Feed Forward Error Back Propagation Artificial Neural Networks

CSE

Malaviya National Institute of Technology Jaipur

February 14, 2025

Connectionist Models

Consider humans:

- Neuron switching time ~ .001 second
- Number of neurons ~ 10^{10}
- Connections per neuron ~ 10^{4-5}
- Scene recognition time ~ .1 second

$\rightarrow$ much parallel computation

Properties of artificial neural nets (ANN's):

- Many neuron-like threshold switching units
- Many weighted interconnections among units
- Highly parallel, distributed process
- Emphasis on tuning weights automatically

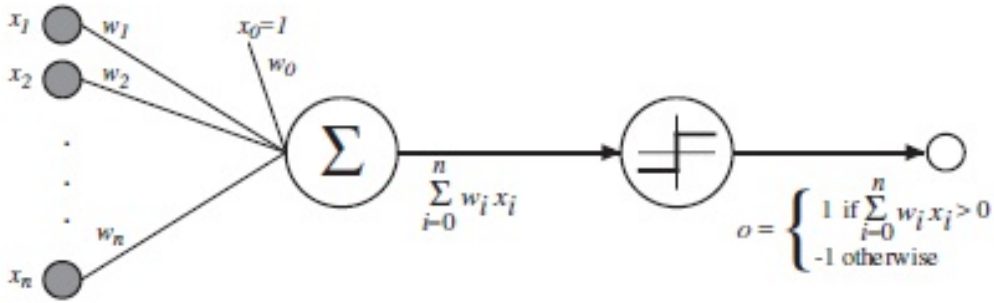
When to Consider Neural Networks

- Input is high-dimensional discrete or real-valued (e.g. raw sensor input)
- Output is discrete or real valued
- Output is a vector of values
- Possibly noisy data
- Form of target function is unknown
- Human readability of result is unimportant

Examples:

- Speech phoneme recognition
- Image classification
- Financial prediction

Perceptron



$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Sometimes we'll use simpler vector notation:

$$o(\vec{x}) = \begin{cases} 1 & \text{if } \vec{w} \cdot \vec{x} > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Perceptron Training Rule

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

where

$$\Delta w_i = \eta(t - o)x_i$$

Where:

- $t = c(\vec{x})$ is target value
- o is perceptron output
- η is small constant (e.g., .1) called *learning rate*

Perceptron Training Rule

Can prove it will converge

- If training data is linearly separable
- and η sufficiently small

Perceptron: Example

Data

Input I_1	Input I_2	Target O/T
0	0	0 [-1]
0	1	0 [-1]
1	0	0 [-1]
1	1	1 [+1]

Network Parameters

Learning Rate=0.4

Input Nodes=2 Output Node =1

Initial Weights $[w_1\ w_2] = [0.6, -0.2]$

Activation Function Step Function

Activation Threshold 1.5

Perceptron: Example

Connection weights:

Step	Input Values	Net Input	Output	Weight Adjustment	Updated Weights
1	(0,0)	0.0	-1	$w_1 = 0.4 * (-1 + 1) * 0$ $w_2 = 0.4 * (-1 + 1) * 0$	0.6 -0.2
2	(0,1)	-0.2	-1	$w_1 = 0.4 * (-1 + 1) * 0$ $w_2 = 0.4 * (-1 + 1) * 1$	0.6 -0.2
3	(1,0)	0.6	-1	$w_1 = 0.4 * (-1 + 1) * 1$ $w_2 = 0.4 * (-1 + 1) * 0$	0.6 -0.2
4	(1,1)	0.4	-1	$w_1 = 0.4 * (1 + 1) * 1$ $w_2 = 0.4 * (1 + 1) * 1$	1.4 0.6
5	(0,0)	0.0	-1	$w_1 = 0.4 * (-1 + 1) * 0$ $w_2 = 0.4 * (-1 + 1) * 0$	1.4 0.6
6	(0,1)	0.6	-1	$w_1 = 0.4 * (-1 + 1) * 0$ $w_2 = 0.4 * (-1 + 1) * 1$	1.4 0.6
7	(1,0)	1.4	-1	$w_1 = 0.4 * (-1 + 1) * 1$ $w_2 = 0.4 * (-1 + 1) * 0$	1.4 0.6
8	(1,1)	2	+1	$w_1 = 0.4 * (1 - 1) * 1$ $w_2 = 0.4 * (1 - 1) * 1$	1.4 0.6

Gradient Descent

To understand, consider simpler *linear unit*, where

$$o = w_0 + w_1 x_1 + \cdots + w_n x_n$$

Let's learn w_i's that minimize the squared error

$$E[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

Where D is set of training examples

Gradient Descent

Gradient

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots \frac{\partial E}{\partial w_n}\right]$$

Training rule:

$$\Delta \vec{w} = -\eta \nabla E[\vec{w}]$$

i.e.,

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

Gradient Descent

$$
\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2}\sum_d (t_d - o_d)^2 \\
&= \frac{1}{2}\sum_d \frac{\partial}{\partial w_i}(t_d - o_d)^2 \\
&= \frac{1}{2}\sum_d 2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d) \\
&= \sum_d (t_d - o_d)\frac{\partial}{\partial w_i}(t_d - \vec{w}\cdot\vec{x_d}) \\
\frac{\partial E}{\partial w_i} &= \sum_d (t_d - o_d)(-x_{i,d})
\end{aligned}
$$

Gradient Descent

Gradient-Descent($training\_examples, \eta$)

Each training example is a pair of the form $\langle \vec{x}, t \rangle$, where $\vec{x}$ is the vector of input values, and t is the target output value. η is the learning rate (e.g., 0.05).

- Initialize each w_i to some small random value
- Until the termination condition is met, Do
  - Initialize each Δw_i to zero.
  - For each $\langle \vec{x}, t \rangle$ in *training_examples*, Do
    - Input the instance $\vec{x}$ to the unit and compute the output o
    - For each linear unit weight w_i, Do

$$\Delta w_i \leftarrow \Delta w_i + \eta(t - o)x_i$$

  - For each linear unit weight w_i, Do

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

Gradient-Descent Training Rule Summary

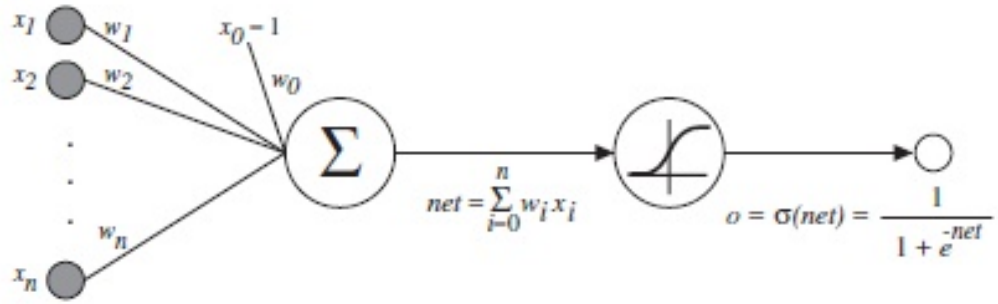
Perceptron training rule guaranteed to succeed if

- Training examples are linearly separable
- Sufficiently small learning rate η

Linear unit training rule uses gradient descent

- Guaranteed to converge to hypothesis with minimum squared error
- Given sufficiently small learning rate η
- Even when training data contains noise
- Even when training data not separable by H

Sigmoid Unit



$\sigma(x)$ is the sigmoid function

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

Nice property: $\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$

We can derive gradient decent rules to train

- One sigmoid unit
- *Multilayer networks* of sigmoid units $\rightarrow$ Backpropagation

Error Gradient for a Sigmoid Unit

$$
\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \\
&= \frac{1}{2} \sum_d \frac{\partial}{\partial w_i} (t_d - o_d)^2 \\
&= \frac{1}{2} \sum_d 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d) \\
&= \sum_d (t_d - o_d) \left(-\frac{\partial o_d}{\partial w_i} \right) \\
&= -\sum_d (t_d - o_d) \frac{\partial o_d}{\partial net_d} \frac{\partial net_d}{\partial w_i}
\end{aligned}
$$

Error Gradient for a Sigmoid Unit

But we know:

$$\frac{\partial o_d}{\partial net_d} = \frac{\partial \sigma(net_d)}{\partial net_d} = o_d(1 - o_d)$$

$$\frac{\partial net_d}{\partial w_i} = \frac{\partial(\vec{w} \cdot \vec{x}_d)}{\partial w_i} = x_{i,d}$$

So:

$$\frac{\partial E}{\partial w_i} = -\sum_{d \in D}(t_d - o_d)o_d(1 - o_d)x_{i,d}$$

Error Backpropagation Algorithm

Initialize all weights to small random numbers. Until satisfied, Do

- For each training example, Do
  1. Input the training example to the network and compute the network outputs
  2. For each output unit k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

  3. For each hidden unit h

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{h,k}\delta_k$$

  4. Update each network weight $w_{i,j}$

$$w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j}$$

where

$$\Delta w_{i,j} = \eta \delta_j x_{i,j}$$

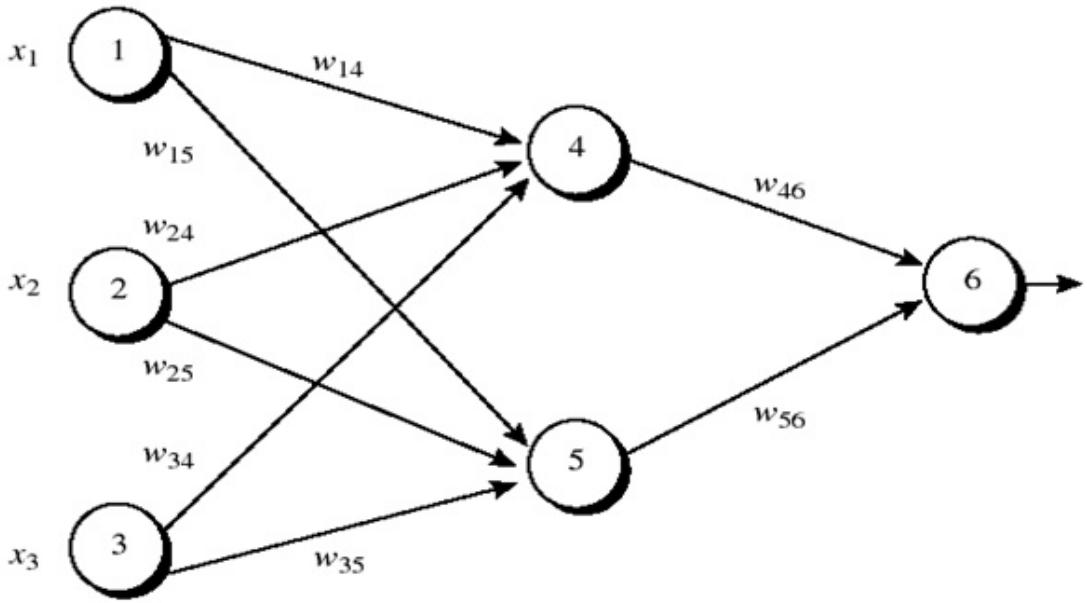
More on Backpropagation

- Gradient descent over entire *network* weight vector
- Easily generalized to arbitrary directed graphs
- Will find a local, not necessarily global error minimum
  - In practice, often works well (can run multiple times)
- Often include weight *momentum* α

$$\Delta w_{i,j}(n) = \eta \delta_j x_{i,j} + \alpha \Delta w_{i,j}(n-1)$$

- Minimizes error over *training* examples
  - Will it generalize well to subsequent examples?
- Training can take thousands of iterations $\rightarrow$ slow!
- Using network after training is very fast

Backpropagation: Example



An example of a multilayer feed-forward neural network. Assume that the learning rate η is 0.9 and the first training example, X = (1,0,1) whose class label is 1.

The sigmoid function is applied to hidden layer and output layer.

Backpropagation: Example

Connection wieghts:

w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	w_{46}	w_{56}	w_{04}	w_{05}	w_{06}
0.2	-0.3	0.4	0.1	-0.5	0.2	-0.3	-0.2	-0.4	0.2	0.1

Backpropagation: Example

Net Input and Output Calculations

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

Error Calculations at Each Node

Unit j	Error δ_j
6	(0.474)(1-0.474)(1-.474)= 0.1311
5	(0.525)(1-0.525)(0.1311)(-0.2)= -0.0065
4	(0.332)(1-0.322)(0.1311)(-0.3)= -0.0087

Backpropagation: Example

Weight Update Calculations

Weight	New Value
w_{46}	-03+(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
w_{06}	0.1 + (0.9)(0.1311) = 0.218
w_{05}	0.2 + (0.9)(-0.0065)=0.1941
w_{04}	-0.4 +(0.9)(-0.0087) = -0.408

Convergence of Backpropagation

Gradient descent to some local minimum

- Perhaps not global minimum...
- Add momentum
- Stochastic gradient descent
- Train multiple nets with different inital weights

Nature of convergence

- Initialize weights near zero
- Therefore, initial networks near-linear
- Increasingly non-linear functions possible as training progresses