Neural Networks

Feng-Cheng Yang 楊烽正

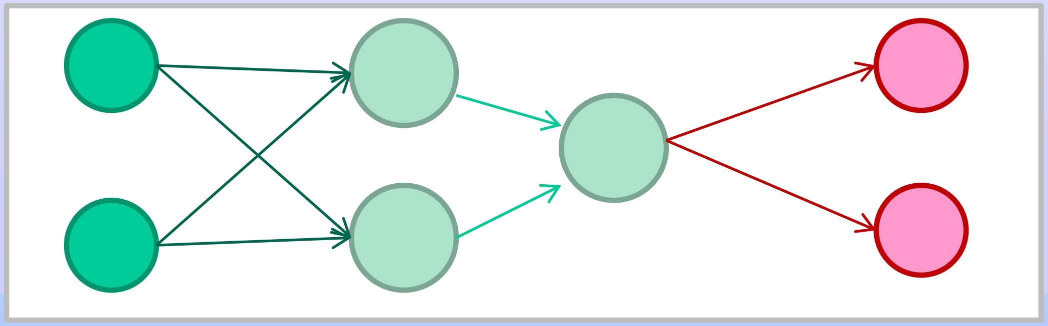
computer automation LAB

Enthusiasm • Leadership • Innovation • Teamwork • Excellence

NTU 國立臺灣大學工業工程學研究所

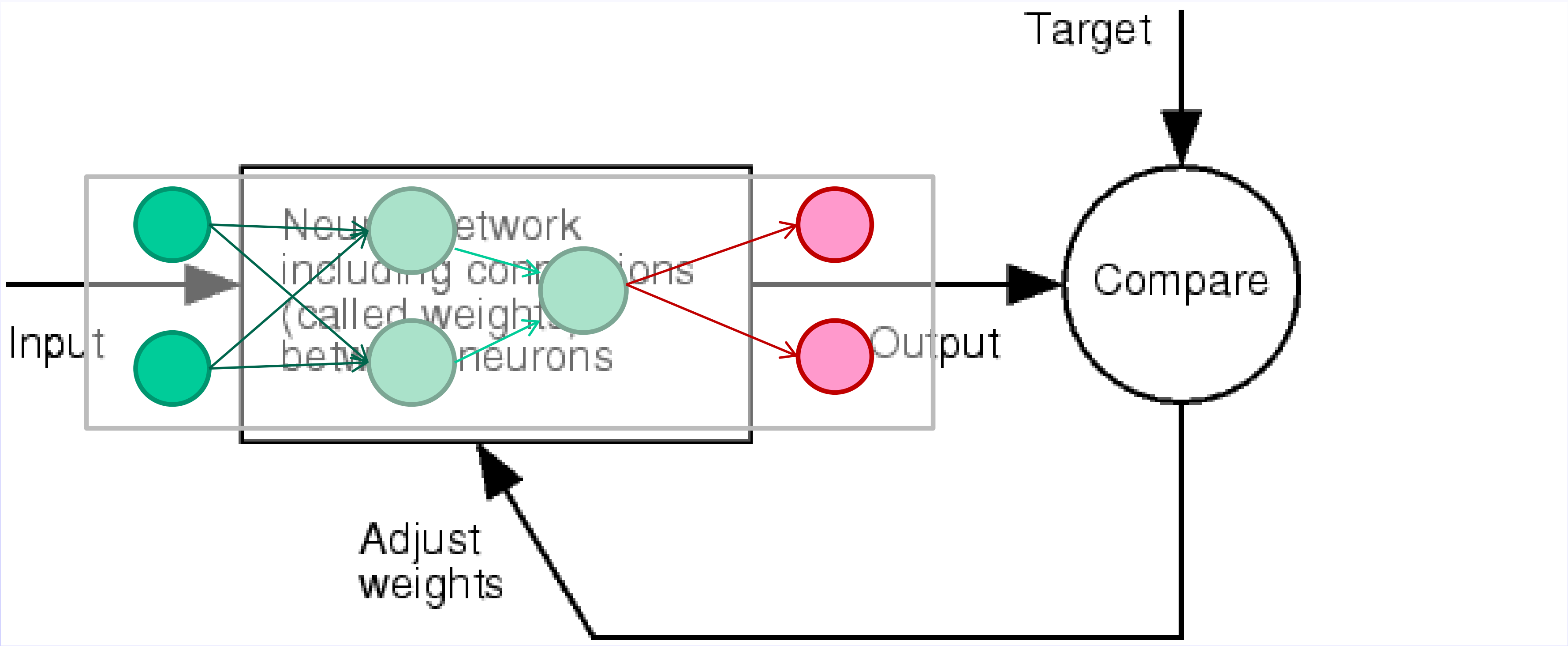
What Is Neural Networks?

- Consists of simple operation elements operating in parallel
  - Elements are inspired by biological nervous systems
  - Network function is determined by the connections between elements
- A neural network is trained to perform a particular function
  - Adjust the values of the connections (weights) between elements
    - Based on a comparison of the output and the target, until the network output matches the target
  - Many such input/target pairs are needed to train a network
    - Training data set

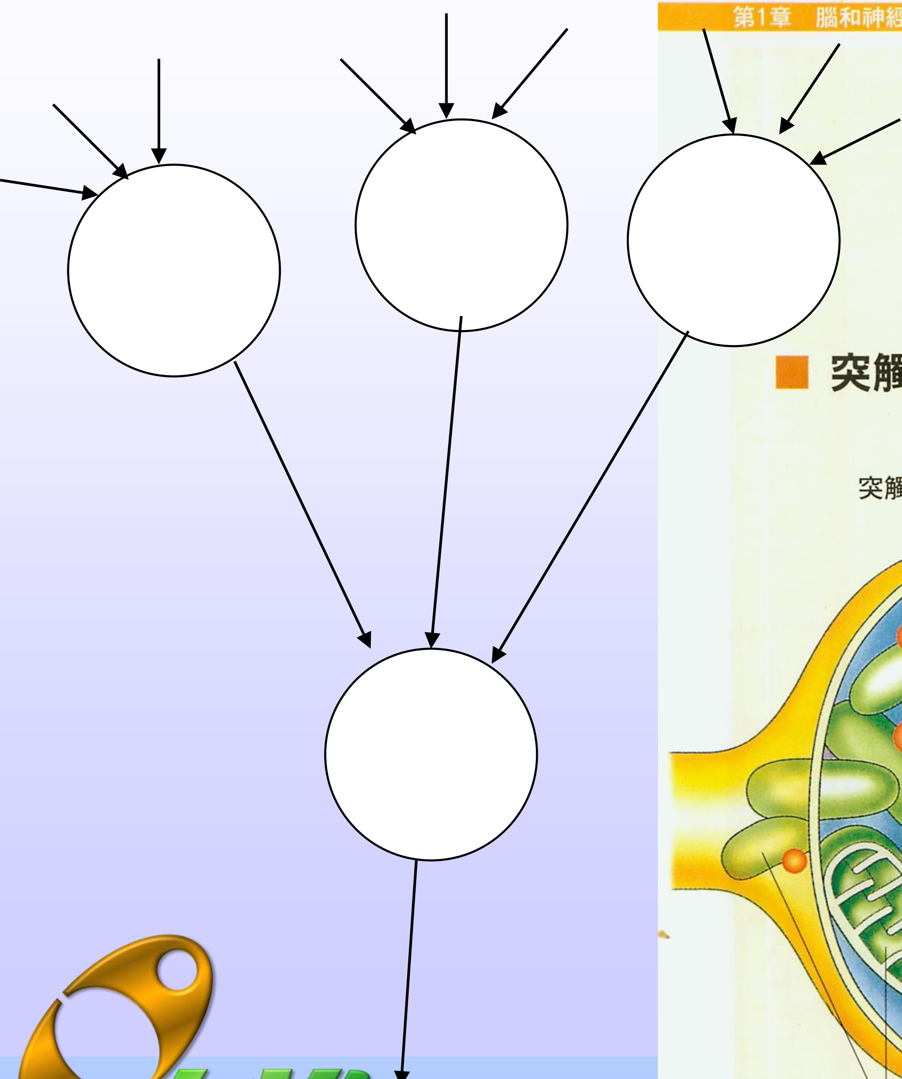


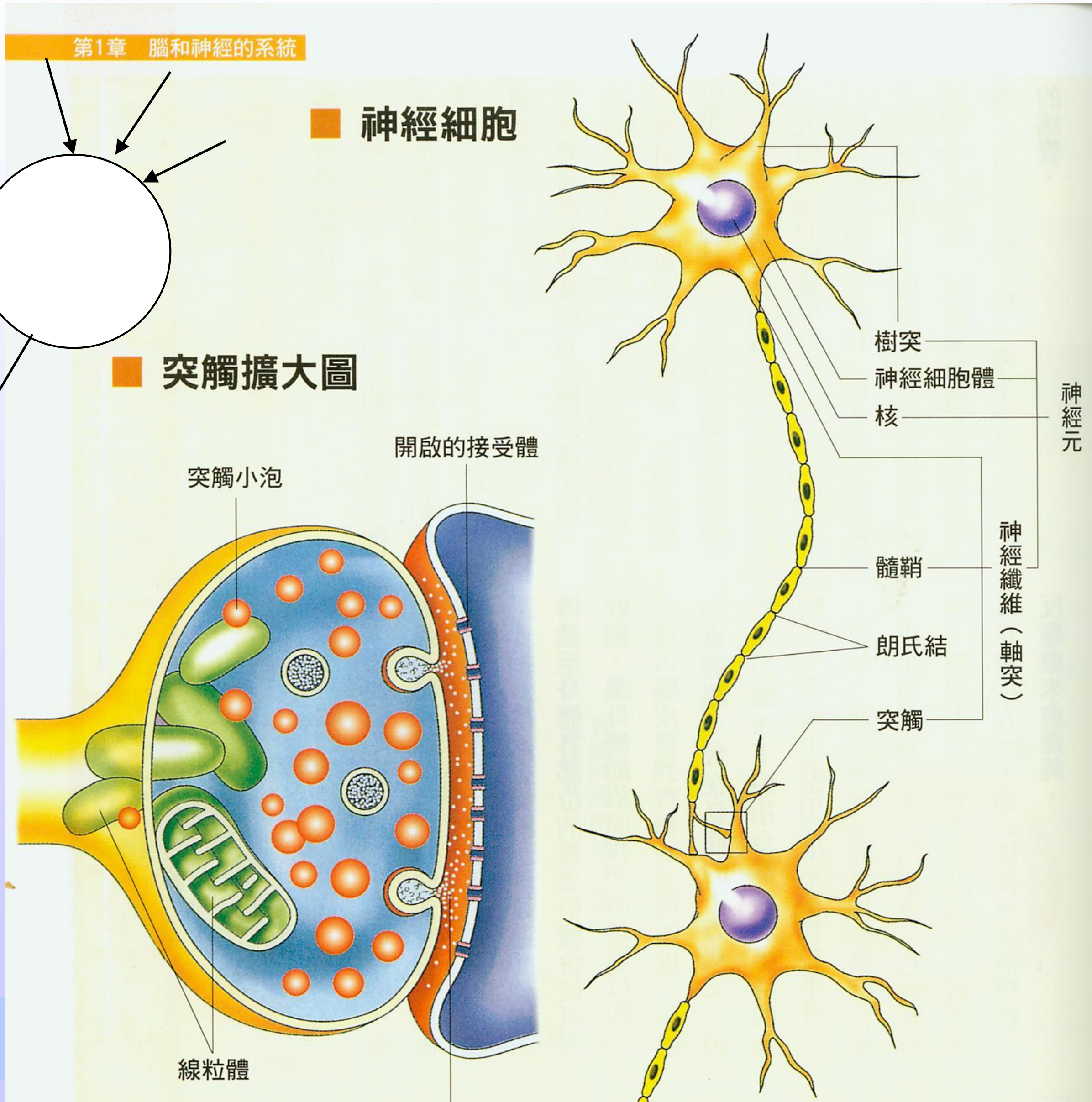
Neural Networks

- **Topics**
  - **Introduction**
  - **Perception**
  - **Adaline**
  - **Backpropagation MLPs**
  - **Discussion**


Target
Compare
Input
Output
Adjust
weights

Biological Neuron and Artificial Neuron





Supervised NN

- ***Supervised* training methods**
  - **Batch training**
    - Change weight and bias based on an entire set (batch) of input vectors
  - **Incremental training**
    - Change the weights and biases after each individual input vector
    - On line training
    - Adaptive training
  - **Application fields**
    - Pattern recognition, identification, classification, speech, vision, and control system
  - **Solve problems that are difficult for conventional computers or human beings**
    - Engineering, financial, and other practical applications.

Unsupervised NN

- ***Unsupervised training*** **techniques**
  - **Identify groups of data**
- ***Direct design*** **methods**
  - **Training is not required**
  - **Certain kinds of linear networks and Hopfield networks**

History

- History of some seven decades
  - Solid application only in the past thirty years
  - The field turns to deep learning now from 2010
- McCulloch an Pitts (1943)
  - A biological brain neuron model
- Hebb (1949)
  - Learning rule
- Rosenblatt (1957,1962)
  - Perceptron; the first neural network model
  - Delta rule (LMS, Least Mean Square rule)
- Widrow (1960)
  - Adaline (Adaptive Linear Element)
- Minsky and Papert (1969)
  - Book “Perceptrons”: kills the NN research
  - 10-15 blackness years (for supervised NNs)

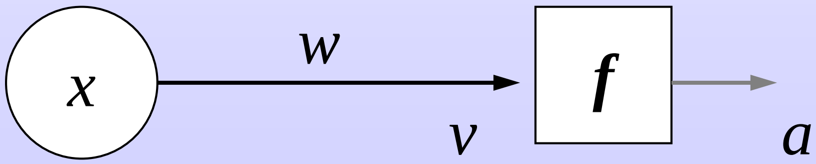
- **Grossberg (1972)**
  - **ART (Adaptive Resonance Theory) NN**
- **Kohonen (1978)**
  - **SOM (Self-Organizing Map) NN**
- **Hopfield (1982)**
  - **HNN, Hopfield NN**
- **Hinton (1984)**
  - **Boltzmann machine**
- **Hopfield and Tank (1985)**
  - **HTN, Hopfield-Tank NN for optimization**

- Rumelhart (1985)
  - BPN, Back Propagation NN
  - Generalized Delta Rule
- Grossberg (1988)
  - ART2
- ICNN, International Conference on NN (1988)
- Specht (1988)
  - PNN, Probabilistic NN
- Kohonen (1988)
  - LVQ, Learning Vector Quantization NN
- Den Bout and Miller (1988)
  - ANN, Annealed NN

- Cortes and Vapnik (1995)
  - Support-Vector network
- LeCun and Bengio (1995)
  - Convolutional networks
    - A special MLP
- 2010-
  - Deep neural networks
  - Deep Learning
  - AI

Neuron Model

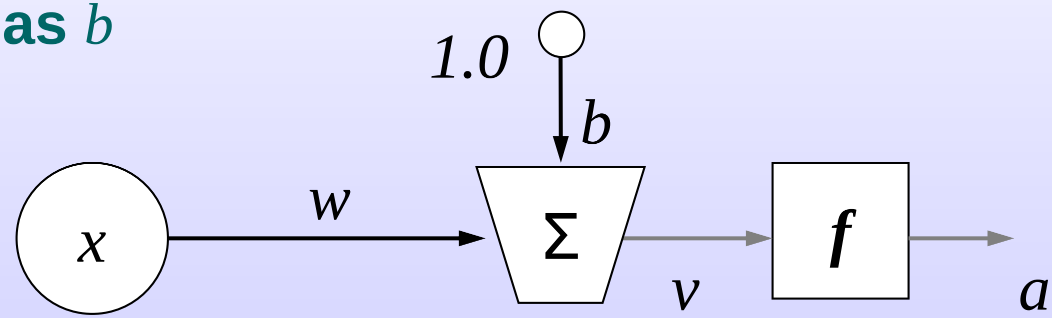
- **Simple Neuron without bias**
  - **A neuron with a single scalar input and no bias**
  - **Input** x **is transmitted through a connection that multiplies its strength by the scalar weight** w **to form the product** wx



$$a = f(wx) = f(v)$$

Simple Neurons

- **Simple Neuron with bias**
  - **Bias can be added to the product** wx
  - **Bias is much like a weight, except that it has a constant input of 1**
  - **Net** v **is the sum of the weighted input** wx **and the bias** b



$$a = f(wx + b) = f(v)$$

Transfer Function and Parameters

- **Transfer function (Activation function)**
  - **A step function or a sigmoid function**
  - **The only input is the net value**
  - **Takes the argument v and produces the output a**
- **Adjustable scalar parameters of the neuron**
  - **Weights w and bias b**
  - **Adjusted so that the network exhibits some desired or interesting behavior**
  - **One can train the network to do a particular job**
    - Adjusting the weight or bias parameters

Supervised Learning Neural Networks

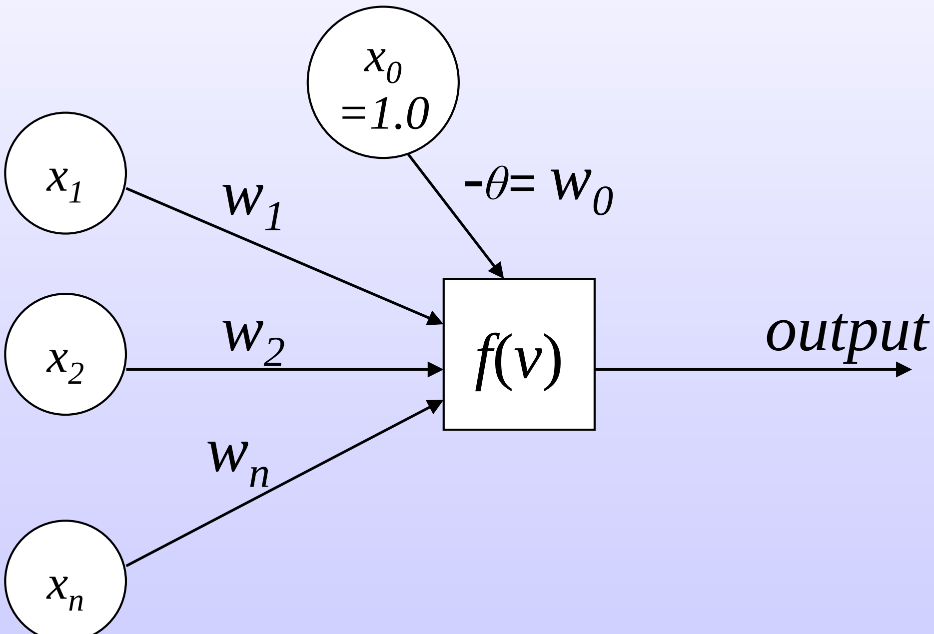
- Problems with known desired input-output data sets
- Adjustable parameters
- Has a supervised learning rule
- Supervised Learning or Mapping networks

Perceptrons

- **Designed by Rosenblatt (1962)**
  - **The earliest neural model**
  - **For pattern recognition (data classification)**
  - **Single neuron layer and an input layer**
    - Input layer is “feature detectors”
    - Neurons classify the given input patterns
  - **Derived from a biological brain neuron model**
    - McCulloch and Pitts (1943)

Learning Method of Perceptron

- **Adjust the relevant connection strengths (weights) w_i and a threshold value θ**



$$v = \sum_{i=1}^{n} w_i x_i - \theta = \sum_{i=1}^{n} w_i x_i + w_0 x_0 = \sum_{i=0}^{n} w_i x_i, \quad x_0 = 1$$

computer automation

Perceptron

- **Value x_i is usually either binary $(0,1)$ or bipolar $(-1,1)$**
  - $x_i = 1$**, active or excitatory**
  - $x_i = 0$ **or** -1**, inactive or inhibitory**
- **Value x_i can be real number in perceptrons**
  - **Classification**
  - **A hyper plane**
- **Output is a linear threshold element**

Perceptron

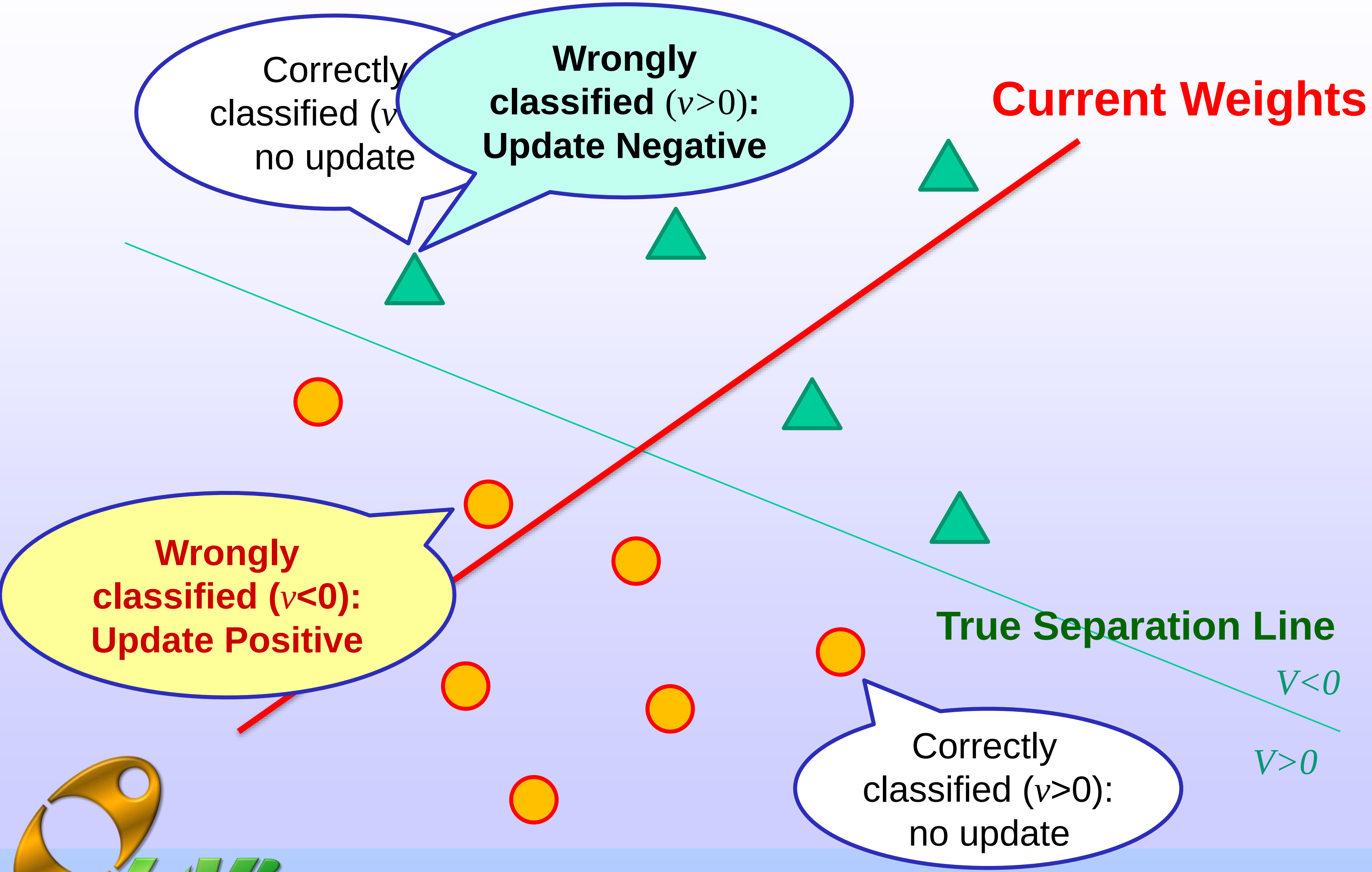
- $f(.)$ is the activation (transfer) function
  - A signum function (for bipolar output)
  - A hard-limit transfer function

$$f(v) = \operatorname{sgn}(v) = \begin{cases} 1, \text{if } v > 0 \\ -1, \text{otherwise} \end{cases}$$

  - A step function (for binary output)
  - A hard-limit transfer function

$$f(v) = \operatorname{step}(v) = \begin{cases} 1, \text{if } v > 0 \\ 0, \text{otherwise} \end{cases}$$

Perceptron


Correctly
classified (v
no update
Wrongly classified (v>0): Update Negative
Current Weights
Wrongly classified (v<0): Update Positive
True Separation Line
V<0
V>0
Correctly classified (v>0): no update

Basic Learning algorithm

- 1. Randomly assign weights and biases
- 2. Select an input vector x with known output
- 3. Compute the computed output from the NN
  - If the output is incorrect, modify all connection weights (including the bias) w_i
  - Applying a learning rate to reduce the update amounts
- 4. Continue Steps 2 and 3 until no weight modifications sustained for all the input vectors

Points (x_1, x_2) are separated by line

$$L\left(x_1, x_2\right) = w_1^* x_1 + w_2^* x_2 + w_0^* = 0,$$

and with output value $y = \begin{cases} 1, \text{if } L\left(x_1, x_2\right) > 0 \\ 0, L\left(x_1, x_2\right) \le 0 \end{cases}$

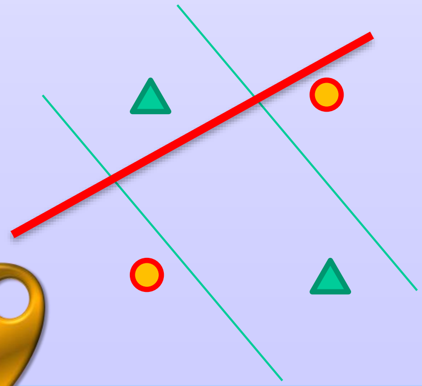
$$v = \sum_{i=0}^{n} w_i x_i,\ f\left(v\right) = \text{step}\left(v\right) = \begin{cases} 1, \text{if } v > 0 \\ 0, v \le 0 \end{cases},$$

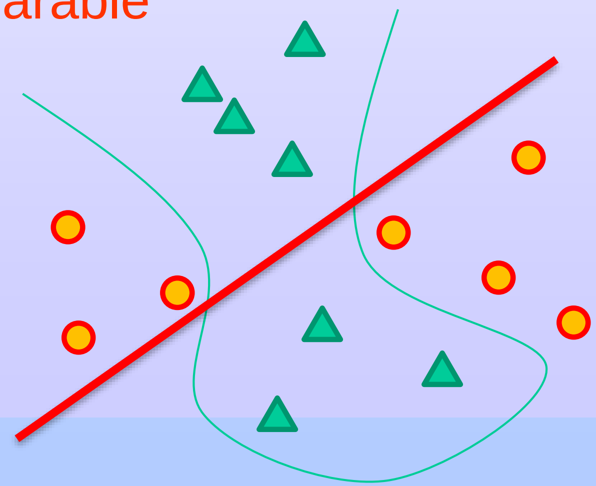
$$w_i = w_i + \Delta w_i = w_i + \eta\left(y - f\left(v\right)\right) x_i,$$

$$y - f\left(v\right) = \begin{cases} 1-1=0 \text{ or } 0-0=0, \text{if the training point } \mathbf{x} \\ \text{is correctly classified;} \\ 1-0=1, \text{ wrongly classified with } v<0 \\ 0-1=-1, \text{ wrongly classified } v>0 \end{cases}$$

Basic Learning algorithm

- Roughly based on gradient descent
- Perceptron convergence theorem
  - Proved by Rosenblatt (1962)
- Limitation
  - Classification must be linear separable
  - Can not deal with the Exclusive-OR Problem
    - Is not linearly separable



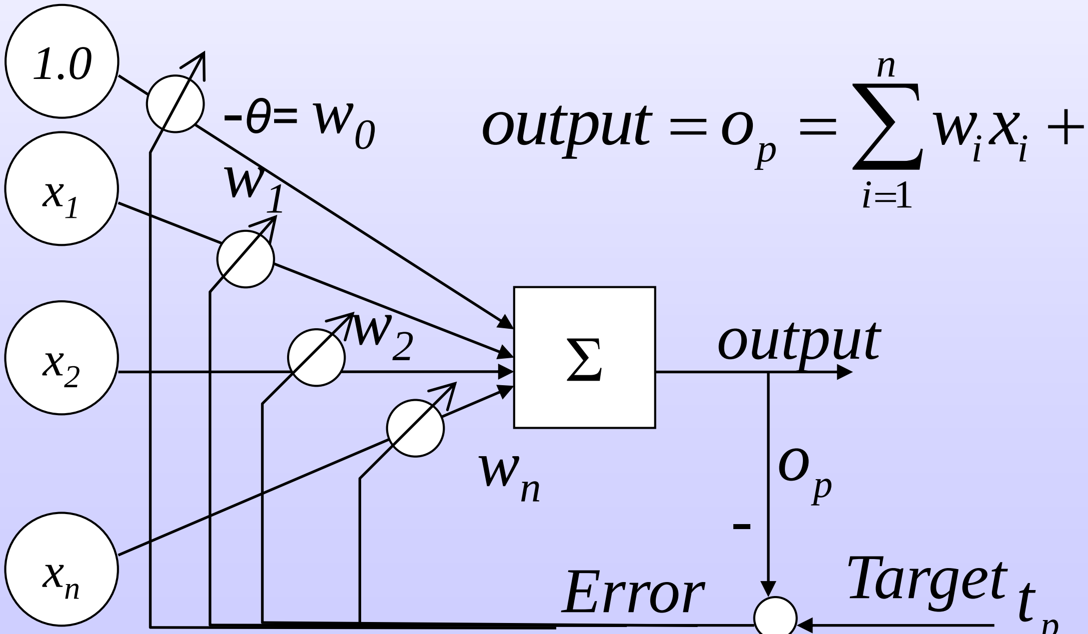


Limitations of Perceptrons

- **Single layer perceptron**
  - **A principal NN component**
  - **Non-differentiability of the hard-limiter activation function**
    - Signum and step functions
  - **The learning strategy is not obvious**
- **Lack of suitable training methods**
  - **From 1960s to1980s, until …**
- **Backpropagation training method for MLPs**
  - **Rumelhart et al. (1986)**

ADALINE

- **Adaptive Linear Element (Adaline)**
  - **By Widrow and Hoff (1960)**
  - **A classical example of the simplest intelligent self-learning system**



$$output = o_p = \sum_{i=1}^{n} w_i x_i + w_0$$

$$E_p = \left(t_p - o_p\right)^2$$

- **A linear model with $n+1$ linear parameters**
- **Learning method**
  - **The best method: the least-squares methods**
    - All points are subject to computing errors
    - Error minimization algorithm is required
    - Tradeoff: extensive computation
  - **Widrow and Hoff introduced "Delta Rule" for adjusting the weights**

$$E_p = \left(t_p - o_p\right)^2$$

$$\frac{\partial E_p}{\partial w_i} = -2\left(t_p - o_p\right)\frac{d\left(\sum_{i=1}^{n} w_i x_i + w_0\right)}{dw_i} = -2\left(t_p - o_p\right)x_i$$

- **Decrease E_p to descent the error,**

$$\Delta_p w_i = -k \frac{\partial E_p}{\partial w_i} = \eta \left(t_p - o_p \right) x_i$$

- **Delta Rule**
  - **Minimize squared errors for a single input**
  - **Widrow-Hoff Learning Rule**
    - Simplicity
    - Distributed learning
    - Support on-line learning (pattern-by-pattern)

Adaline and Madaline

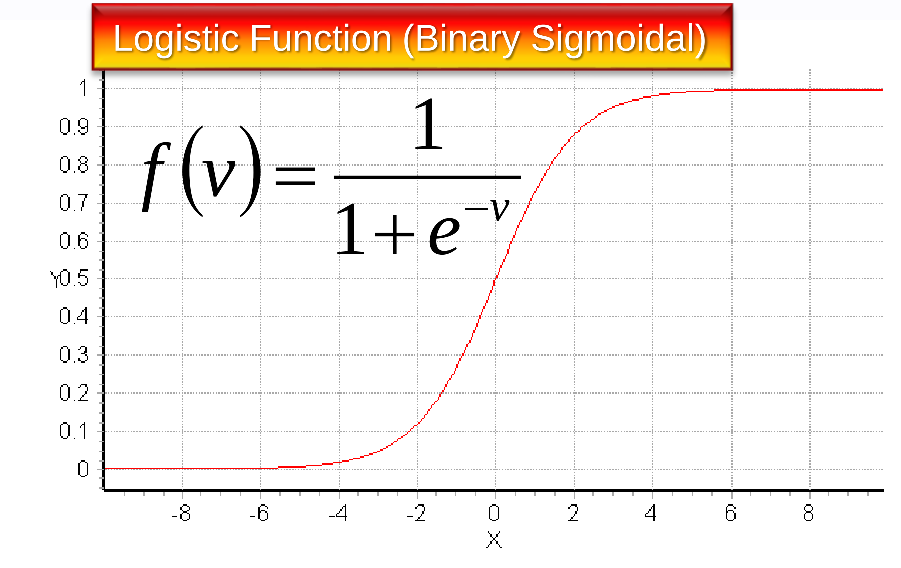
- Adaline+delta rule
  - 1960s
  - Suitable for simple hardware implementation
- Two or more Adaline components are integrated
  - Used for adaptive noise cancellation
  - Adaptive inverse control

A Backpropagation MLP

- MLP (Multiple Layer Perceptron)
- Node (Neuron) function
  - A composite of the weighted sum and a differentiable nonlinear activation function
- Three of the most commonly used activation functions
  - Logistic function, Hyperbolic tangent function, Identity function

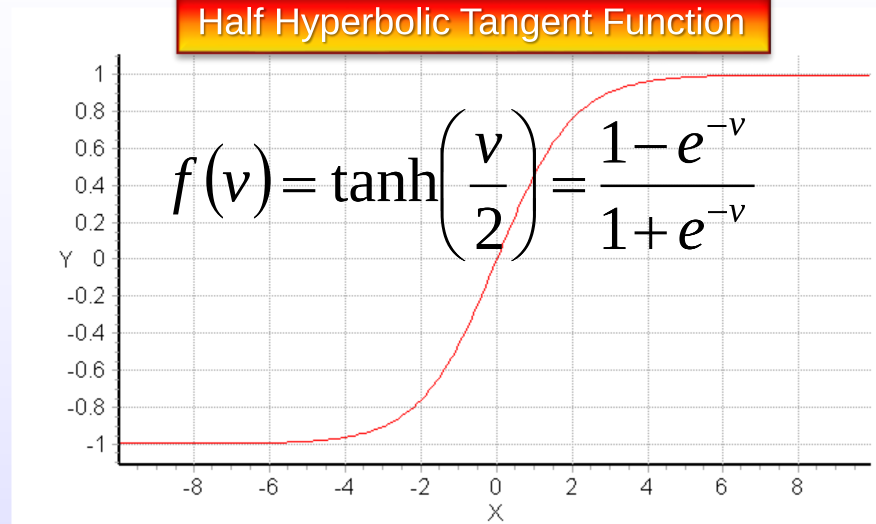
Transfer Functions for Back Propagation MLPs

Logistic Function (Binary Sigmoidal)



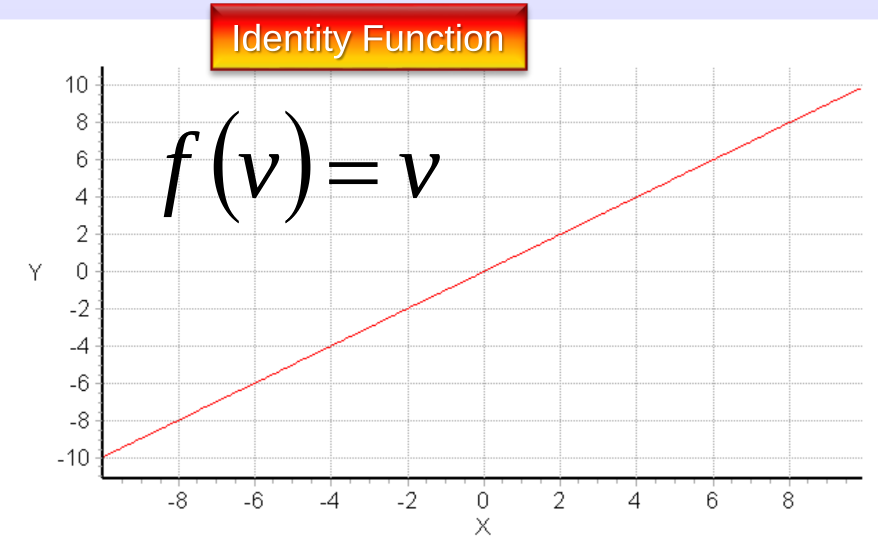
$$f(v) = \frac{1}{1+e^{-v}}$$

Half Hyperbolic Tangent Function



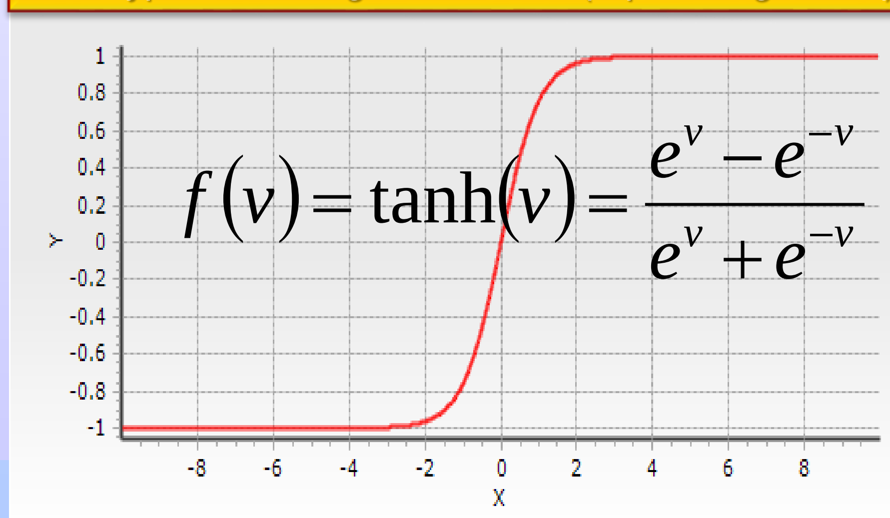
$$f(v) = \tanh\left(\frac{v}{2}\right) = \frac{1-e^{-v}}{1+e^{-v}}$$

Identity Function



$$f(v) = v$$

Full Hyperbolic Tangent Function (Bipolar Sigmoidal)



$$f(v) = \tanh(v) = \frac{e^{v}-e^{-v}}{e^{v}+e^{-v}}$$

$$\frac{1}{1+e^{-v}}$$

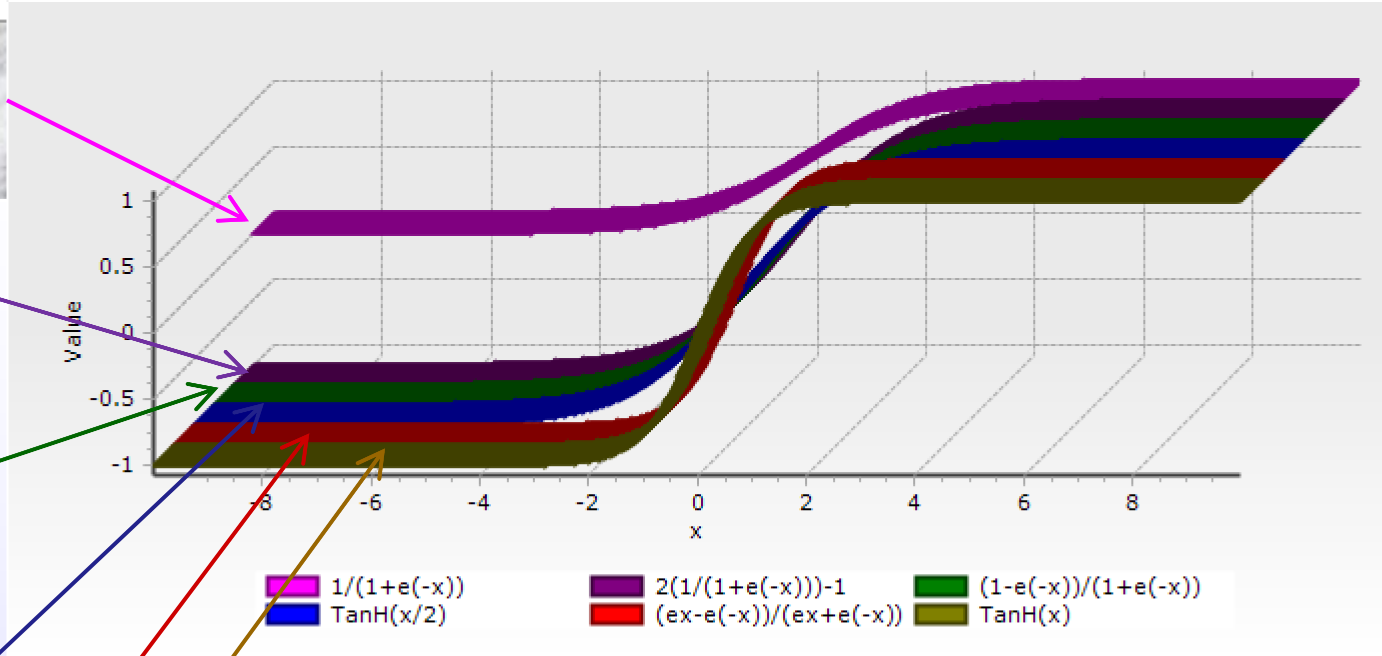
$$\frac{2}{1+e^{-v}}-1$$

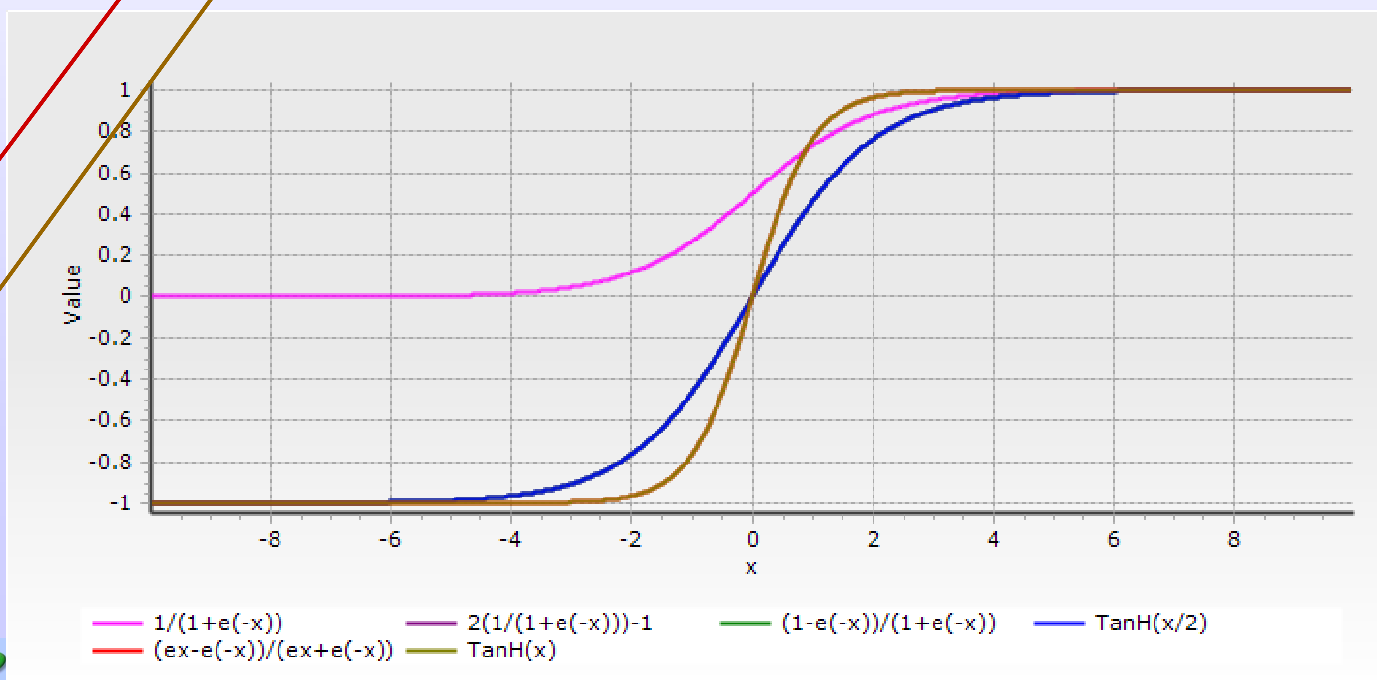
$$\frac{1-e^{-v}}{1+e^{-v}}$$

$$\tanh\left(\frac{v}{2}\right)$$

$$\frac{e^{v}-e^{-v}}{e^{v}+e^{-v}}$$

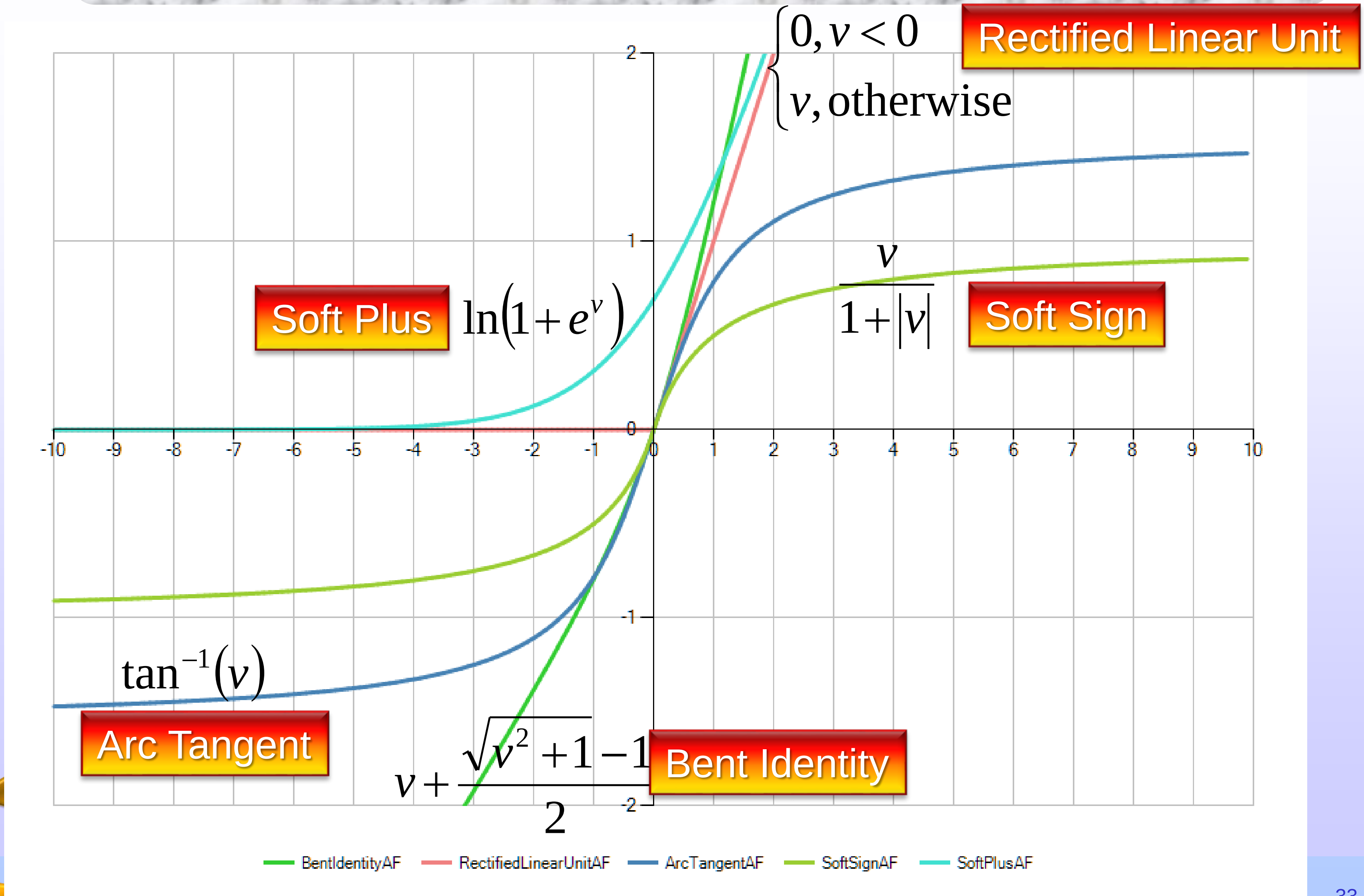
$$\tanh(v)$$





- Logistic and Hyperbolic tangent functions
  - Approximate the signum and step functions
  - Smooth, nonzero derivatives with respect to input signals
  - Referred to as “Squashing Functions”
  - Also called “Sigmoidal Functions”
    - Hyperbolic tangent → Bipolar sigmoidal
    - Logistic → Binary sigmoidal
- Identity function
  - Continuous valued function not limited to [0,1] or [-1,1]
  - No squashing function
  - Linear node

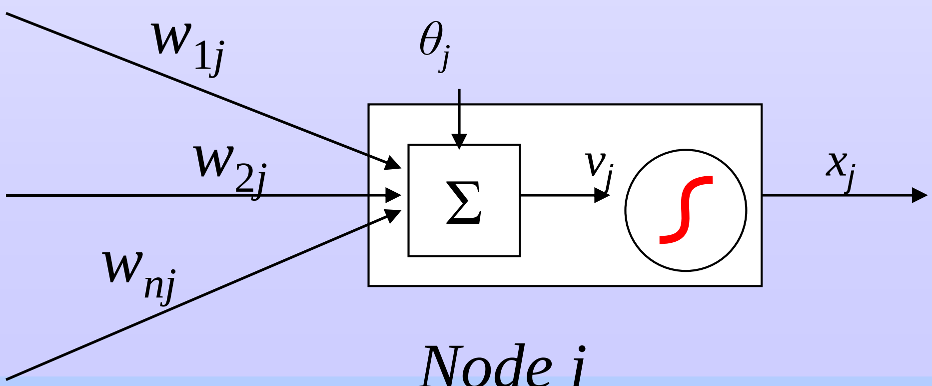
Transfer Functions in Deep Learning NNs



Backpropagation Learning Rule

- **Assume logistic transfer function is used**
- **A net input v of a node is defined as the weighted sum of the incoming signals plus a bias**

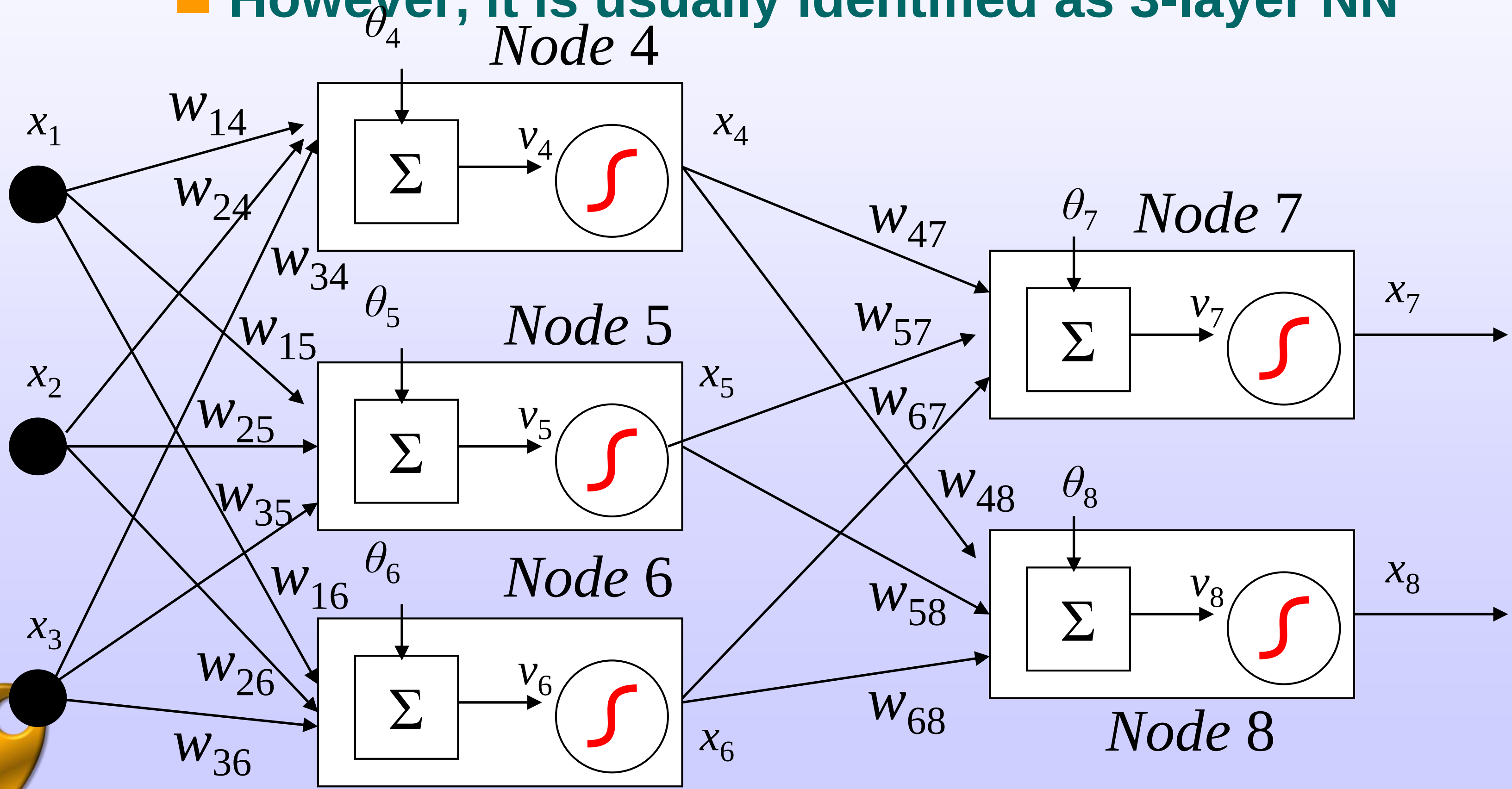
$$v_j = \sum_i w_{ij} x_i + \theta_j, \quad x_j = f\left(v_j\right) = \frac{1}{1+e^{-v_j}}$$



Node j

Sample MLP Neural Network

- **Two-layer backpropagation**
  - **Input layer is not counted as a physical layer**
  - **However, it is usually identified as 3-layer NN**



Backward Error Propagation

- **Backward error propagation=Backprogagation (BP)=Generalized Delta Rule (GDR)**

$$E=\sum_k\left(d_k-x_k\right)^2,k=1,\ldots,\text{number of output neurons}$$

- **E: squared error measure, d_k : the desired output for node k, x_k: actual output of node k**
- **The gradient vector of E with respect to the neuron values and weights are**

$$\nabla E=\left[\frac{\partial E}{\partial x_k}\right]=\left[\frac{\partial E}{\partial x_1}\ \frac{\partial E}{\partial x_2}\cdots\right]\qquad \nabla E=\left[\frac{\partial^+ E}{\partial w_{ki}}\right]=\left[\frac{\partial^+ E}{\partial w_{11}}\ \frac{\partial^+ E}{\partial w_{12}}\cdots\right]$$

- **The partial gradient vector of E with respect to the net input v_j is**

$$\nabla E=\left[\frac{\partial^+ E}{\partial v_j}\right]=\left[\frac{\partial^+ E}{\partial v_1}\ \frac{\partial^+ E}{\partial v_2}\cdots\right]$$

Ordered Derivative

$$z = g(x, y)$$

if x and y are independent

$$\nabla g = \left[\frac{\partial z}{\partial x}, \frac{\partial z}{\partial y}\right], \frac{\partial z}{\partial x} = \frac{\partial g(x, y)}{\partial x}, \frac{\partial z}{\partial y} = \frac{\partial g(x, y)}{\partial y}$$

if $y = f(x)$

$$\frac{\partial z}{\partial y} = \frac{\partial g(x, y)}{\partial y}, \text{yet}$$

$$\frac{\partial z}{\partial x} = \frac{\partial^{+} z}{\partial x} = \left.\frac{\partial g(x, y)}{\partial x}\right|_{y=f(x)} + \left.\frac{\partial g(x, y)}{\partial y}\right|_{y=f(x)} \cdot \frac{\partial y}{\partial x}$$

$$= \left.\frac{\partial g(x, y)}{\partial x}\right|_{y=f(x)} + \left.\frac{\partial g(x, y)}{\partial y}\right|_{y=f(x)} \cdot \frac{\partial f(x)}{\partial x}$$

Example

$$z = g\left(x, y\right) = 3x^2 y$$

if x and y are independent

$$\frac{\partial z}{\partial x} = \frac{\partial\left(3x^2 y\right)}{\partial x} = 6xy \Leftarrow \text{Right!} \quad \frac{\partial z}{\partial y} = \frac{\partial\left(3x^2 y\right)}{\partial y} = 3x^2 \Leftarrow \text{Right!}$$

$$\text{if } y = f(x) = (4x+2),\ z' \equiv 3x^2(4x+2) = 12x^3 + 6x^2, \quad \frac{\partial z'}{\partial x} = 36x^2 + 12x$$

$$\frac{\partial z}{\partial x} = \left.\frac{\partial\left(3x^2 y\right)}{\partial x}\right|_{y=f(x)} = 6xy\Big|_{y=f(x)} = 6x\left(4x+2\right) = 24x^2 + 12x \neq 36x^2 + 12x \Leftarrow \text{Wrong!}$$

$$\frac{\partial^+ z}{\partial x} \Leftarrow \text{Right!} \quad \frac{\partial^+ z}{\partial x} = \frac{\partial^+\left(3x^2 y\right)}{\partial x} = \left.\frac{\partial\left(3x^2 y\right)}{\partial x}\right|_{y=f(x)} + \left.\frac{\partial\left(3x^2 y\right)}{\partial y}\right|_{y=f(x)} \cdot \frac{\partial f\left(x\right)}{\partial x}$$

$$= 6xy\Big|_{y=f(x)} + 3x^2\Big|_{y=f(x)} \frac{\partial\left(4x+2\right)}{\partial x}$$

$$= 6x\left(4x+2\right) + 3x^2 \cdot 4 = 24x^2 + 12x + 12x^2 = 36x^2 + 12x$$

x is independent; y_i depends on x

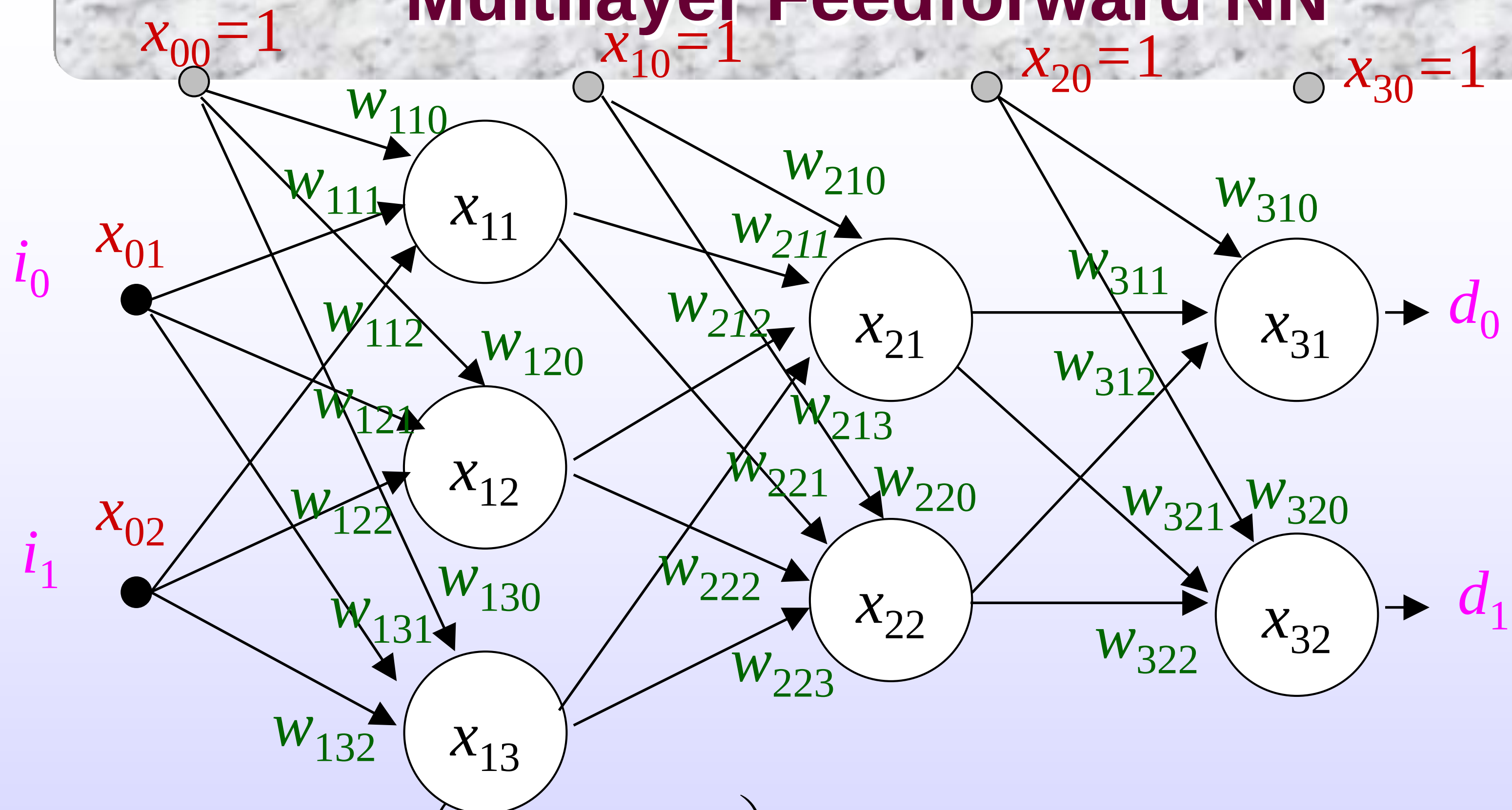
$$z = g\left(x, y_1, y_2, \ldots, y_n\right)$$

$$y_i = f_i\left(x\right)$$

$$\frac{\partial^+ z}{\partial x} = \left.\frac{\partial g}{\partial x}\right|_{y_i = f_i(x)} + \left.\frac{\partial g}{\partial y_1}\right|_{y_i = f_i(x)} \cdot \frac{\partial f_1\left(x\right)}{\partial x} + \ldots + \left.\frac{\partial g}{\partial y_n}\right|_{y_i = f_i(x)} \cdot \frac{\partial f_n\left(x\right)}{\partial x}$$

$$= \left.\frac{\partial g}{\partial x}\right|_{y_i = f_i(x)} + \sum_{i=1}^{n} \left.\frac{\partial g}{\partial y_i}\right|_{y_i = f_i(x)} \cdot \frac{\partial f_i\left(x\right)}{\partial x}$$

Multilayer Feedforward NN



$$x_{l,k} = f\left(\sum_{i=0}^{n_{l\text{-}1}} w_{l,k,i} \cdot x_{l\text{-}1,i}\right), x_{l,0} = 1.0,\ l = 1,2,\ldots,L,\ k = 1,2,\ldots,n_l$$

$$x_{l,k} = f\left(v_{l,k}\right), v_{l,k} = \sum_{i=0}^{n_{l\text{-}1}} w_{l,k,i} \cdot x_{l\text{-}1,i}$$

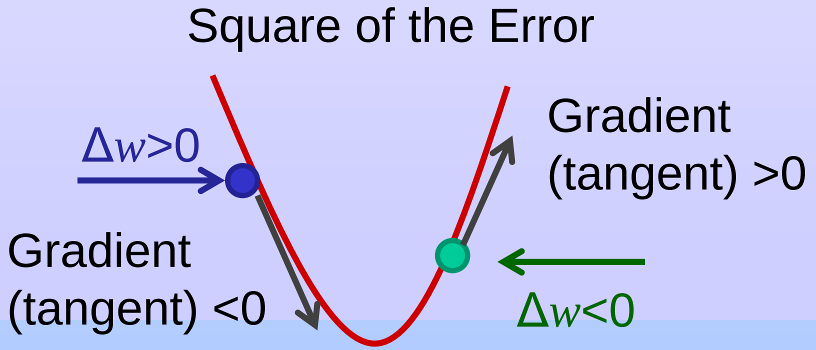
n_l : The number of neurons in layer l

$l = 0$: input layer; $l = L$: output layer

Output neuron values depends on neuron values of the previous layers

Learning Method

- **Adjustable parameters**
  - $w_{l,k,i} \leftarrow w_{l,k,i} + \Delta w_{l,k,i}$
- **Objectives**
  - **Are different in various learning methods**
- **The Least Square of the Error**
  - **The steepest descent approach**
    - Find the gradient of the error with respect to $w_{l,k,i}$
    - Update $w_{l,k,i}$ with $w_{l,k,i} + \Delta w_{l,k,i}$
    - $\Delta w_{l,k,i}$ = -Step size x gradient



Derivative of Errors

$$E = \sum_{k=1}^{n_L} \left(d_k - x_{L,k}\right)^2, \rightarrow \text{find } \frac{\partial E}{\partial w_{L,k,i}}$$

Output neuron values depends on neuron values of the previous layers

Since the computed values are $x_{l,k}$, find $\frac{\partial E}{\partial v_{L,k}}$ and $\frac{\partial^+ E}{\partial v_{l,k}}$ first.

For logistic sigmoidal activation $f(\cdot)$:

$$\frac{\partial f(v)}{\partial v} = \frac{\partial \left(\frac{1}{1+e^{-v}} \right)}{\partial v} = \left(\frac{1}{1+e^{-v}} \right) \left(1 - \frac{1}{1+e^{-v}} \right) = f(v)\left(1 - f(v)\right)$$

Define $\varepsilon_{L,k} = \frac{\partial E}{\partial v_{L,k}} = \frac{\partial \left(\sum_{k=1}^{n_L} \left(d_k - x_{L,k}\right)^2 \right)}{\partial v_{L,k}} = -2\left(d_k - x_{L,k}\right) \frac{\partial x_{L,k}}{\partial v_{L,k}}$

$$= -2\left(d_k - x_{L,k}\right) \frac{\partial f\left(v_{L,k}\right)}{\partial v_{L,k}} = -2\left(d_k - x_{L,k}\right) f\left(v_{L,k}\right)\left(1 - f\left(v_{L,k}\right)\right)$$

$$= -2\left(d_k - x_{L,k}\right)\left(x_{L,k}\right)\left(1 - x_{L,k}\right)$$

computer automation

Derivatives Derivation Details

$$\frac{\partial f(v)}{\partial v}=\frac{\partial}{\partial v}\left(\frac{1}{1+e^{-v}}\right)=\frac{-\frac{d}{dv}\left(1+e^{-v}\right)}{\left(1+e^{-v}\right)^2}=\frac{-\left(e^{-v}\frac{d(-v)}{dv}\right)}{\left(1+e^{-v}\right)^2}=\frac{-\left(-e^{-v}\right)}{\left(1+e^{-v}\right)\left(1+e^{-v}\right)}$$

$$=\left(\frac{1}{1+e^{-v}}\right)\left(\frac{e^{-v}}{1+e^{-v}}\right)=\left(\frac{1}{1+e^{-v}}\right)\left(\frac{1+e^{-v}-1}{1+e^{-v}}\right)=\left(\frac{1}{1+e^{-v}}\right)\left(1-\frac{1}{1+e^{-v}}\right)$$

$$=f(v)\left(1-f(v)\right)$$

$$\frac{d}{dv}\left(\frac{1}{X}\right)=\frac{-\frac{d}{dv}X}{X^2},$$

$$X=1+e^{-v},-\frac{d}{dv}X=-\frac{d}{dv}\left(1+e^{-v}\right)=-e^{-v}\frac{d(-v)}{dv}=e^{-v}=1+e^{-v}-1=X-1$$

$$\frac{d}{dv}\left(\frac{1}{X}\right)=\frac{X-1}{X^2}=\frac{1}{X}\frac{X-1}{X}=\frac{1}{X}\left(1-\frac{1}{X}\right)=\frac{1}{1+e^{-v}}\left(1-\frac{1}{1+e^{-v}}\right)$$

Derivatives Derivation Details

For full hyperbolic tangent activation $f\left(v\right)=\dfrac{e^{v}-e^{-v}}{e^{v}+e^{-v}}$:

$$\frac{\partial f\left(v\right)}{\partial v}=\frac{\partial}{\partial v}\left(\frac{e^{v}-e^{-v}}{e^{v}+e^{-v}}\right)=\frac{\left(e^{v}+e^{-v}\right)\cdot\dfrac{d}{dv}\left(e^{v}-e^{-v}\right)-\left(e^{v}-e^{-v}\right)\cdot\dfrac{d}{dv}\left(e^{v}+e^{-v}\right)}{\left(e^{v}+e^{-v}\right)^{2}}=$$

$$\frac{\left(e^{v}+e^{-v}\right)\left(e^{v}+e^{-v}\right)-\left(e^{v}-e^{-v}\right)\left(e^{v}-e^{-v}\right)}{\left(e^{v}+e^{-v}\right)^{2}}=\frac{\left(e^{v}+e^{-v}\right)^{2}-\left(e^{v}-e^{-v}\right)^{2}}{\left(e^{v}+e^{-v}\right)^{2}}=\frac{A^{2}-B^{2}}{A^{2}}$$

$$=\frac{(A-B)(A+B)}{A\cdot A}=\left(\frac{A-B}{A}\right)\left(\frac{A+B}{A}\right)=\left(1-\frac{B}{A}\right)\left(1+\frac{B}{A}\right)$$

$$=\left(1-\frac{e^{v}-e^{-v}}{e^{v}+e^{-v}}\right)\left(1+\frac{e^{v}-e^{-v}}{e^{v}+e^{-v}}\right)=\left(1+f\left(v\right)\right)\left(1-f\left(v\right)\right)$$

$$\varepsilon_{L-1,k} = \frac{\partial^{+} E}{\partial v_{L-1,k}} = \left.\frac{\partial E}{\partial v_{L-1,k}}\right|_{v_{L,k}=\varphi\left(v_{L-1,k}\right)} + \sum_{i=1}^{n_L} \frac{\partial E}{\partial v_{L,i}} \frac{\partial v_{L,i}}{\partial v_{L-1,k}}$$

$$= 0 + \sum_{i=1}^{n_L} \varepsilon_{L,i} \frac{\partial v_{L,i}}{\partial v_{L-1,k}} = \sum_{i=1}^{n_L} \varepsilon_{L,i} \frac{\partial \sum_{j=0}^{n_{L-1}} w_{L,i,j} \cdot x_{L-1,j}}{\partial x_{L-1,k}} \frac{\partial x_{L-1,k}}{\partial v_{L-1,k}}$$

$$= \sum_{i=1}^{n_L} \varepsilon_{L,i} w_{L,i,k} \left(\frac{\partial x_{L-1,k}}{\partial v_{L-1,k}} \right) = \frac{\partial f\left(v_{L-1,k}\right)}{\partial v_{L-1,k}} \sum_{i=1}^{n_L} \varepsilon_{L,i} w_{L,i,k}$$

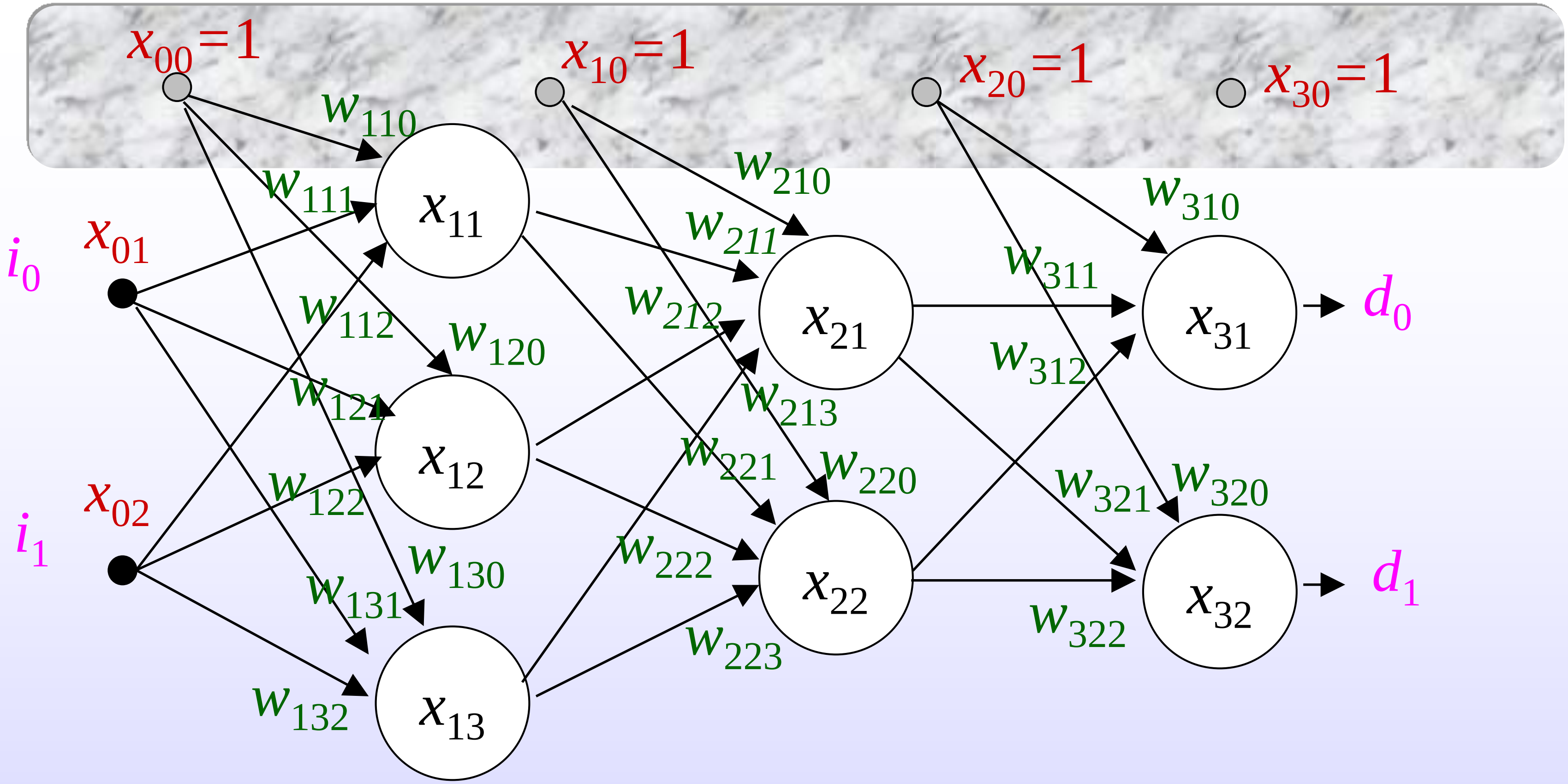
$$= \left(x_{L-1,k}\right)\left(1 - x_{L-1,k}\right) \sum_{i=1}^{n_L} \varepsilon_{L,i} w_{L,i,k}$$

Therefore, for sigmoidal activation, in general

$$\varepsilon_{L,k} = -2\left(d_k - x_{L,k}\right)\left(x_{L,k}\right)\left(1 - x_{L,k}\right),\ k = 1, 2, \ldots, n_L$$

$$\varepsilon_{l'-1,k} = \left(x_{l'\text{-}1,k}\right)\left(1 - x_{l'\text{-}1,k}\right) \sum_{i=1}^{n_{l'}} \varepsilon_{l',i} w_{l',i,k} ; k = 1, 2, \ldots, n_{l'-1} ; l' = 2, \ldots, L \quad \text{or}$$

$$\varepsilon_{l,k} = \left(x_{l,k}\right)\left(1 - x_{l,k}\right) \sum_{i=1}^{n_{l+1}} \varepsilon_{l+1,i} w_{l+1,i,k} ,\ k = 1, 2, \ldots, n_l ; l = 1, \ldots, L-1$$



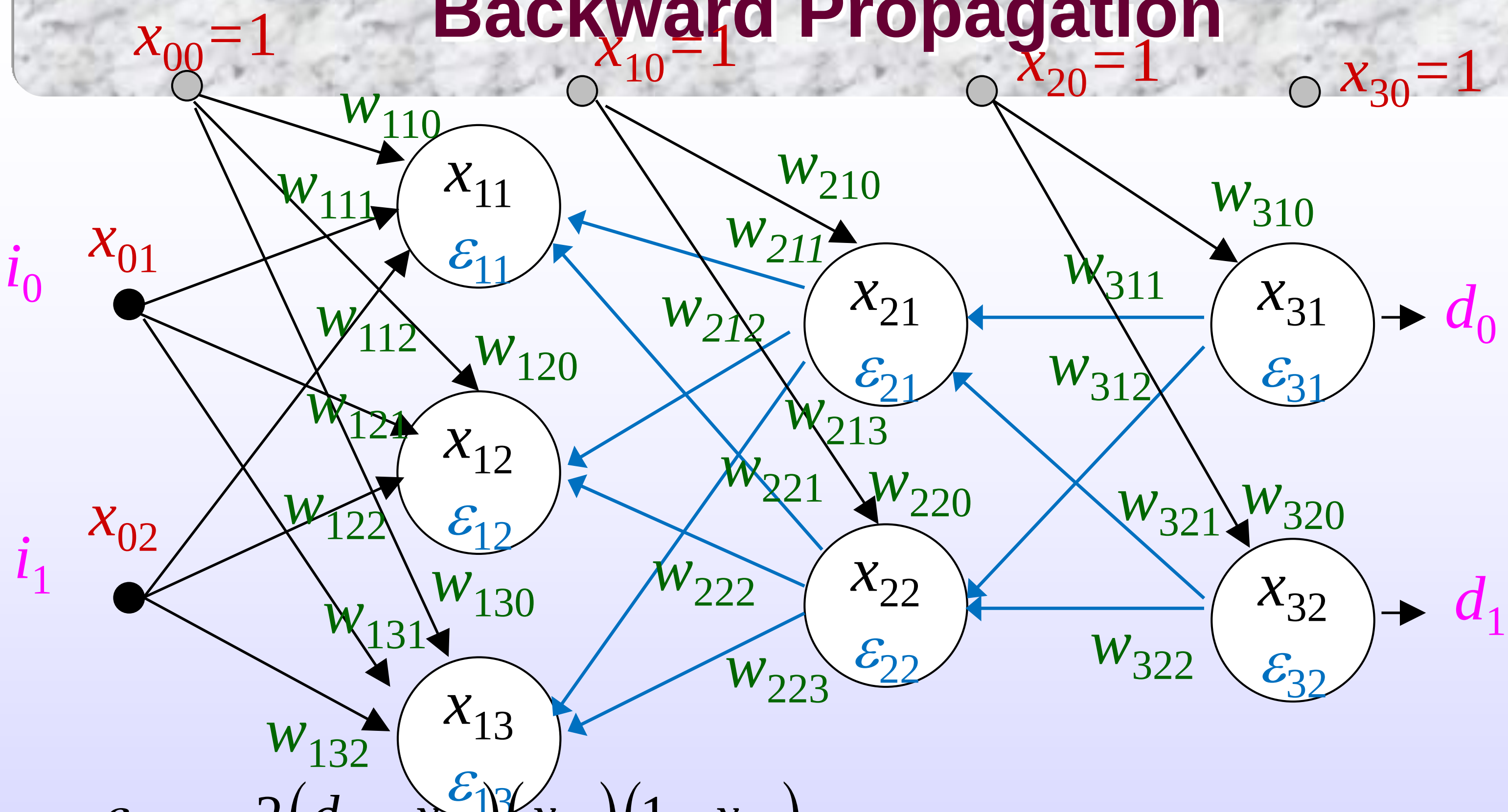
$$\varepsilon_{3,1} = -2\left(d_0 - x_{3,1}\right)\left(x_{3,1}\right)\left(1 - x_{3,1}\right)$$

$$\varepsilon_{3,2} = -2\left(d_1 - x_{3,2}\right)\left(x_{3,2}\right)\left(1 - x_{3,2}\right)$$

$$\varepsilon_{2,1} = \left(x_{2,1}\right)\left(1 - x_{2,1}\right)\sum_{i=1}^{n_3} \varepsilon_{3,i} w_{3,i,1} = \left(x_{2,1}\right)\left(1 - x_{2,1}\right)\left(\varepsilon_{3,1} w_{3,1,1} + \varepsilon_{3,2} w_{3,2,1}\right)$$

$$\varepsilon_{2,2} = \left(x_{2,2}\right)\left(1 - x_{2,2}\right)\sum_{i=1}^{n_3} \varepsilon_{3,i} w_{3,i,2} = \left(x_{2,2}\right)\left(1 - x_{2,2}\right)\left(\varepsilon_{3,1} w_{3,1,2} + \varepsilon_{3,2} w_{3,2,2}\right)$$

Backward Propagation



$$\varepsilon_{3,1}=-2\left(d_0-x_{3,1}\right)\left(x_{3,1}\right)\left(1-x_{3,1}\right)$$

$$\varepsilon_{3,2}=-2\left(d_1-x_{3,2}\right)\left(x_{3,2}\right)\left(1-x_{3,2}\right)$$

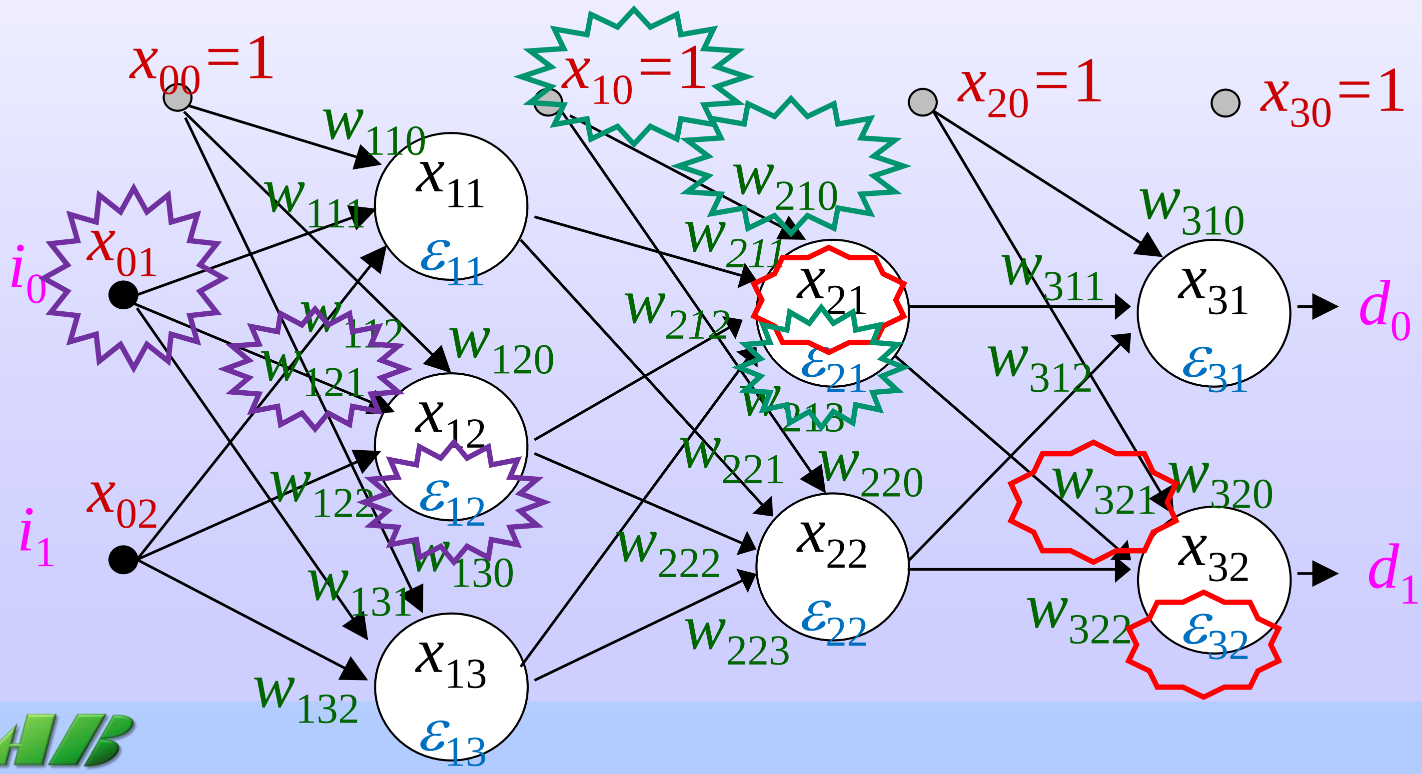
$$\varepsilon_{2,1}=\left(x_{2,1}\right)\left(1-x_{2,1}\right)\sum_{i=1}^{n_3}\varepsilon_{3,i}w_{3,i,1}=\left(x_{2,1}\right)\left(1-x_{2,1}\right)\left(\varepsilon_{3,1}w_{3,1,1}+\varepsilon_{3,2}w_{3,2,1}\right)$$

$$\varepsilon_{2,2}=\left(x_{2,2}\right)\left(1-x_{2,2}\right)\sum_{i=1}^{n_3}\varepsilon_{3,i}w_{3,i,2}=\left(x_{2,2}\right)\left(1-x_{2,2}\right)\left(\varepsilon_{3,1}w_{3,1,2}+\varepsilon_{3,2}w_{3,2,2}\right)$$

Parameter Update

$$\Delta w_{l,k,i} = -\eta \frac{\partial E}{\partial w_{l,k,i}} = -\eta \frac{\partial^{+} E}{\partial v_{l,k}} \frac{\partial v_{l,k}}{\partial w_{l,k,i}} = -\eta \varepsilon_{l,k} \frac{\partial v_{l,k}}{\partial w_{l,k,i}}$$

$$= -\eta \varepsilon_{l,k} \frac{\partial \sum_{j=0}^{n_{l\text{-}1}} w_{l,k,j} \cdot x_{l\text{-}1,j}}{\partial w_{l,k,i}} = -\eta \varepsilon_{l,k} x_{l\text{-}1,i},\ \ i = 0,1,2,\dots,n_l$$



Practical Implementation

$$w_{l,k,i} \leftarrow w_{l,k,i} - \eta \varepsilon_{l,k} x_{l\text{-}1,i}; \forall l = 1, 2, ..., L; \forall k = 1, \ldots, n_l; i = 0, 1, 2, ..., n_{l-1};$$

When an epoch is completed:

$\eta \leftarrow \alpha\eta, 0 < \alpha \leq 1.0; \alpha = 0.9$ is suggested

DEMO

Speeding Up Training

- Use momentum term

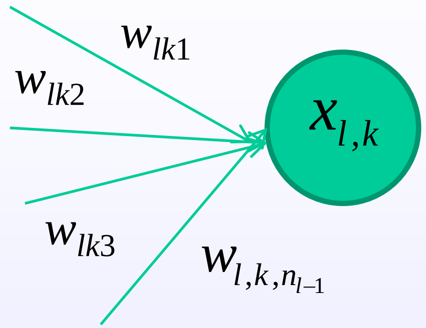
$$\Delta w_{l,k,i} = -\eta \nabla_{w_{l,k,i}} E + \alpha \Delta w_{l,k,i}^{prev}, 0.1 \le \alpha \le 1.0$$

$\Delta w_{l,k,i}^{prev}$ is the previous update amount

- Smooth weight updating
- Resist erratic weight changes
  - Changes from gradient noises or high error frequencies
- Not guaranteed
  - Usually problem dependent

- **Weight update normalization**

$$\Delta \mathbf{w}_{l,k} = -\gamma \frac{\nabla_{w_{l,k}} E}{\left\| \nabla_{w_{l,k}} E \right\|},$$



  - **Unified the moving distance of the weight vector to γ**
    - γ is set from history or error measure
    - No learning rate is applied

- **Others**
  - **Quick-propagation algorithm, delta-bar-delta approach, extended Kalman filter method, second-order optimization, optimal filtering approach**

Discussion about MLPs

- **Different Activation Functions**
- **Using Hyperbolic tangent function**
  - **Learning speed is faster than logistic functions, in general**
  - **Output scaling should be applied**
    - Not using [-1,1]
    - Using [-0.9, 0.9] to avoid an infinite value changed in the weight update
  - **Input scaling should be applied to restrict the input within the corresponding range**

$$f(v) = \tanh(v) = \frac{e^{v} - e^{-v}}{e^{v} + e^{-v}}$$

$$\frac{df(v)}{dv} = \left(1 + f(v)\right)\left(1 - f(v)\right)$$

Neuron Saturation

- **Output value of a neuron is too large**
  - **Consequents**
    - Derivative of the activation function is zero
    - Amount of parameter value change is too small
    - No contribution to the weight update
  - **Causes**
    - Net value is too large
    - Learning rate is too large
    - Moment factor is too large
    - Non-extreme range of the activation function is too narrow

Overcome Neuron Saturation

- **Rescale the activation values**
  - 0.9[-1,1]=[-0.9, 0.9]
- **Add a minimal value to the derivative of the activation function, to have non-zero weight changes**

$$\varepsilon_{L,k} = -2\left(d_k - x_{L,k}\right)\left(\left(x_{L,k}\right)\left(1 - x_{L,k}\right) + 0.01\right),\ k = 1, 2, \ldots, n_L$$

$$\varepsilon_{l\text{-}1,k} = \left(x_{l\text{-}1,k}\left(1 - x_{l\text{-}1,k}\right) + 0.01\right)\sum_{i=1}^{n_l} \varepsilon_{l,i} w_{l,i,k},\ k = 1, 2, \ldots, n_l; \text{or}$$

$$\varepsilon_{l,k} = \left(x_{l\text{-}1,k}\left(1 - x_{l\text{-}1,k}\right) + 0.01\right)\sum_{i=1}^{n_{l+1}} \varepsilon_{l+1,i} w_{l+1,i,k},\ k = 1, 2, \ldots, n_l; l = 1, \ldots, L - 1$$

Initial Weights and Learning Rates

- **Initial weight and bias setting**
  - **Should be uniformly distributed across a small range, usually [-1,1] (for both binary and bipolar NN)**
    - A too large parameter will easily make a neuron saturated
      - Small error
    - A too small parameter generates small gradient
      - Small initial learning rate
- **Learning rate rescaling**
  - **Learning rate in front layers should be larger than rear ones**
    - Error signals are propagated from the rear ones

Number of Hidden Layers

- One hidden layer
  - With a sigmoidal nonlinear function can approximate any continuous function
- Two hidden layers
  - Can form arbitrary complex decision regions to separate different classes
- Large number of hidden layers
  - Deep learning in dealing with large scale of problem; e.g., image classification or recognition

Variants: Different Error Measures

- **Cubic values of errors**

$$E_p = \sum_k \left(d_k - x_k\right)^3$$

$$\varepsilon_{L,k} = -3\left(d_k - x_{L,k}\right)^2 \frac{\partial f\left(v_{L-1,k}\right)}{\partial v_{L-1,k}}$$

- **Quartic values of errors**

$$E_p = \sum_k \left(d_k - x_k\right)^4$$

$$\varepsilon_{L,k} = -4\left(d_k - x_{L,k}\right)^3 \frac{\partial f\left(v_{L-1,k}\right)}{\partial v_{L-1,k}}$$

Batch Training

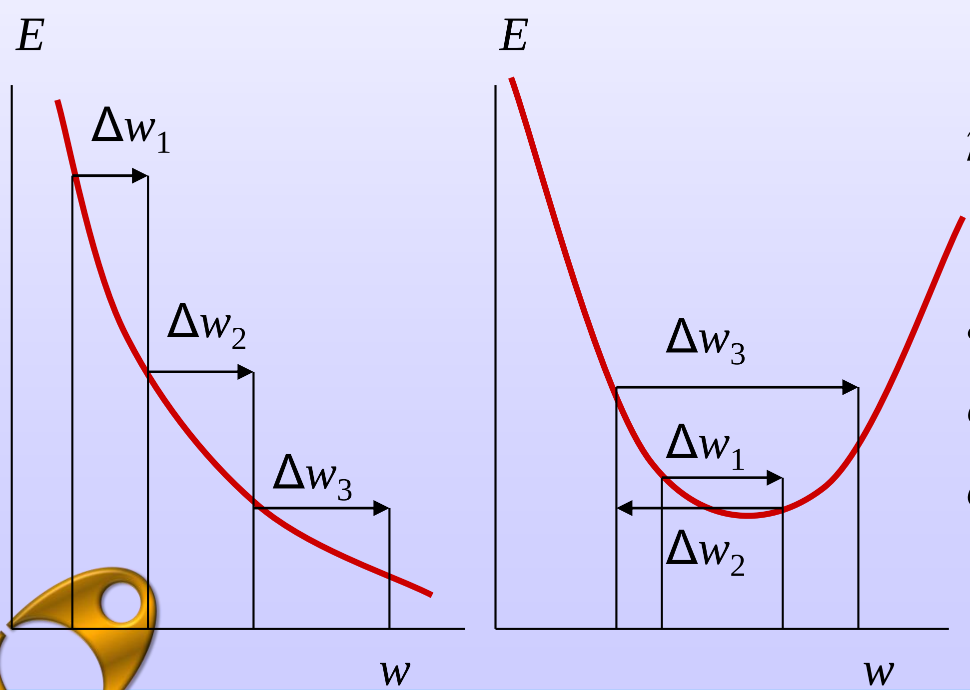
- Weights are updated when the set of all the input vectors are fed and the update amounts are cumulated
  - Since the change amounts are large, they should be reduced

$$\Delta w_{l,k,i} = \frac{\sum_{p=1}^{N} \Delta w_{l,k,i}^{p}}{\sqrt{N}}$$

  - The value of learning rate is also decreased batch by batch

Customized Learning Rate Adjustments

- **Delta-Bar-Delta**
  - Increase the learning rate when weight changes are in the same direction
  - Each weight has its learning rate



$$\eta_{l,k,i} = \begin{cases} \eta_{l,k,i}^{prev} + \delta, \text{if } \tilde{\varepsilon} \cdot \Delta w_{l,k,i} > 0 \\ \phi \cdot \eta_{l,k,i}^{prev}, \text{if } \tilde{\varepsilon} \cdot \Delta w_{l,k,i} < 0 \\ \eta_{l,k,i}^{prev}, \text{else} \end{cases},$$

$$\tilde{\varepsilon} = (1-\xi)\Delta w_{l,k,i} + \xi \cdot \Delta w_{l,k,i}^{prev}$$

$$\xi \in [0.3, 0.7]$$

$$\delta \in [0.05, 0.2], \phi \in [0.4, 0.6],$$

Importance Differentials of the Training Data

- **The training influences are different for different training data**
  - **Order sorting method**
    - Data are sorted in an importance increasing order
    - The weight change is multiplied by the importance factor of the training data

$$\Delta w_{l,k,i} = \left(\frac{p}{N}\right)^{\alpha} \Delta w_{l,k,i},$$

p is the sorting order of the training data
N is the total number of training data
α is a curve factor, $\alpha > 0$

Local Minimum Convergence Problem

- **The deepest descent error minimization approach can not avoid local convergence**
- **Small perturbation is added to the weight change**

$$w_{l,k,i} = w_{l,k,i} + \Delta w_{l,k,i} + \upsilon,$$
υ is a randomly generated perturbation

Speed Up Training

- Avoid backpropagation for small error
  - Set an small error tolerance
  - When a training datum produces a error smaller than the tolerance, no computation for weights adjustment
- Scale the real number computation to integral computation

Advanced Training Algorithm

- Threat the training as an optimization problem
- To minimize the defined error function with respect to the weights
  - Weights are the optimization variables
  - Error function is the objective function which is to minimize

Derivative-based or Derivative-free optimization techniques can be applied

- **Derivative-based Methods**
  - Newton's Method
    - Classical Newton
    - Modified Newton
    - Quasi-Newton
  - Conjugate Gradient Method
- **Derivative-free Methods**
  - Genetic Algorithm
  - Simulated Annealing
  - Random Search
  - Downhill Simplex Search
  - Other Heuristic Algorithms

Computing Assignment 005

- Design a general back-propagation MLP that allows different transfer functions, different numbers of layers and neurons implementations
  - Use data from UCI machine intelligence repository to train and test your NNs
  - Or use data in cal format
  - Provide possible visualization support

- Each attribute of the input data set should be preprocessed first before training
  - Either linearly map the values within [0, 1] or [-1, 1] depending on the transfer function used
- Batch-training or on-line training can be specified
- Number of output neurons is depended on the number of classes, if a UCI data set (classification data set) is used
- Root Mean Squared Error can be computed and used for stop condition

Programming Notes

- In-line Elapsed RMS of error
  - Average square root of error square for each output neuron on each training instance up to now

$C:$ number of training instances executed so far

$e^{(c)}:$ sum of error square on output neurons of training instance c

$e:$ cumulated error square

$$e^{(c)} = \sum_{k=1}^{n_L} \left(d_k^{(c)} - x_{L,k}^{(c)} \right)^2$$

$$e = \sum_{c=1}^{C} e^{(c)}$$

$$E^{elasped} = \sqrt{\frac{e}{C \cdot n_L}}$$

Programming Notes

- Epoch RMS of error
  - The average square root of error square for each output neuron on each training instance within an epoch

J : number of training data

$e^{(j)}$: sum of error squares on output neurons of training instance j

$e_{(p)}$: cumulated error square in epoch p ,

initialized to 0 at the beginning of each epoch

$$e^{(j)} = \sum_{k=1}^{n_L} \left(d_k^{(j)} - x_{L,k}^{(j)} \right)^2$$

$$e_{(p)} = \sum_{j=1}^{J} e^{(j)}$$

$$E^{epoch} = \sqrt{\frac{e_{(p)}}{J \cdot n_L}}, \text{evaluated at the end of an epoch}$$