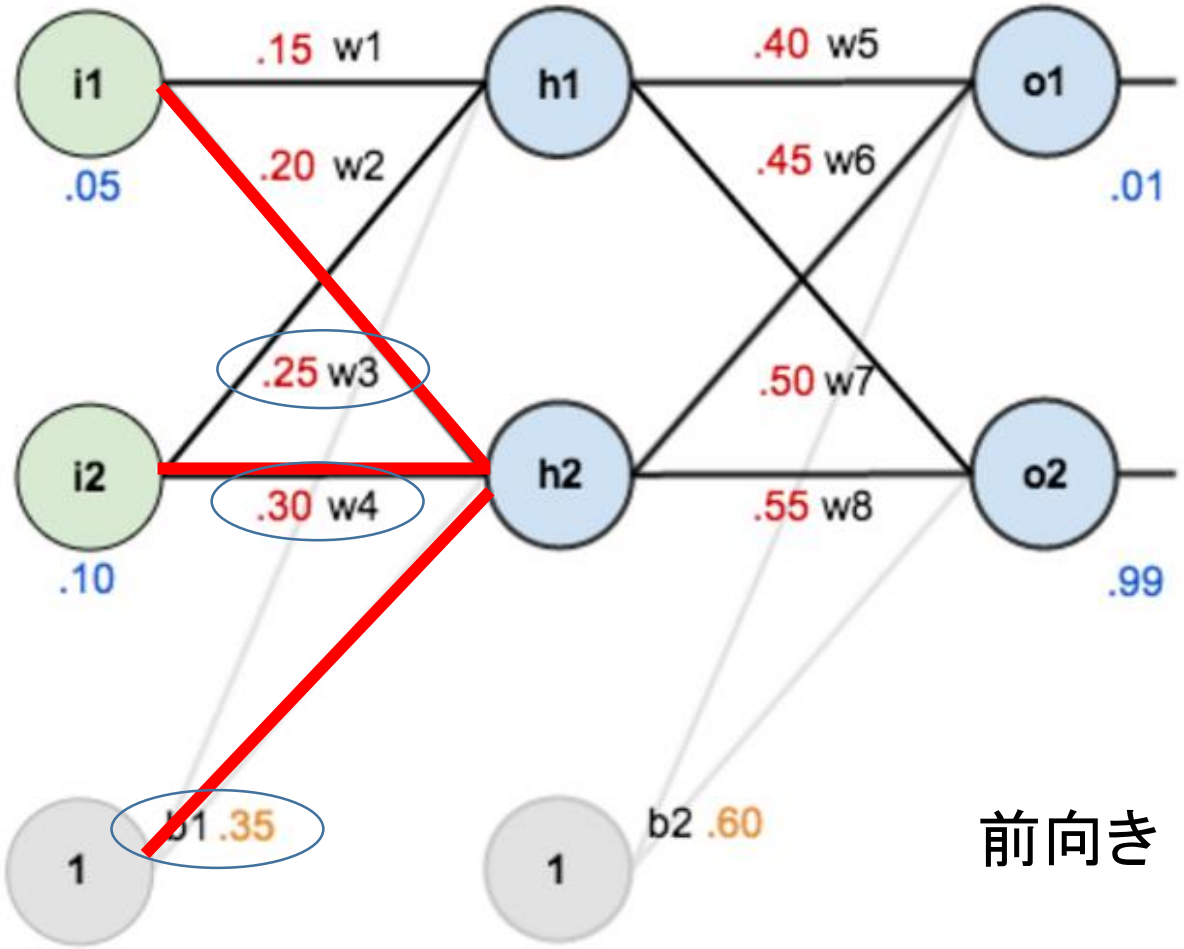
$$net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775$$

- これをシグモイド関数に入れる。

$$out_{h1} = \frac{1}{1+e^{-net_{h1}}} = \frac{1}{1+e^{-0.3775}} = 0.593269992$$

- この隠れ層ニューロンからの出力out_{h1}を、出力層への入力にする。

誤差逆伝播法（h_2導出）

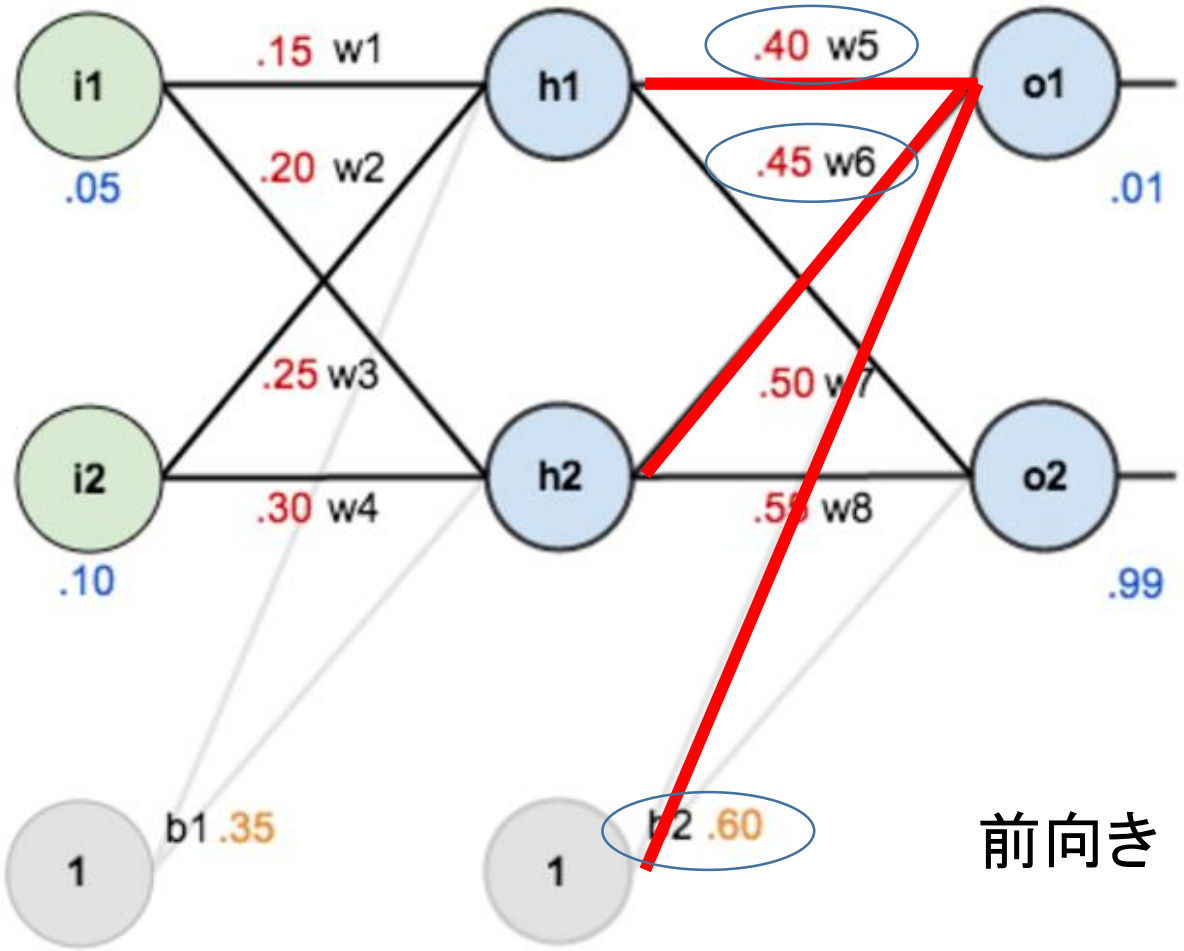


前向き

誤差逆伝播法（h_2）

- h_1の時と同様に
- net_{h2}=w_3*i_1+w_4*i_2+b_1*1
  =0.25*0.05+0.3*0.1+0.35*1=0.3925
- これをシグモイド関数に入れる。
- out_{h2}=1/(1 + $e^{-0.3925}$) = 0.596884378

誤差逆伝播法（出力層o_1）



誤差逆伝播法（o_1導出）

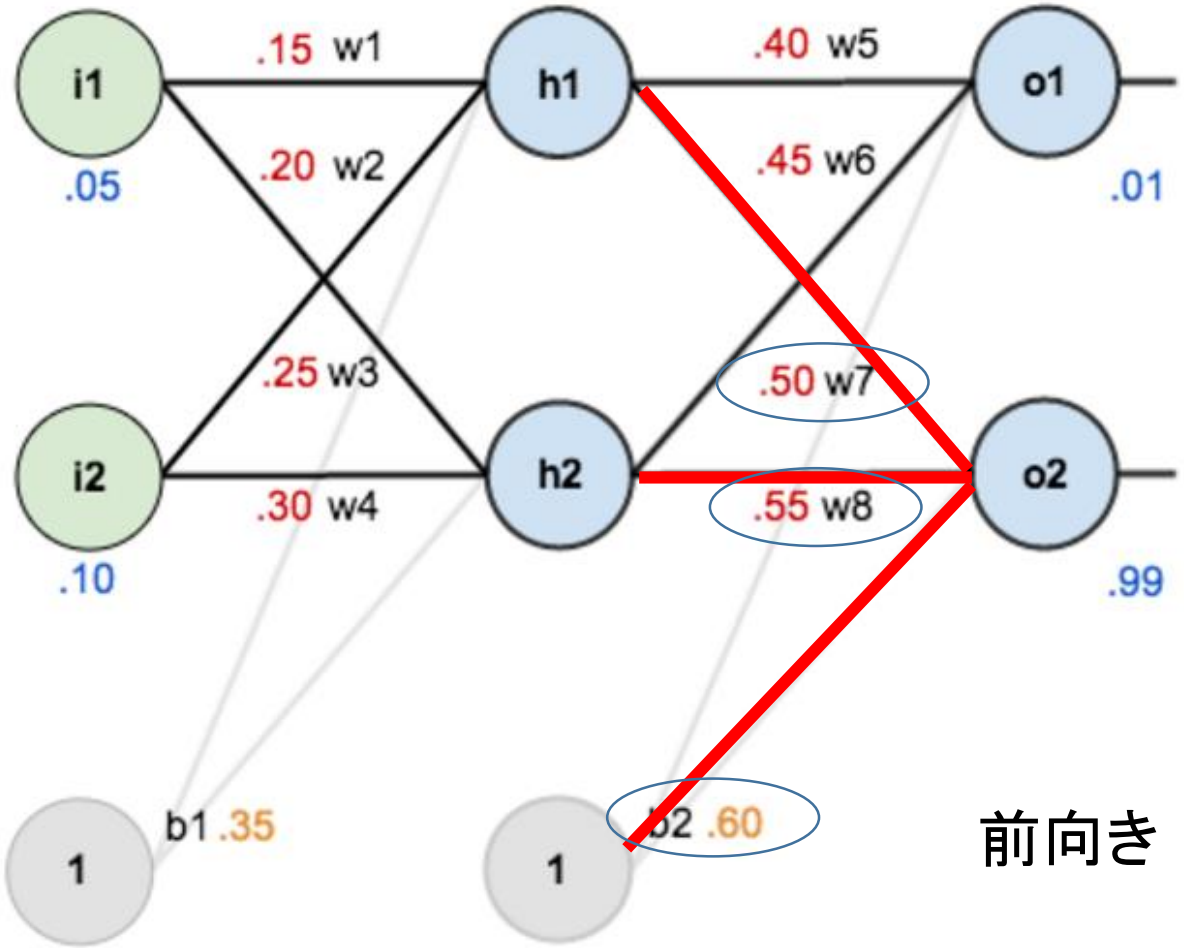
- 先程得た隠れ層の出力out_{h1}, out_{h2}を入力とし、出力層での出力を出す。

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$net_{o1} = 0.4 * 0.593269992 + 0.45 * 0.596884378 + 0.6 * 1 = 1.105905967$$

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}} = \frac{1}{1+e^{-1.105905967}} = 0.75136507$$

誤差逆伝播法（o_2導出）


i1
i2
h1
h2
o1
o2
1
1
.15 w1
.20 w2
.25 w3
.30 w4
.40 w5
.45 w6
.50 w7
.55 w8
b1 .35
b2 .60
.05
.10
.01
.99
前向き

誤差逆伝播法（o_2）

- o_1の時と同様に
- net_{o2}=w_7*out_{h1}+w_8*out_{h2}+b_2*1

  =0.5*0.593269992+0.55*0.596884378+0.6*1=1.2249214
- これをシグモイド関数に入れる。
- out_{o2}=1/(1 + $e^{-1.2249214}$) = 0.772928465

総誤差計算

- 出力層の各ニューロンにおける、それぞれの出力と実際に期待した値との差の二乗に1/2をかけたものの合計を求める。

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

総誤差計算

- 総誤差は、今回はE_{o1}とE_{o2}の合計を求めればいいので、分けて考えて、

$$E_{o1} = \frac{1}{2}(target_{o1} - out_{o1})^2 = \frac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083$$

- E_{o1}と同様に

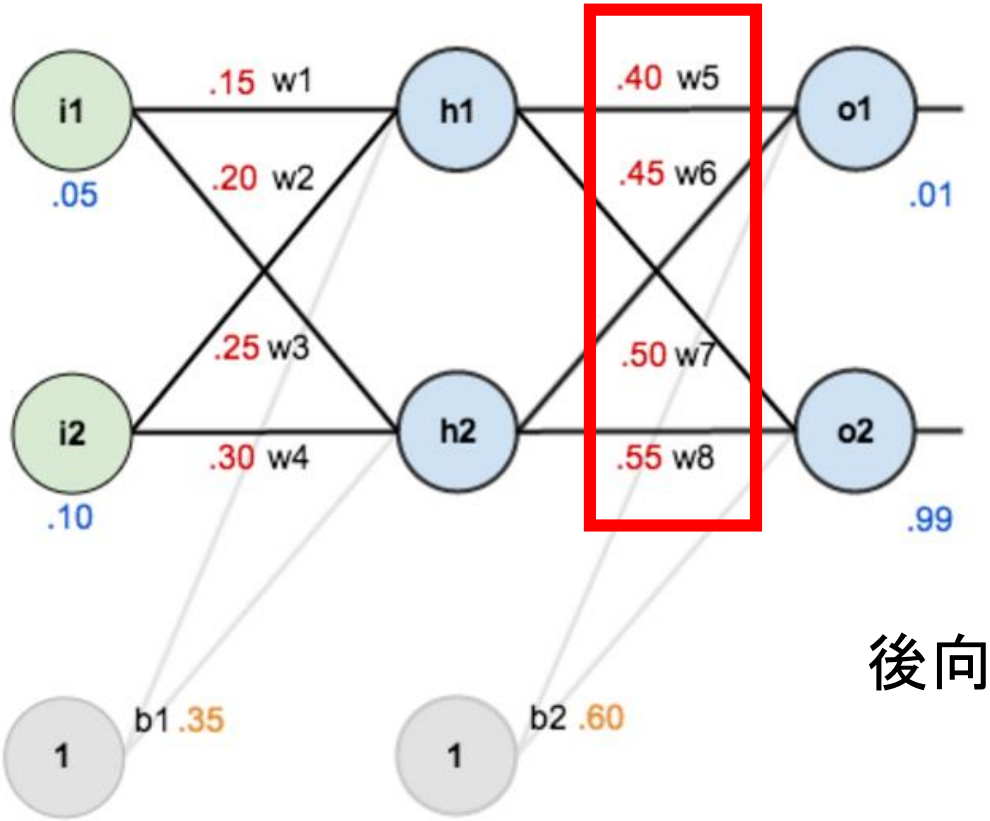
$$E_{o2} = 0.023560026$$

- 2つを足し合わせる。

$$E_{total} = E_{o1} + E_{o2} = 0.274811083 + 0.023560026 = 0.298371109$$

重みの更新

- ここまでの計算結果を、今度は出力層から前の層へ次々と利用して、各重みを更新する。後向き。

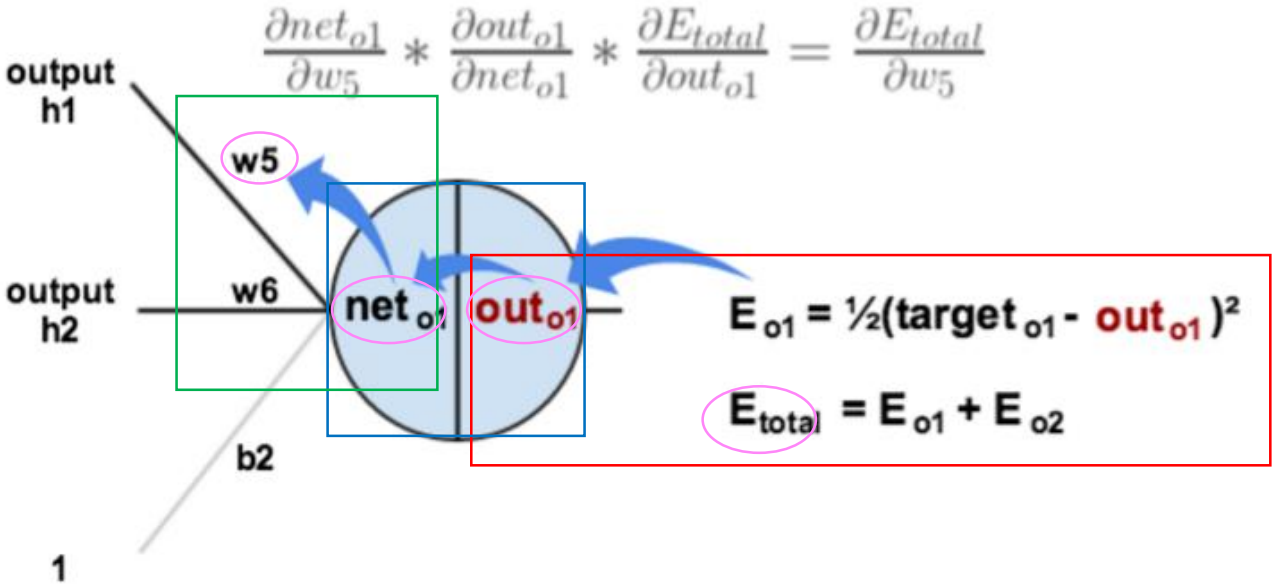


後向き

重みの更新(出力層)

- まず、出力層において、各重みを更新する。
  w_5に関しては次式に基づき求める。

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$



重みの更新(w_5)

- E_{total}をout_{o1}について微分した式の値を求める。

$$E_{total} = \frac{1}{2}(target_{o1} - out_{o1})^2 + \frac{1}{2}(target_{o2} - out_{o2})^2$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = 2 * \frac{1}{2}(target_{o1} - out_{o1})^{2-1} * -1 + 0$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = -(target_{o1} - out_{o1}) = -(0.01 - 0.75136507) = 0.74136507$$

重みの更新(w_5)

- out_{o1}の式をnet_{o1}について微分したものの値を求める。

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}}$$

$$\frac{\partial out_{o1}}{\partial net_{o1}} = out_{o1}(1 - out_{o1}) = 0.75136507(1 - 0.75136507) = 0.186815602$$

$$\varphi(z) = \frac{1}{1+e^{-z}}$$

which has a nice derivative of:

$$\frac{\partial \varphi}{\partial z} = \varphi(1-\varphi)$$

⇐

φ = 1/(e^(-z) + 1) = (e^(-z) + 1)^(-1)

∂φ/∂z = e^(-z)/(e^(-z) + 1)^2
= 1/(e^(-z) + 1)(e^(-z)+1-1)/(e^(-z) + 1)
= φ (1- φ)

重みの更新(w_5)

- 既に求めたneto1の式を、w_5について微分したものの値を求める。

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$\frac{\partial net_{o1}}{\partial w_5} = 1 * out_{h1} * w_5^{(1-1)} + 0 + 0 = out_{h1} = 0.593269992$$

重みの更新(w_5)

- 3つを全てまとめて計算する。

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

$$\frac{\partial E_{total}}{\partial w_5} = 0.74136507 * 0.186815602 * 0.593269992 = 0.082167041$$

重みの更新（出力層）

- w_5を次式に基づき更新する。η（学習率:0<η≤1）は0.5と今回は設定している。

$$w_5^+ = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5} = 0.4 - 0.5 * 0.082167041 = 0.35891648$$

- w_6, w_7, w_8も同様にすると、

$$w_6^+ = 0.408666186$$

$$w_7^+ = 0.511301270$$

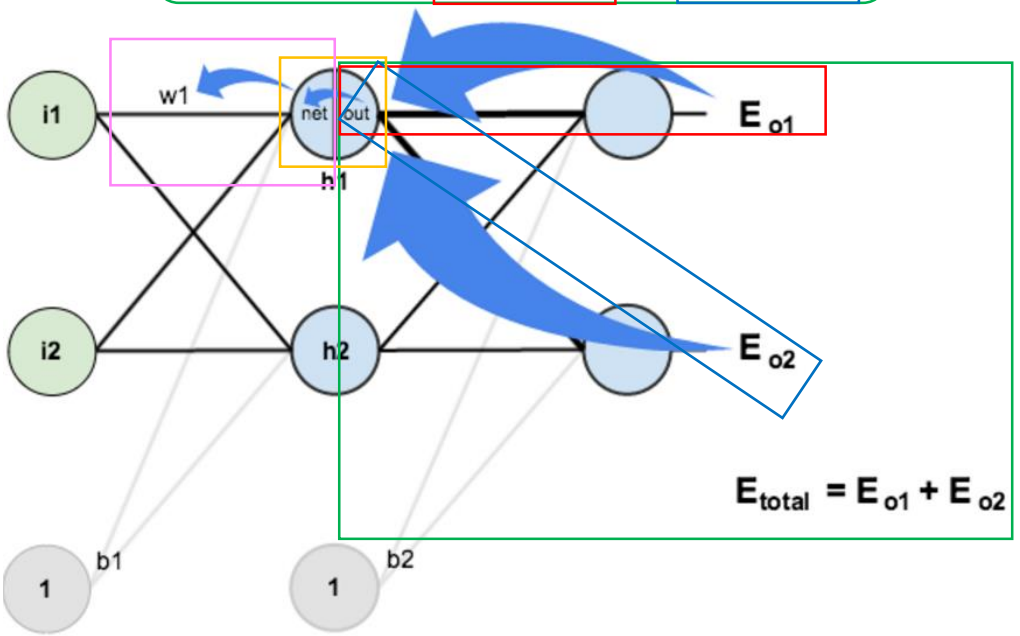
$$w_8^+ = 0.561370121$$

重みの更新(隠れ層)

- 次に、隠れ層において重みを更新する。w_1を例として。

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$



後向き

重みの更新(w_1)

- まずは、緑の部分を求める。その中の赤の部分について。

$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}}$$

- 部分ごとに求め、計算する。

$$\frac{\partial E_{o1}}{\partial net_{o1}} = \frac{\partial E_{o1}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} = 0.74136507 * 0.186815602 = 0.138498562$$

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$\frac{\partial net_{o1}}{\partial out_{h1}} = w_5 = 0.40$$

$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}} = 0.138498562 * 0.40 = 0.055399425$$

重みの更新（w_1）

- 青の部分について、同様に行う。

$$\frac{\partial E_{o2}}{\partial out_{h1}} = -0.019049119$$

- まとめて

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}} = 0.055399425 + -0.019049119 = 0.036350306$$

重みの更新(w_1)

- 残りの黄色部分について。

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$out_{h1} = \frac{1}{1+e^{-net_{h1}}}$$

$$\frac{\partial out_{h1}}{\partial net_{h1}} = out_{h1}(1 - out_{h1}) = 0.59326999(1 - 0.59326999) = 0.241300709$$

$$\varphi(z) = \frac{1}{1 + e^{-z}}$$

which has a nice derivative of:

$$\frac{\partial \varphi}{\partial z} = \varphi(1 - \varphi)$$

⇦

φ = 1/(e^(-z) + 1) = (e^(-z) + 1)^(-1)

∂φ/∂z = e^(-z)/(e^(-z) + 1)^2
= 1/(e^(-z) + 1)(e^(-z)+1-1)/(e^(-z) + 1)
= φ (1- φ)

重みの更新(w_1)

- 残りのピンク部分について。

$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

$$\frac{\partial net_{h1}}{\partial w_1} = i_1 = 0.05$$

重みの更新(w_1)

- 全てをまとめる。

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial w_1} = 0.036350306 * 0.241300709 * 0.05 = 0.000438568$$

重みの更新(w_1)

- w_1を更新する(η:0.5)。

$$w_1^+ = w_1 - \eta * \frac{\partial E_{total}}{\partial w_1} = 0.15 - 0.5 * 0.000438568 = 0.149780716$$

- 他も同様に

$$w_2^+ = 0.19956143$$

$$w_3^+ = 0.24975114$$

$$w_4^+ = 0.29950229$$

繰り返し

- 更新した重みで、5ページ目からの流れを繰り返し、また誤差を出し、重みを更新するという流れを設定回数だけ繰り返していく。
- プログラムでは1万回繰り返し、実行してみると少しずつ総誤差の値が小さくなっていることが分かる。

演習

- スライド内で計算の過程を端折った部分や、2回目、3回目の誤差を紙に式を書き表しながら、計算して出してみる。
- プログラムの結果と比較して、大体合っているか確認してみると良い。
- プログラムを解読してみる。

3年生輪読予定表改訂版

担当箇所	名前	発表予定日
1-1	保科	5/12
1-3	猪狩	5/12
2-1	村田 西澤	5/19
2-2 2-3	西ヶ谷 坂中	5/19
3-1	米倉 田之井	5/26
3-2	松田 清水	5/26

3年生輪読予定表改訂版

担当箇所	名前	発表予定日
3-3	宿野 前原 村田	6/2
4章	門木 山下 上野 數見	6/9 6/16
5章	下窪 黒川 関根 坂本	6/16

3年生向け補足

- プログラムのデモンストレーション。

3年生向け補足(変更可能パラメータ)

- xを自分の使用データに合わせて設定する。[None,2]の数字の部分。データ1つにおける次元数。
- 多クラスの分類を行う場合は、
  y_ = tf.placeholder(tf.float32, [None, 20])　(20はクラス数、次元数)等として正解クラスの入れ場所も定義しておく。

```
num_units1 = 2
num_units2 = 2

x = tf.placeholder(tf.float32, [None, 2])

w1 = tf.Variable(tf.truncated_normal([2, num_units1]))
b1 = tf.Variable(tf.zeros([num_units1]))
hidden1 = tf.nn.tanh(tf.matmul(x, w1) + b1)

w2 = tf.Variable(tf.truncated_normal([num_units1, num_units2]))
b2 = tf.Variable(tf.zeros([num_units2]))
hidden2 = tf.nn.tanh(tf.matmul(hidden1, w2) + b2)

w0 = tf.Variable(tf.zeros([num_units2, 1]))
b0 = tf.Variable(tf.zeros([1]))
p = tf.nn.sigmoid(tf.matmul(hidden2, w0) + b0)
```

3年生向け補足(変更可能パラメータ)

- 多クラスの分類を行う場合は、
  y = tf.nn.softmax(tf.matmul(h4, w_o) + b_o)
  等として推定クラスの入れ場所も定義しておく。これは出力層に該当する。
  下のコードの場合、pがそれに該当する(多クラス分類ではないが)。

```
num_units1 = 2
num_units2 = 2

x = tf.placeholder(tf.float32, [None, 2])

w1 = tf.Variable(tf.truncated_normal([2, num_units1]))
b1 = tf.Variable(tf.zeros([num_units1]))
hidden1 = tf.nn.tanh(tf.matmul(x, w1) + b1)

w2 = tf.Variable(tf.truncated_normal([num_units1, num_units2]))
b2 = tf.Variable(tf.zeros([num_units2]))
hidden2 = tf.nn.tanh(tf.matmul(hidden1, w2) + b2)

w0 = tf.Variable(tf.zeros([num_units2, 1]))
b0 = tf.Variable(tf.zeros([1]))
p = tf.nn.sigmoid(tf.matmul(hidden2, w0) + b0)
```

3年生向け補足(変更可能パラメータ)

- ユニット数(num_units、層毎に設定)
  該当する層の重みやバイアスと関連付ける。
- 隠れ層(中間層)の数
  層毎に、重み(w)、バイアス(b)や層の結びつきの関係(hidden)をちゃんと書くこと。(特に、最終層における参照している隠れ層を、一番最後の隠れ層にする等。)

```
num_units1 = 2
num_units2 = 2

x = tf.placeholder(tf.float32, [None, 2])

w1 = tf.Variable(tf.truncated_normal([2, num_units1]))
b1 = tf.Variable(tf.zeros([num_units1]))
hidden1 = tf.nn.tanh(tf.matmul(x, w1) + b1)

w2 = tf.Variable(tf.truncated_normal([num_units1, num_units2]))
b2 = tf.Variable(tf.zeros([num_units2]))
hidden2 = tf.nn.tanh(tf.matmul(hidden1, w2) + b2)

w0 = tf.Variable(tf.zeros([num_units2, 1]))
b0 = tf.Variable(tf.zeros([1]))
p = tf.nn.sigmoid(tf.matmul(hidden2, w0) + b0)
```

3年生向け補足(変更可能パラメータ)

- 学習率（3行目の0.001の部分）
  データ数や学習回数等が膨れ上がった際に、下げる方向で調整する。

```
t = tf.placeholder(tf.float32, [None, 1])
loss = -tf.reduce_sum(t*tf.log(p) + (1-t)*tf.log(1-p))
train_step = tf.train.GradientDescentOptimizer(0.001).minimize(loss)
correct_prediction = tf.equal(tf.sign(p-0.5), tf.sign(t-0.5))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

3年生向け補足(変更可能パラメータ)

- correct_prediction用の設定は、分類対象によって使用関数を変える。二項分類ならばシグモイド関数、多クラスならばソフトマックス関数が良い。予測と正解の一致具合を計る正答判定に関しても記述に違いがある。後者は推定結果と正解のラベルから最も数値が高いものを取り出して比較することを意味する。
  例:予測が[0.0,0.2,0.3,0.0,0.5]⇒　4
  正解ラベル[0,0,0,0,1]⇒4　4=4,つまり両者一致。

```
    p = tf.nn.sigmoid(tf.matmul(hidden2, w0) + b0)
correct_prediction = tf.equal(tf.sign(p-0.5), tf.sign(t-0.5))
```

```
      y = tf.nn.softmax(tf.matmul(h4, w_o) + b_o)
correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
```

```
labels=[
[1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
]
```

3年生向け補足(変更可能パラメータ)

- signやargmax等、比較して判定に使用できる記述。
  参照:
  http://mirai-tec.hatenablog.com/entry/2016/02/22/001459

```
    p = tf.nn.sigmoid(tf.matmul(hidden2, w0) + b0)
correct_prediction = tf.equal(tf.sign(p-0.5), tf.sign(t-0.5))
```

```
      y = tf.nn.softmax(tf.matmul(h4, w_o) + b_o)
correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
```

```
labels=[
[1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
]
```

3年生向け補足(変更可能パラメータ)

- 最適化・学習回数(2000の部分)。
- また、ミニバッチ学習で1度あたりに取り出すデータの数を決めるとしたら(ミニバッチサイズ)、学習部分の中になる。(乱数を発生させて、そのインデックスのデータを取り出す等、各自工夫する。)

```
i = 0
for _ in range(2000):
    i += 1
    sess.run(train_step, feed_dict={x:train_x, t:train_t})
    if i % 100 == 0:
        loss_val, acc_val = sess.run(
            [loss, accuracy], feed_dict={x:train_x, t:train_t})
        print ('Step: %d, Loss: %f, Accuracy: %f'
               % (i, loss_val, acc_val))
```

次回

- 誤差逆伝播法の応用？
- 輪読(4章)。