

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

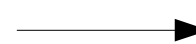
予測

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



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

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

- \mathbf{x} : 入力
- $\boldsymbol{\varphi}(\mathbf{x})$: 素性関数のベクトル $\{\varphi_1(\mathbf{x}), \varphi_2(\mathbf{x}), \dots, \varphi_I(\mathbf{x})\}$
- \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？」

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

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

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

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

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

- 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

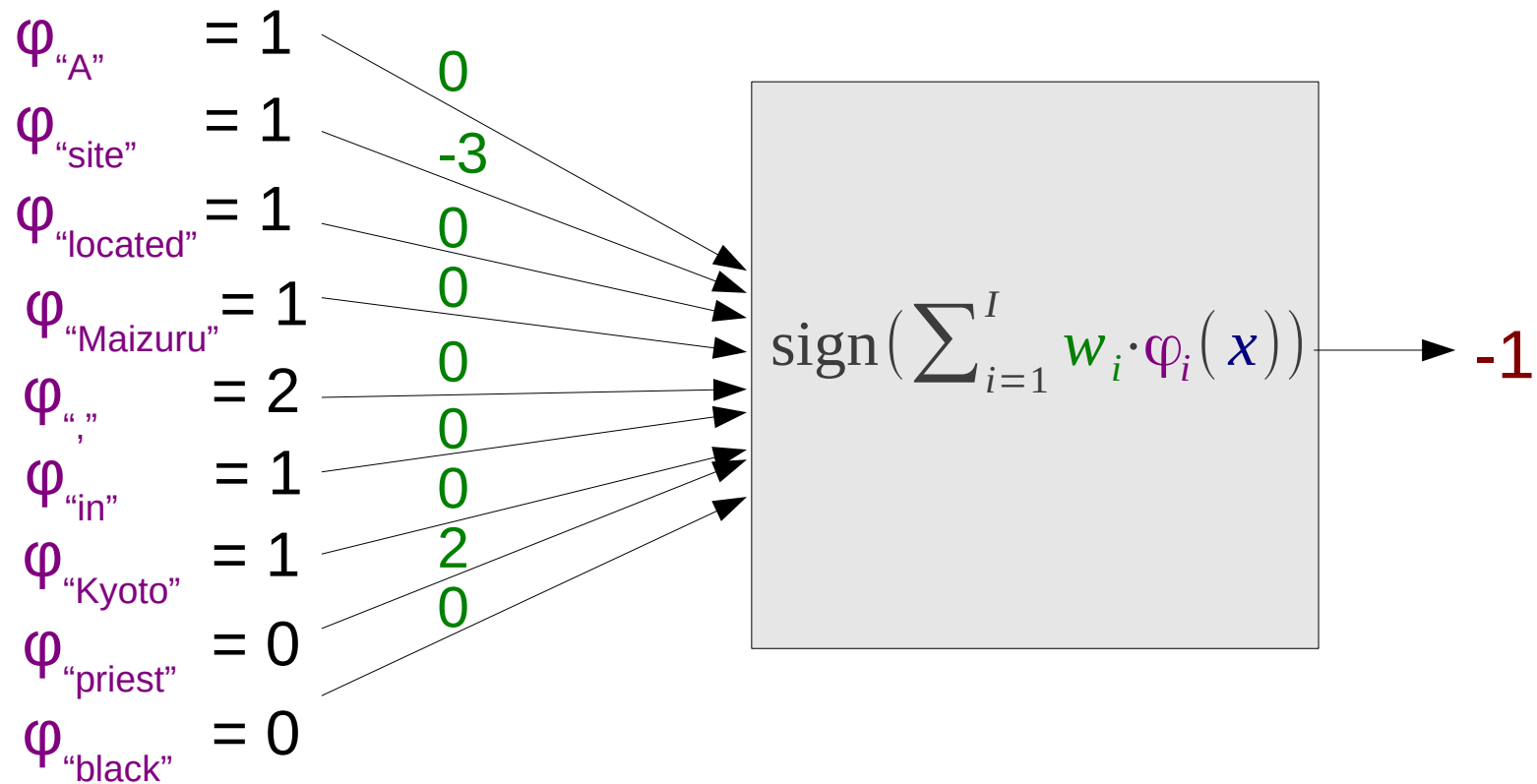
重み付き和の計算

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

$\varphi_{\text{unigram "A"}}(x) = 1$		$W_{\text{unigram "a"}} = 0$		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 =  $w * \phi(x)$   
    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
```

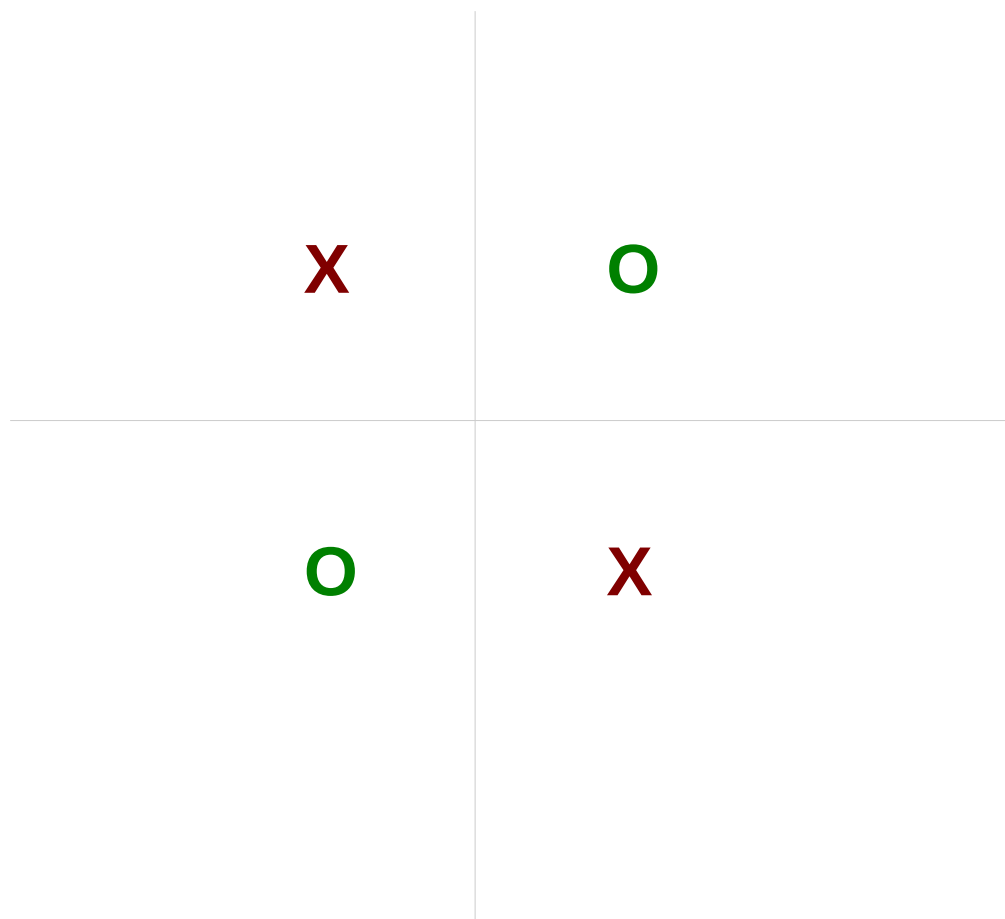
パーセプトロン予測の疑似コード

```
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

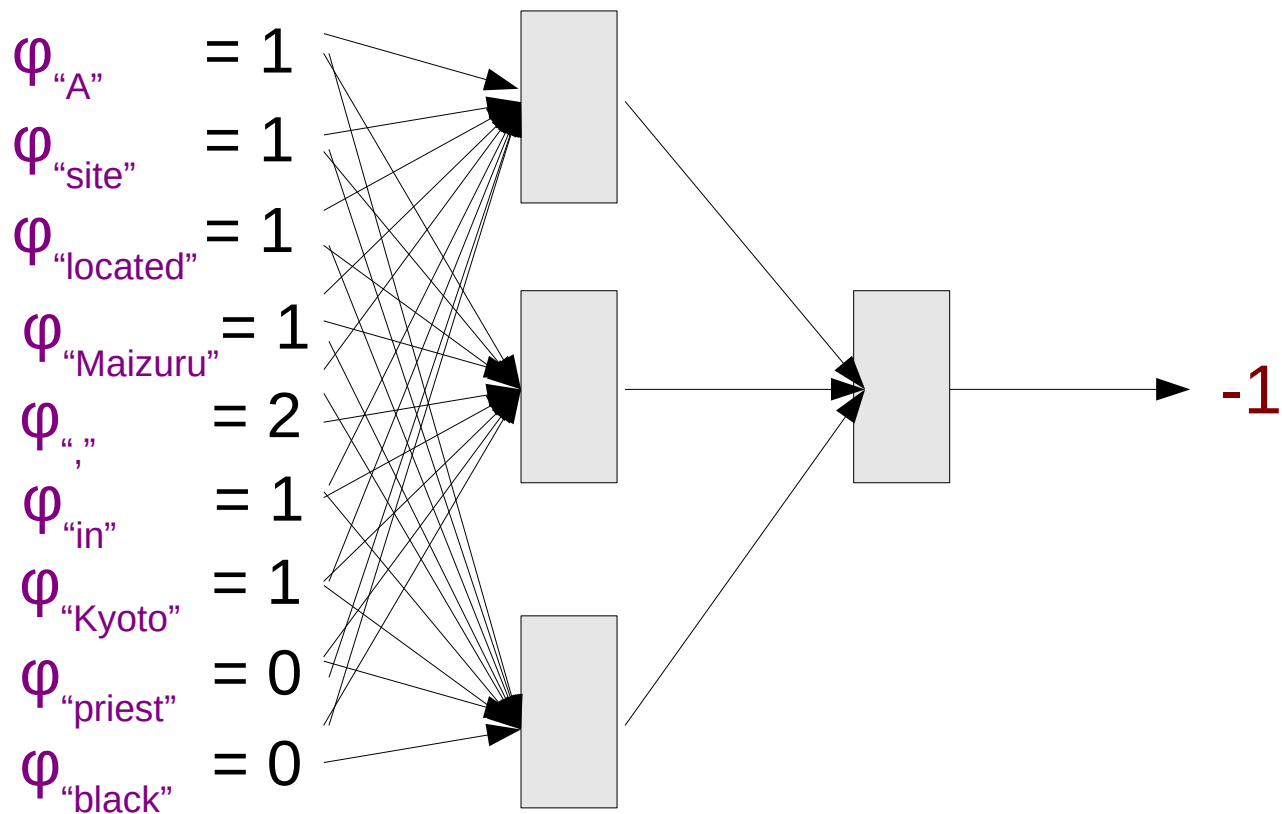
問題：線形分類のみ

- 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

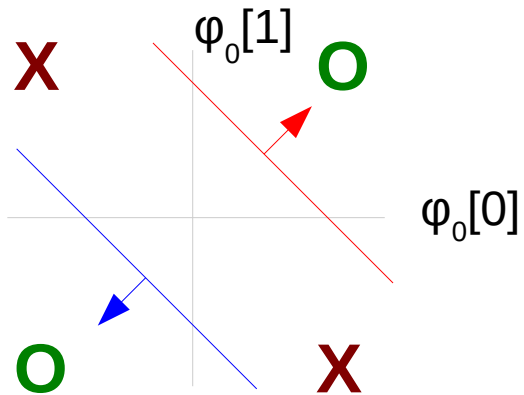
- 複数のパーセプトロンをつなげる



- モチベーション：線形でない関数も表現可能！

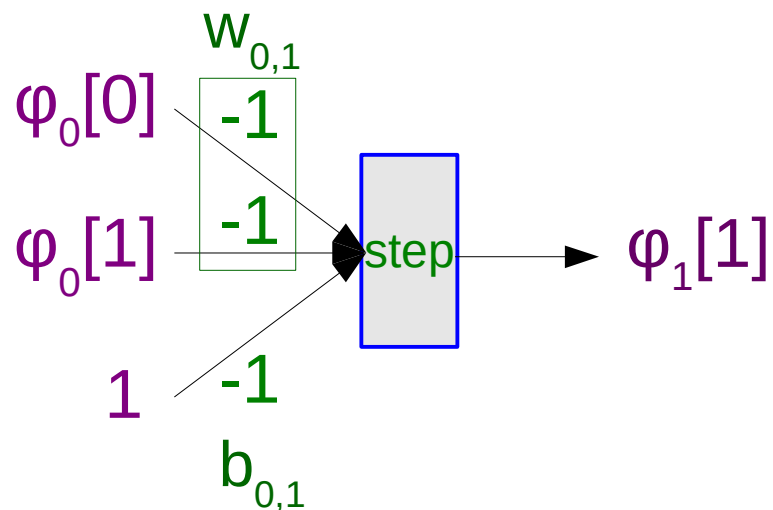
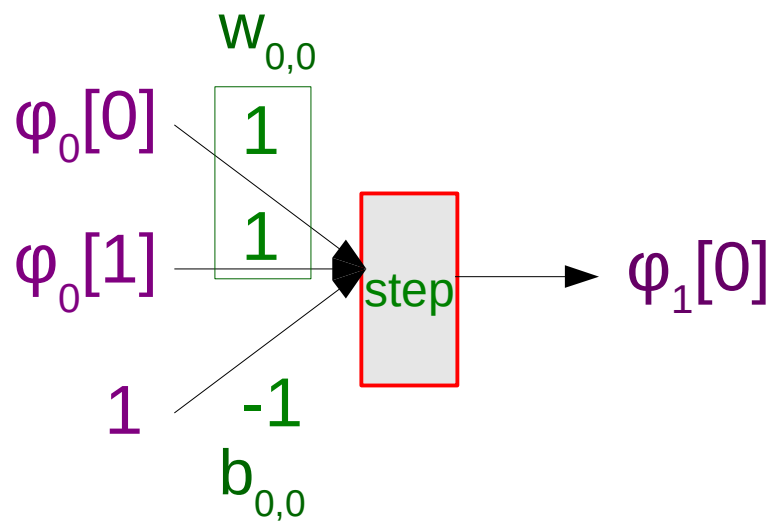
- 2つの分類器を作成

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



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

例:

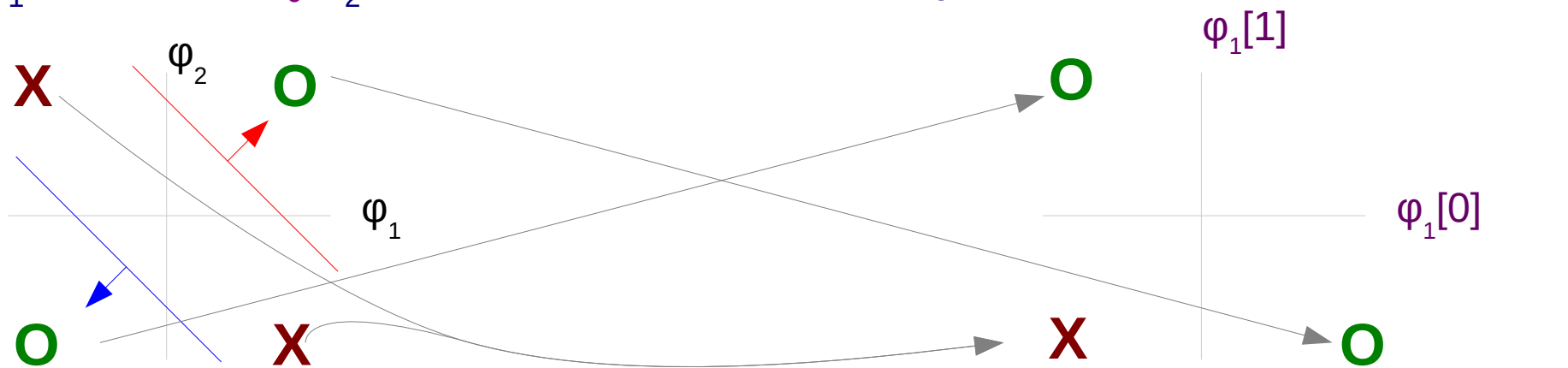


例:

- 分類器は新しい素性空間へマッピング

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

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



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

$$\varphi_1(x_1) = \{-1, -1\}$$

$$\varphi_1(x_2) = \{1, -1\}$$

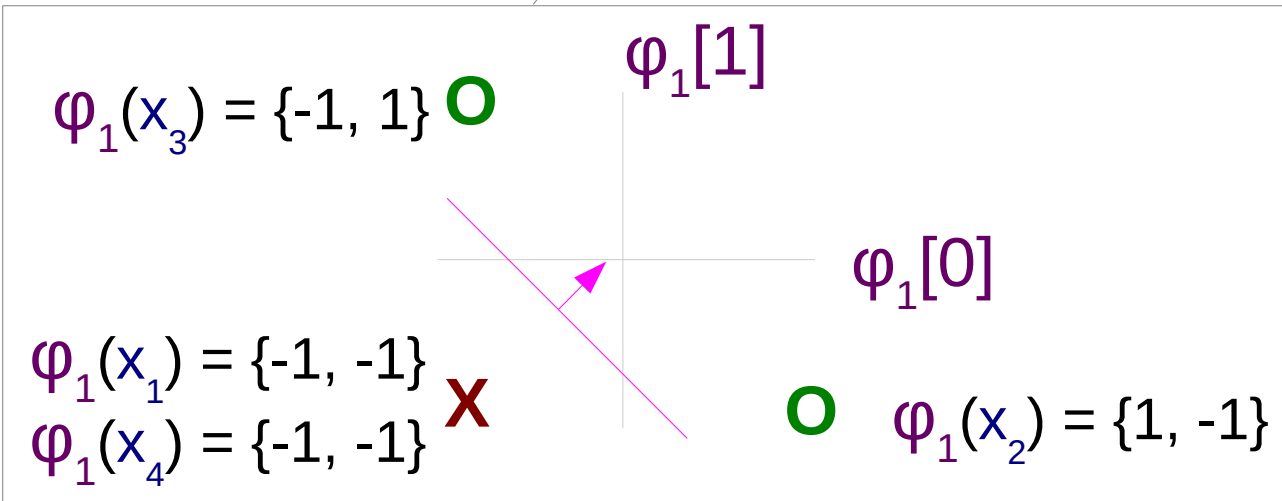
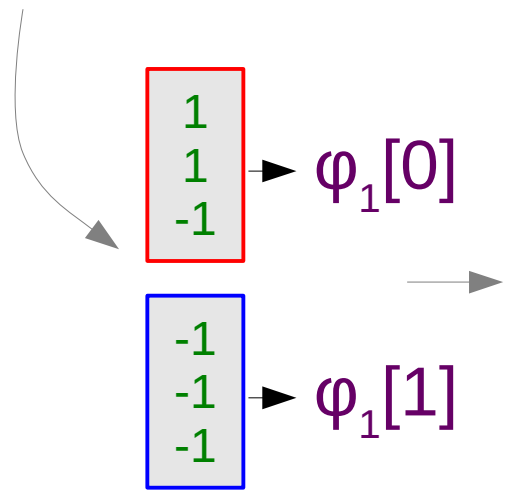
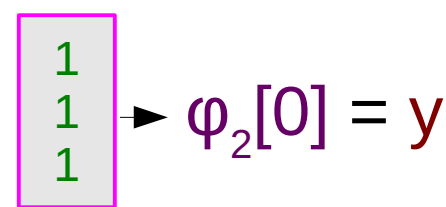
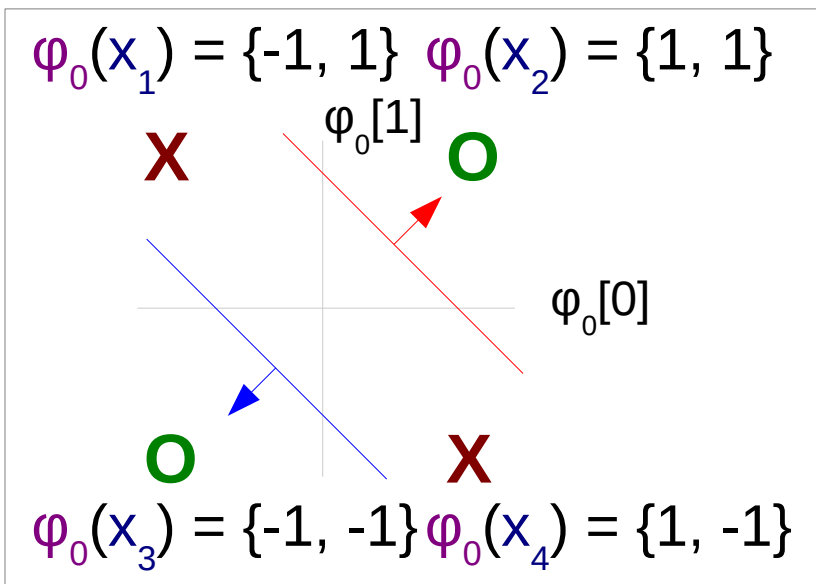
$$\varphi_1(x_4) = \{-1, -1\}$$

$$\begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} \rightarrow \varphi_1[0]$$

$$\begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix} \rightarrow \varphi_1[1]$$

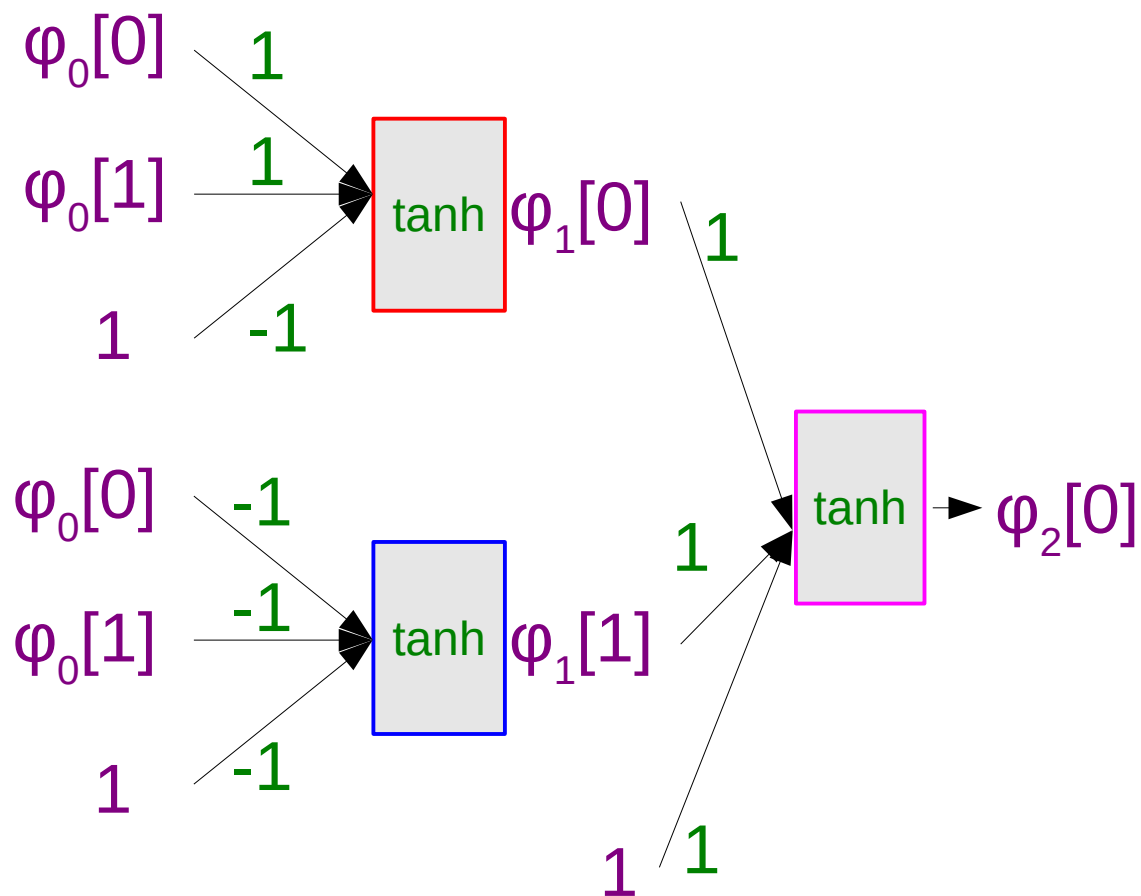
例：

- 新しい空間で、事例が分類可能に！



例：

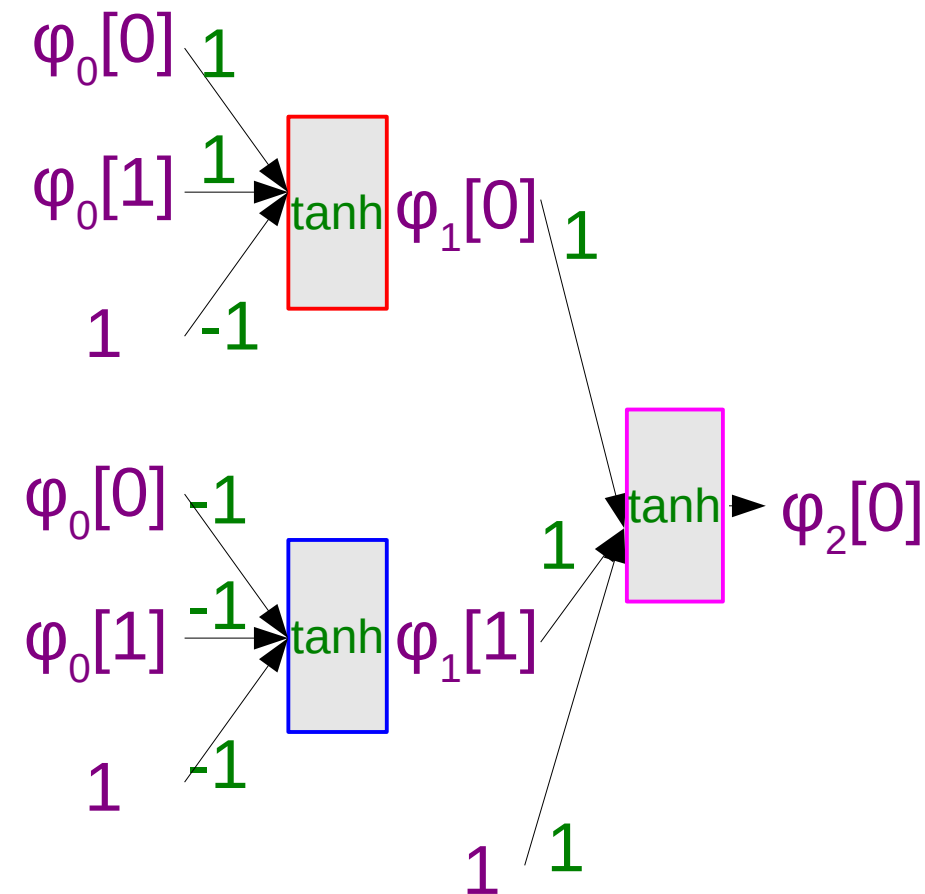
- 最終的なニューラルネット



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

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

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

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

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

2層ニューラルネットの例 (行列編)

入力

$$\boldsymbol{\varphi}_0 = \text{np.array}([1, -1])$$

一層目の計算

$$\mathbf{w}_0 = \text{np.array}([[1, 1], [-1, -1]])$$

$$\mathbf{b}_0 = \text{np.array}([-1, -1])$$

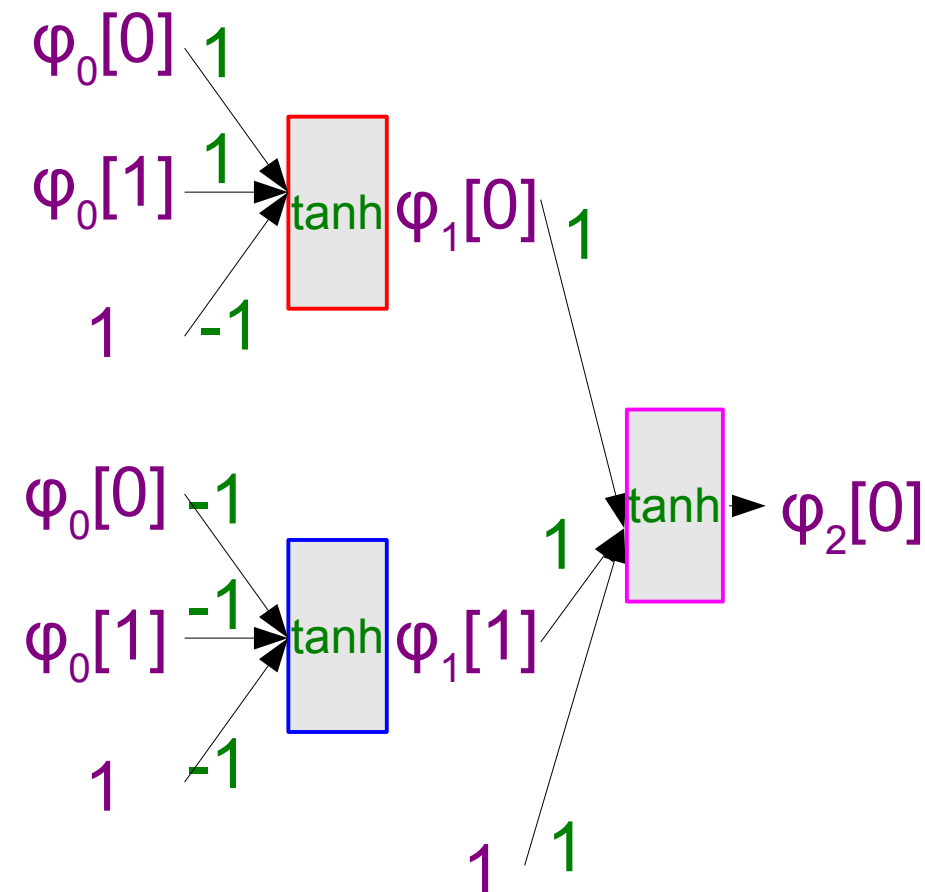
$$\boldsymbol{\varphi}_1 = \text{np.tanh}(\text{np.dot}(\mathbf{w}_0, \boldsymbol{\varphi}_0) + \mathbf{b}_0)$$

2層目の計算

$$\mathbf{w}_1 = \text{np.array}([[1, 1]])$$

$$\mathbf{b}_1 = \text{np.array}([-1])$$

$$\boldsymbol{\varphi}_2 = \text{np.tanh}(\text{np.dot}(\mathbf{w}_1, \boldsymbol{\varphi}_1) + \mathbf{b}_1)$$



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:  
     $w, b = \text{network}[i]$   
    # 前の層の値に基づいて値を計算  
     $\varphi[i] = \text{np.tanh}( \text{np.dot}( w, \varphi[i-1] ) + b )$   
  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

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

正解 システム出力

- エラーの勾配：

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

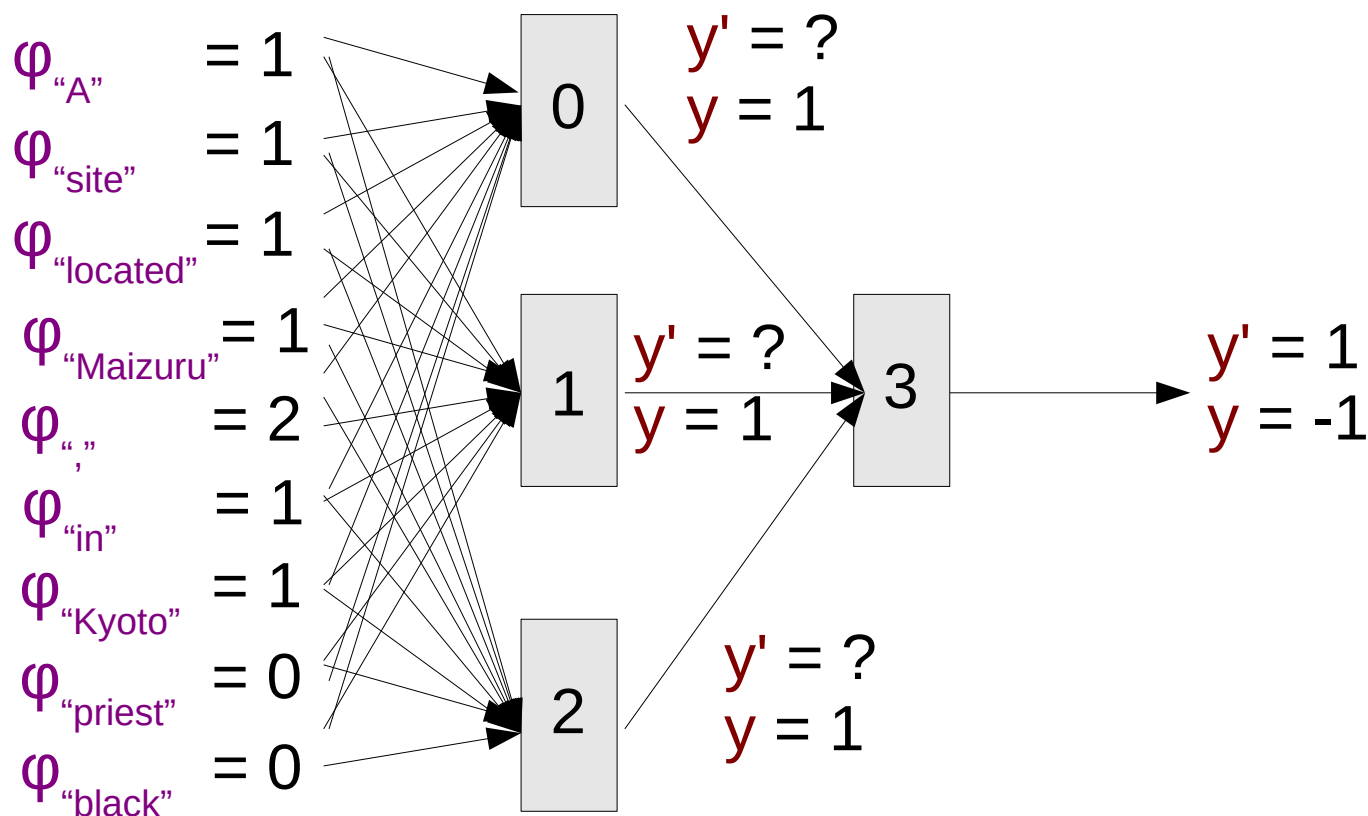
- 各重みを更新：

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

- λ は学習率

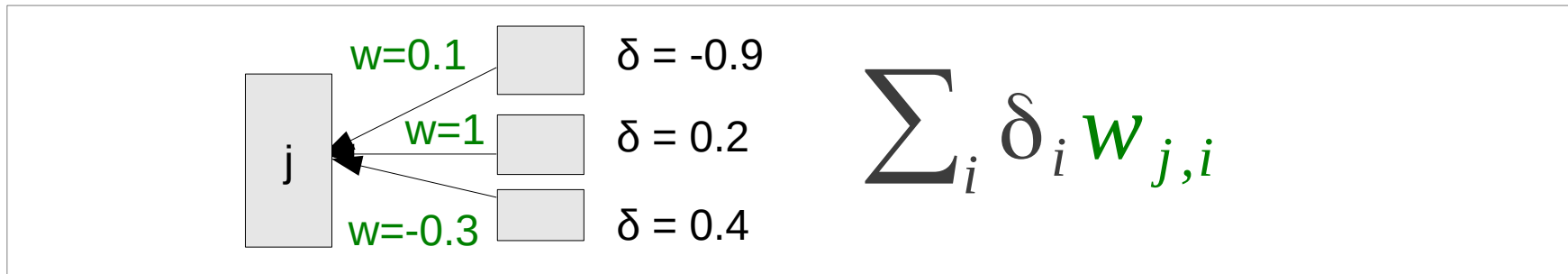
問題：正解は分からない！

- NN では出力層のみで正解が与えられる

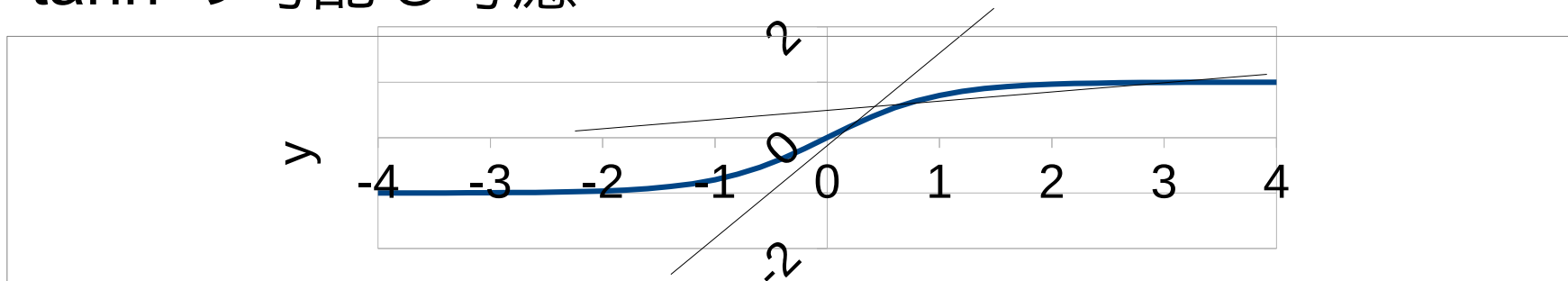


解決策：逆伝搬法

- 出力層からエラーを後ろへ伝搬



- tanh の勾配も考慮



$$d \tanh(\varphi(x) * w) = 1 - \tanh(\varphi(x) * w)^2 = 1 - y_j^2$$

- 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

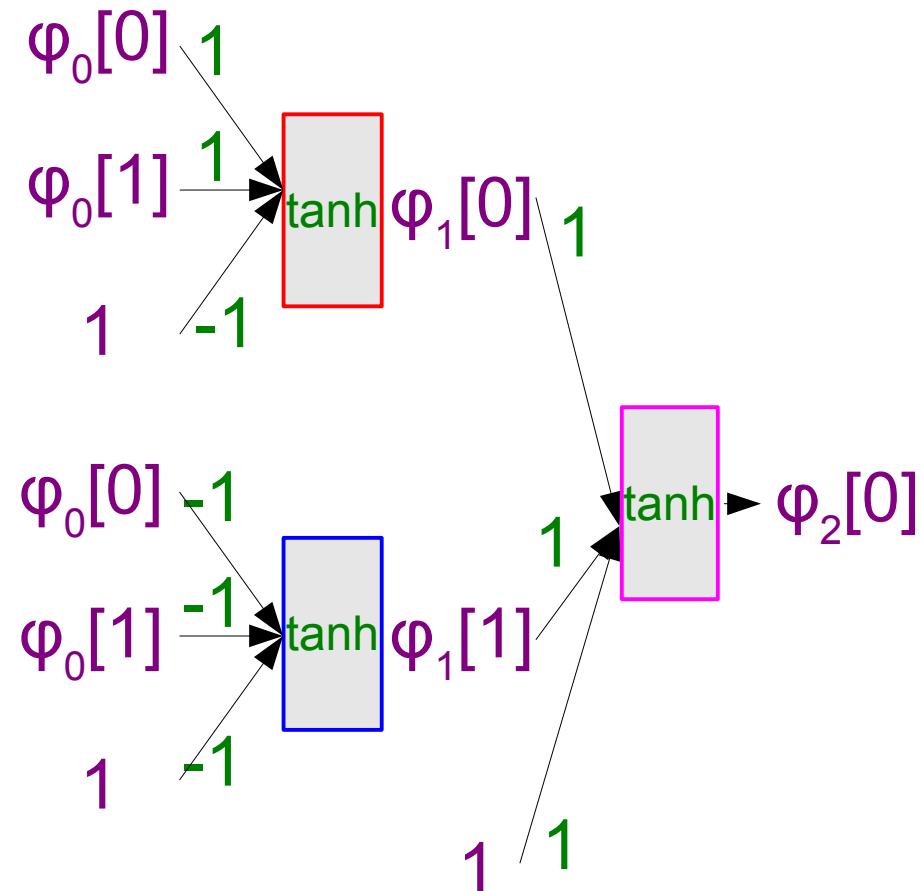
$$\delta'_2 = \delta_2 * (1 - \phi_2^2)$$

$$\delta_1 = \text{np.dot}(\delta'_2, \mathbf{w}_1)$$

0層目の計算

$$\delta'_1 = \delta_1 * (1 - \phi_1^2)$$

$$\delta_0 = \text{np.dot}(\delta'_1, \mathbf{w}_0)$$



逆伝搬のコード

```
backward_nn(net,  $\varphi$ ,  $y'$ )
    J = len(net)
    create array  $\delta = [ 0, 0, \dots, \text{np.array}([y' - \varphi[J][0]]) ]$  # length J+1
    create array  $\delta' = [ 0, 0, \dots, 0 ]$ 
    for i in J-1 .. 0:
         $\delta'[i+1] = \delta[i+1] * (1 - \varphi[i+1]^2)$ 
         $w, b = \text{net}[i]$ 
         $\delta[i] = \text{np.dot}(\delta'[i+1], w)$ 
    return  $\delta'$ 
```

重み更新

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

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

- 学習率をかけ、重みを更新

$$\mathbf{w}_i += \lambda * -\text{derr}/d\mathbf{w}_i$$

- バイアス項は単純に δ' と同等

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

$$\mathbf{b}_i += \lambda * -\text{derr}/d\mathbf{b}_i$$

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
         $\varphi$  = forward_nn(net,  $\varphi_0$ )
         $\delta'$  = backward_nn(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、 $-0.1 \sim 0.1$ 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 ($*= 0.9$ や $*= 0.5$)
- 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： $derr/dw_i$
つまり、この重みを少しだけ (ω だけ) 揺らせば

$w_i = x$		$w_i = x + \omega$	
の場合	なら	の場合	が成り立つはず！
$err = y$		$err = y + \omega * derr/dw_i$	

有限差分法で、 w_i を揺らしてみても、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!