Neural Network II

Week 8

Team Homework Assignment #10

- Read pp. 327 – 334.
- Do Example 6.9.
- Explore neural network tools and try to use a tool for solving Example 6.9 (or you can do R programming for solving Example 6.9)
- beginning of the lecture on Friday March25th.

Keywords for ANN

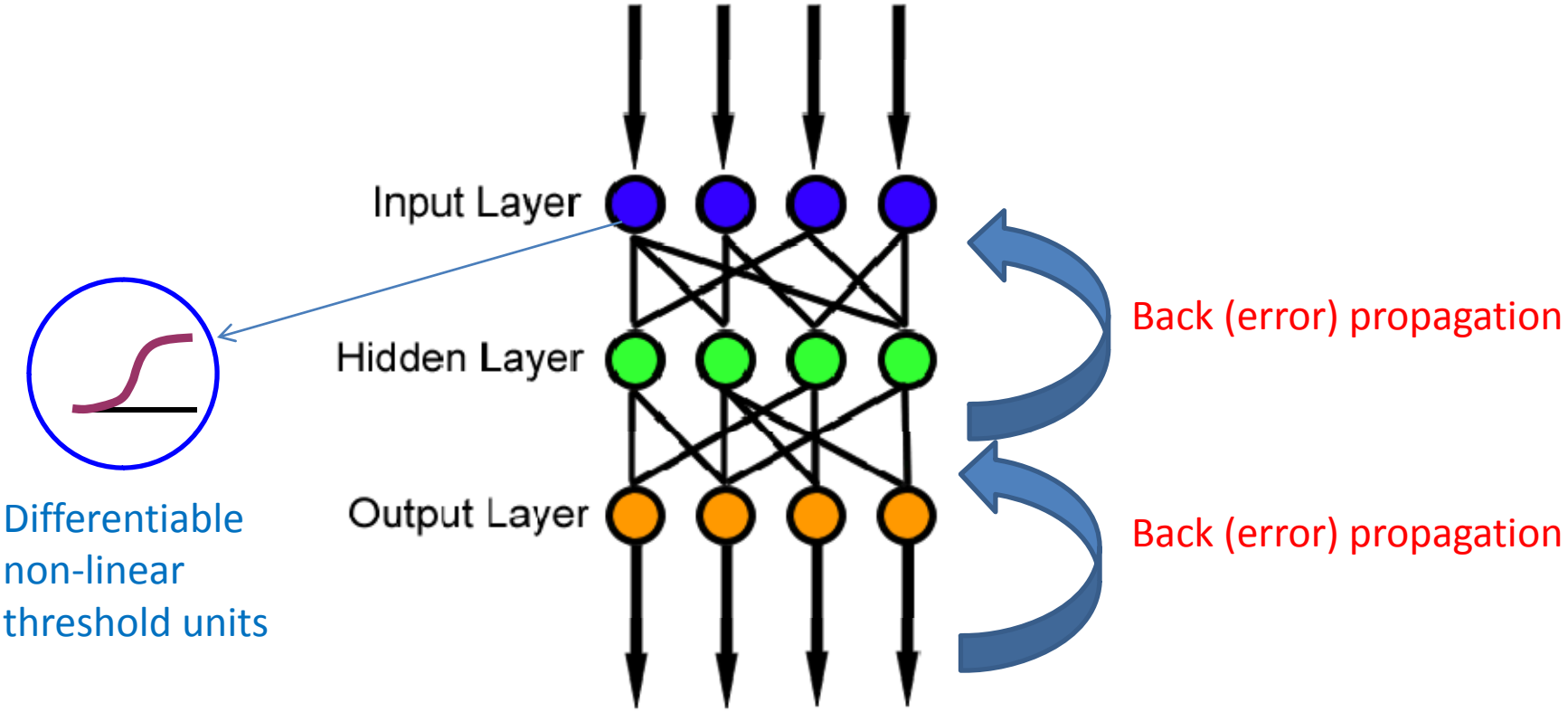
- Gradient
- Gradient descent
- Delta rule
- Mean squared error
- Differentiation
- Derivative
- Partial derivative
- Chain rule
- General power rule

Non-linearly Separable Training Data Set

- If the training examples are not linearly separable, the **delta rule** converges toward a best-fit approximation to the target concept.
- The key idea behind the delta rule is to use **gradient descent** to search the hypothesis space of possible weight vectors to find the weights that best fit the training data.

Neural Network Design (1)

- **Architecture:** the pattern of nodes and connections between them. Normally the network consists of a layered topology with units in any layer receiving input from all units in the previous layer. The most common layered topology is an input layer, 1 or 2 hidden layers, and an output layer. → Multilayer feed-forward
- **Activation function:** the function that produces an output based on the input values received by a node. This is also fixed. It can be the sigmoid function, hyperbolic tangent among other possibilities. → Differentiable non-linear threshold units
- **Learning algorithm:** (training method) the method for determining the weights of the connections. → Backpropagation



In a feed forward network information always moves one direction; it never goes backwards.

Neural Network Design (2)

- Decide the network topology: # of units in the *input layer*, # of *hidden layers* (if more than one), # of units in *each hidden layer*, and # of units in the *output layer*
- Normalizing the input values for each attribute measured in the training tuples to [0.0 – 1.0], if possible
- Initialize the values of weighs to [-1.0 ~ 1.0] and the values of bias
- In general, one output unit is used
- Once a network has been trained and its accuracy is unacceptable, repeat the training process with a *different network topology* or a *different set of initial weights*

Neural Network Design (3)

- The Structure of Multilayer Feed-Forward Network
  - The network is feed-forward in that none of the weighted cycles back to an input unit or to an output unit of a previous layer.
  - It is fully connected in that each unit provides input each unit in the next forward layer
  - Consist of an input layer, one or more hidden layers, and an output layer
  - Each layer is made up of units
  - The inputs to the network correspond to the attributes measured for each training tuple

What Unit Should We Use at Each Node?

- Multiple layers of linear units still produce a linear units. We need non-linearity at the level of the individual node.
- The perceptron is a linear threshold function. It is not differentiable at the threshold. Hence, we can't learn its weights using gradient descent.
- We need a **differentiable threshold** unit.

How Does a Multilayer **Feed Forward** Neural Network Work? (1)

1. **Feed forward training of input patterns**
   - The inputs are fed simultaneously into the units making up the input layer
   - These inputs pass through the input layer and then weighted and fed simultaneously to a second layer of units, known as a hidden layer.
   - The weighted outputs of the last hidden layer are input to units making up the output layer, which emits the network's prediction for given tuples

How Does a Multilayer **Feed Forward** Neural Network Work? (2)

2. **Backpropagation of errors**

   Each output node compares its activation with the desired output. The error is propagated backwards to upstream nodes.

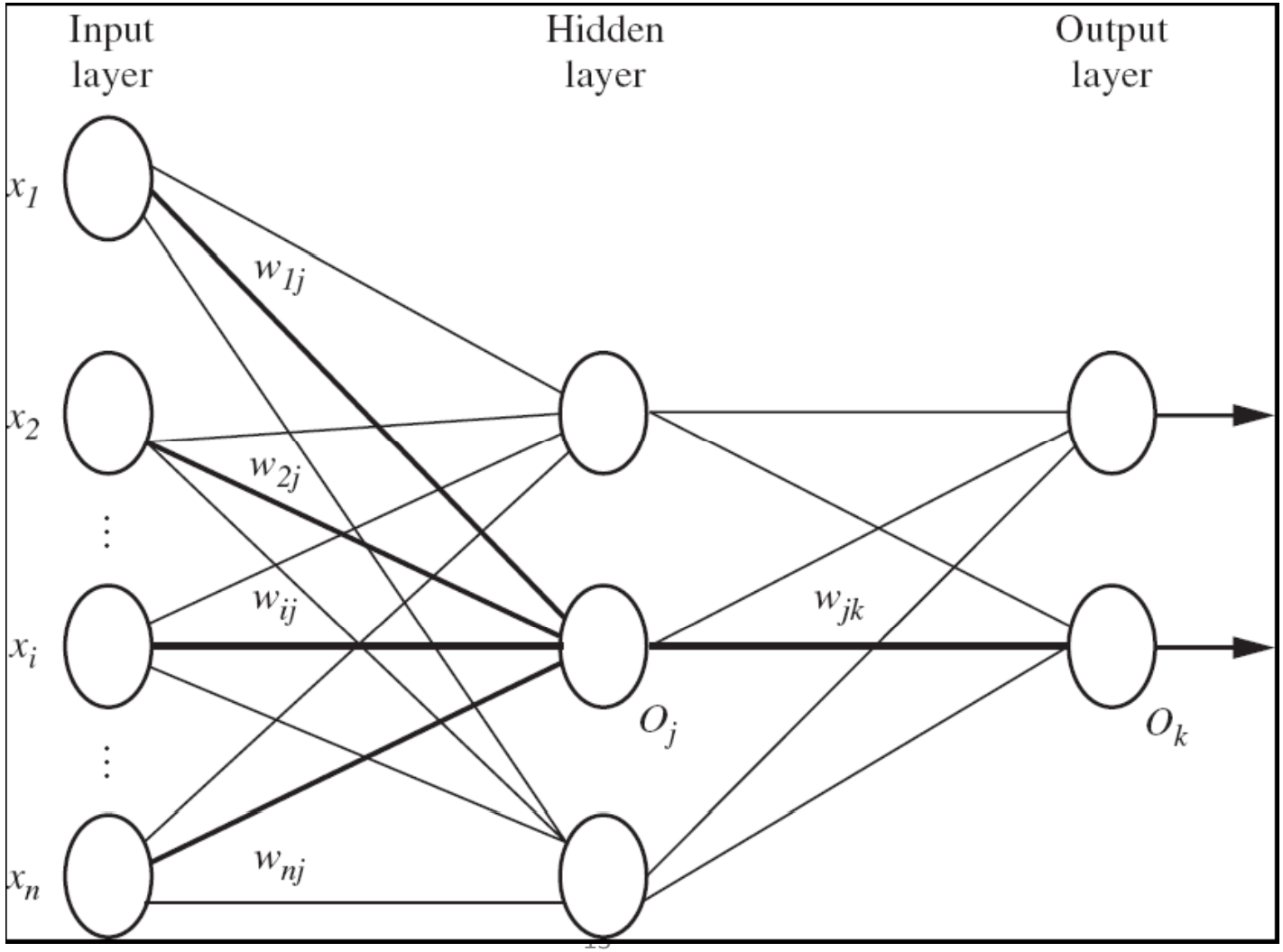
3. **Weight adjustment**

   The weights of all links are computed simultaneously based on the error propagated backwards.

Actual Algorithm for a 3-layer Network (Only One Hidden Layer)

```
Initialize the weights in the network (often randomly)
  Do
      For each example e in the training set
          O = neural-net-output(network, e) ; forward pass
          T = teacher output for e
          Calculate error (T - O) at the output units
          Compute delta_wh for all weights from hidden layer to output layer; backward pass
          Compute delta_wi for all weights from input layer to hidden layer; backward pass continued
          Update the weights in the network
   Until all examples classified correctly or stopping criterion satisfied
  Return the network
```

A Multilayer Feed-Forward Network

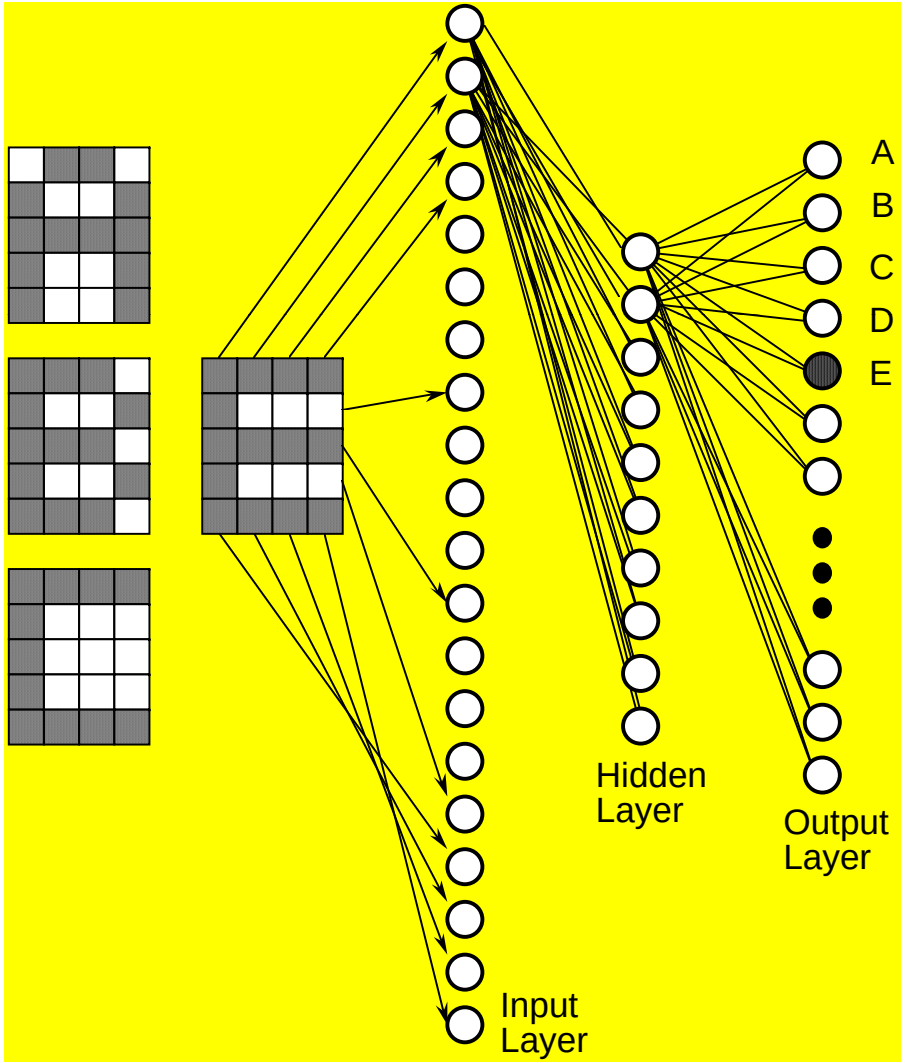


ANN Applications

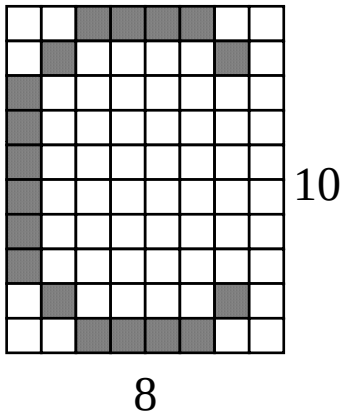
- OCR
- Engine Management
- Navigation
- Signature Recognition
- Sonar Recognition
- Stock Market Prediction
- Mortgage Assessment

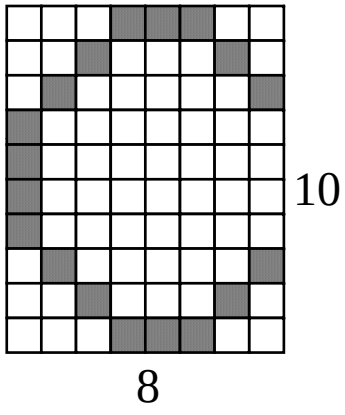
OCR

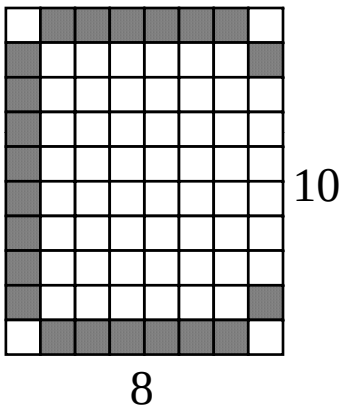
- Feed forward network
- Trained using Back-propagation



OCR for 8x10 characters



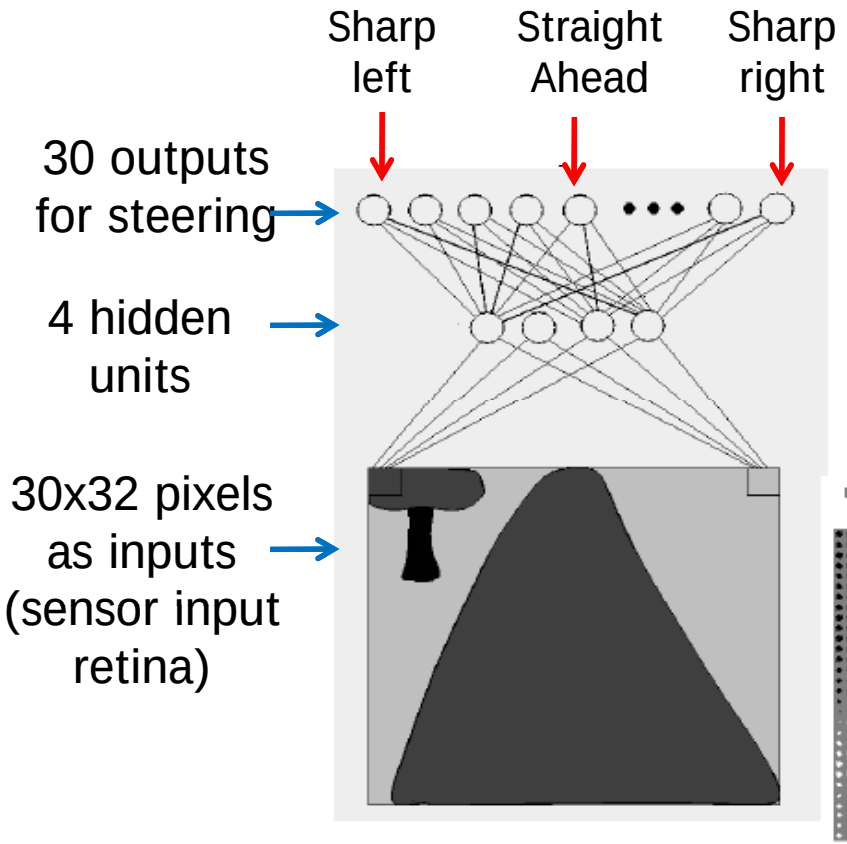


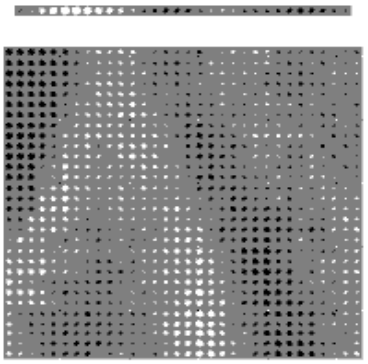


Engine Management

- The behavior of a car engine is influenced by a large number of parameters
  - temperature at various points
  - fuel/air mixture
  - lubricant viscosity.
- Major companies have used neural networks to dynamically tune an engine depending on current settings.

ALVINN





Neural network learning to steer an **autonomous vehicle**. The **ALVINN** system uses Backpropagation to learn to steer an autonomous vehicle (photo at top right) driving at speed up to 70 miles per hour. The diagram on the left shows how the image of a forward-mounted camera is mapped to 960 neural network inputs, which are fed forward to 4 hidden units, connected to 30 output units. Network output encoded the commanded steering direction. The figure on the right shows weight values for one of the hidden units in this network. The 30 x 32 weights into the hidden unit are displayed in the large matrix, with white blocks indicating positive and black indicating negative weights. The weights from this hidden unit to the 30 output units are depicted by the smaller rectangular block directly above the large block. As can be seen from these output weights, activation of this particular hidden unit encourages a turn toward the left.

Signature Recognition

- Each person's signature is different.
- There are structural similarities which are difficult to quantify.
- One company has manufactured a machine which recognizes signatures to within a high level of accuracy.
  - Considers speed in addition to gross shape.
  - Makes forgery even more difficult.

Sonar Target Recognition

- Distinguish mines from rocks on sea-bed
- The neural network is provided with a large number of parameters which are extracted from the sonar signal.
- The training set consists of sets of signals from rocks and mines.

Stock Market Prediction

- “Technical trading” refers to trading based solely on known statistical parameters; e.g. previous price
- Neural networks have been used to attempt to predict changes in prices.
- Difficult to assess success since companies using these techniques are reluctant to disclose information.

Mortgage Assessment

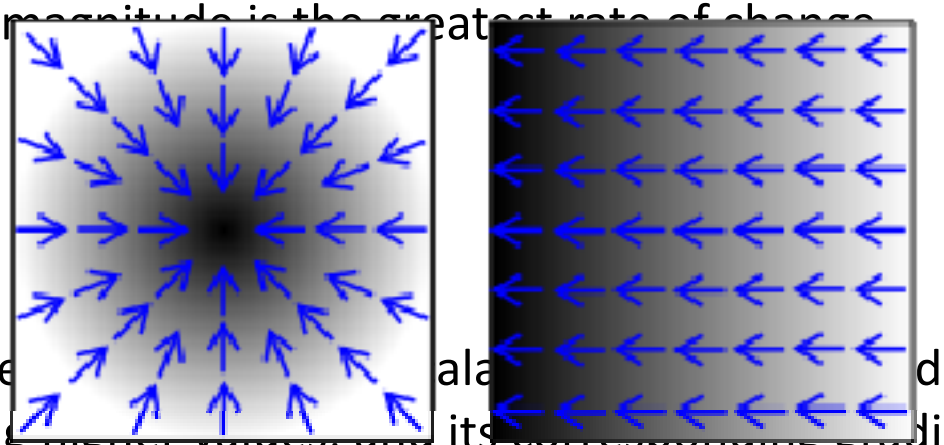
- Assess risk of lending to an individual.
- Difficult to decide on marginal cases.
- Neural networks have been trained to make decisions, based upon the opinions of expert underwriters.
- Neural network produced a 12% reduction in delinquencies compared with human experts.

Learning the Connection Weights

- How can we learn the connection weights in a multilayer feed forward network?
  - **Gradient descent** is a good general search technique over continuously parameterized hypotheses.
  - We have to define the error of the network and this error has to be **differentiable** with respect to the parameters of the hypothesis (weights for ANNs).

Gradient

- In vector calculus, the **gradient** of a scalar field is a vector field which points in the direction of the **greatest rate of increase** of the scalar field, and whose magnitude is the greatest rate of change.



- In the above ala d white, black representing higher values, and its corresponding gradient is represented by blue arrows.

Gradient

Formal definition

The gradient (or gradient vector field) of a scalar function $f(x)$ with respect to a vector variable $x = (x_1, \ldots, x_n)$ is denoted by ∇f or $\vec{\nabla} f$ where ∇ (the nabla symbol) denotes the vector differential operator, del. The notation $\operatorname{grad}(f)$ is also used for the gradient.

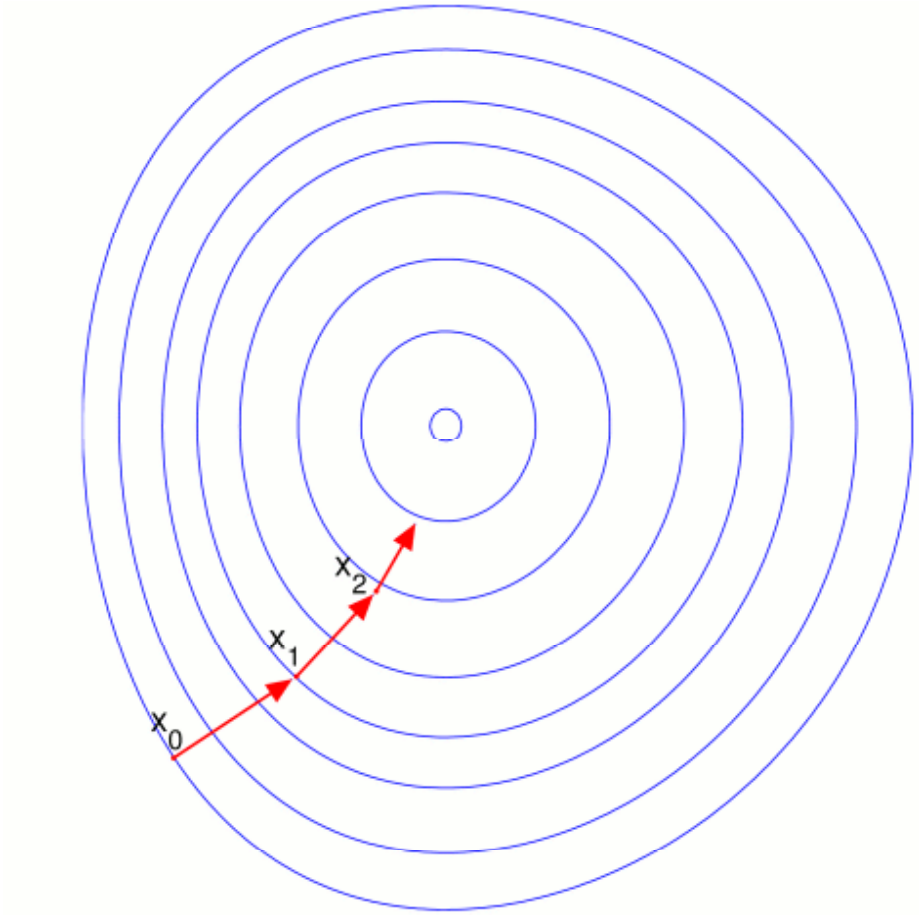
By definition, the gradient is a vector field whose components are the partial derivatives of f. That is:

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n} \right).$$

(Here the gradient is written as a row vector, but it is often taken to be a column vector; note also that when a function has a time component, the gradient often refers simply to the vector of its spatial derivatives only.)

The dot product $(\nabla f)_x \cdot v$ of the gradient at a point x with a vector v gives the directional derivative of f at x in the direction v. It follows that the gradient of f is orthogonal to the level sets of f. This also shows that, although the gradient was defined in terms of coordinates, it is actually invariant under orthogonal transformations, as it should be, in view of the geometric interpretation given above.

Gradient



Gradient Descent

- Gradient descent is also known as **steepest descent**, or the method of steepest descent.
- Gradient descent is an **optimization** algorithm. To find a **local minimum** of a function using gradient descent, one takes steps proportional to the negative of the gradient (or the approximate gradient) of the function at the current point. If instead one takes steps proportional to the gradient, one approaches a local maximum of that function; the procedure is then known as **gradient ascent**.

Delta Rule

- The **delta rule** can be states as: The adjustment made to weight factor of an input neuron connection is proportional to the product of the error signal and the input value of the connection in question
- The key idea behind the delta rule is to use **gradient descent** to search the hypothesis space of possible weight vectors to find the weights that best fit the training data.
- Delta rule is important because it provides the basis for the **backpropagation** algorithm, which can learn networks with many interconnected units.

Error

- An error exists at the output of a neuron *j* at iteration n (i.e., presentation of the n^{th} training sample)
  - $e_j(n) = t_j(n) - y_j(n)$
- Define the instantaneous value of the error for neuron j is
  - $(1/2)e_j^2(n)$
- The total error for the entire network is obtained by summing instantaneous values over all neurons
  - $E(n) = (1/2)\sum e_j^2(n)$

$$E(\vec{w}) \equiv \frac{1}{2}\sum_{d \in D}(t_d - y_d)^2 \qquad \text{.....(1)}$$

$$\nabla E(\vec{w}) \equiv [\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_n}] \qquad \text{.....(2)}$$

$$\vec{w} \leftarrow \vec{w} + \Delta \vec{w}$$

$$\Delta \vec{w} = -\eta \nabla E(\vec{w})$$

gradient descent

$$w_i \leftarrow w_i + \Delta w_i$$

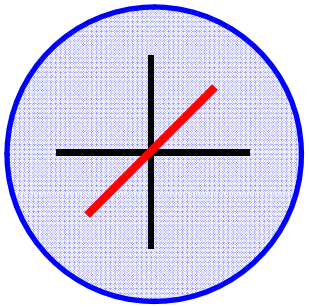
$$where\ \Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{d\in D}(t_d - y_d)^2$$

$$= \frac{1}{2}\sum_{d\in D}\frac{\partial}{\partial w_i}(t_d - y_d)^2$$

$$= \frac{1}{2}\sum_{d\in D}2(t_d - y_d)\frac{\partial}{\partial w_i}(t_d - y_d)$$

$$= \sum_{d\in D}(t_d - y_d)\frac{\partial}{\partial w_i}(t_d - \vec{w}\cdot\vec{x}_d)$$



Linear function

$$\frac{\partial E}{\partial w_i} = \sum_{d\in D}(t_d - y_d)(-x_i)$$

$$\Delta w_i = -\eta\frac{\partial E}{\partial w_i} = \eta\sum_{d\in D}(t_d - y_d)x_i$$

$$w_i \leftarrow w_i + \eta\sum_{d\in D}(t_d - y_d)(x_i)$$

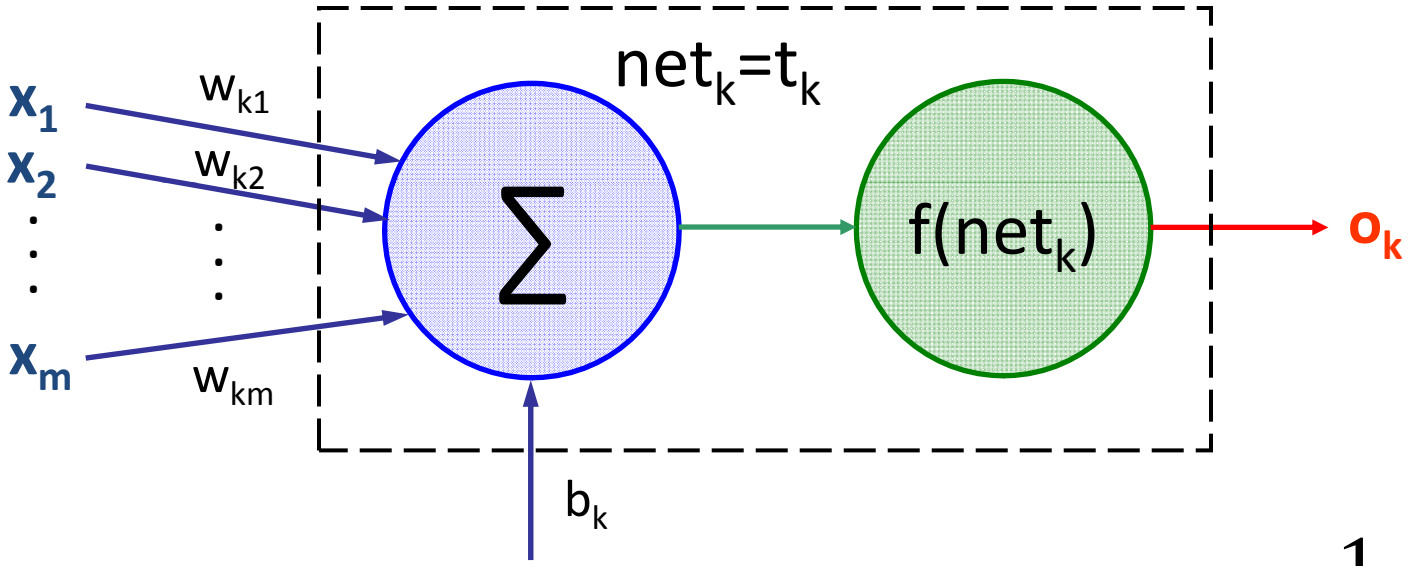
Delta rule learning (training)

$$\Delta w_{ji} = -\eta\frac{\partial E}{\partial w_i} = \eta(t_d - y_d)x_i$$

$$w_{ji} \leftarrow w_i + \eta(t_d - y_d)(x_i)$$

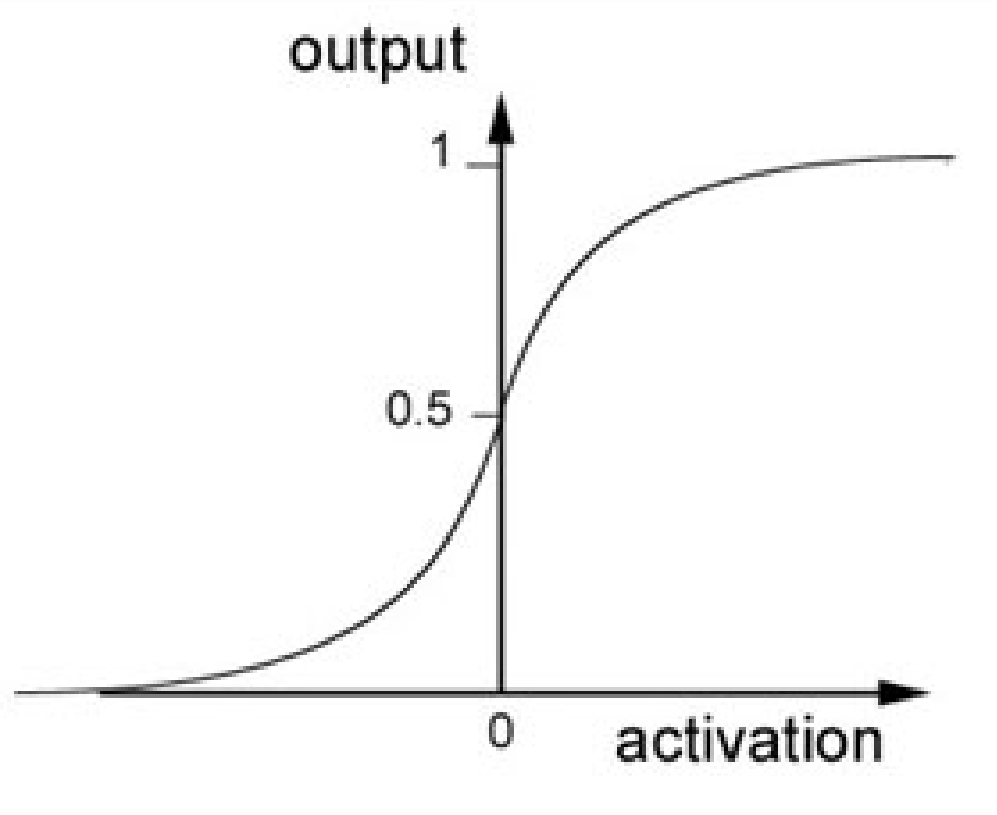
Sigmoid Unit (1)

k_{th} sigmoid unit



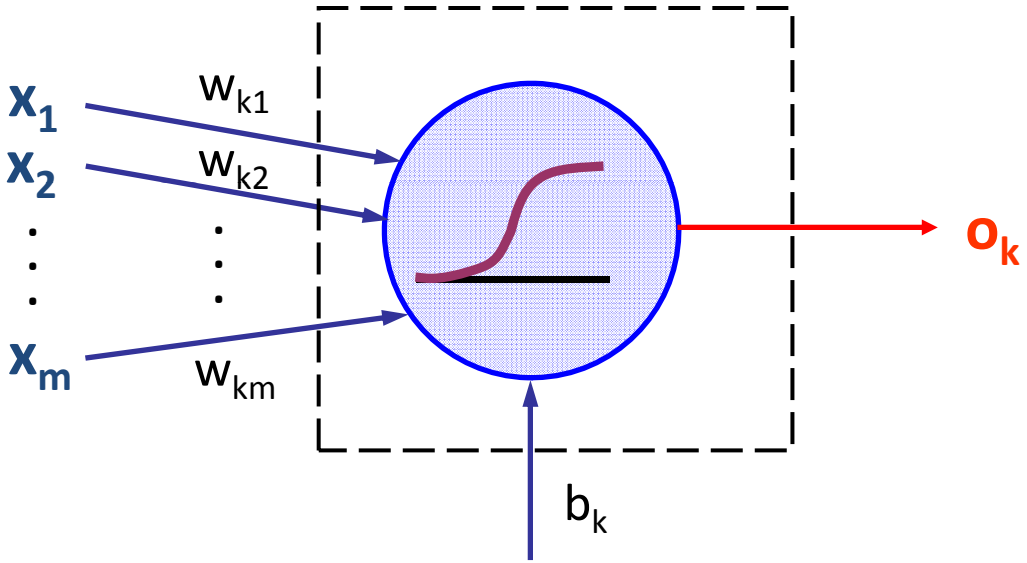
$$o_k = f(t_k) = \frac{1}{1+e^{-y_k}}$$

Sigmoid Function



Sigmoid Unit (2)

k_{th} sigmoid unit



$$o_k = f(y_k) = \frac{1}{1+e^{-y_k}}$$

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{d\in D}(t_d - o_d)^2$$

$$= \frac{1}{2}\sum_{d\in D}\frac{\partial}{\partial w_i}(t_d - o_d)^2$$

$$= \frac{1}{2}\sum_{d\in D}2(t_d - o_d)\frac{\partial(t_d - o_d)}{\partial w_i}$$

$$= -\sum_{d\in D}(t_d - o_d)\frac{\partial o_d}{\partial w_i}$$

$$= -\sum_{d\in D}(t_d - o_d)\frac{\partial o_d}{\partial y_d}\frac{\partial y_d}{\partial w_i}$$ ← chain rule

$$= -\sum_{d\in D}(t_d - o_d)\frac{\partial}{\partial y_d}\left(\frac{1}{1+e^{-y_d}}\right)\frac{\partial(\sum_{i=0}^{n} x_i w_i)}{\partial w_i}$$ ← Sigmoid function

$$= -\sum_{d\in D}(t_d - o_d)o_d(1-o_d)x_i$$

Continue….

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} = \eta \sum_{d \in D} (t_d - o_d) o_d (1 - o_d) x_i$$

$$w_i \leftarrow w_i + \eta \sum_{d \in D} (t_d - o_d) o_d (1 - o_d) x_i$$

Delta rule learning (training)

$$\Delta w_{ji} = \eta (t_d - o_d) o_d (1 - o_d) x_i$$

$$w_{ji} \leftarrow w_i + \eta (t_d - o_d) o_d (1 - o_d) x_i$$

Derivative

$f(x) = x^r \qquad f'(x) = rx^{r-1} \qquad \frac{d}{dx}e^x = e^x. \qquad \frac{d}{dx}e^{f(x)} = f'(x)e^{f(x)}.$

Rules for finding the derivative

Main article: Differentiation rules

In many cases, complicated limit calculations by direct application of Newton's difference quotient can be avoided using differentiation rules. Some of the most basic rules are the following.

Constant rule: if *f*(*x*) is constant, then

$f' = 0$

Sum rule:

$(af + bg)' = af' + bg'$ for all functions *f* and *g* and all real numbers *a* and *b*.

Product rule:

$(fg)' = f'g + fg'$ for all functions *f* and *g*.

Quotient rule:

$$\left(\frac{f}{g}\right)' = \frac{f'g - fg'}{g^2}$$

Chain rule: If $f(x) = h(g(x))$, then

$f'(x) = h'(g(x))g'(x)$.

How Long Should You Train The Network?

- Typically, many iterations are needed (often thousands).
- The goal is to achieve a balance between correct responses for the training patterns and correct responses for new patterns. (That is, a balance between memorization and generalization.)
- If you train the network for too long, then you run the risk of overfitting.
- Possible stopping conditions:
  - Fixed number of iterations
  - Threshold on training set error (e.g., 5%)
  - Increased error on a validation set

Comments on Training (1)

- No convergence guarantee, may oscillate or reach a local minima
- However, in practice, many large networks have been adequately trained on large amounts of data for realistic problems.
- Adding momentum to the update helps avoid local minima.

Comments on Training (2)

- To avoid local minima, run several trials from different random weights and:
  - Take the result with the best training or validation performance, OR
  - Build a committee of networks that vote during testing, possibly weighting vote by training or validation accuracy
- Backpropagation easily generalizes to acyclic networks with any number of hidden node layers and even to any directed acyclic network (no organized layers).

Neural Network -- Strength

- High tolerance to noisy data
- Ability to classify untrained patterns
- Well-suited for continuous-valued inputs and outputs
- Successful on a wide array of real-world data
- Algorithms are inherently parallel
- Techniques have recently been developed for the extraction of rules from trained neural networks

Neural Network -- Weakness

- Long training time
- Require a number of parameters typically best determined empirically, e.g., the network topology or “structure”.
- Poor interpretability: Difficult to interpret the symbolic meaning behind the learned weights and of “hidden units” in the network

Exercise – Example 6.9

- Figure 6.18 shows a multilayer feed-forward neural network. Let the learning rate be 0.9. The initial weight and bias values of the network are given in Table 6.3, along with the first training tuple, $\boldsymbol{X}$ = (1, 0, 1), whose class label is 1.
- The example shows the calculations for backpropagation, given the first training tuple, $\boldsymbol{X}$. The tuple is fed into the network, and the net input and output of each unit are computed. These values are shown in Table 6.4. The error of each unit is computed and propagated backward. The error values are shown in Table 6.5. The weight and bias updates are shown in Table 6.6

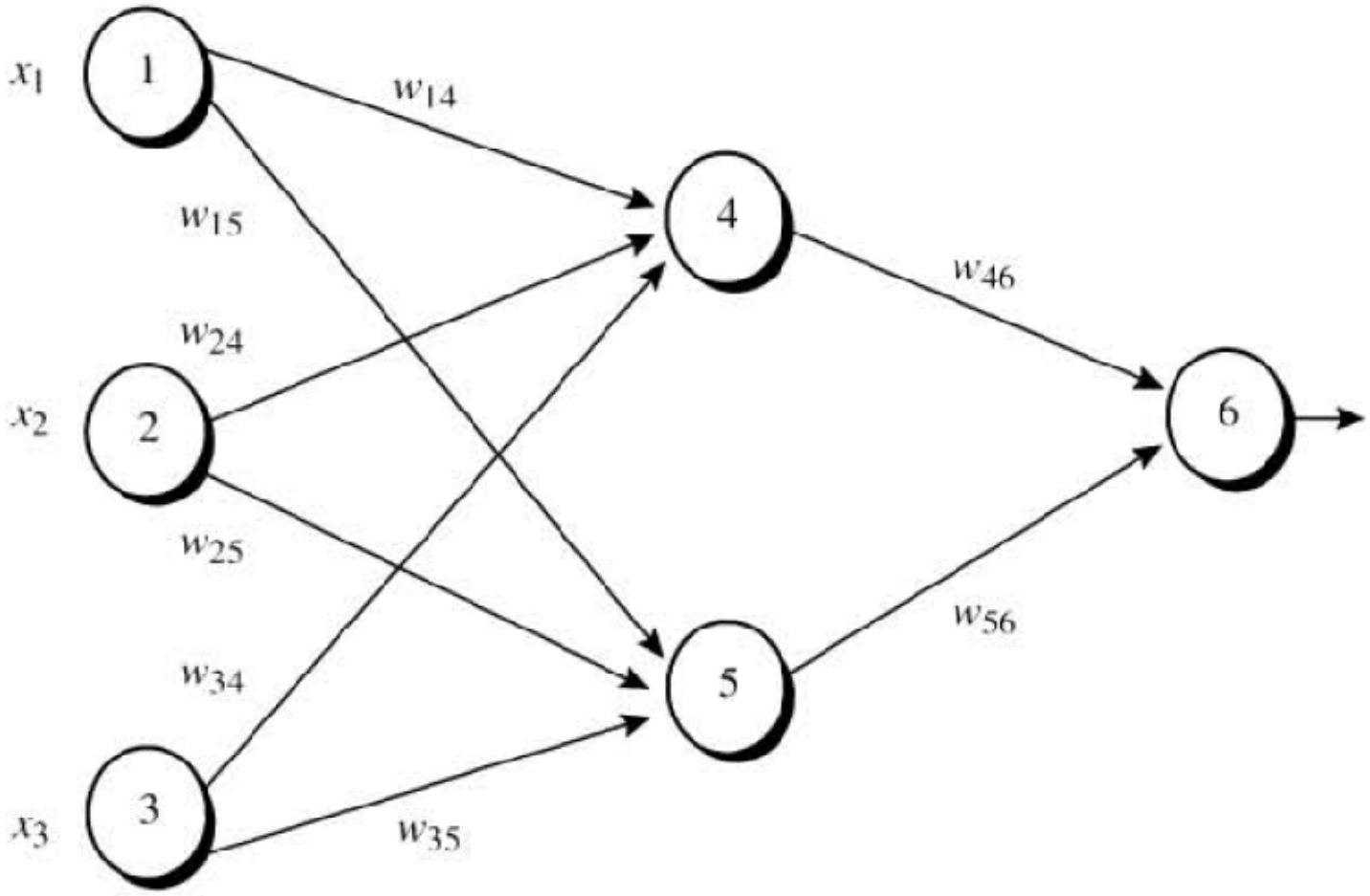


Figure 6.18 An example of a multilayer feed-forward neural network.

x_1	x_2	x_3	w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	w_{46}	w_{56}	θ_4	θ_5	θ_6
1	0	1	0.2	–0.3	0.4	0.1	–0.5	0.2	–0.3	–0.2	–0.4	0.2	0.1

Table 6.3 Initial input, weight, and bias values.

Unit j	Net input, I_j	Output, O_j
4	$0.2 + 0 - 0.5 - 0.4 = -0.7$	$1/(1 + e^{0.7}) = 0.332$
5	$-0.3 + 0 + 0.2 + 0.2 = 0.1$	$1/(1 + e^{-0.1}) = 0.525$
6	$(-0.3)(0.332) - (0.2)(0.525) + 0.1 = -0.105$	$1/(1 + e^{0.105}) = 0.474$

Table 6.4 The net input and output calculations.

Unit j	Err j
6	$(0.474)(1 - 0.474)(1 - 0.474) = 0.1311$
5	$(0.525)(1 - 0.525)(0.1311)(-0.2) = -0.0065$
4	$(0.332)(1 - 0.332)(0.1311)(-0.3) = -0.0087$

Table 6.5 Calculation of the error at each node.

Weight or bias	New value
w_{46}	$-0.3 + (0.9)(0.1311)(0.332) = -0.261$
w_{56}	$-0.2 + (0.9)(0.1311)(0.525) = -0.138$
w_{14}	$0.2 + (0.9)(-0.0087)(1) = 0.192$
w_{15}	$-0.3 + (0.9)(-0.0065)(1) = -0.306$
w_{24}	$0.4 + (0.9)(-0.0087)(0) = 0.4$
w_{25}	$0.1 + (0.9)(-0.0065)(0) = 0.1$
w_{34}	$-0.5 + (0.9)(-0.0087)(1) = -0.508$
w_{35}	$0.2 + (0.9)(-0.0065)(1) = 0.194$
θ_6	$0.1 + (0.9)(0.1311) = 0.218$
θ_5	$0.2 + (0.9)(-0.0065) = 0.194$
θ_4	$-0.4 + (0.9)(-0.0087) = -0.408$

Table 6.6 Calculation for weight and bias updating.