

# Backpropagation learning algorithm

1. Initialize weights & thresholds to small random values.

$$W1_{ij} = \text{random}(-0.5, 0.5) \quad i = 0, \dots, A; \quad j = 1, \dots, B$$

$$W2_{ij} = \text{random}(-0.5, 0.5) \quad i = 0, \dots, B; \quad j = 1, \dots, C$$

$$x_0 = -1.0$$

$$h_0 = -1.0$$

2. Choose an input/output pair  $(\bar{x}, \bar{y})$  from the training set, where  $\bar{x} = (x_1, \dots, x_A)$  and  $\bar{y} = (y_1, \dots, y_C)$ . Assign activation levels to input units.

3. Determine activation levels of hidden units.

$$h_j = f \left( \sum_{i=0}^A W1_{ij} \cdot x_i \right) \quad j = 1, \dots, B$$

4. Determine activation levels of output units.

$$o_j = f \left( \sum_{i=0}^B W2_{ij} \cdot h_i \right) \quad j = 1, \dots, C$$

5. Determine how to adjust weights between hidden and output layer to reduce error for this training example.

$$E2_j = k \cdot o_j(1 - o_j)(y_j - o_j) \quad j = 1, \dots, C$$

6. Determine how to adjust weights between input and hidden layer to reduce error for this training example.

$$E1_j = k \cdot h_j \cdot (1 - h_j) \sum_{i=1}^C E2_i \cdot W2_{ji} \quad j = 1, \dots, B$$

7. Adjust weights between hidden and output layer.

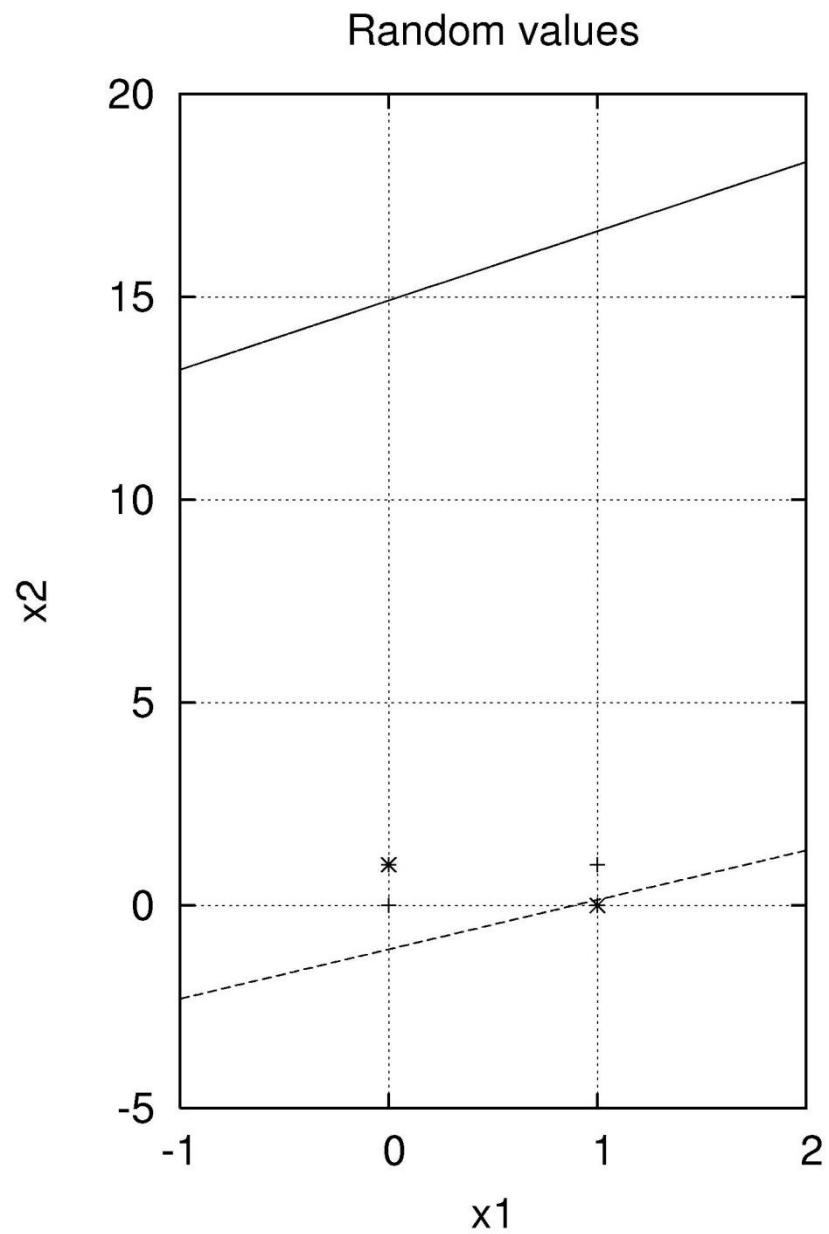
$$W2_{ij} = W2_{ij} + \text{LearningRate} \cdot E2_j \cdot h_i \quad i = 0, \dots, B; \quad j = 1, \dots, C$$

8. Adjust weights between input and hidden layer.

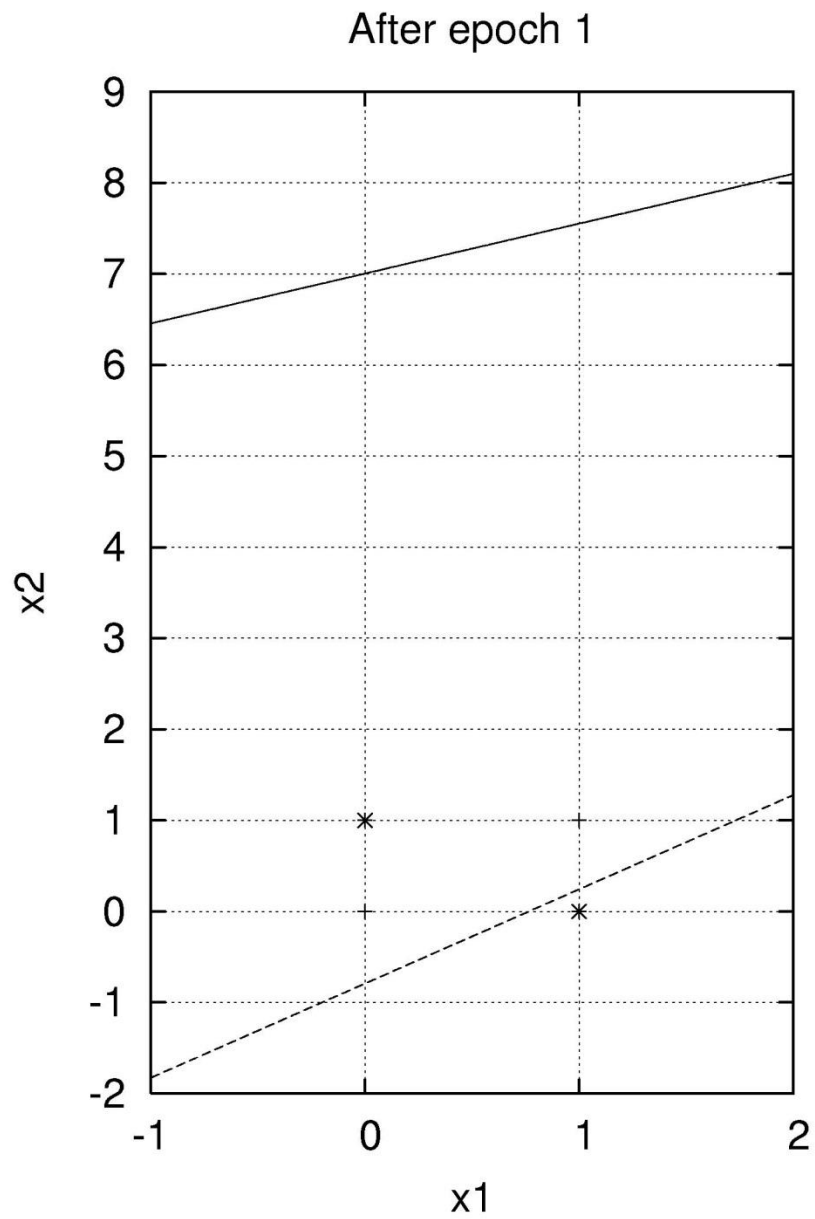
$$W1_{ij} = W1_{ij} + \text{LearningRate} \cdot E1_j \cdot x_i \quad i = 0, \dots, A; \quad j = 1, \dots, B$$

9. If stopping criteria not met, goto step 2 and repeat.

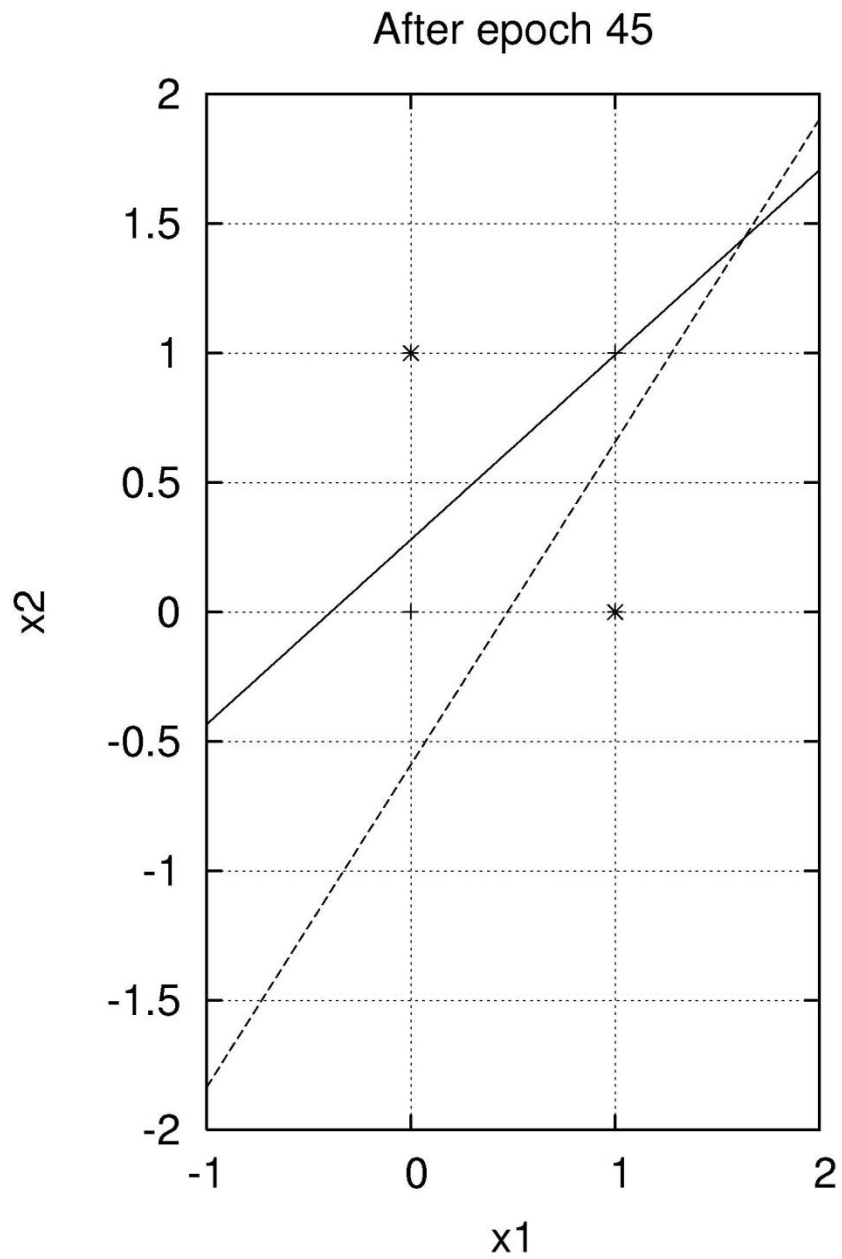
# XOR Example



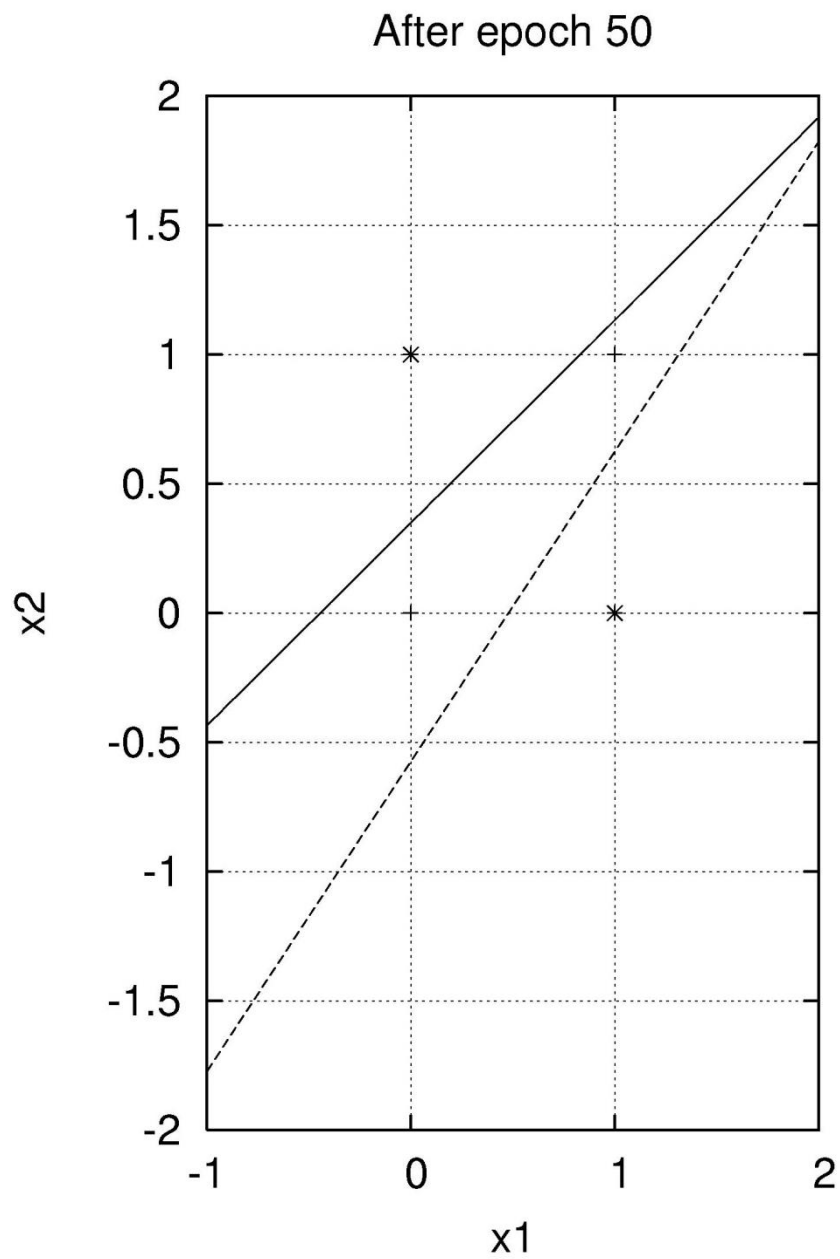
# XOR Example



## XOR Example



## XOR Example



# Parameters

- Learning rate
  - range: 0.05 to 0.35
  - constant vs dynamic (e.g., 1/iteration)

- Multiplicative constant  $k$

$$f(x) = \frac{1}{1 + e^{-kx}}$$

- Hidden units

# Stopping criteria

- Maximum number of epochs
  - epoch is one time through the training set
  - in case of no convergence
- Error is acceptably small
  - error on training set or on separate validation set
  - classification error, *or*
  - continuous error
- can distinguished between training and deployment level
  - e.g., train so that
    - $[0, 0.2] \rightarrow 0$
    - $[0.8, 1] \rightarrow 1$
  - deploy
    - $[0, 0.5) \rightarrow 0$
    - $[0.5, 1] \rightarrow 1$

# Preventing overfitting

- Early stopping
  - split training set into training and validation
  - stop learning when error on validation set increases for a specified number of iterations
- Regularization
  - add a penalty term to the error function that penalizes large weights



# Architectures

- scale inputs/outputs
  - map inputs/outputs to  $[0, 1]$ , *or*
  - mean zero, standard deviation of one
- unary, binary, and discretized real-valued encodings



Jeeves is a valet to Bertie Wooster. On some days, Bertie likes to play tennis and asks Jeeves to lay out his tennis things and book the court. Jeeves would like to be able to predict whether Bertie will play tennis (and so be a better valet). Each morning over the last two weeks, Jeeves has recorded whether Bertie played tennis on that day and various attributes of the weather.

Day	Outlook	Temp	Humidity	Wind	Tennis?
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

Can Jeeves learn to predict Bertie's tennis playing?



Jeeves would like to evaluate the classifier he has come up with for predicting whether Bertie will play tennis. Each morning over the next two weeks, Jeeves records the following data.

Day	Outlook	Temp	Humidity	Wind	Tennis?
1	Sunny	Mild	High	Strong	No
2	Rain	Hot	Normal	Strong	No
3	Rain	Cool	High	Strong	No
4	Overcast	Hot	High	Strong	Yes
5	Overcast	Cool	Normal	Weak	Yes
6	Rain	Hot	High	Weak	Yes
7	Overcast	Mild	Normal	Weak	Yes
8	Overcast	Cool	High	Weak	Yes
9	Rain	Cool	High	Weak	Yes
10	Rain	Mild	Normal	Strong	No
11	Overcast	Mild	High	Weak	Yes
12	Sunny	Mild	Normal	Weak	Yes
13	Sunny	Cool	High	Strong	No
14	Sunny	Cool	High	Weak	No

How well does Jeeves predict Bertie's tennis playing?

# *Unary* encoding of Jeeves data

- Inputs:

Outlook	$x_1$	$x_2$	$x_3$
Sunny	1	0	0
Overcast	0	1	0
Rain	0	0	1

Temp	$x_4$	$x_5$	$x_6$
Hot	1	0	0
Mild	0	1	0
Cool	0	0	1

Humidity	$x_7$	$x_8$
High	1	0
Normal	0	1

Wind	$x_9$	$x_{10}$
Weak	1	0
Strong	0	1

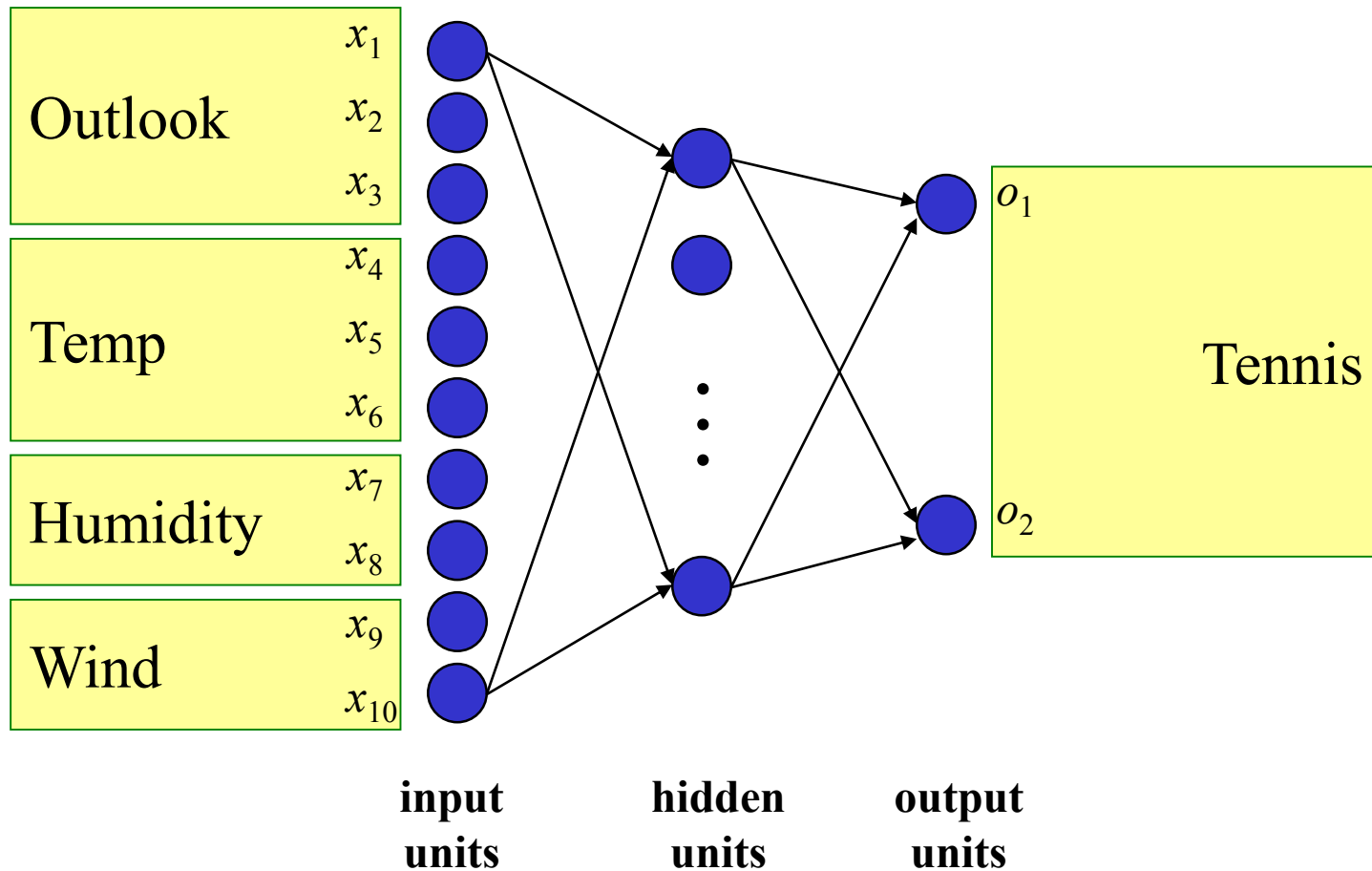
- Output:

Tennis	$y_1$	$y_2$
yes	1	0
no	0	1

## Training data using *unary* encoding

Day	Outlook	Temp	Humidity	Wind	Tennis?
	$x_1, x_2, x_3$	$x_4, x_5, x_6$	$x_7, x_8$	$x_9, x_{10}$	$y_1, y_2$
1	1, 0, 0	1, 0, 0	1, 0	1, 0	0, 1
2	1, 0, 0	1, 0, 0	1, 0	0, 1	0, 1
3	0, 1, 0	1, 0, 0	1, 0	1, 0	1, 0
4	0, 0, 1	0, 1, 0	1, 0	1, 0	1, 0
5	0, 0, 1	0, 0, 1	0, 1	1, 0	1, 0
6	0, 0, 1	0, 0, 1	0, 1	0, 1	0, 1
7	0, 1, 0	0, 0, 1	0, 1	0, 1	1, 0
8	1, 0, 0	0, 1, 0	1, 0	1, 0	0, 1
9	1, 0, 0	0, 0, 1	0, 1	1, 0	1, 0
10	0, 0, 1	0, 1, 0	0, 1	1, 0	1, 0
11	1, 0, 0	0, 1, 0	0, 1	0, 1	1, 0
12	0, 1, 0	0, 1, 0	1, 0	0, 1	1, 0
13	0, 1, 0	1, 0, 0	0, 1	1, 0	1, 0
14	0, 0, 1	0, 1, 0	1, 0	0, 1	0, 1

# Network for *unary* encoding of Jeeves data



# *Binary* encoding of Jeeves data

- Inputs:

Outlook	$x_1$	$x_2$
Sunny	1	0
Overcast	0	1
Rain	1	1

Humidity	$x_5$
High	1
Normal	0

Temp	$x_3$	$x_4$
Hot	1	0
Mild	0	1
Cool	1	1

Wind	$x_6$
Weak	1
Strong	0

- Output:

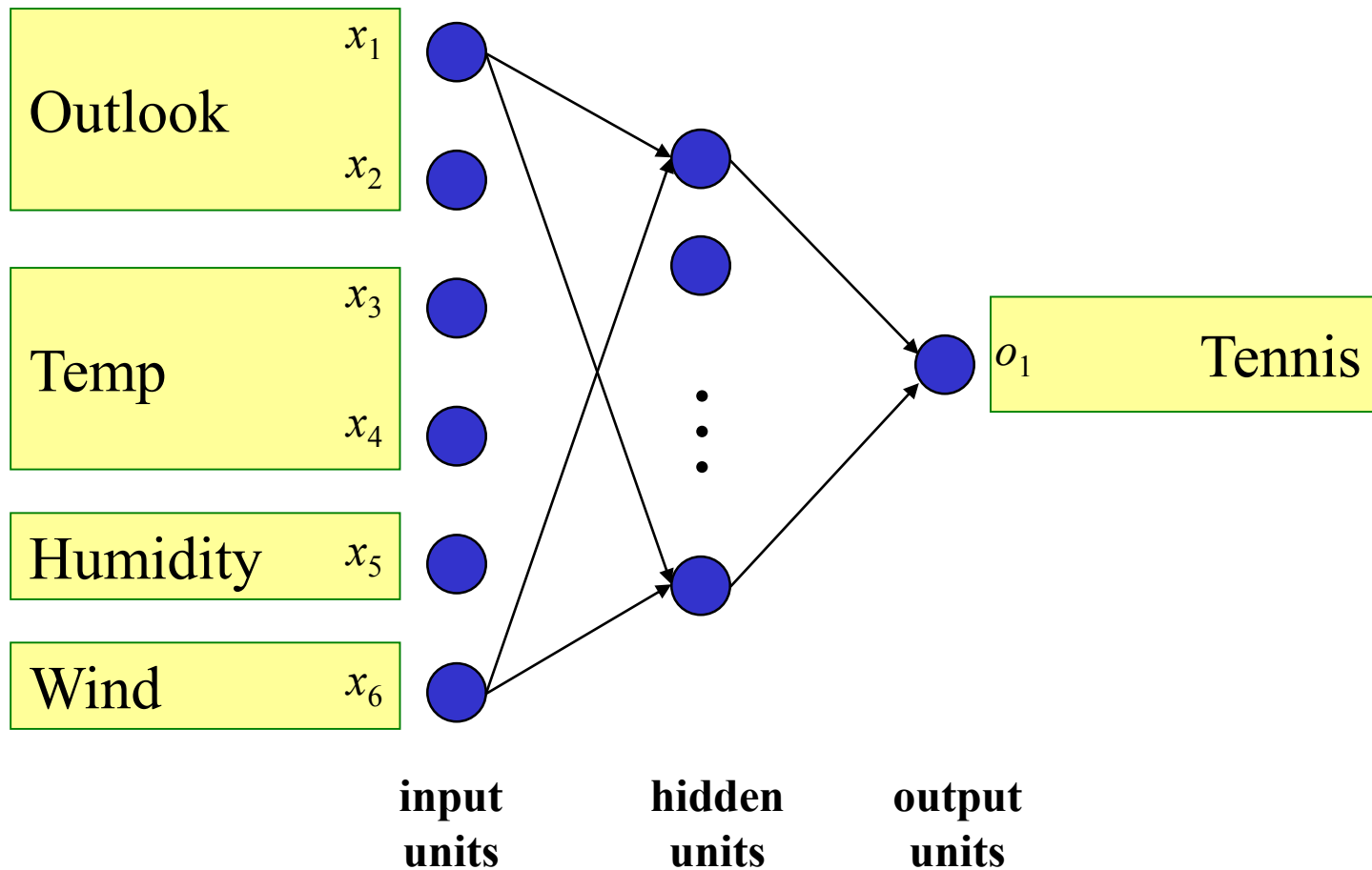
Tennis	$y_1$
yes	1
no	0

## Training data using *binary* encoding

Day	Outlook	Temp	Humidity	Wind	Tennis?
	$x_1, x_2$	$x_3, x_4$	$x_5$	$x_6$	$y_1$
1	1, 0	1, 0	1	1	0
2	1, 0	1, 0	1	0	0
3	0, 1	1, 0	1	1	1
4	1, 1	0, 1	1	1	1
5	1, 1	1, 1	0	1	1
6	1, 1	1, 1	0	0	0
7	0, 1	1, 1	0	0	1
8	1, 0	0, 1	1	1	0
9	1, 0	1, 1	0	1	1
10	1, 1	0, 1	0	1	1
11	1, 0	0, 1	0	0	1
12	0, 1	0, 1	1	0	1
13	0, 1	1, 0	0	1	1
14	1, 1	0, 1	1	0	0



# Network for *binary* encoding of Jeeves data



# *Real-valued* encoding of Jeeves data

- Inputs:

Outlook	$x_1$
Sunny	0.833
Overcast	0.500
Rain	0.167

Humidity	$x_3$
High	0.833
Normal	0.167

Temp	$x_2$
Hot	0.833
Mild	0.500
Cool	0.167

Wind	$x_4$
Weak	0.833
Strong	0.167

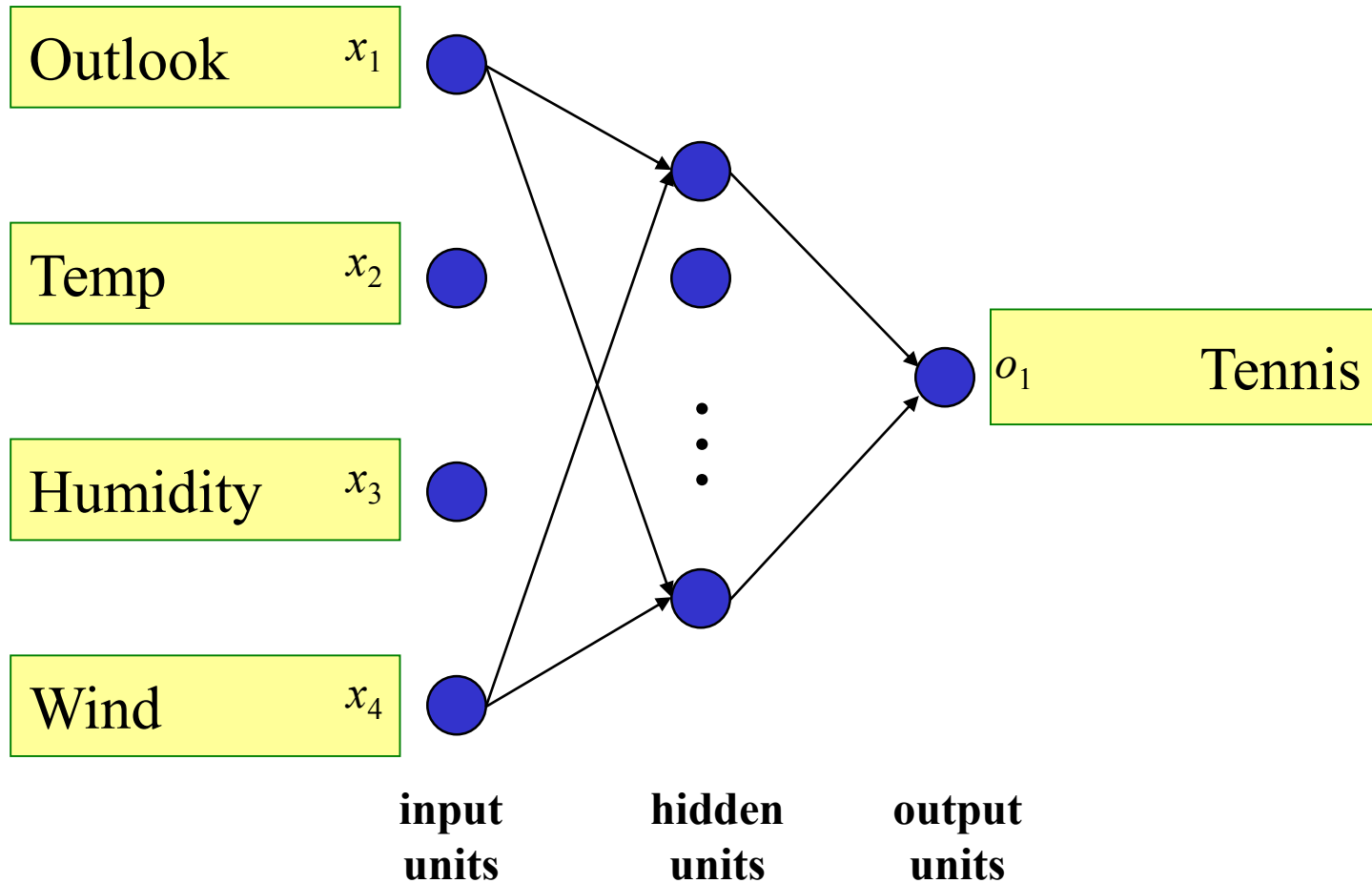
- Output:

Tennis	$y_1$
yes	0.833
no	0.167

## Training data using *real-valued* encoding

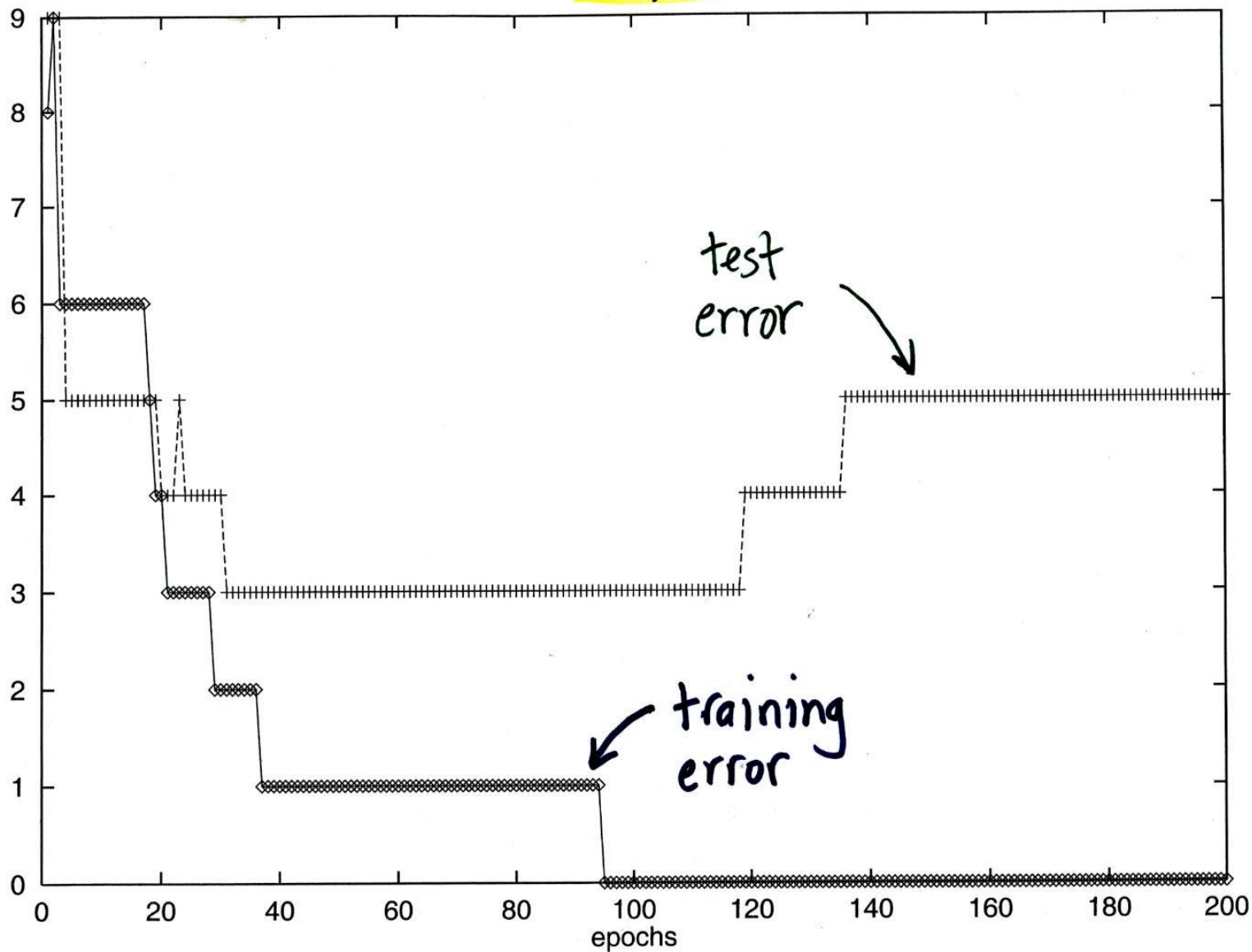
Day	Outlook	Temp	Humidity	Wind	Tennis?
	$x_1$	$x_2$	$x_3$	$x_4$	$y_1$
1	0.833	0.833	0.833	0.833	0.167
2	0.833	0.833	0.833	0.167	0.167
3	0.500	0.833	0.833	0.833	0.833
4	0.167	0.500	0.833	0.833	0.833
5	0.167	0.167	0.167	0.833	0.833
6	0.167	0.167	0.167	0.167	0.167
7	0.500	0.167	0.167	0.167	0.833
8	0.833	0.500	0.833	0.833	0.167
9	0.833	0.167	0.167	0.833	0.833
10	0.167	0.500	0.167	0.833	0.833
11	0.833	0.500	0.167	0.167	0.833
12	0.500	0.500	0.833	0.167	0.833
13	0.500	0.833	0.167	0.833	0.833
14	0.167	0.500	0.833	0.167	0.167

# Network for *real-valued* encoding of Jeeves data

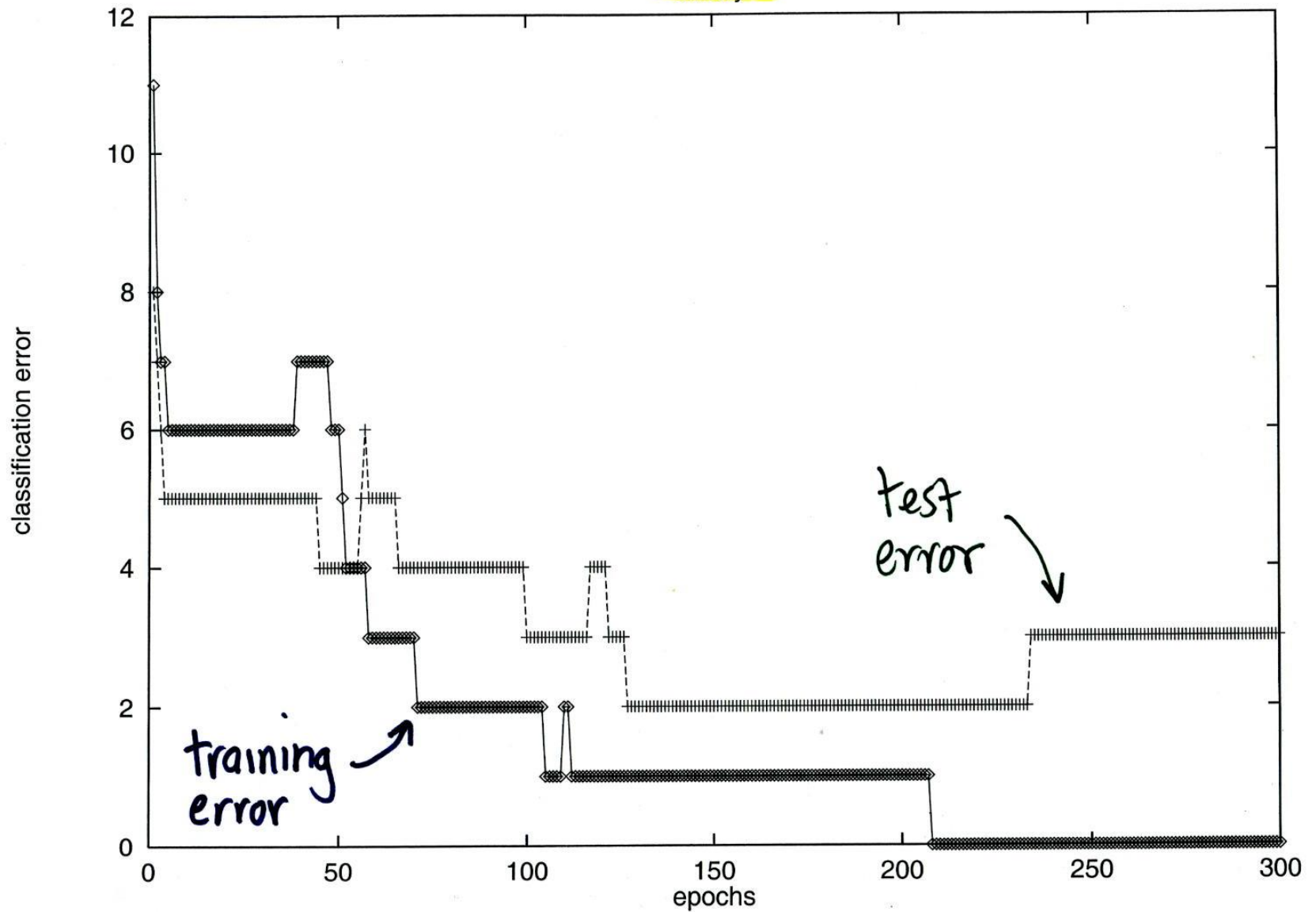


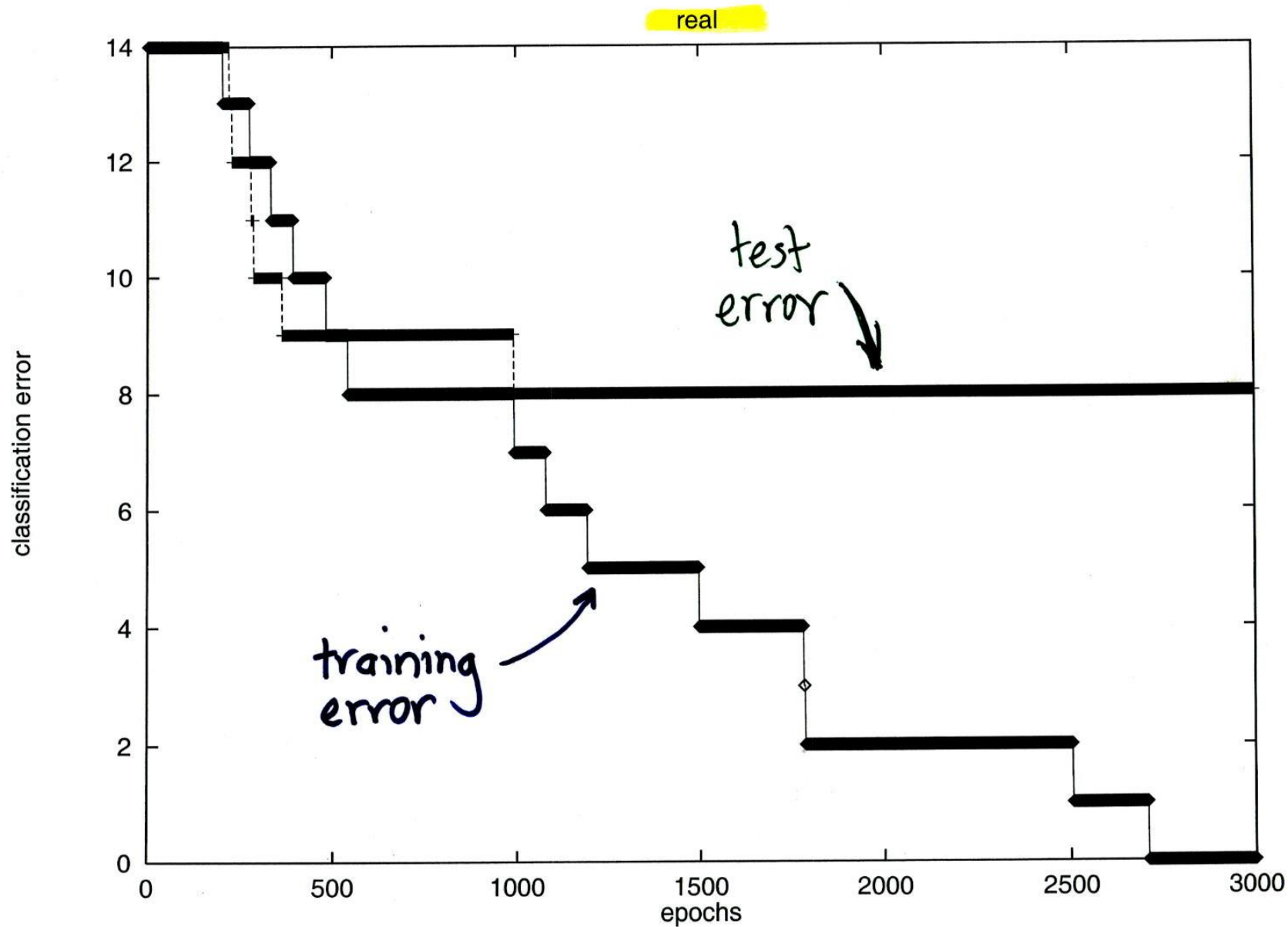
unary

classification error



binary





# Practical Considerations: Training Data

- Size of the data must be large
  - if the problem lacks adequate data, the use of neural networks is not recommended
- Training data must be representative
- Divide data into training set, validation and test set



# Practical Considerations: Network Sizing

- How many units in input layer?
  - decide on features
  - decide on unary, binary, or discretized real-valued inputs
- How many hidden units are needed?
  - use as few as possible, small fraction of input units
  - if learning doesn't converge, try with more units
- How many units in output layer?
  - decide on unary, binary, or discretized real-valued outputs

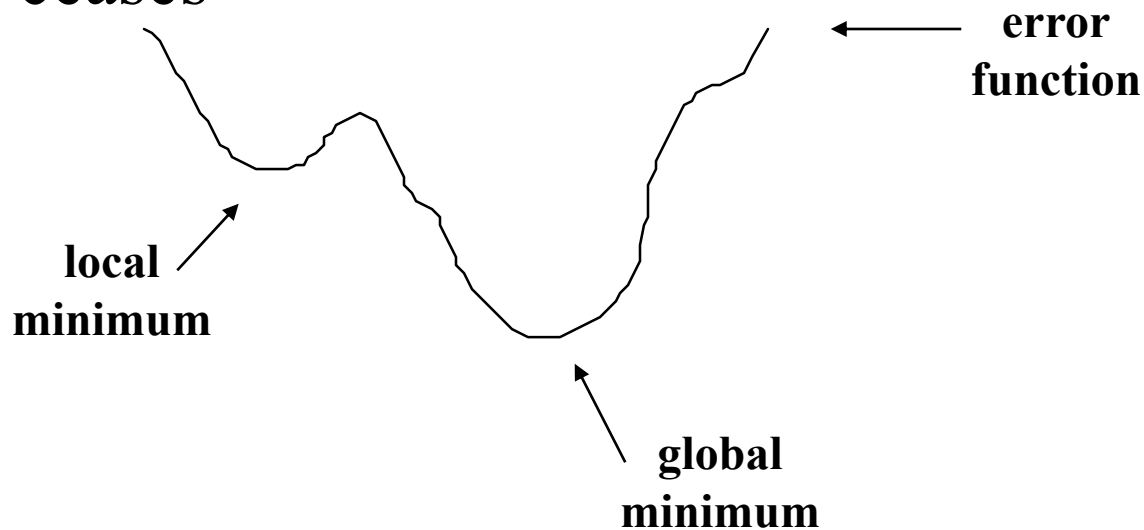
# Practical Considerations: Training the Network

- The weights are initialized to small random values in the range -0.5 to 0.5
- Many iterations through training set (millions) may be needed before stable solution found
  - each iteration through the training set is called an epoch

# Practical Considerations:

## Local minima

- The learning rule is not guaranteed to converge
- The network may settle into a local minima, where learning ceases



# Practical Considerations:

## Local minima

- If learning ceases and error is still too high:
  - start again with new random weights
  - add more hidden nodes to the network
  - experiment with various learning parameters
    - learning rate
    - “squashing” the sigmoid function
    - momentum
- In general, much experimentation is needed before the network converges to an acceptable solution

# Example Applications of Neural Networks


- NETtalk for pronouncing English text  
(Sejnowski & Rosenberg '87)
- Paint-quality inspection
- Vision-based autonomous driving  
(Carnegie-Mellon University)

# NETtalk

- Input layer: 203 units (7 x 29; “a...z,. ”)

**A window 7 characters wide is moved over the text.  
The network learns to pronounce the middle letter.**

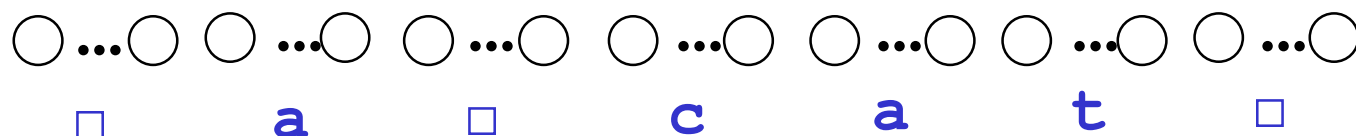
there once was a cat that

window 

**The sound of a letter depends on its context. For example, the letter “a” is pronounced differently in “mean”, “lamb”, and “class”.**

# NETtalk

- Sample input:



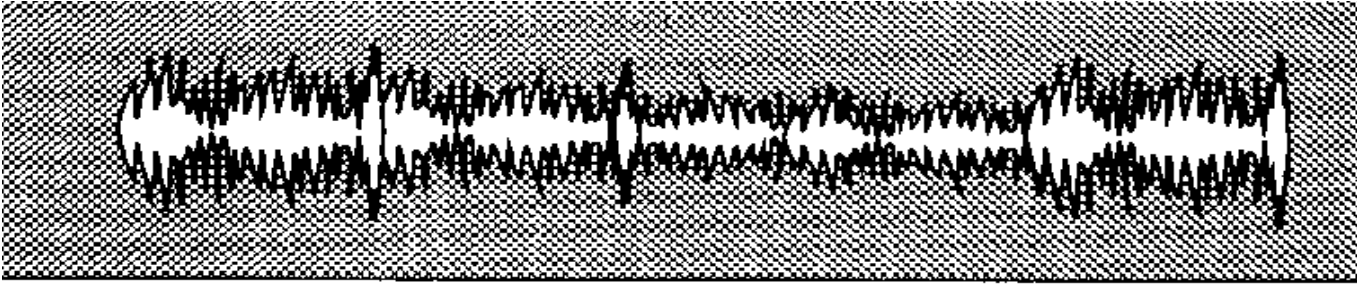
- Hidden layer: 80 units
- Output layer: 26 units
  - one for each phoneme in English (a phoneme is a basic sound in a language)
- Once trained, the network achieves:
  - 90% correct pronunciation for training set
  - 80%-87% correct on new data

# Paint-Quality Inspection

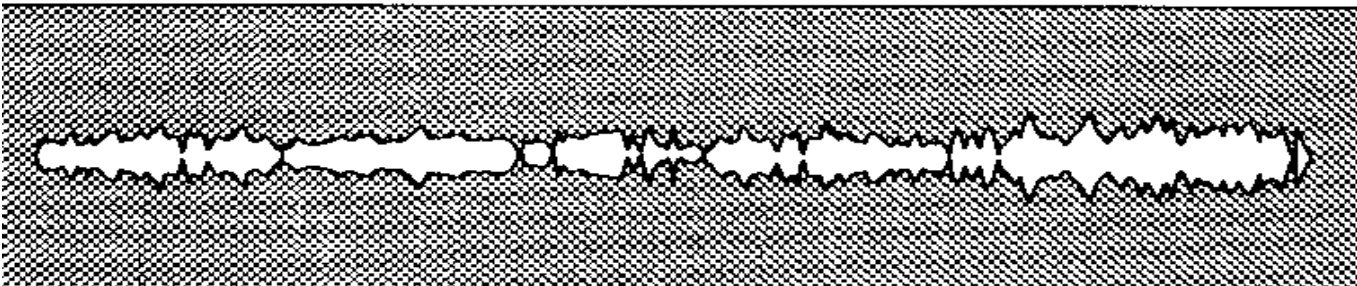
- Inspection of painted surfaces such as car body panels
- Method:
  - reflect laser beam off panel onto projection screen
  - poor paint job (ripples, orange peel, low shine) will give a diffuse image
  - neural network to categorize using 20 categories ranging from best possible finish to worst possible finish



# Two typical scatter patterns



**(a) A poor-quality finish**



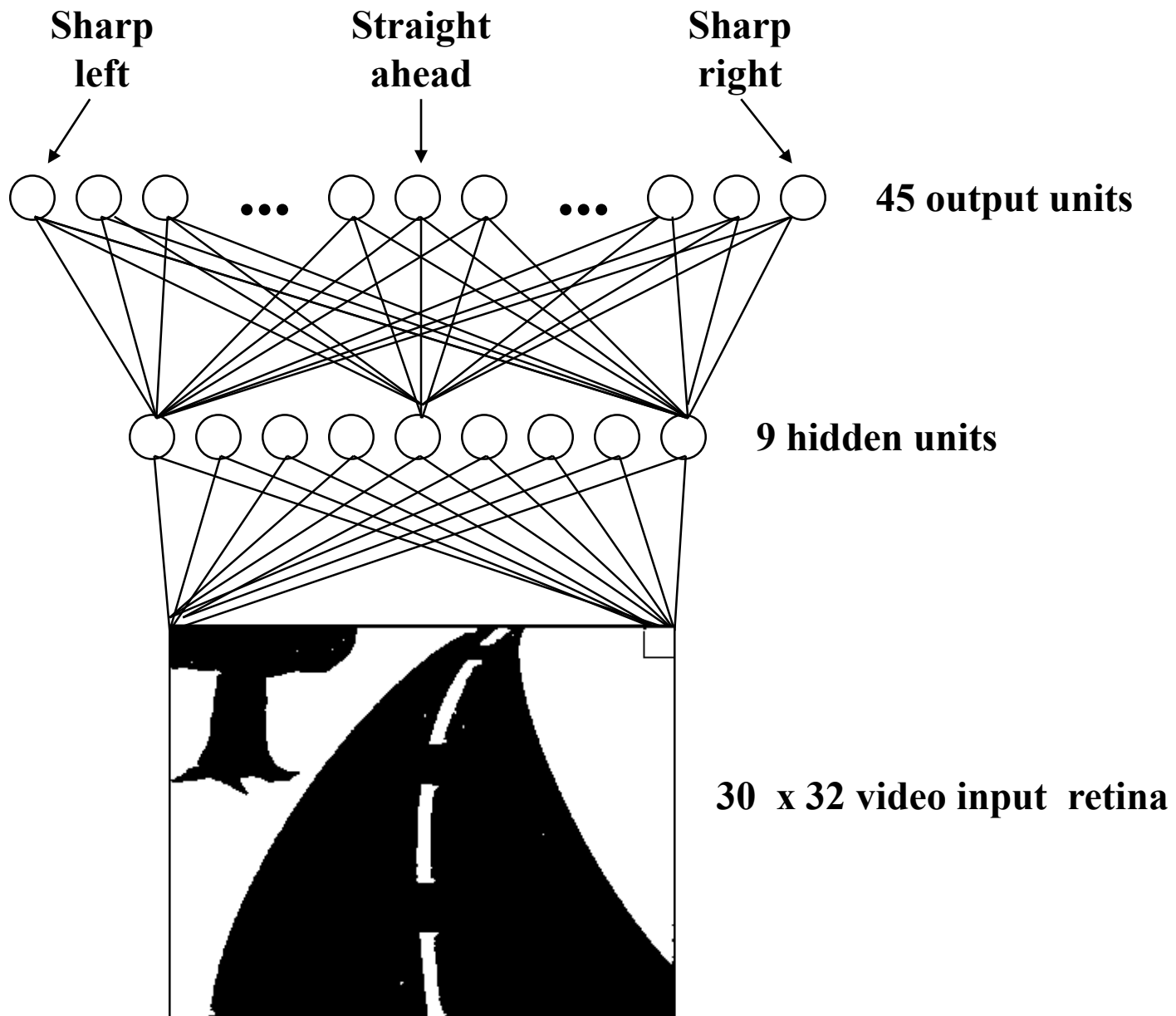
**(b) A better-quality finish**

# Neural network for paint-quality inspection

- Input layer: 900 units (30x30 pixels)
  - each pixel has one of 256 values
  - scaled to a floating point number in the range 0 to 1
- Hidden layer: 50 units
- Output layer: 1 unit
  - a floating point number in the range 0 to 1
  - subdivided into 20 subintervals ranging from the best possible finish to the worst possible finish
- Network was trained on 130,000 patterns
- Results consistent with human experts

# Vision-based autonomous driving

- ALVINN: Autonomous Land Vehicle In a Neural Net
- Automatic road following (driverless driving) using color vision
- Input to the network: image information from a video camera
- Output of the network: vehicle heading such that the vehicle stays on the road



# Neural network for autonomous driving

- The ALVINN network is trained “on-the-fly” as the van is driven by a person down a highway
  - road images give input
  - desired output is the person’s steering actions
- ALVINN has been successfully driven on various types of road (paved, dirt, single-lane, multi-lane) and in various weather conditions
- Experiment: autonomous driving at 55 mph for more than 90 miles on a highway near Pittsburgh