

NLP Programming Tutorial 10 - Neural Networks

Graham Neubig
Nara Institute of Science and Technology (NAIST)

Prediction Problems

Given x , predict y

Example we will use:

- Given an introductory sentence from Wikipedia
- Predict **whether the article is about a person**

Given

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



Predict

Yes!

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



No!

- This is **binary classification** (of course!)

Linear Classifiers

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

- x : the input
- $\boldsymbol{\phi}(x)$: vector of feature functions $\{\phi_1(x), \phi_2(x), \dots, \phi_I(x)\}$
- \mathbf{w} : the weight vector $\{w_1, w_2, \dots, w_I\}$
- y : the prediction, +1 if “yes”, -1 if “no”
 - ($\text{sign}(v)$ is +1 if $v \geq 0$, -1 otherwise)

Example Feature Functions: Unigram Features

- Equal to “number of times a particular word appears”

x = A site , located in Maizuru , Kyoto

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

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

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

$$\varphi_{\text{unigram "the"}}(x) = 0 \quad \varphi_{\text{unigram "temple"}}(x) = 0$$

...

The rest
are all 0

- For convenience, we use feature names ($\varphi_{\text{unigram "A"}}$) instead of feature indexes (φ_1)

Calculating the Weighted Sum

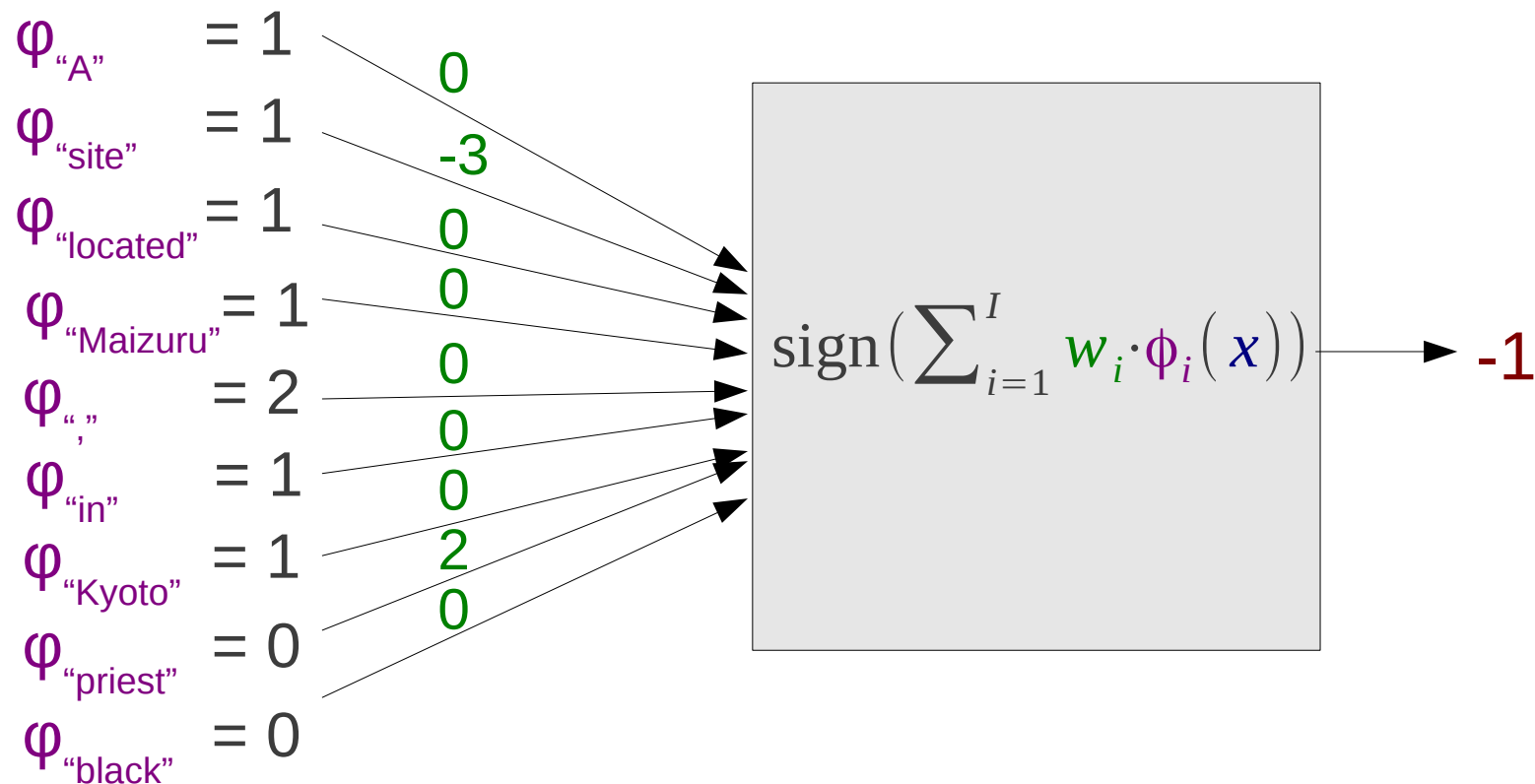
x = A site , located in Maizuru , Kyoto

$\varphi_{\text{unigram "A"}}(x) = 1$		$W_{\text{unigram "a"}} = 0$		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!

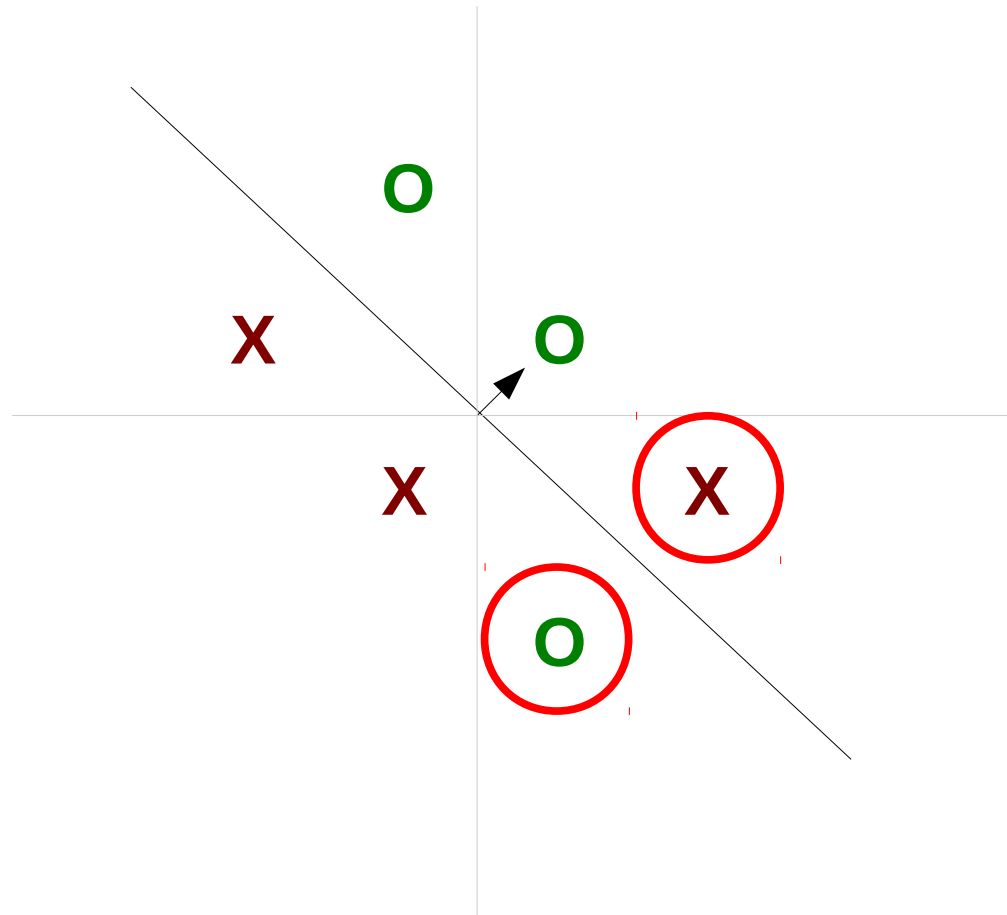
The Perceptron

- Think of it as a “machine” to calculate a weighted sum



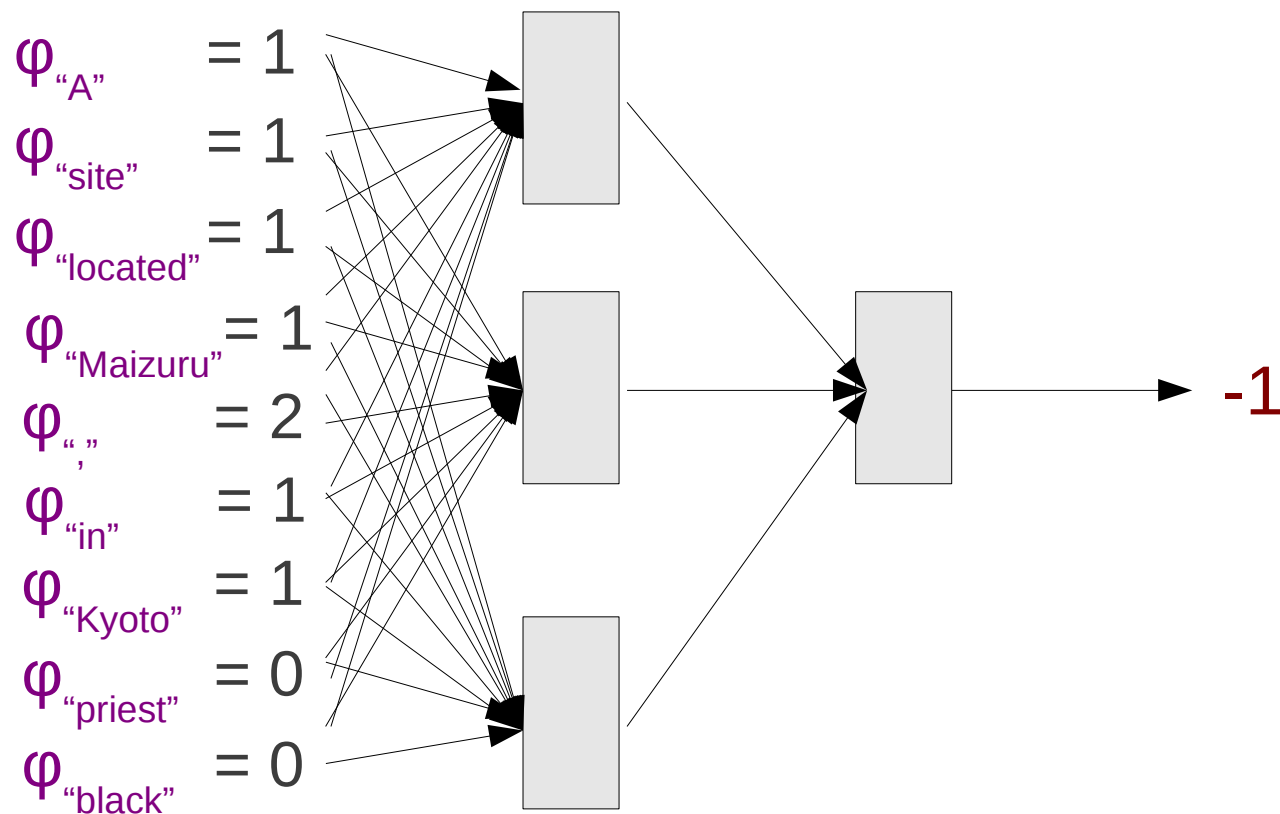
Problem: Linear Constraint

- Perceptron cannot achieve high accuracy on non-linear functions



Neural Networks

- Neural networks connect multiple perceptrons together

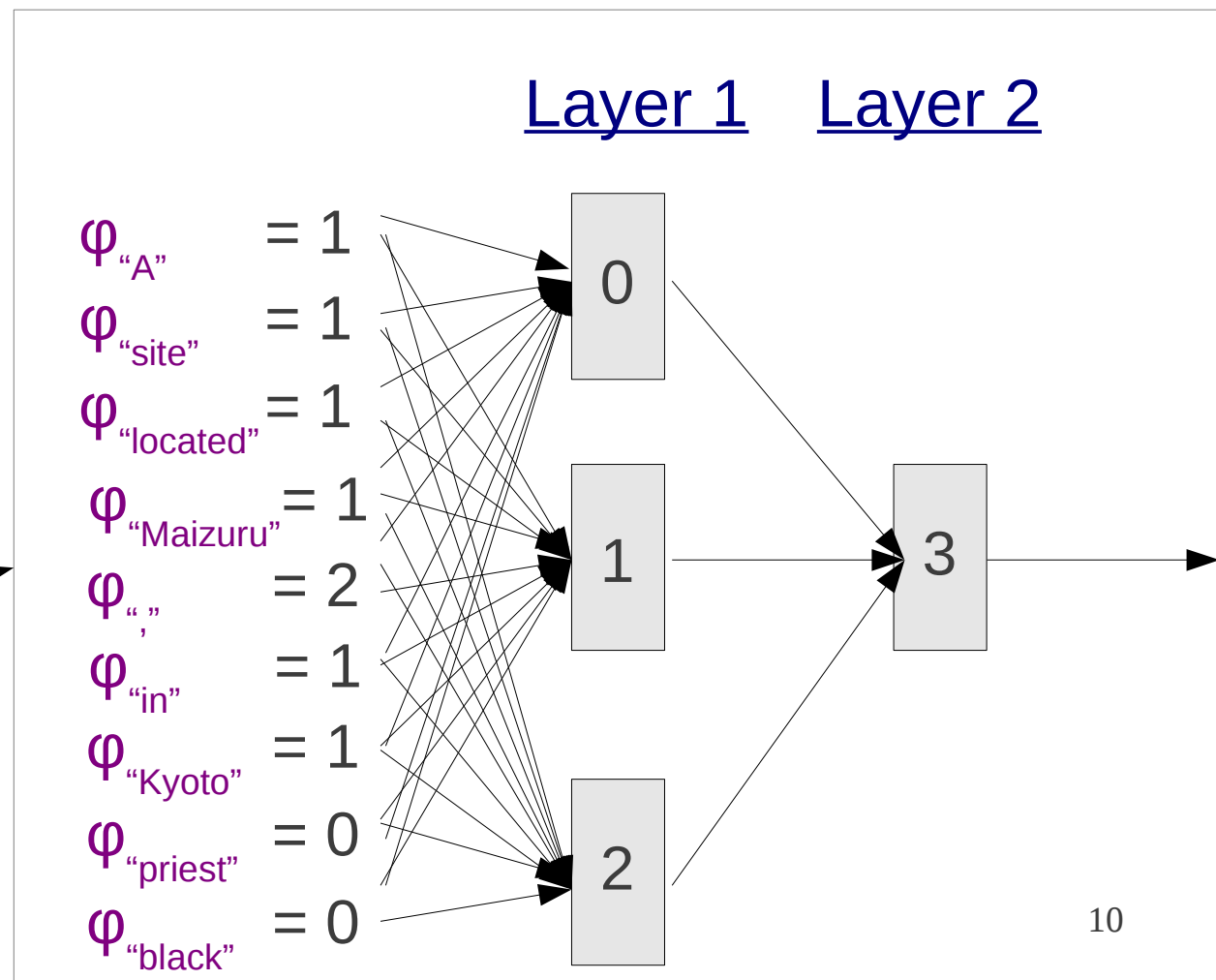


- Motivation: Can express non-linear functions

Representing a Neural Network

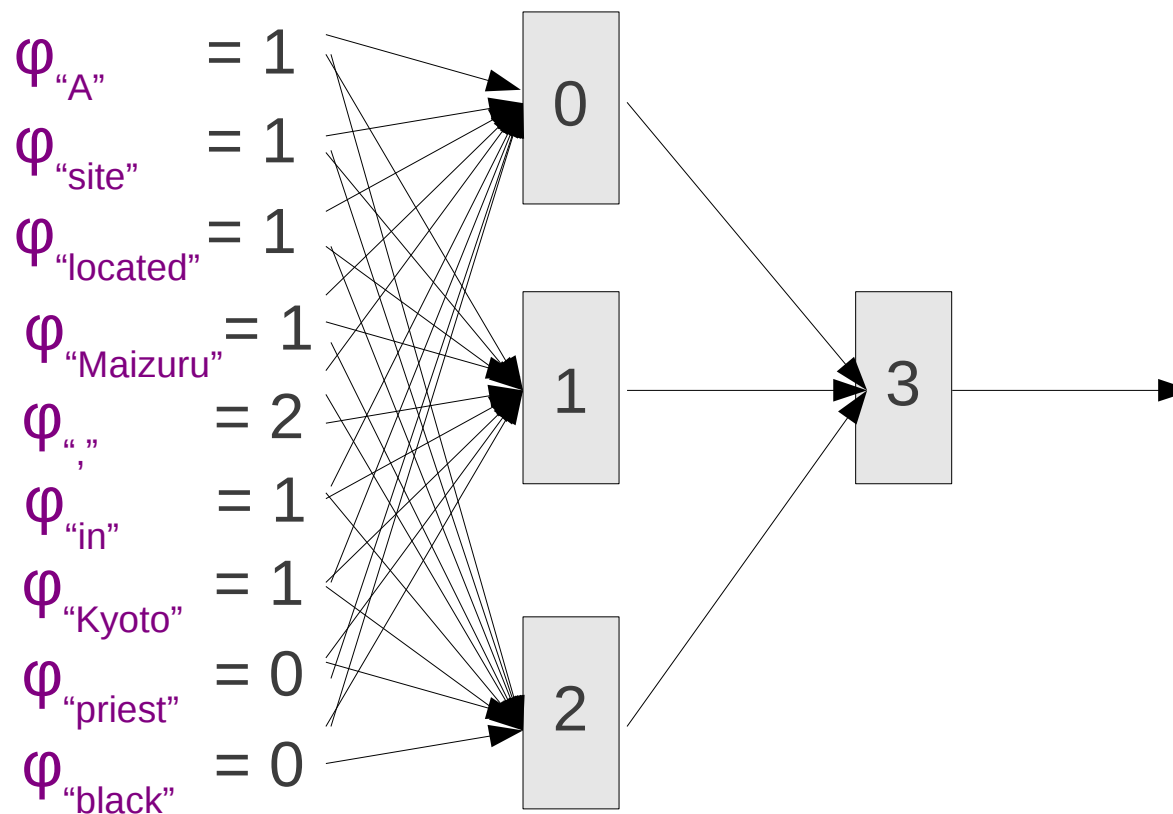
- Assume network is fully connected and in layers
- Each perceptron:
 - A layer ID
 - A weight vector

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



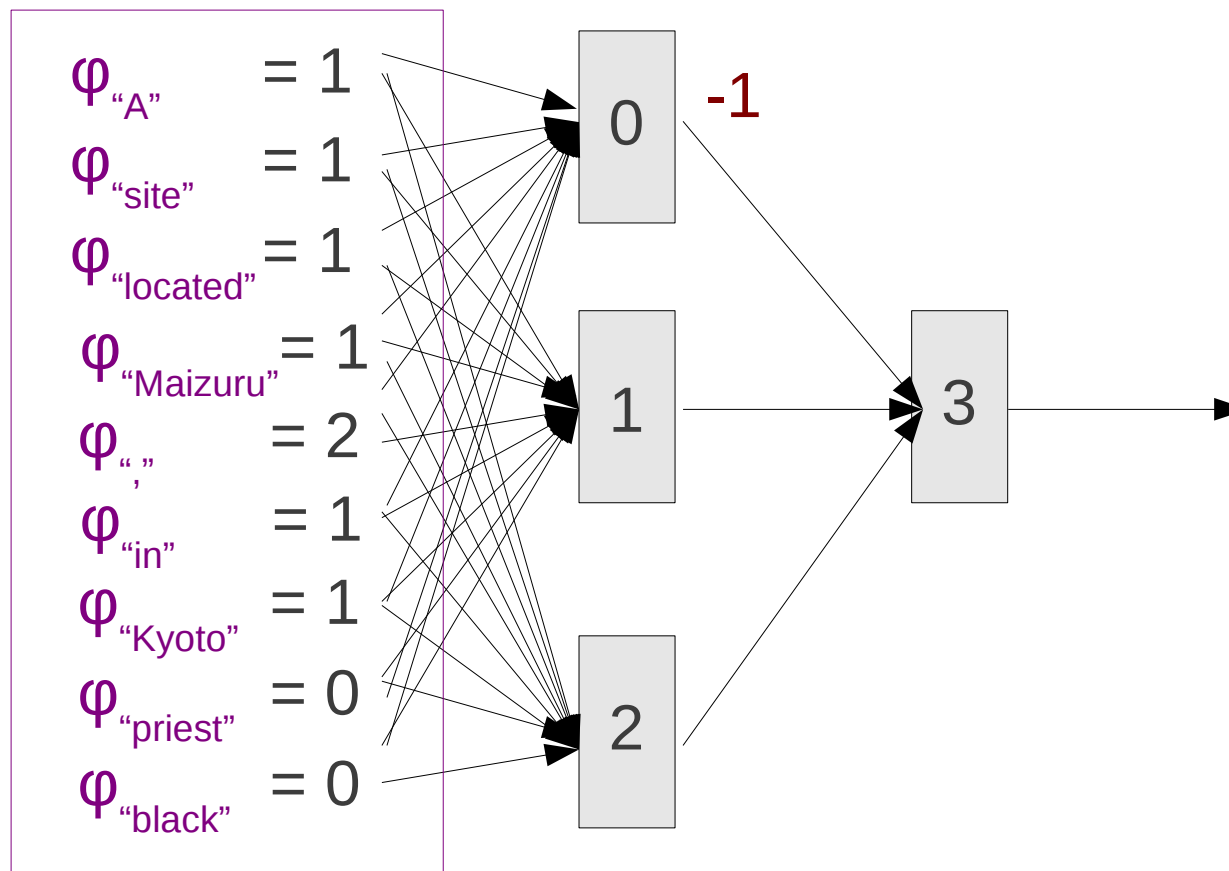
Neural Network Prediction Process

- Predict one perceptron at a time using previous layer



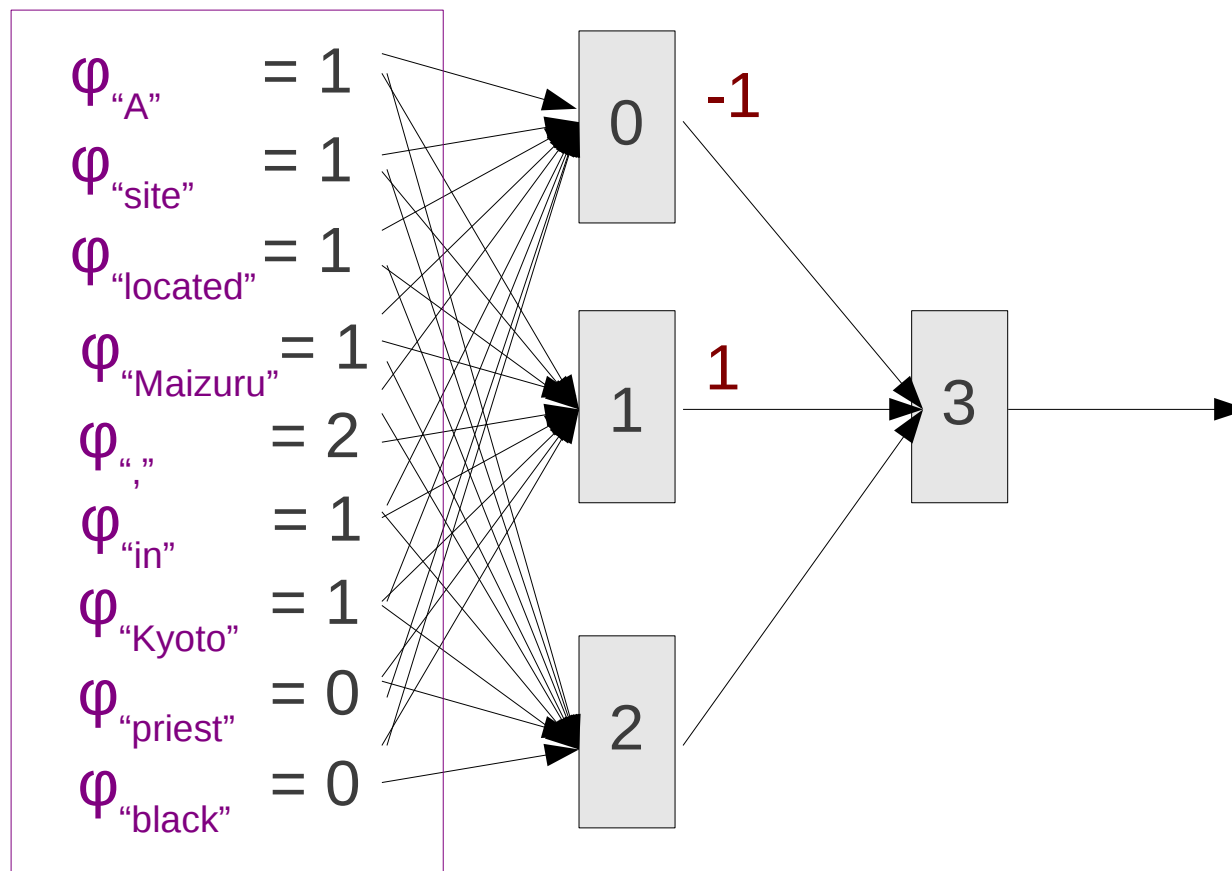
Neural Network Prediction Process

- Predict one perceptron at a time using previous layer



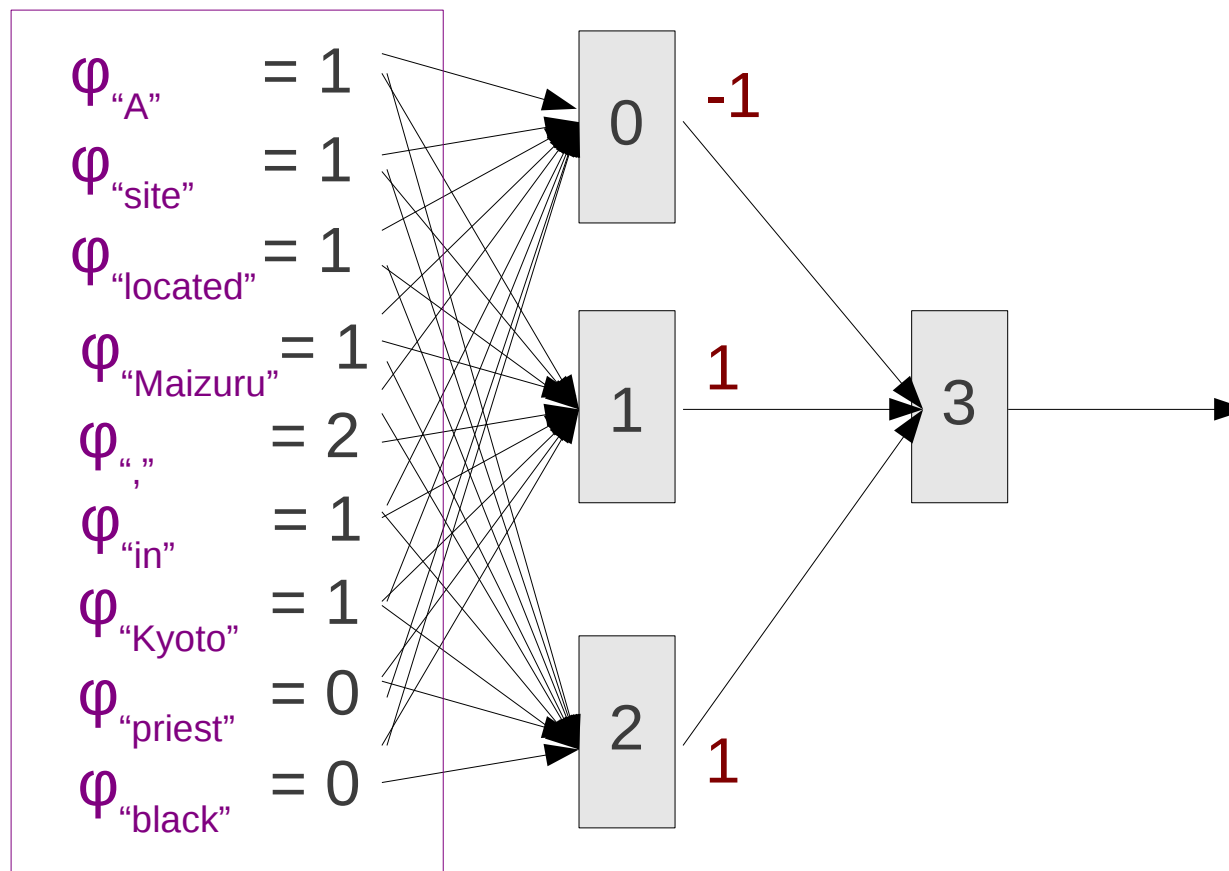
Neural Network Prediction Process

- Predict one perceptron at a time using previous layer



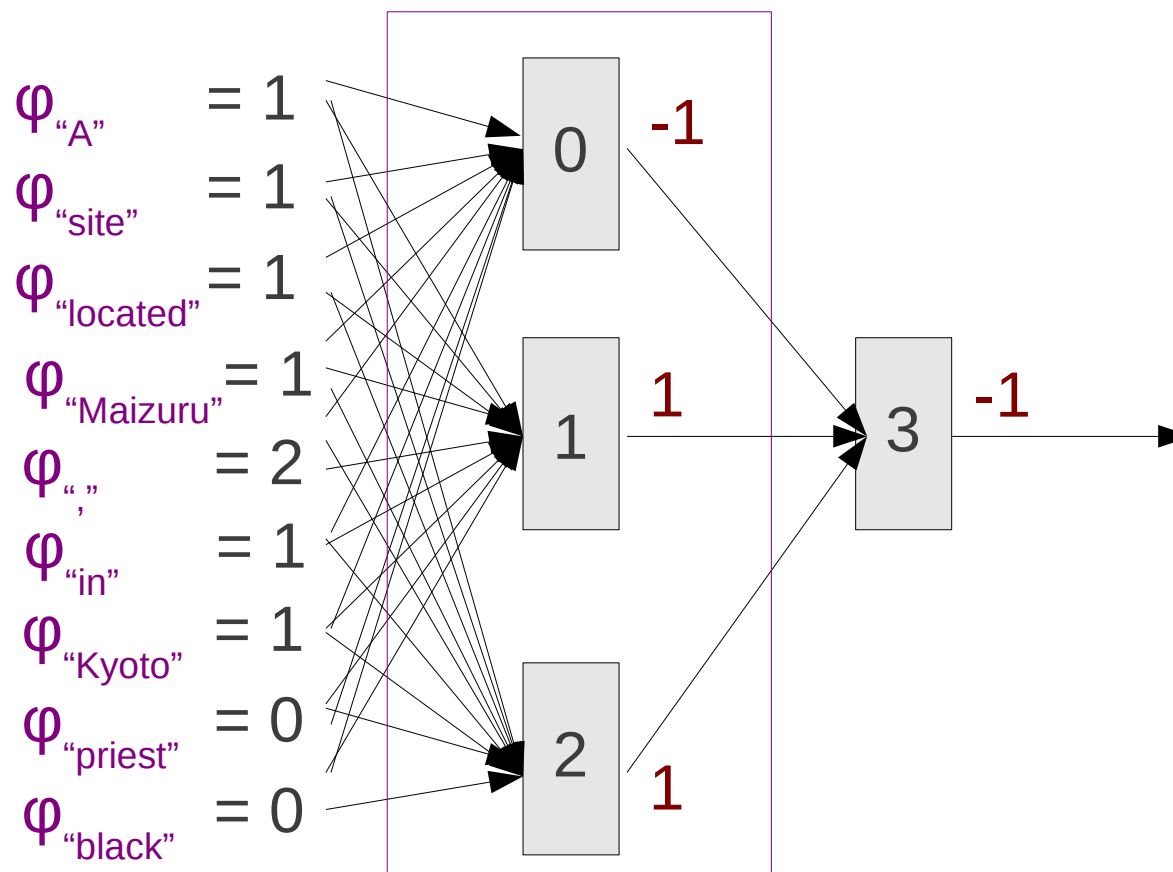
Neural Network Prediction Process

- Predict one perceptron at a time using previous layer



Neural Network Prediction Process

- Predict one perceptron at a time using previous layer



Review:

Pseudo-code for Perceptron Prediction

```

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

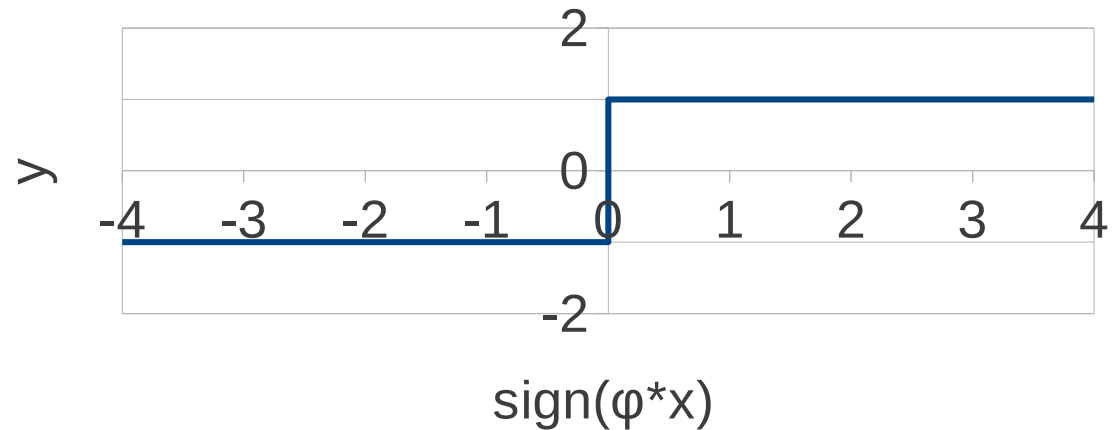

Pseudo-Code for NN Prediction

```
PREDICT_NN(network, phi)  
  y = [ phi, {}, {} ... ] # activations for each layer  
  for each node i:  
    layer, weight = network[i]  
    # predict the answer with the previous perceptron  
    answer = PREDICT_ONE(weight, y[layer-1])  
    # save this answer as a feature for the next layer  
    y[layer][i] = answer  
  return the answer for the last perceptron
```

Neural Network Activation Functions

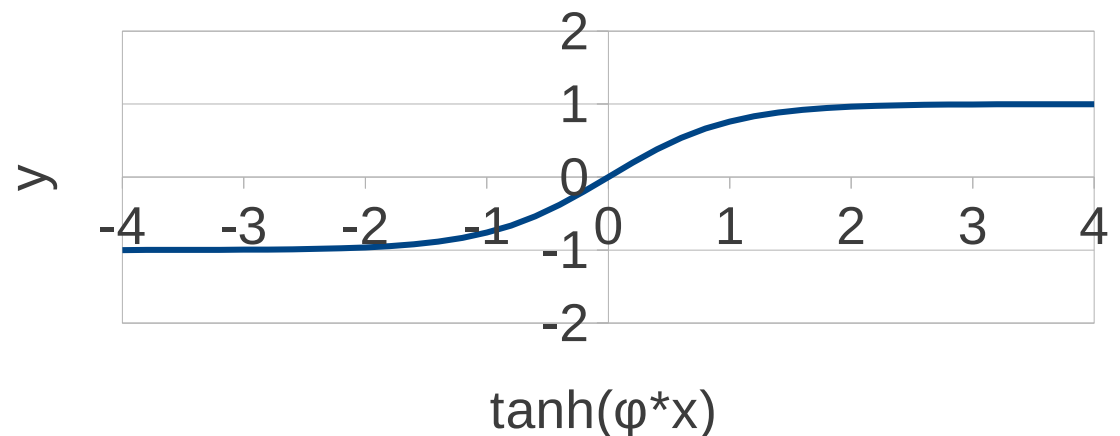
- Previously described NN uses step function

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



- Step function is not differentiable → use tanh

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



Python:

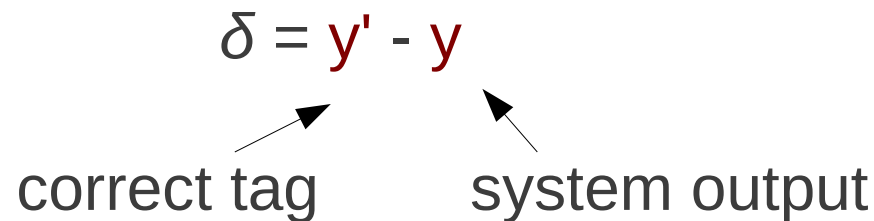
```
from math import tanh
tanh(x)
```

Learning a Perceptron w/ tanh

- First, calculate the error:

$$\delta = y' - y$$

correct tag system output



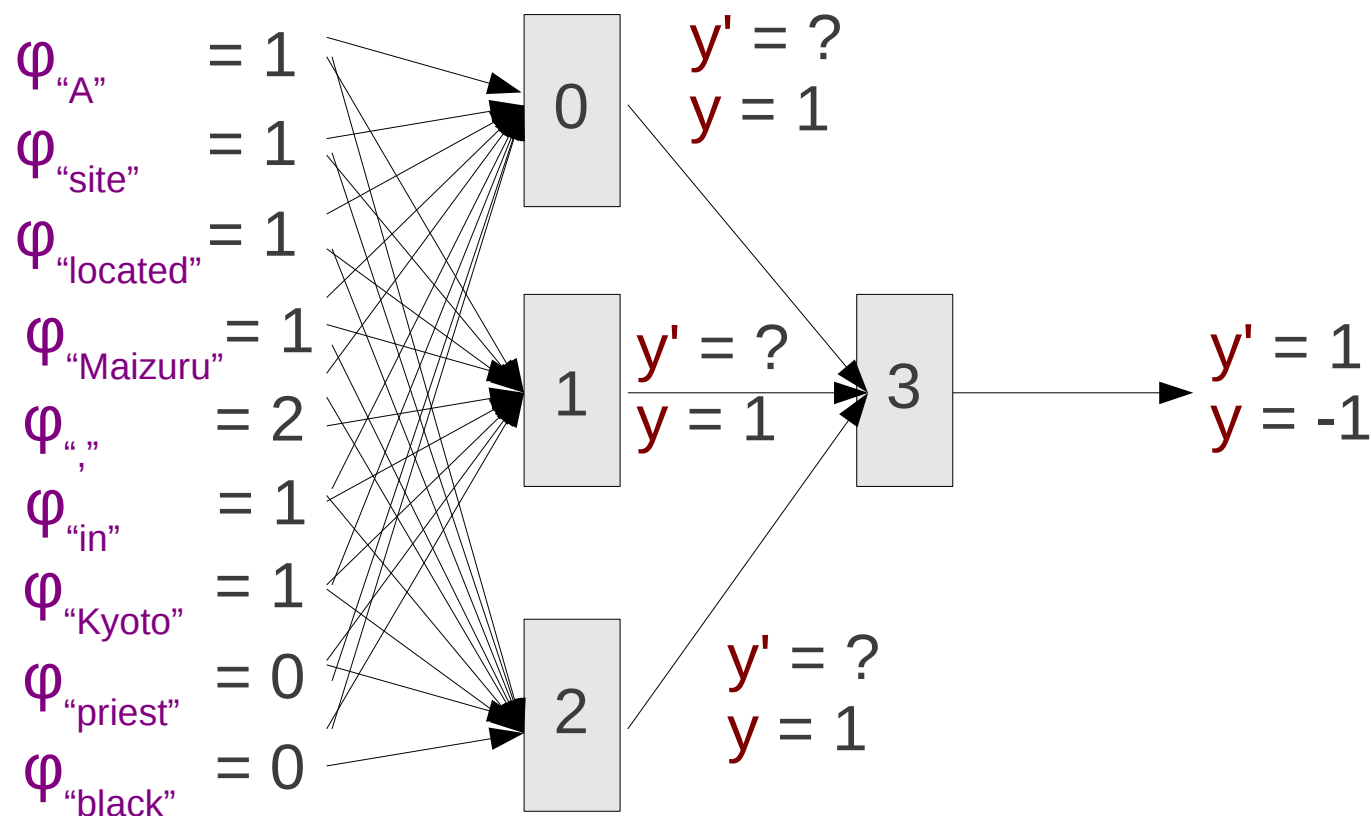
- Update each weight with:

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

- Where λ is the learning rate
- (For step function perceptron $\delta = -2$ or $+2$, $\lambda = 1/2$)

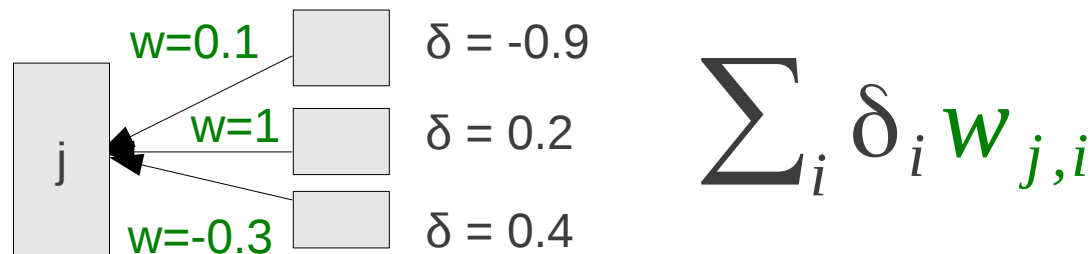
Problem: Don't Know Correct Answer!

- For NNs, only know correct tag for last layer

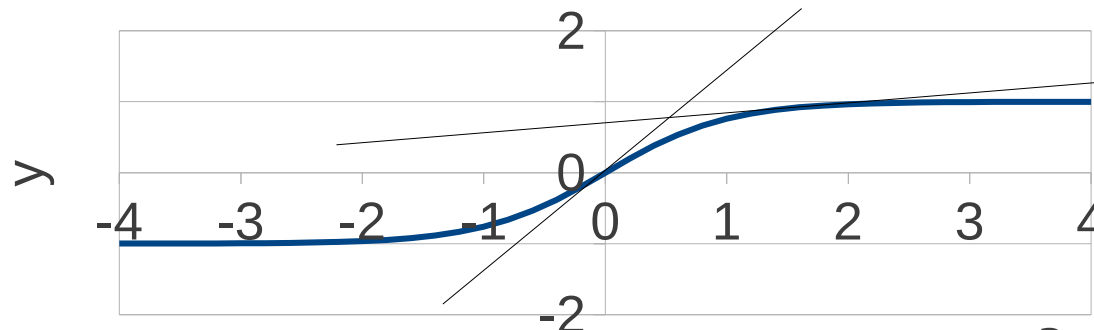


Answer: Back-Propogation

- Pass error backwards along the network



- Also consider gradient of tanh



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

- Combine:

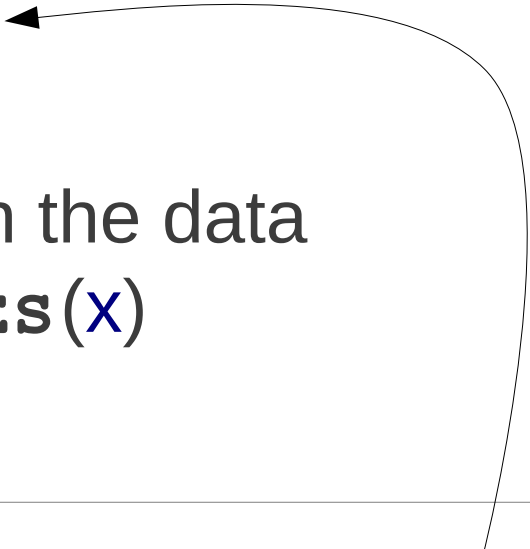
$$\delta_j = (1 - y_j^2) \sum_i \delta_i w_{j,i}$$

Back Propagation Code

```
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] +=  $\lambda * \delta_j * val$ 
```

Training process

```
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)
```



- For previous perceptron, we initialized weights to zero
- In NN: randomly initialize weights (so not all perceptrons are identical)

Exercise

Exercise (1)

- Write two programs
 - train-nn: Creates a neural network model
 - test-nn: Reads a neural network model
- Test train-nn
 - Input: test/03-train-input.txt
 - Use one iteration, one hidden layer, two hidden nodes
 - Calculate updates by hand and make sure they are correct

Exercise (2)

- **Train** a model on data/titles-en-train.labeled
- **Predict** the labels of data/titles-en-test.word
- **Grade** your answers
 - `script/grade-prediction.py data-en/titles-en-test.labeled your_answer`
- **Compare:**
 - With a single perceptron/SVM classifiers
 - With different neural network structures

Thank You!