

自然言語処理プログラミング勉強会 7 ニューラルネット

Graham Neubig
奈良先端科学技術大学院大学 (NAIST)

予測問題

x が与えられた時
 y を予測する

今回の例

- Wikipedia 記事の最初の 1 文が与えられた時
- その記事が人物についての記事かどうかを予測

与えられた情報

Gonso was a Sanron sect priest (754-827)
in the late Nara and early Heian periods.

予測

Yes!

Shichikuzan Chigogataki Fudomyoo is
a historical site located at Magura, Maizuru
City, Kyoto Prefecture.



No!

- これはもちろん、2 値予測

線形識別器

$$\begin{aligned}y &= \text{sign}(\mathbf{w} \cdot \boldsymbol{\varphi}(x)) \\&= \text{sign}\left(\sum_{i=1}^I w_i \cdot \varphi_i(x)\right)\end{aligned}$$

- x : 入力
- $\boldsymbol{\varphi}(x)$: 素性関数のベクトル $\{\varphi_1(x), \varphi_2(x), \dots, \varphi_I(x)\}$
- \mathbf{w} : 重みベクトル $\{w_1, w_2, \dots, w_I\}$
- y : 予測値、「yes」なら +1、「no」なら -1
 - $\text{sign}(v)$ は 「 $v \geq 0$ 」 の場合 +1、そうでない場合 -1

素性関数の例：1-gram 素性

- 「事例において、ある単語が何回現れるか？」

$x = \text{A site , located in Maizuru , Kyoto}$

$$\varphi_{\text{unigram "A"}}(x) = 1 \quad \varphi_{\text{unigram "site"}}(x) = 1 \quad \varphi_{\text{unigram ","}}(x) = 2$$

$$\varphi_{\text{unigram "located"}}(x) = 1 \quad \varphi_{\text{unigram "in"}}(x) = 1$$

$$\varphi_{\text{unigram "Maizuru"}}(x) = 1 \quad \varphi_{\text{unigram "Kyoto"}}(x) = 1$$

$$\varphi_{\text{unigram "the"}}(x) = 0 \quad \varphi_{\text{unigram "temple"}}(x) = 0 \quad \dots \quad \left. \right\} \text{ 残りはすべて } 0$$

- 便宜のため、素性 $\text{ID}(\varphi_1)$ の代わりに、素性の名前 ($\varphi_{\text{unigram "A"}}$) を利用

重み付き和の計算

$x = \text{A site , located in Maizuru , Kyoto}$

$\varphi_{\text{unigram "A"}}(x)$	= 1	$w_{\text{unigram "a"}}$	= 0	0	+
$\varphi_{\text{unigram "site"}}(x)$	= 1	$w_{\text{unigram "site"}}$	= -3	-3	+
$\varphi_{\text{unigram "located"}}(x)$	= 1	$w_{\text{unigram "located"}}$	= 0	0	+
$\varphi_{\text{unigram "Maizuru"}}(x)$	= 1	$w_{\text{unigram "Maizuru"}}$	= 0	0	+
$\varphi_{\text{unigram ","}}(x)$	= 2 *	$w_{\text{unigram ","}}$	= 0	= 0	+
$\varphi_{\text{unigram "in"}}(x)$	= 1	$w_{\text{unigram "in"}}$	= 0	0	+
$\varphi_{\text{unigram "Kyoto"}}(x)$	= 1	$w_{\text{unigram "Kyoto"}}$	= 0	0	+
$\varphi_{\text{unigram "priest"}}(x)$	= 0	$w_{\text{unigram "priest"}}$	= 2	0	+
$\varphi_{\text{unigram "black"}}(x)$	= 0	$w_{\text{unigram "black"}}$	= 0	0	+

...

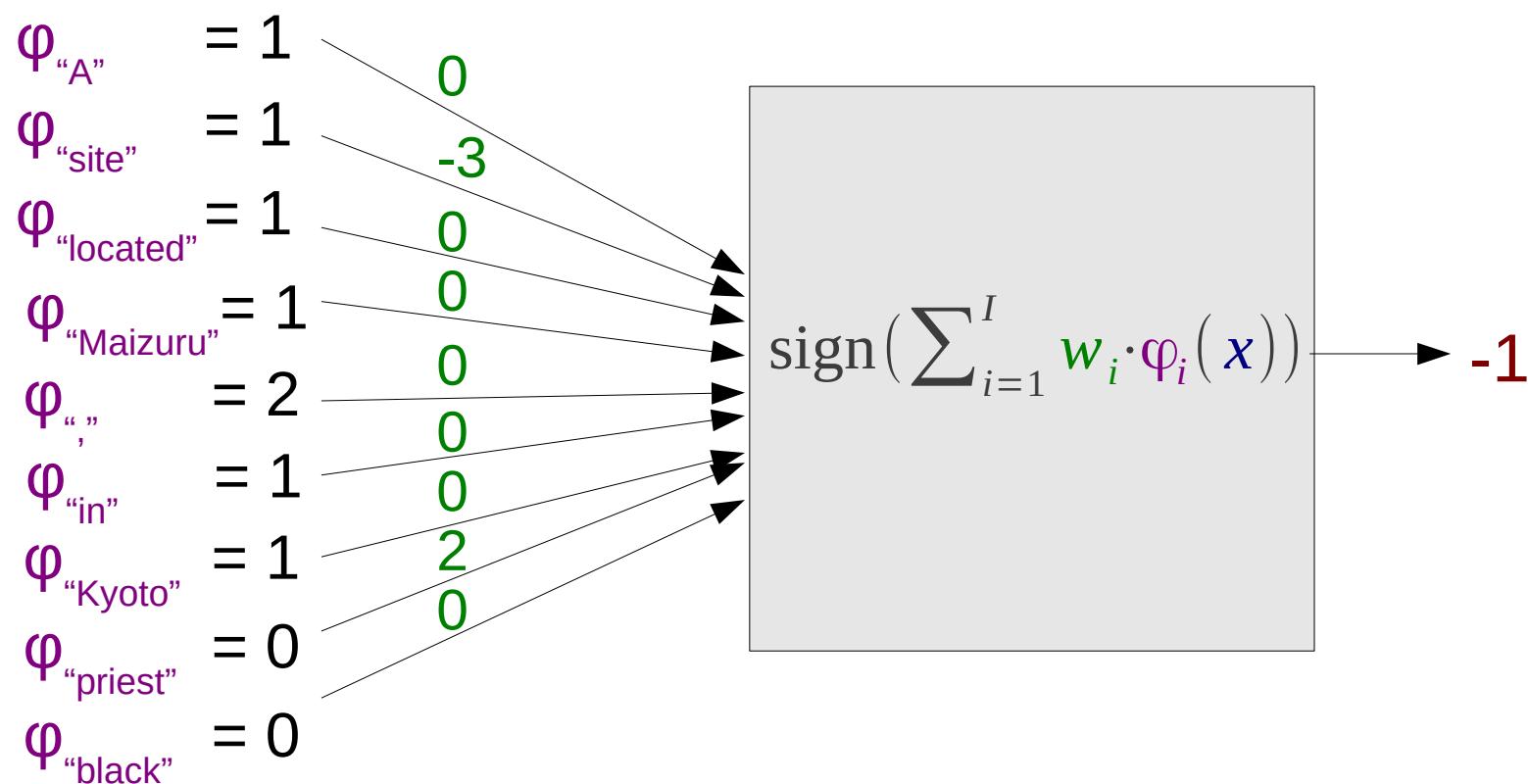
...

=

-3 → No!

パーセプトロン

- 重み付き和を計算する「機械」として考える



numpy でパーセプトロン

numpy とは

- Python で利用できる、強力な計算ライブラリ
- ベクトルや行列の掛け算などが簡単
- [SciPy](#) というパッケージの一部（SciPy は機械学習アルゴリズムなども実装）

numpy の使用例（ベクトル編）

```
import numpy as np

a = np.array( [20,30,40,50] )
b = np.array( [0,1,2,3] )
print(a - b)                      # Subtract each element
print(b ** 2)                      # Take the power of each element
print(10 * np.tanh(b))             # Hyperbolic tangent * 10 of each element
print(a < 35)                      # Check if each element is less than 35
```

numpy の使用例（行列編）

```
import numpy as np

A = np.array( [[1,1],
               [0,1]] )
B = np.array( [[2,0],
               [3,4]] )

print(A * B)          # elementwise product
print(np.dot(A,B))    # dot product
print(B.T)            # transpose
```

パーセプトロンの予測

```
predict_one(w, phi)
    score = 0
    for each name, value in phi          # score = w* $\varphi(\mathbf{x})$ 
        if name exists in w
            score += value * w[name]
    return (1 if score >= 0 else -1)
```

↓ numpy

```
predict_one(w, phi)
    score = np.dot( w, phi )
    return (1 if score[0] >= 0 else -1)
```

素性の ID 化

- numpy は密行列を利用 → 素性を整数 ID に変えたい !

ids = defaultdict(lambda: len(*ids*)) # ID に変換するトリック !

CREATE_FEATURES(*x*):

create list *phi*

split *x* **into** words

for word **in** words

phi[*ids*[“UNI:”+word]] += 1

return *phi*

素性の初期化

- 素性の数だけのベクトルを初期化
- ゼロで初期化

```
w = np.zeros(len(ids))
```

- [-0.5,0.5] でランダムに初期化

```
w = np.random.rand(len(ids)) - 0.5
```

パーセプトロン学習の疑似コード

```
# 素性の数を数えて、重みを初期化する
create map ids
for each labeled pair x, y in the data
    create_features(x)
    w = np.zeros(len(ids))

# 学習を行う
for / iterations
    for each labeled pair x, y in the data
        phi = create_features(x)
        y' = predict_one(w, phi)
        if y' != y
            update_weights(w, phi, y)

print w to weight_file
print ids to id_file
```

パーセプトロン予測の疑似コード

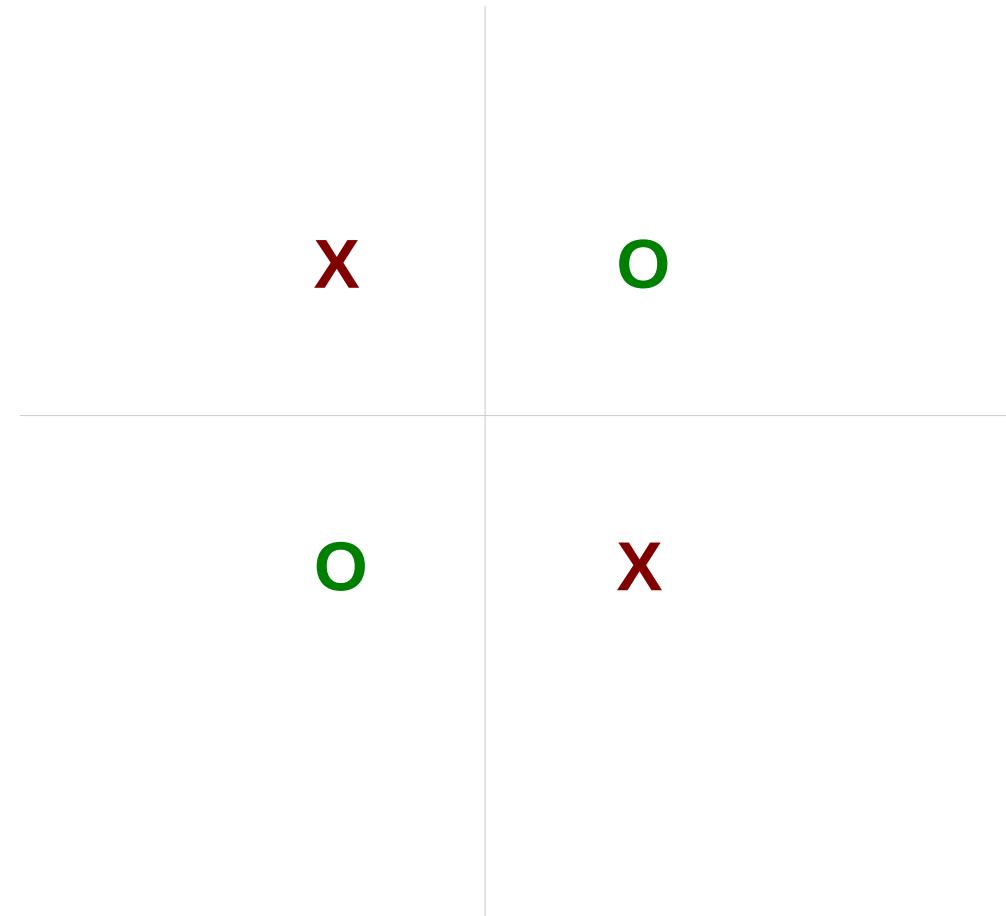
```
read ids from id_file
read w from weights_file

for each example x in the data
    phi = create_features(x)
    y' = predict_one(w, phi)
```

ニューラルネット

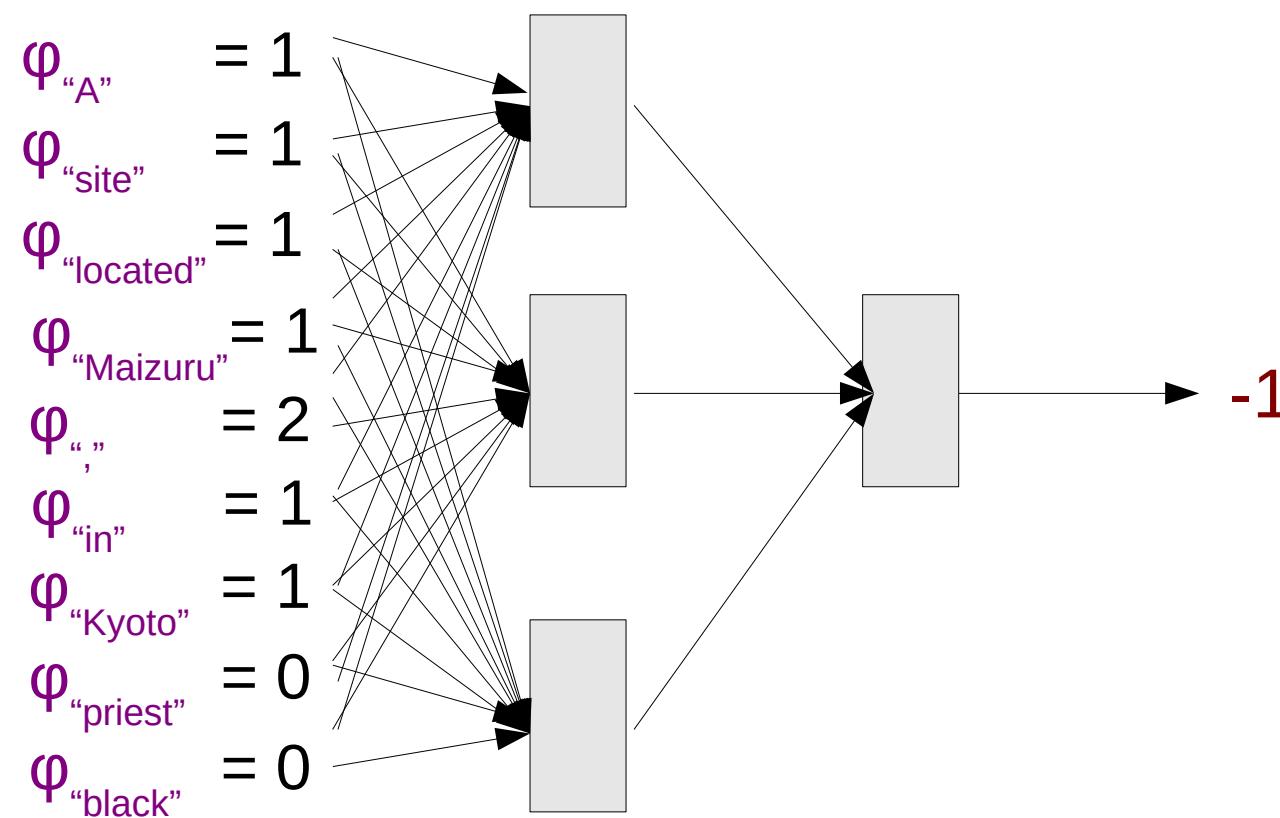
問題：線形分類のみ

- 線形分離不可能な問題に対して高い精度は実現不可



ニューラルネット

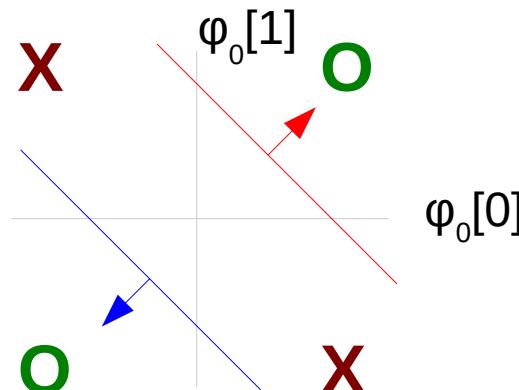
- 複数のパセプトロンをつなげる



- モチベーション：線形でない関数も表現可能！

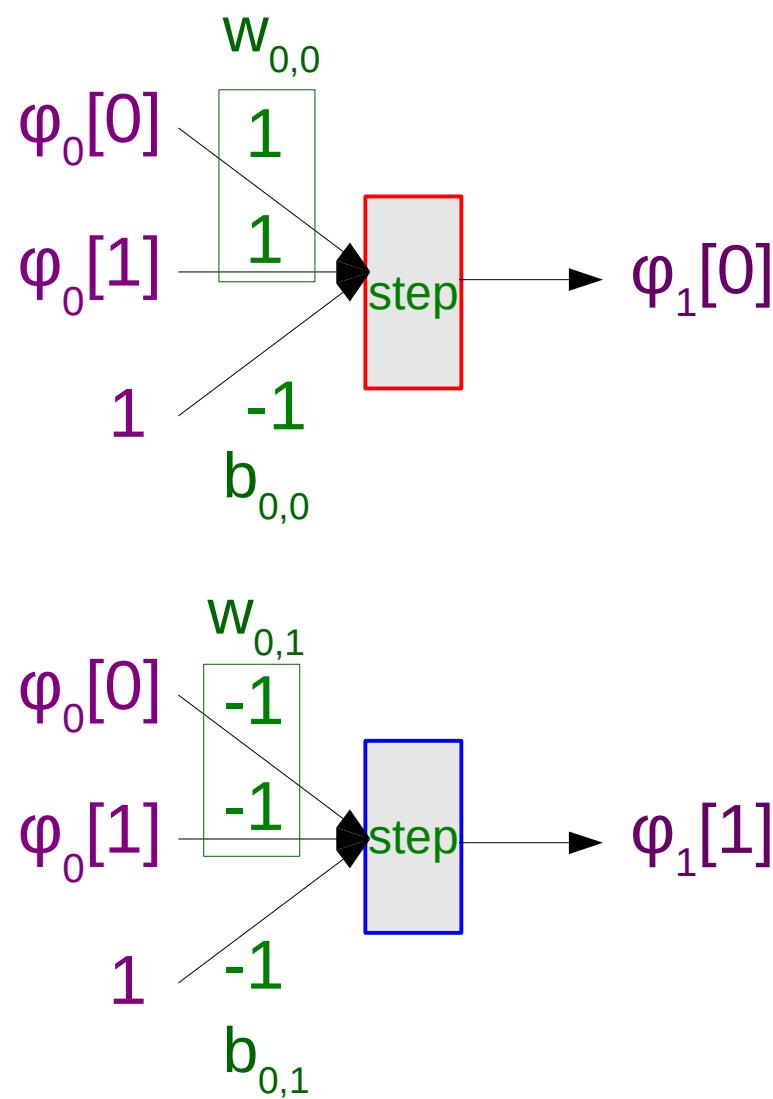
- 2つの分類器を作成

$$\varphi_0(x_1) = \{-1, 1\} \quad \varphi_0(x_2) = \{1, 1\}$$



$$\varphi_0(x_3) = \{-1, -1\} \quad \varphi_0(x_4) = \{1, -1\}$$

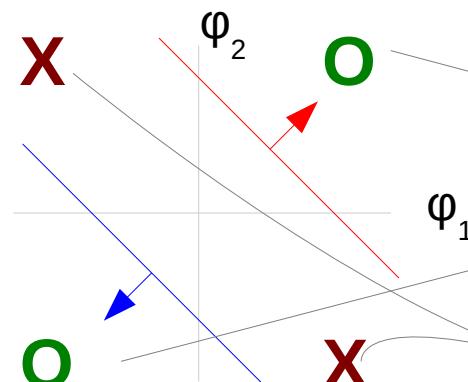
例：



例：

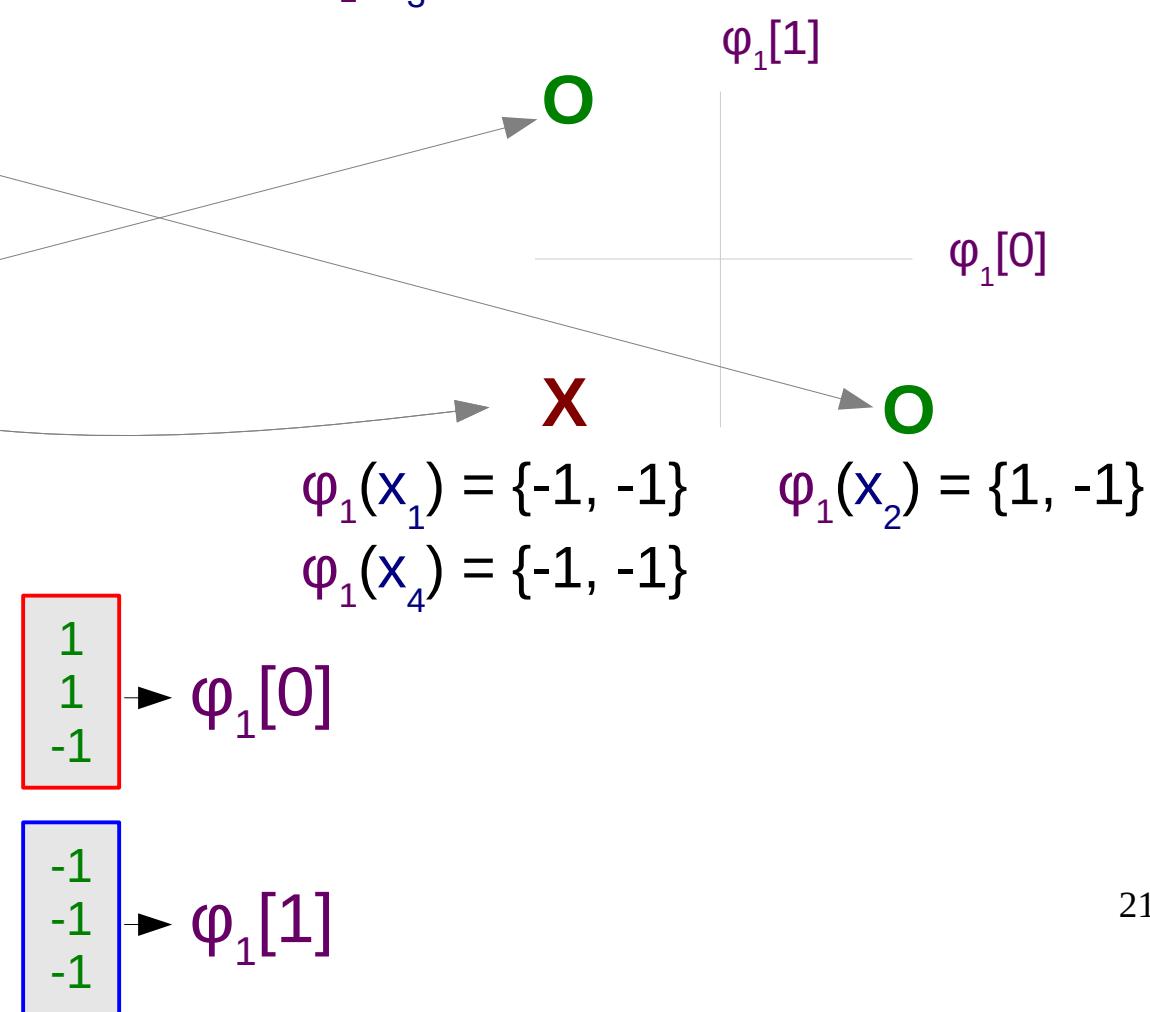
- 分類器は新しい属性空間へマッピング

$$\varphi_0(x_1) = \{-1, 1\} \quad \varphi_0(x_2) = \{1, 1\}$$



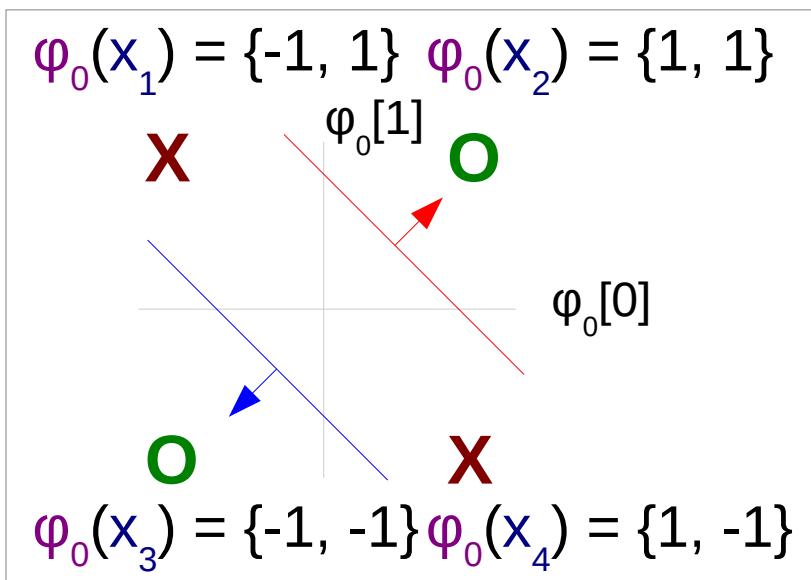
$$\varphi_1(x_3) = \{-1, 1\}$$

$$\varphi_0(x_3) = \{-1, -1\} \quad \varphi_0(x_4) = \{1, -1\}$$

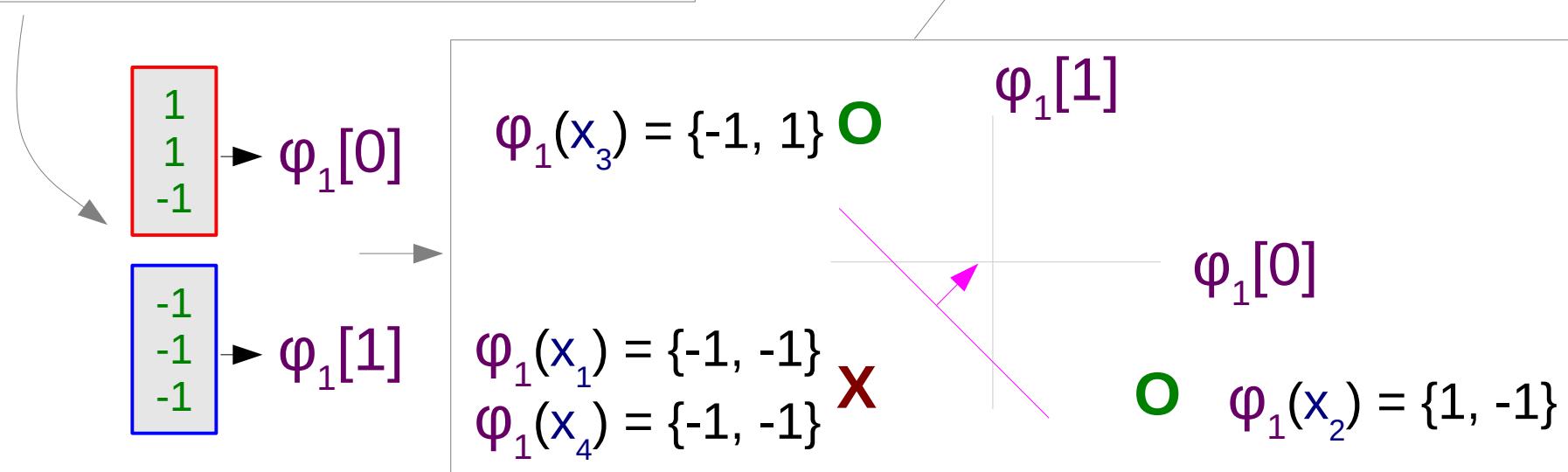


例：

- 新しい空間で、事例が分類可能に！

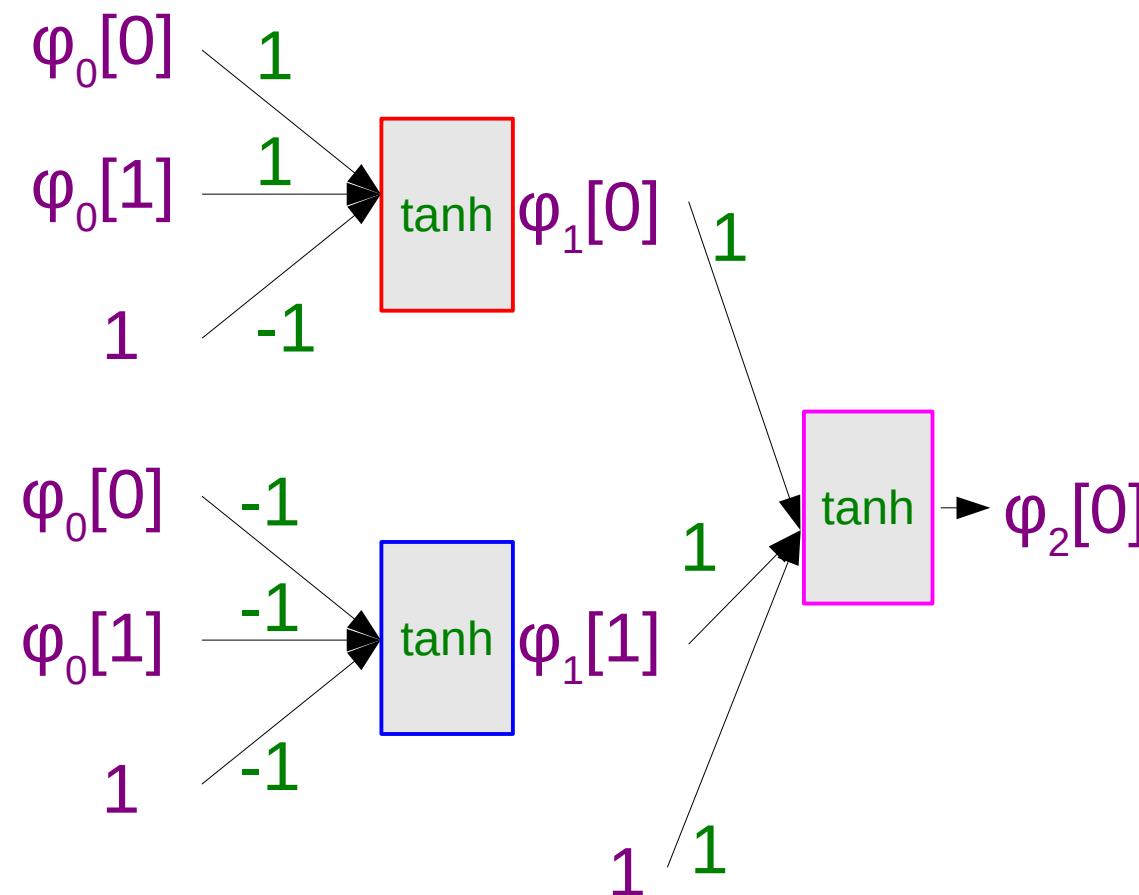


$$\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \rightarrow \varphi_2[0] = y$$



例：

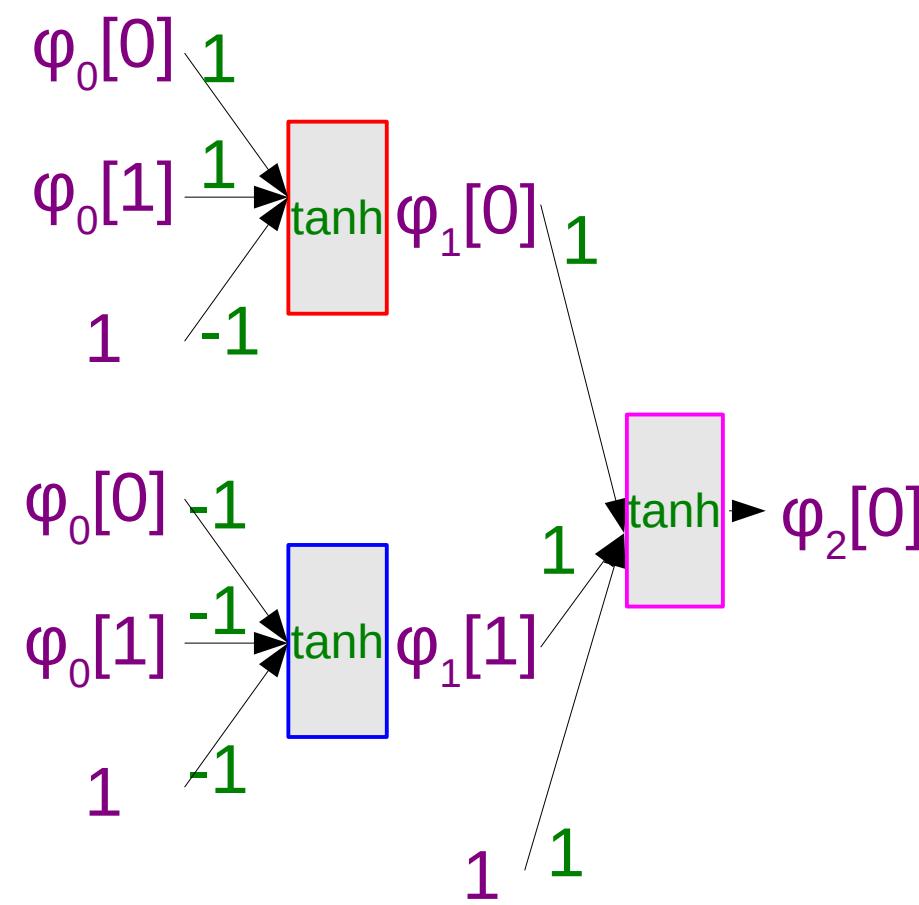
- 最終的なニューラルネット



2層ニューラルネットの例（ベクトル編）

入力

$$\varphi_0 = \text{np.array}([1, -1])$$



一層目の計算

$$w_{0,0} = \text{np.array}([1, 1])$$

$$b_{0,0} = \text{np.array}([-1])$$

$$w_{0,1} = \text{np.array}([-1, -1])$$

$$b_{0,1} = \text{np.array}([-1])$$

$$\varphi_1 = \text{np.zeros}(2)$$

$$\varphi_1[0] = \text{np.tanh}(\varphi_0 w_{0,0} + b_{0,0})[0]$$

$$\varphi_1[1] = \text{np.tanh}(\varphi_0 w_{0,1} + b_{0,1})[0]$$

2層目の計算

$$w_{1,0} = \text{np.array}([1, 1])$$

$$b_{1,0} = \text{np.array}([-1])$$

$$\varphi_2 = \text{np.zeros}(1)$$

$$\varphi_2[0] = \text{np.tanh}(\varphi_1 w_{1,0} + b_{1,0})[0]$$

24

2層ニューラルネットの例（行列編）

入力

$$\Phi_0 = \text{np.array}([1, -1])$$

一層目の計算

$$w_0 = \text{np.array}([[1, 1], [-1, -1]])$$

$$b_0 = \text{np.array}([-1, -1])$$

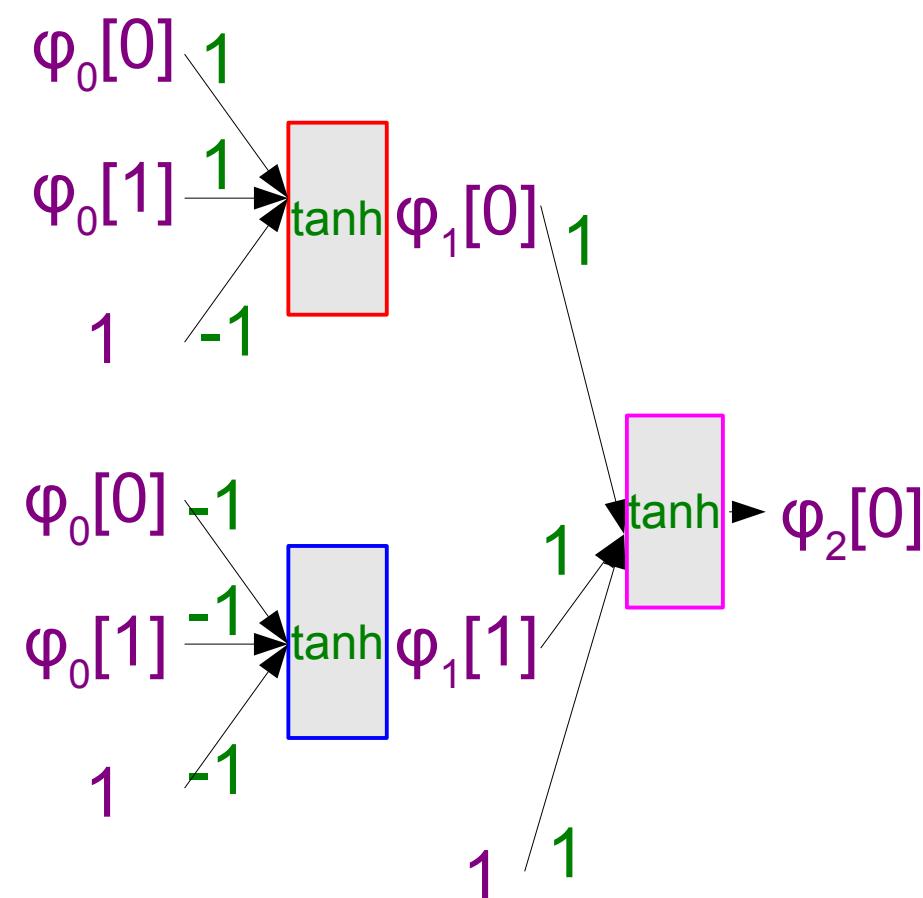
$$\Phi_1 = \text{np.tanh}(\text{np.dot}(w_0, \Phi_0) + b_0)$$

2層目の計算

$$w_1 = \text{np.array}([[1, 1]])$$

$$b_1 = \text{np.array}([-1])$$

$$\Phi_2 = \text{np.tanh}(\text{np.dot}(w_1, \Phi_1) + b_1)$$



ニューラルネットの伝搬コード

```
forward_nn(network,  $\varphi_0$ )
```

```
     $\varphi = [\varphi_0]$  # 各層の値
```

```
    for each layer i in 0 .. len(network)-1:
```

```
        w, b = network[i]
```

```
        # 前の層の値に基づいて値を計算
```

```
         $\varphi[i] = \text{np.tanh}(\text{np.dot}(\mathbf{w}, \varphi[i-1]) + \mathbf{b})$ 
```

```
    return  $\varphi$  # 各層の結果を返す
```

tanh を用いたパーセプトロン学習

- エラー関数：二乗誤差

$$\text{err} = (\mathbf{y}' - \mathbf{y})^2 / 2$$

正解 システム出力

- エラーの勾配：

$$\text{err}' = \delta = \mathbf{y}' - \mathbf{y}$$

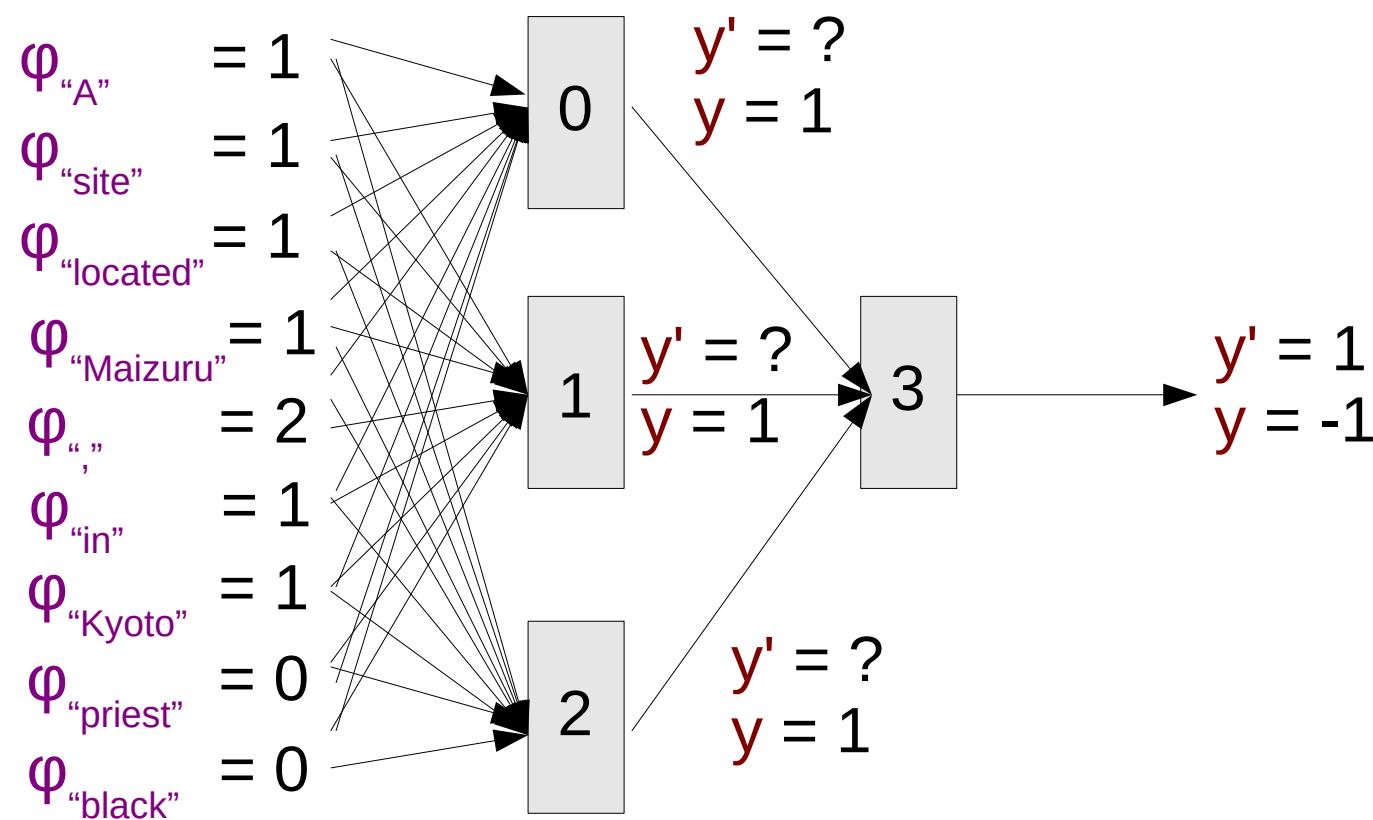
- 各重みを更新：

$$\mathbf{w} \leftarrow \mathbf{w} + \lambda \cdot \delta \cdot \varphi(\mathbf{x})$$

- λ は学習率

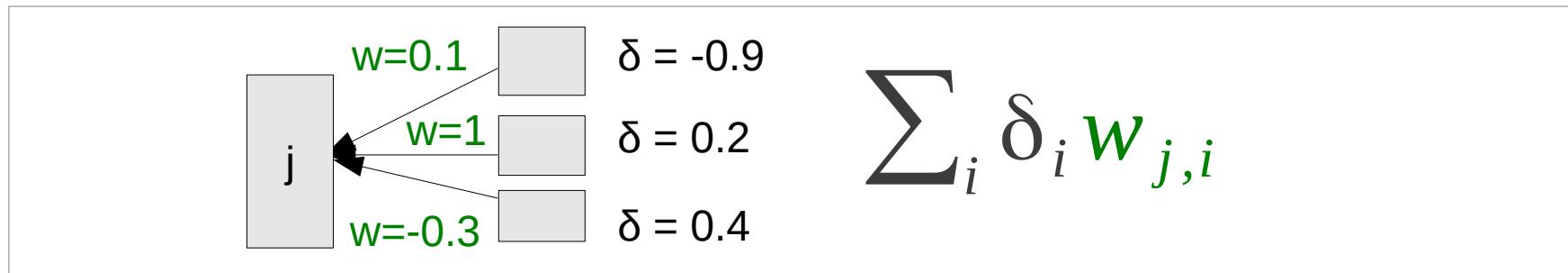
問題：正解は分からぬ！

- NN では出力層のみで正解が与えられる

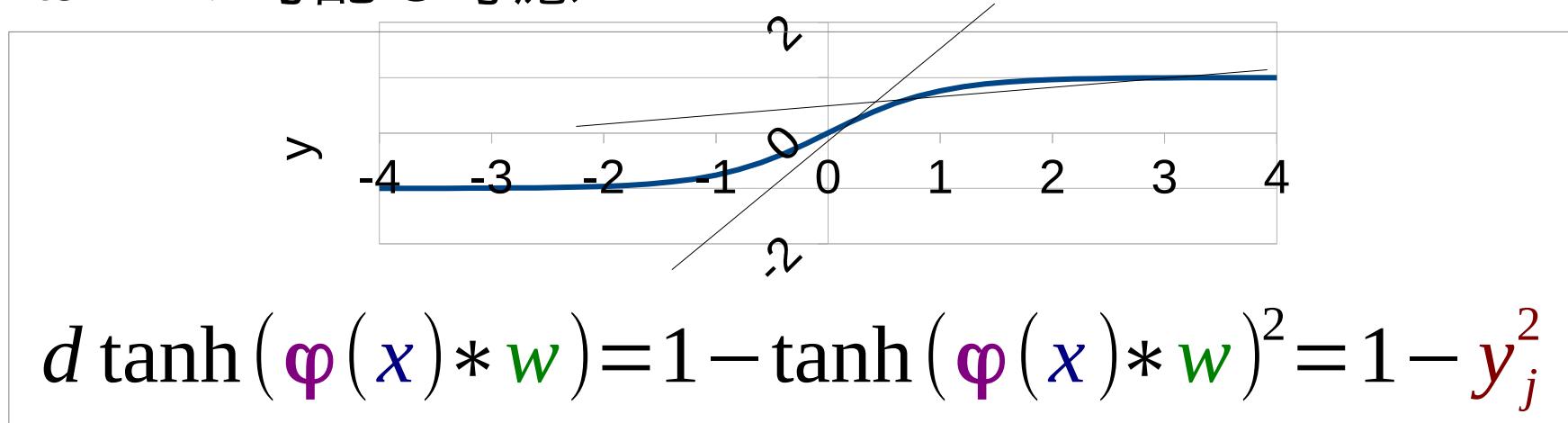


解決策：逆伝搬法

- 出力層からエラーを後ろへ伝搬



- \tanh の勾配も考慮



- 合わせて：

$$\delta_j = (1 - y_j^2) \sum_i \delta_i w_{j,i}$$

逆伝搬の例（行列編）

出力層のエラー

$$\delta_2 = \text{np.array}([y' - y])$$

1層目の計算

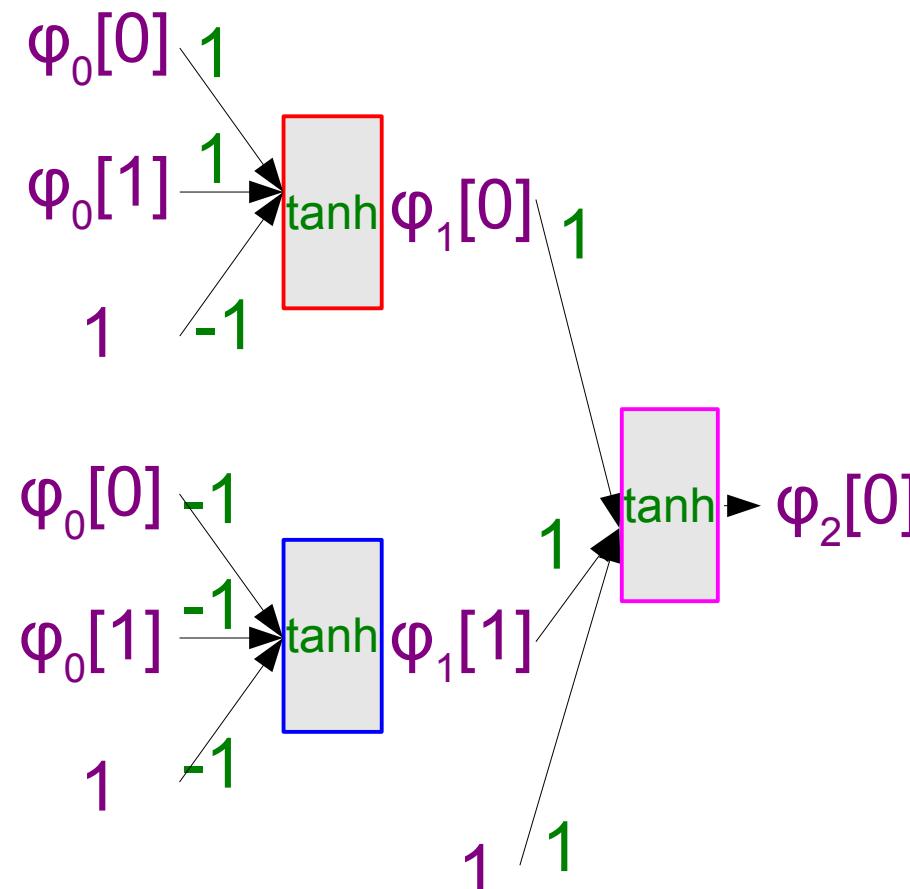
$$\delta'_2 = \delta_2 * (1 - \varphi_2^2)$$

$$\delta_1 = \text{np.dot}(\delta'_2, w_1)$$

0層目の計算

$$\delta'_1 = \delta_1 * (1 - \varphi_1^2)$$

$$\delta_0 = \text{np.dot}(\delta'_1, w_0)$$



逆伝搬のコード

```
backward_nn(net,  $\varphi$ ,  $y'$ )
```

```
J = len(net)
```

```
create array  $\delta$  = [ 0, 0, ..., np.array([ $y'$  –  $\varphi[J][0]$ ]) ] # length J+1
```

```
create array  $\delta'$  = [ 0, 0, ..., 0 ]
```

```
for i in J-1 .. 0:
```

```
     $\delta'[i+1] = \delta[i+1] * (1 - \varphi[i+1]^2)$ 
```

```
    w, b = net[i]
```

```
     $\delta[i] = \text{np.dot}(\delta'[i+1], \mathbf{w})$ 
```

```
return  $\delta'$ 
```

重み更新

- 最後に、重みを更新
- 重み w の勾配、次の δ' と、前の φ の外積で求める

$$-\text{derr}/\text{d}w_i = \text{np.outer}(\delta'_{i+1}, \varphi_i)$$

- 学習率をかけ、重みを更新

$$w_i += \lambda * -\text{derr}/\text{d}w_i$$

- バイアス項は単純に δ' と同等

$$-\text{derr}/\text{d}b_i = \delta'_{i+1}$$

$$b_i += \lambda * -\text{derr}/\text{d}b_i$$

重み更新のコード

```
update_weights(net,  $\varphi$ ,  $\delta'$ ,  $\lambda$ )
    for i in 0 .. len(net)-1:
         $w, b = net[i]$ 
         $w += \lambda * np.outer( \delta[i+1], \varphi[i] )$ 
         $b += \lambda * \delta[i+1]$ 
```

学習の全体像

```
# 素性を作り、ネットワークをランダムな値で初期化
create map ids, array feat_lab
for each labeled pair x, y in the data
    add (create_features(x), y) to feat_lab
initialize net randomly

# 学習を行う
for / iterations
    for each labeled pair  $\varphi_0$ , y in the feat_lab
         $\varphi = \text{forward\_nn}(\text{net}, \varphi_0)$ 
         $\delta' = \text{backward\_nn}(\text{net}, \varphi, y)$ 
        update_weights(net,  $\varphi$ ,  $\delta'$ ,  $\lambda$ )
    print net to weight_file
    print ids to id_file
```

ニューラルネット学習のこつ

学習の安定化

- ニューラルネットはパラメータが多い→学習が不安定
- 重みの初期値：
 - ランダム、 -0.1~0.1 の間の一様分布など
- 学習率：
 - 0.1 から始めることが多い
 - エラーが前イタレーションに比較して増加した場合は学習率を下げる ($\alpha = 0.9$ や $\alpha = 0.5$)
- 隠れ層の大きさ：
 - だいたい色々試して一番精度の良いものを選択

テスト

- **手軽**：エラーの値をプリントし、イタレーションごとにだいたい減ることを確認
- **本気**：有限差分法で勾配を確認

アイデア：

重み更新の際、重み w_i の勾配を計算： $d\text{err}/d\mathbf{w}_i$
つまり、この重みを少しだけ (ω だけ) 揺らせば

$$\begin{array}{ll} \mathbf{w}_i = \mathbf{x} & \mathbf{w}_i = \mathbf{x} + \omega \\ \text{の場合} & \text{の場合} \\ \mathbf{err} = \mathbf{y} & \mathbf{err} = \mathbf{y} + \omega * d\text{err}/d\mathbf{w}_i \end{array}$$

が成り立つはず！

有限差分法で、 \mathbf{w}_i を揺らしてみて、上記が（ $1e-6$ など、一定の誤差内）成り立たなければ、勾配計算のバグがあると判断

詳細：<http://cs231n.github.io/neural-networks-3/>

演習課題

演習課題 (1)

- 実装
 - train-nn: NN を学習するプログラム
 - test-nn: NN を用いて予測するプログラム
- テスト
 - 入力 : test/03-train-input.txt
 - 学習 1 回、隠れ層 1 つ , 隠れ層のノード 2 つ
 - 更新を手で確認

演習課題 (2)

- 学習 data/titles-en-train.labeled
- 予測 data/titles-en-test.word
- 評価
 - script/grade-prediction.py data-en/titles-en-test.labeled your_answer
- 比較
 - 単純なパーセプトロン、SVM、ロジスティック回帰
 - ノード数、初期学習率、ランダムな値の初期レンジ
- チャレンジ
 - 複数の隠れ層を使ったネットの実装
 - エラー増加の場合、学習率を減らす手法を実装

Thank You!