

自然言語処理プログラミング勉強会 10 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

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

- x : 入力
- $\boldsymbol{\phi}(x)$: 素性関数のベクトル $\{\phi_1(x), \phi_2(x), \dots, \phi_I(x)\}$
- \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 = 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 = 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	+

...

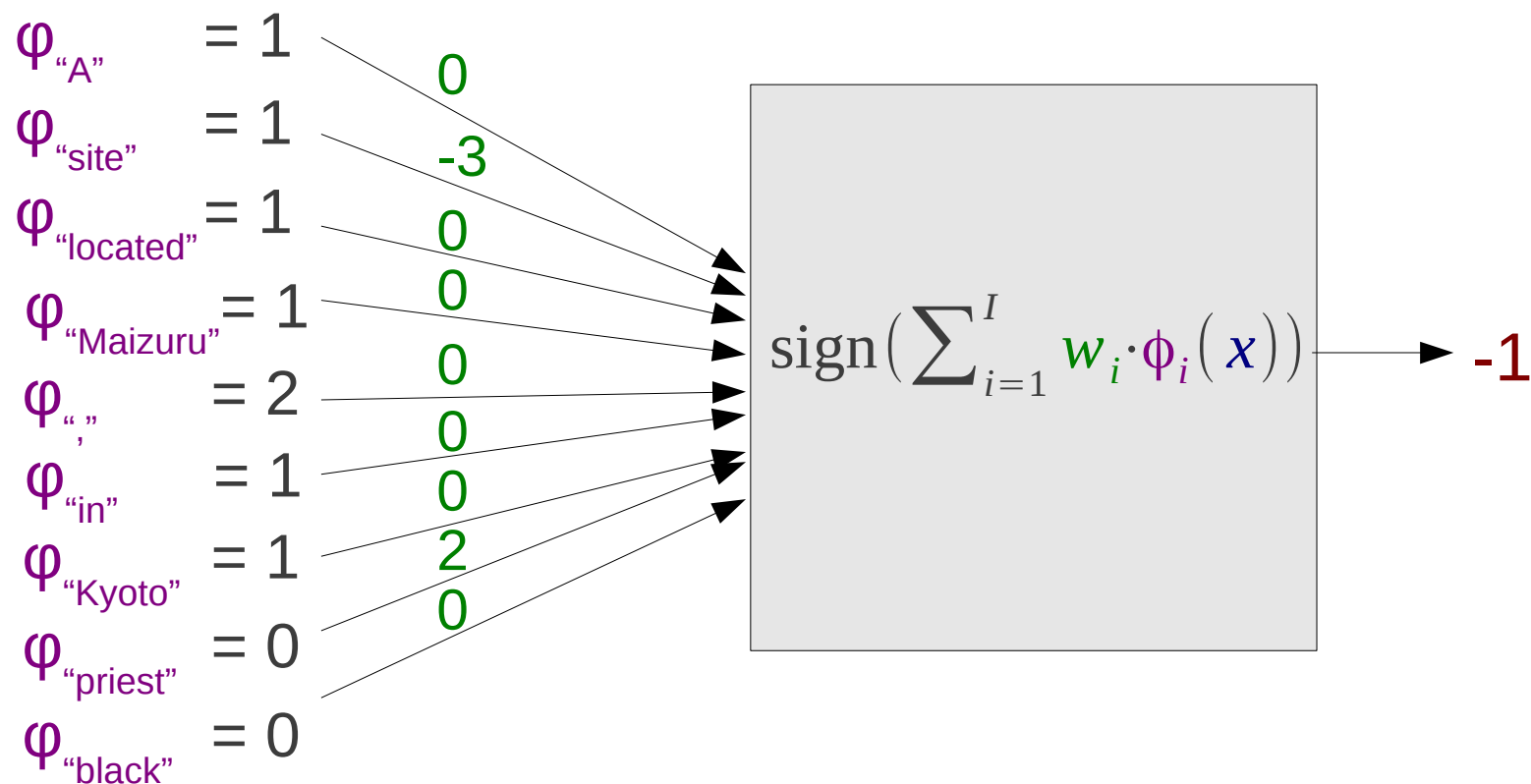
...

=

-3 → No!

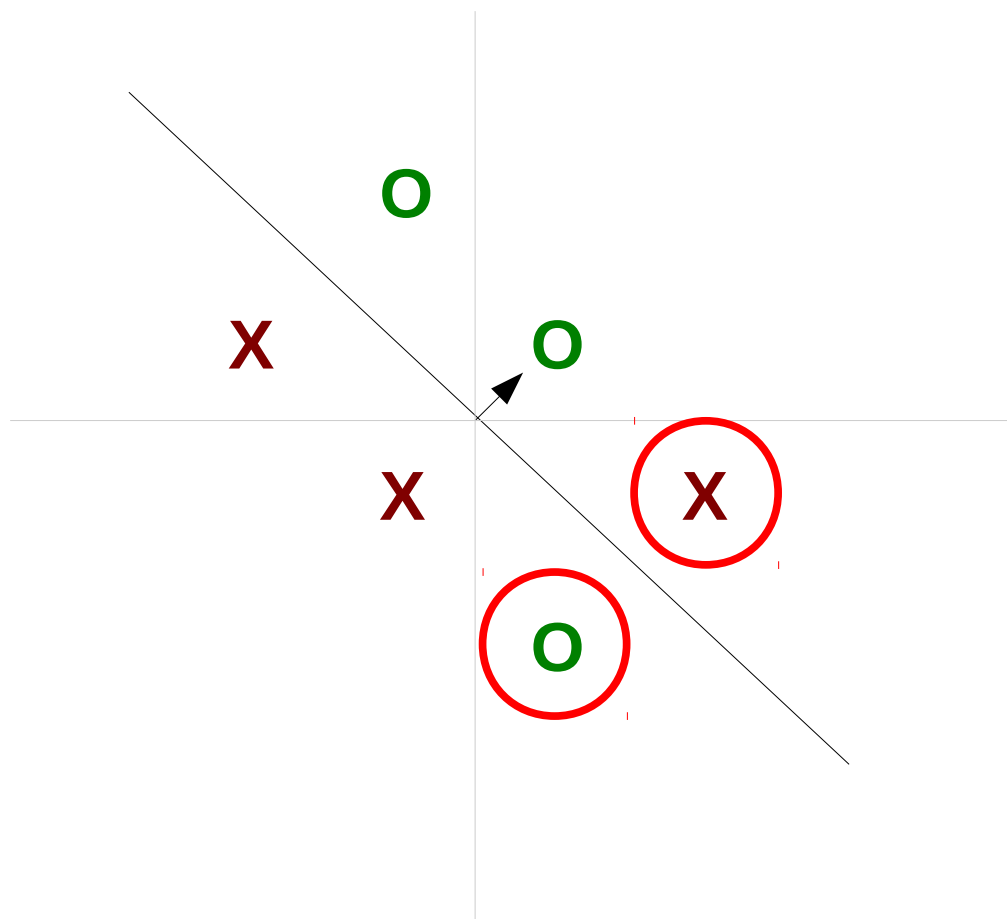
パーセプトロン

- 重み付き和を計算する「機械」として考える



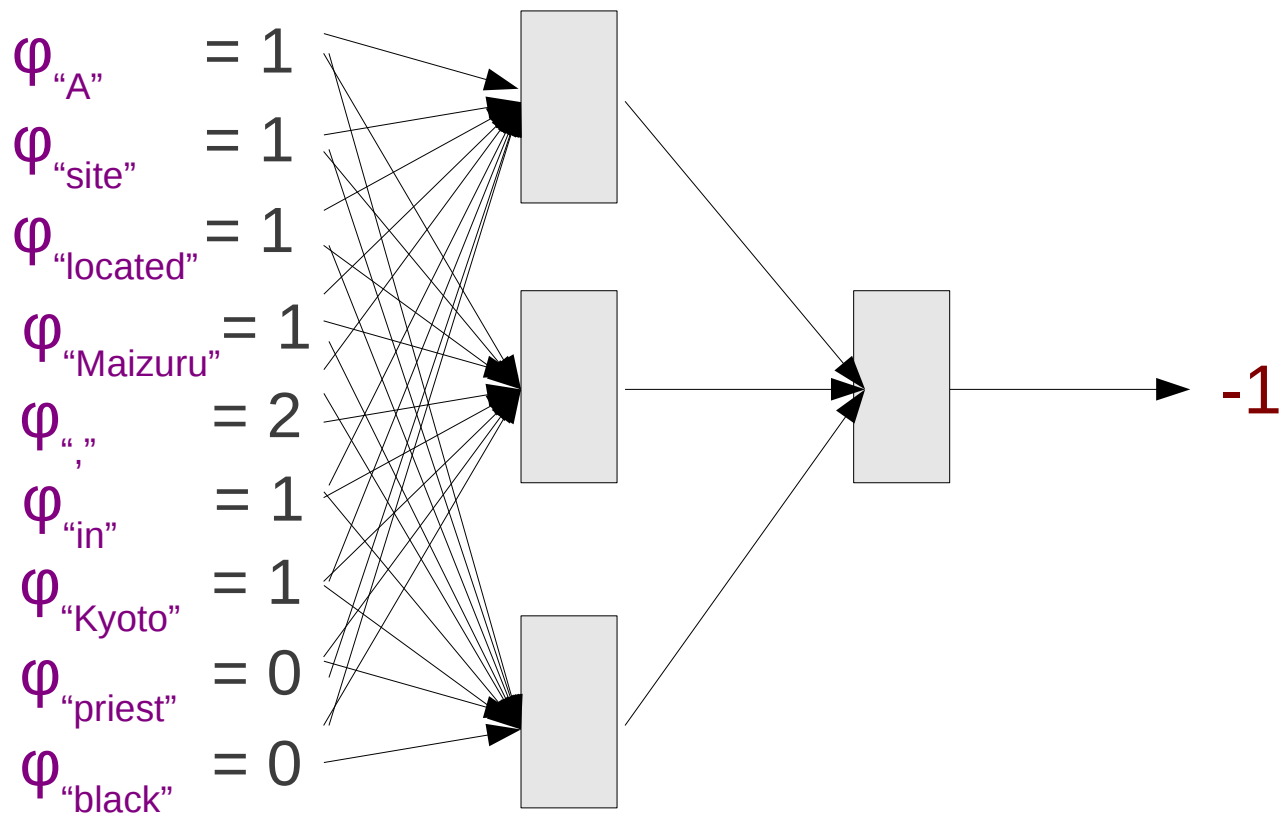
問題：線形分類のみ

- 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

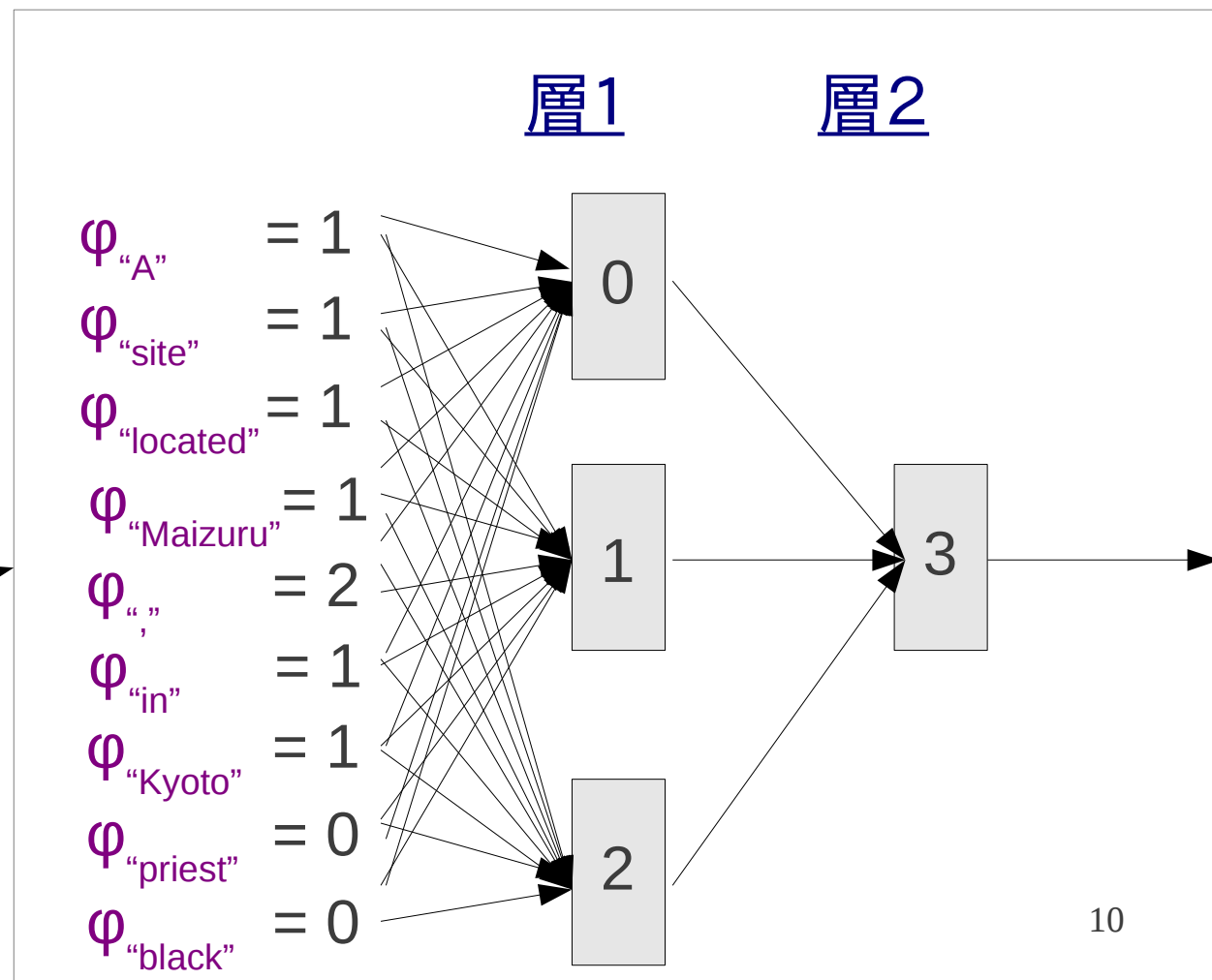


- モチベーション：線形でない関数も表現可能！

ニューラルネットの実装

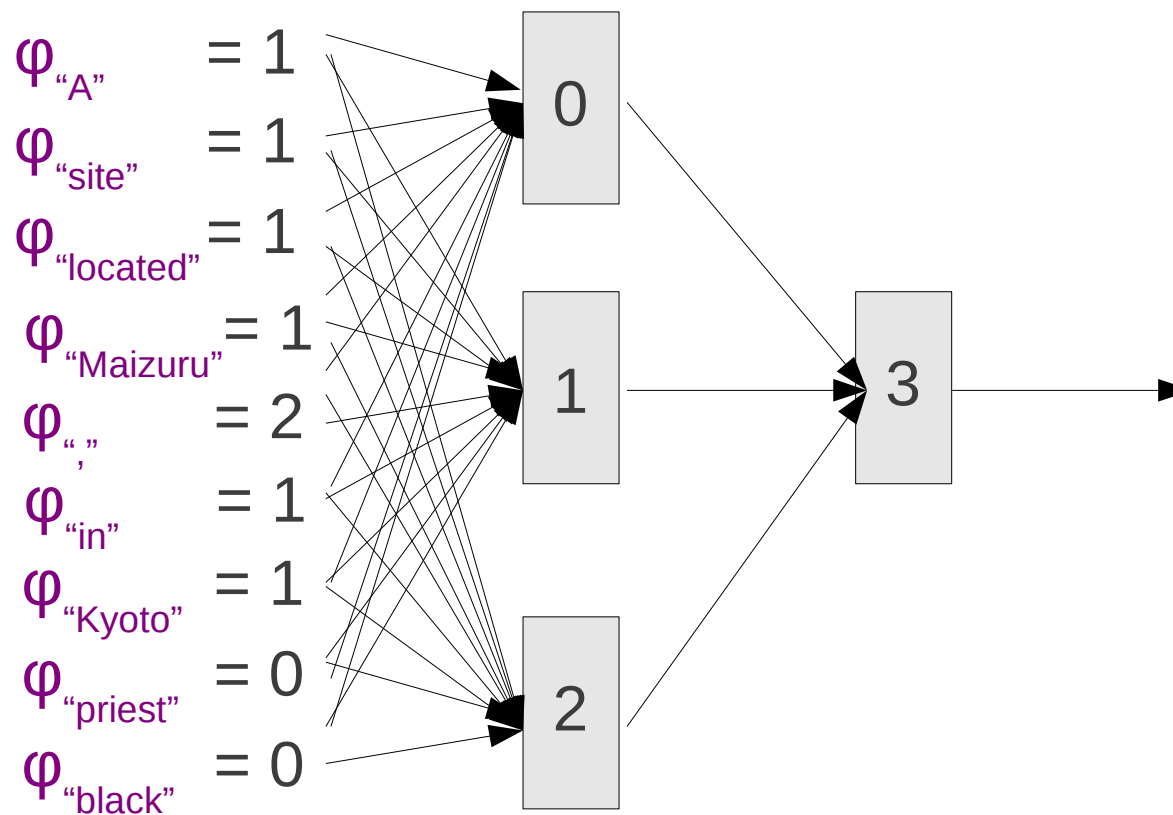
- ネットは複数の層からなり、層の間は全連結
- パーセプトロン：
 - 層の ID
 - 重みベクトル

```
network = [
    (1,  $w_0$ ),
    (1,  $w_1$ ),
    (1,  $w_2$ ),
    (2,  $w_3$ )
]
```



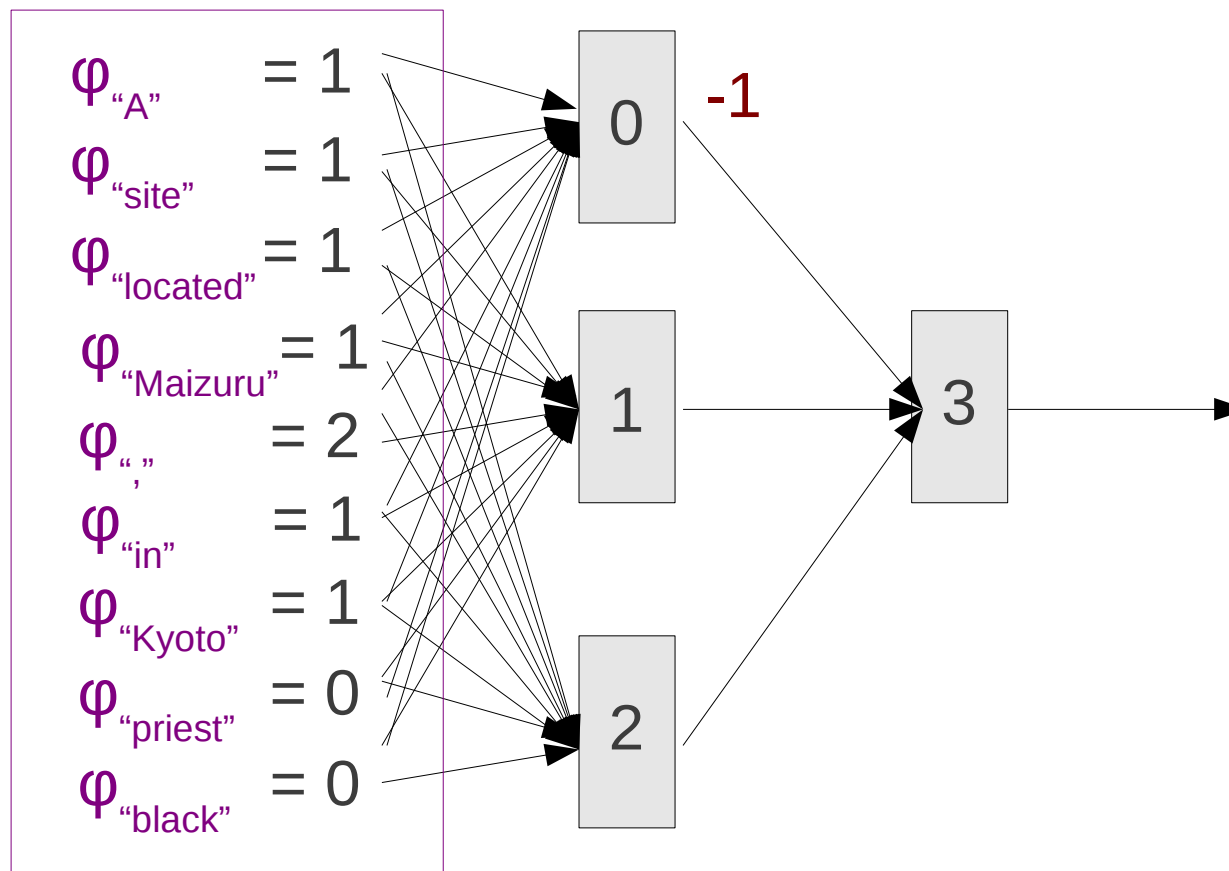
ニューラルネットによる予測

- 各パーセプトロンの値を前の層の結果に基づいて計算



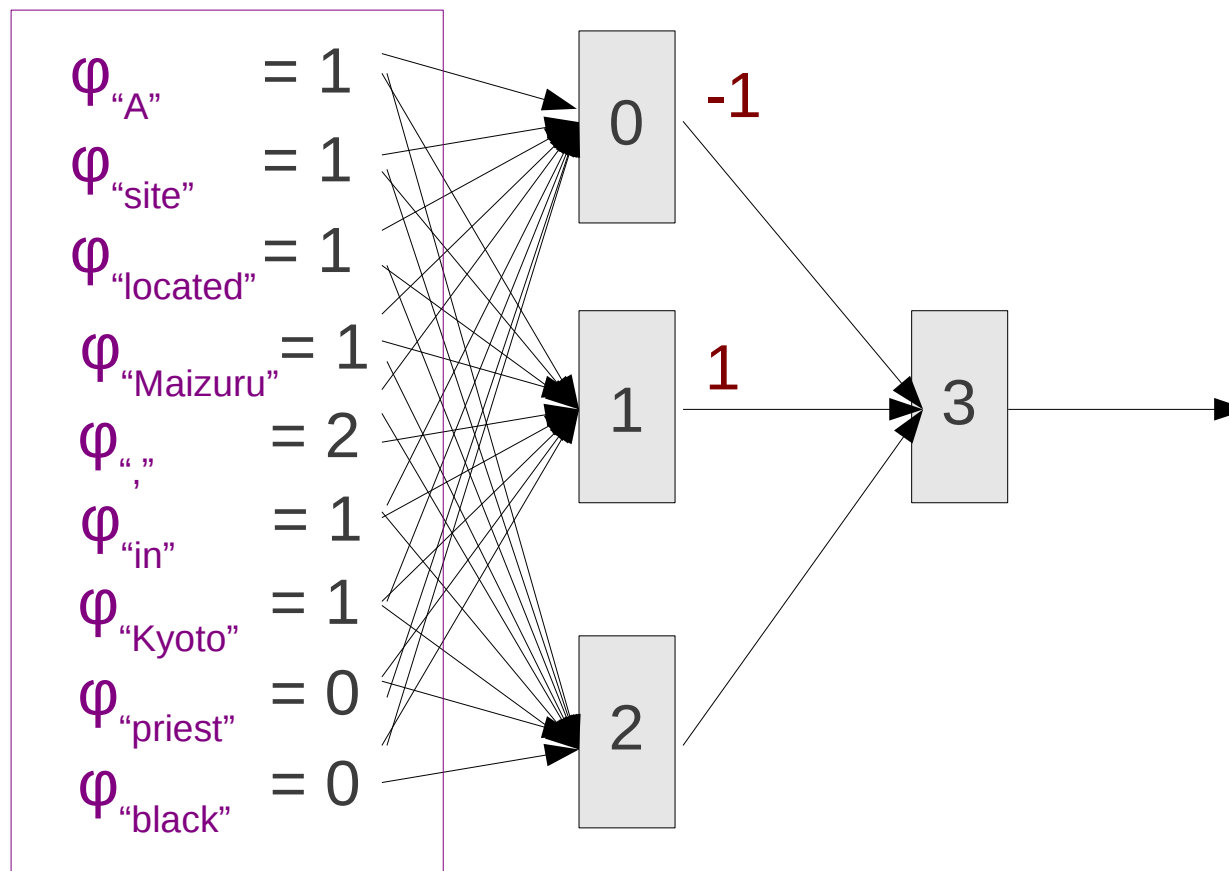
ニューラルネットによる予測

- 各パーセプトロンの値を前の層の結果に基づいて計算



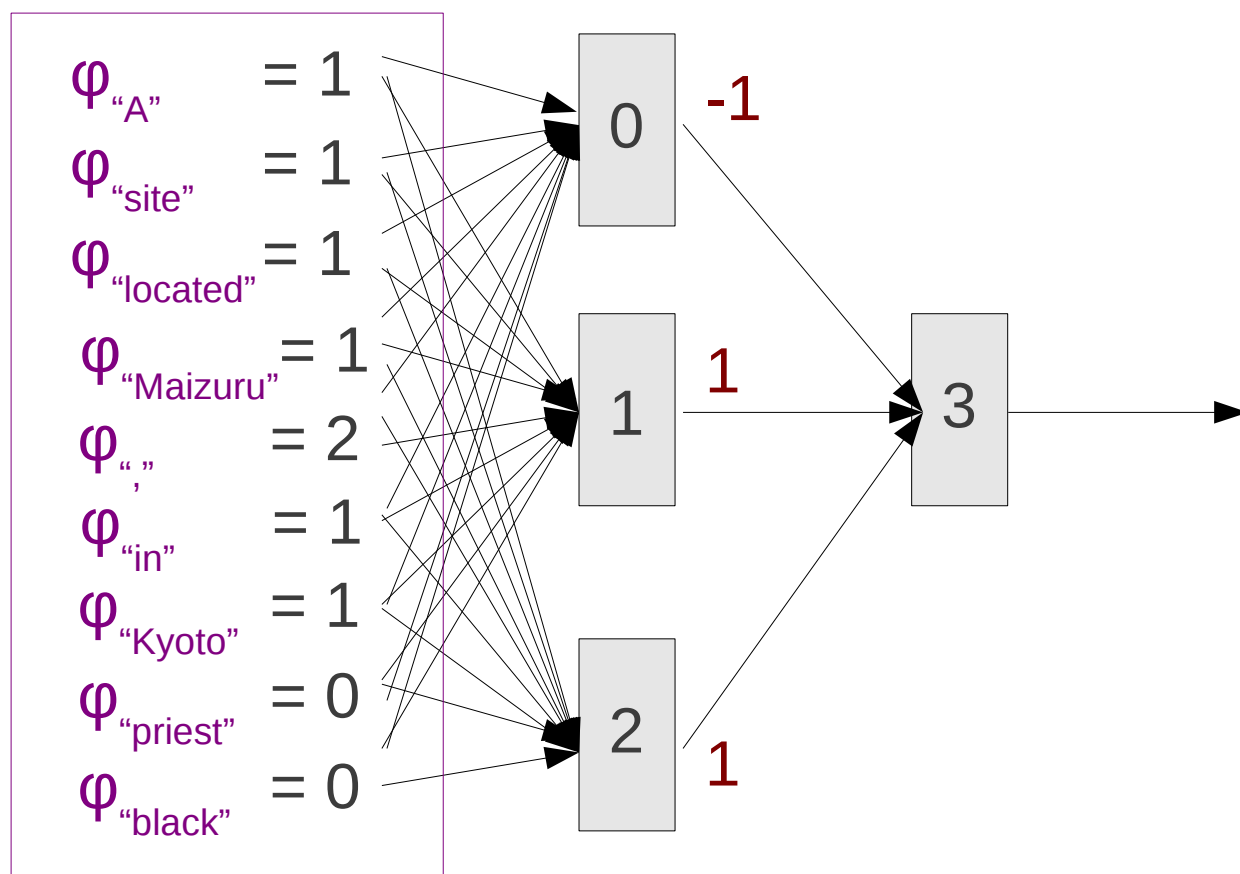
ニューラルネットによる予測

- 各パーセプトロンの値を前の層の結果に基づいて計算



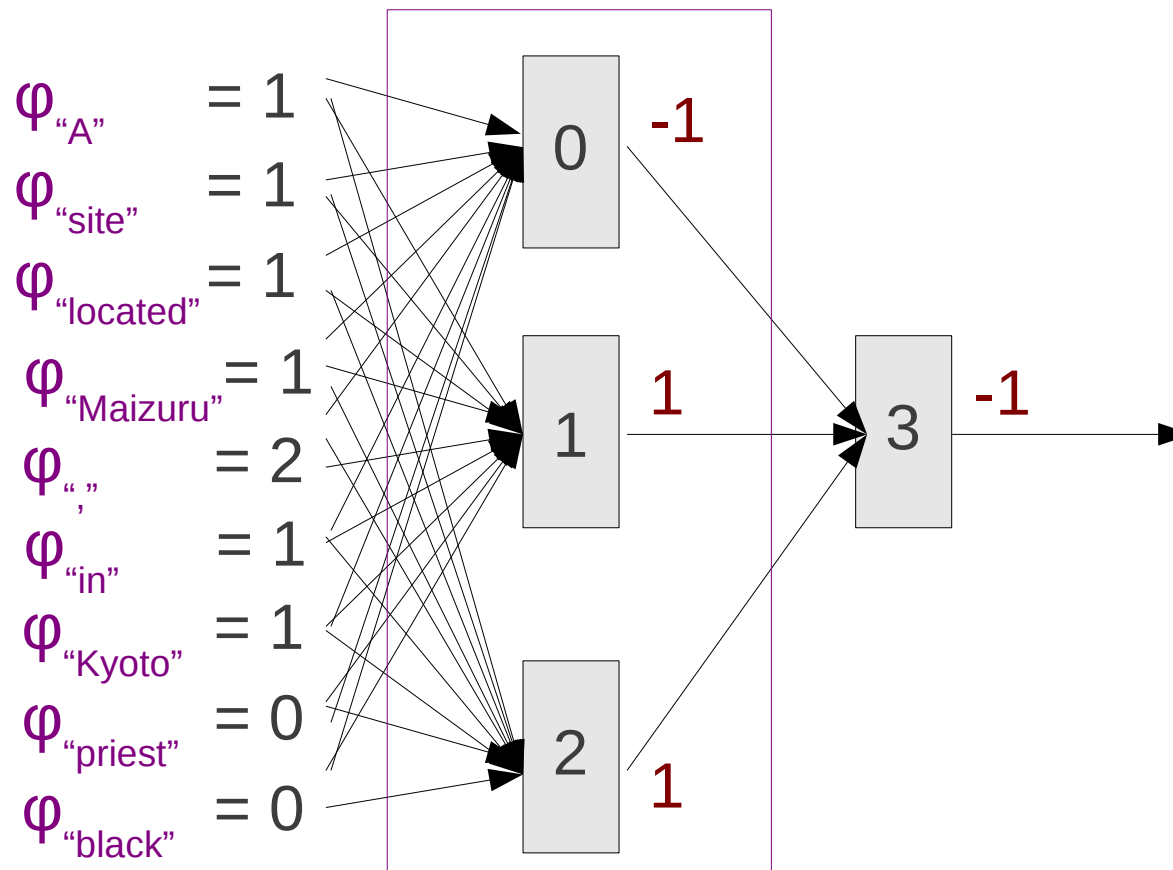
ニューラルネットによる予測

- 各パーセプトロンの値を前の層の結果に基づいて計算



ニューラルネットによる予測

- 各パーセプトロンの値を前の層の結果に基づいて計算



復習： 単一のパーセプトロンの予測コード

```
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]
    if score >= 0
        return 1
    else
        return -1
```

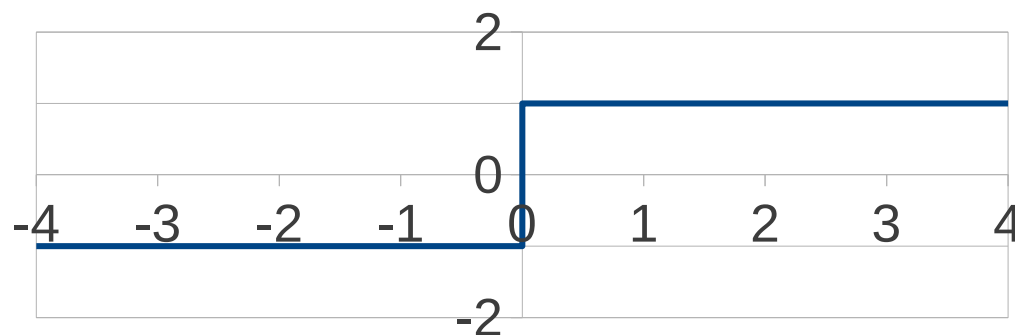

ニューラルネットの予測コード

```
PREDICT_NN(network, phi)  
  y = [ phi, {}, {} ... ] # 各層の値  
  for each node i:  
    layer, weight = network[i]  
    # 前の層の値に基づいて値を計算  
    answer = PREDICT_ONE(weight, y[layer-1])  
    # 次の層に計算された値を保存  
    y[layer][i] = answer  
  return y[-1][0] # 最後の層のパーセプトロンの値
```

パーセプトロンの関数

- 今までの話は step 関数を利用

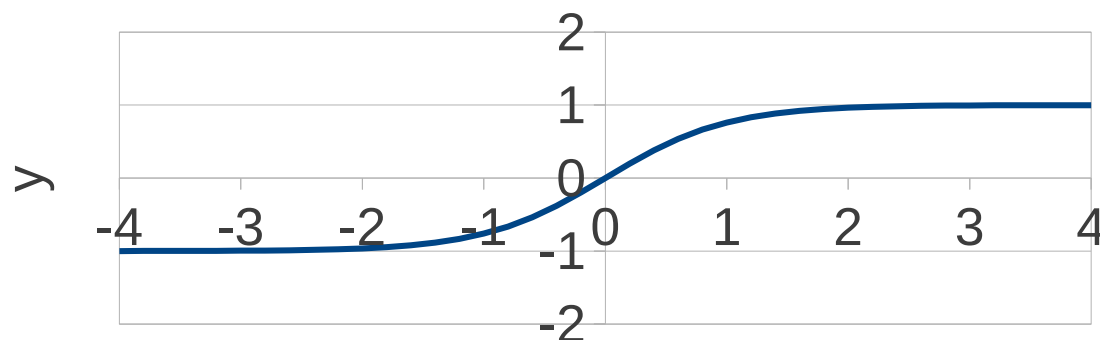
$$y = \text{sign}(\mathbf{w} \cdot \boldsymbol{\phi}(\mathbf{x}))$$



sign($\phi \cdot x$)

- step 関数を微分できない \rightarrow tanh を利用

$$y = \tanh(\mathbf{w} \cdot \boldsymbol{\phi}(\mathbf{x}))$$



tanh($\phi \cdot x$)

Python:

```
from math import tanh
tanh(x)
```

tanh を用いたパーセプトロン学習

- エラーを計算：

$$\delta = y' - y$$

正解 システム出力

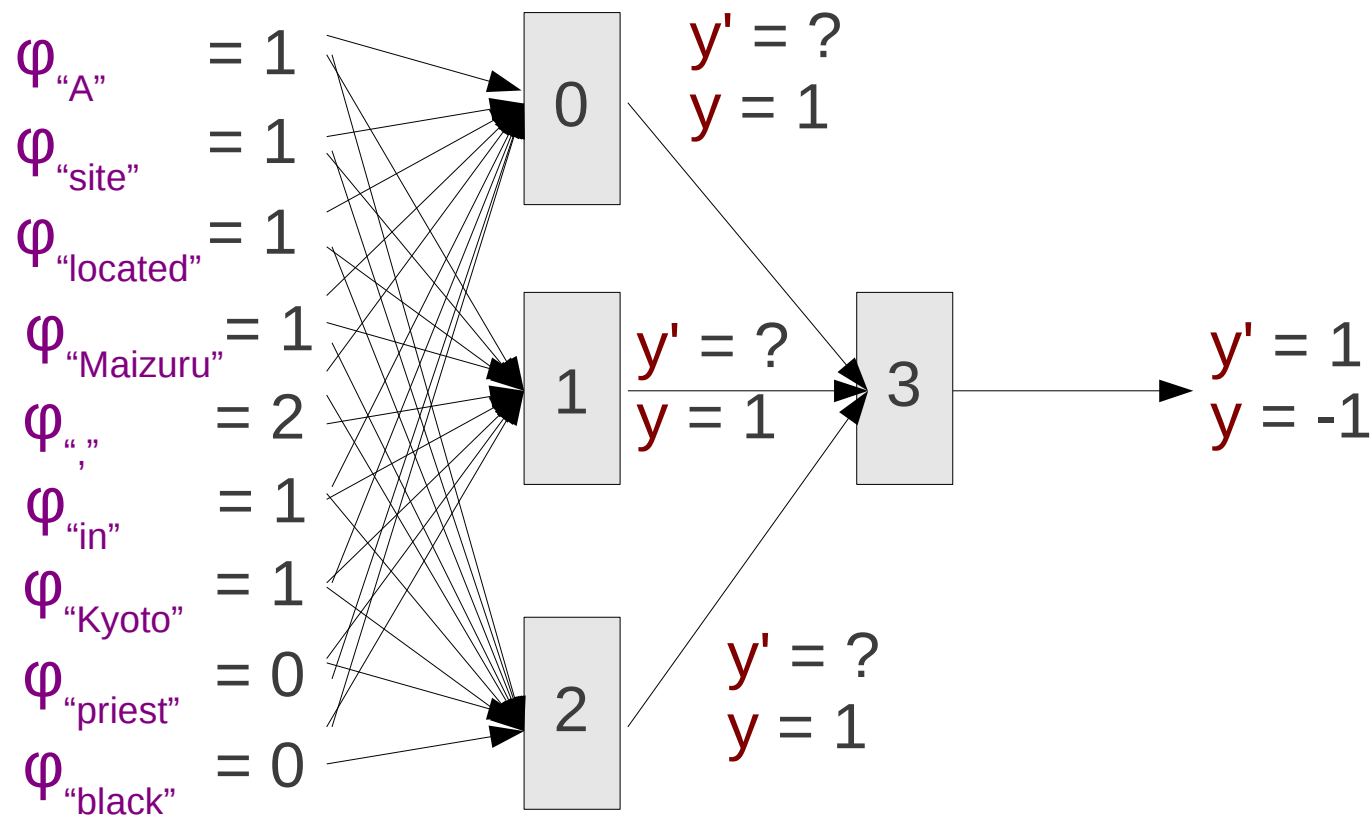
- 各重みを更新：

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

- λ は学習率
- (step 関数パーセプトロンでは $\delta = -2$ or $+2$, $\lambda = 1/2$)

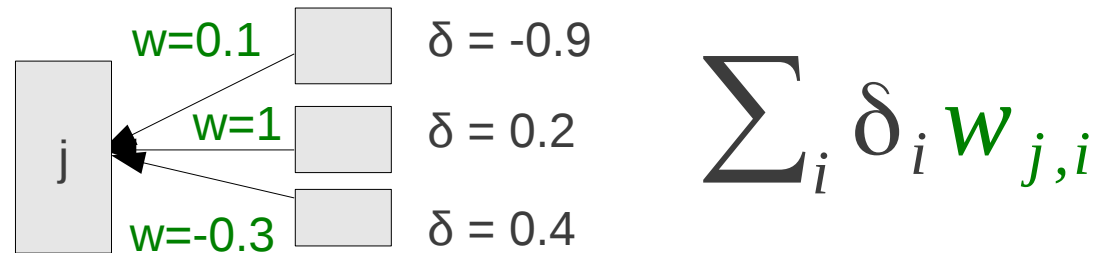
問題：正解は分からない！

- NN では出力層のみで正解が与えられる

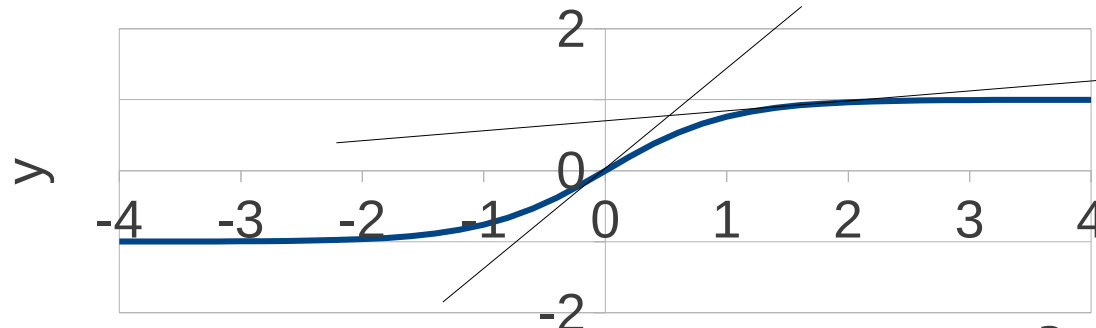


解決策：逆伝搬法

- 出力層からエラーを後ろへ伝搬



- \tanh の勾配も考慮



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

- 合わせて：

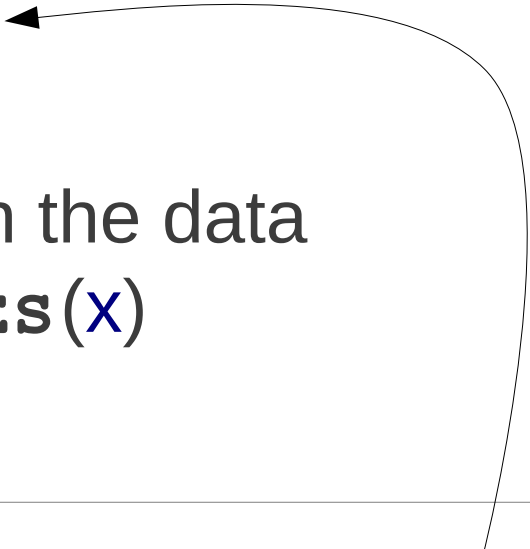
$$\delta_j = (1 - y_j^2) \sum_i \delta_i w_{j,i}$$

逆伝搬のコード

```
UPDATE_NN(network, phi, y')  
  create array  $\delta$   
  calculate y using PREDICT_NN  
  for each node j in reverse order:  
    if j is the last node  
       $\delta_j = y' - y_j$   
    else  
       $\delta_j = (1 - y_j^2) \sum_i \delta_i w_{j,i}$   
  for each node j:  
    layer, w = network[j]  
    for each name, val in y[layer-1]:  
      w[name] +=  $\lambda * \delta_j * val$ 
```

学習コード

```
create network  
randomize network weights  
for / iterations  
  for each labeled pair x, y in the data  
    phi = CREATE_FEATURES(x)  
    UPDATE_NN(w, phi, y)
```



- 単純なパーセプトロンで、重みを 0 へ初期化
- NN ではランダム初期化
(全てのパーセプトロンが同一の値にならないよう)

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
 - 様々な隠れ層の数、ノード数

Thank You!