Lec5: Learning and Adaptation

Outline:

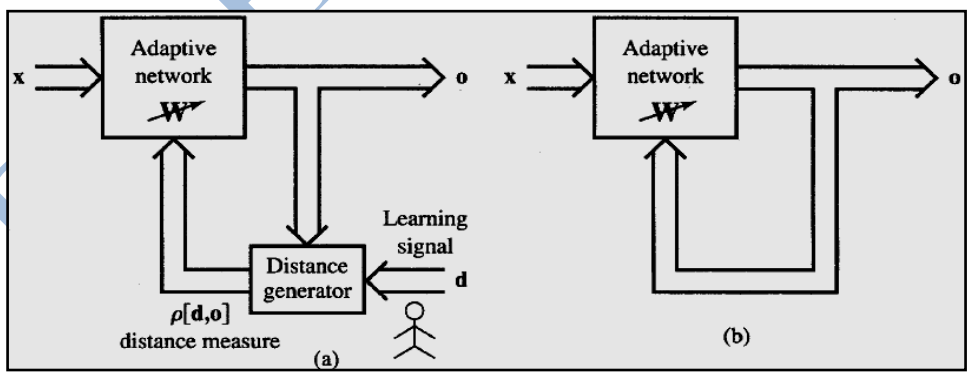
1. **Learning Overview**
2. **Supervised and Unsupervised Learning**
3. **Learning Rule Types**

1. Learning Overview

Each of us acquires and then hones our skills and abilities through the basic Phenomenon of learning. Learning is a fundamental subject for psychologists. In general, learning is a relatively permanent change in behavior brought about by experience. Learning in human beings and animals is an inferred process; we cannot see it happening directly and we can assume that it has occurred by observing changes in performance. Learning in neural networks is a more direct process, and we typically can capture each learning step in a distinct cause-effect relationship

2. Supervised and Unsupervised Learning

The main difference between unsupervised training and supervised training
1. **Unsupervised learning:** A means of modifying the weights of a neural net without specifying the desired output for any input patterns
2. supervised learning: Process of a adjusting the weights in a neural net using a learning algorithm, the desired output is presented to the net



Block diagram for: (a) Supervised learning and (b) Unsupervised learning.

Learning rule	Single weight adjustment Δw_{ij}	Initial weights	Learning	Neuron characteristics	Neuron / Layer
Hebbian	$co_i x_j$ $j = 1, 2, \ldots, n$	0	U	Any	Neuron
Perceptron	$c\,[d_i - \text{sgn}\,(\mathbf{w}_i^t \mathbf{x})]\,x_j$ $j = 1, 2, \ldots, n$	Any	S	Binary bipolar, or Binary unipolar*	Neuron
Delta	$c(d_i - o_i)f'(net_i)x_j$ $j = 1, 2, \ldots, n$	Any	S	Continuous	Neuron
Widrow-Hoff	$c(d_i - \mathbf{w}_i^t \mathbf{x})x_j$ $j = 1, 2, \ldots, n$	Any	S	Any	Neuron
Correlation	$cd_i x_j$ $j = 1, 2, \ldots, n$	0	S	Any	Neuron
Winner-take-all	$\Delta w_{mj} = \alpha(x_j - w_{mj})$ m-winning neuron number $j = 1, 2, \ldots, n$	Random Normalized	U	Continuous	Layer of p neurons
Outstar	$\beta(d_i - w_{ij})$ $i = 1, 2, \ldots, p$	0	S	Continuous	Layer of p neurons

3. Learning Rule Types

1. Hebb Net

The earliest and simplest learning rule for a neural net is generally known as the Hebb rule. Hebb proposed that learning occurs by modification of the synapse strengths (weights) in a manner such that if two interconnected neurons are both "on" at the same time, then the weight between those neurons should be increased. The original statement only talks about neurons firing at the same time (and does not say anything about reinforcing neurons that do not fire at the same time). However, a stronger form of learning occurs if we also increase the weights if both neurons are "off" at the same time. We use this extended Hebb rule [McClelland & Rumelhart, 1988] because of its improved computational power and shall refer to it as the Hebb rule. We shall refer to a single-layer (feed forward) neural net trained using the (extended) Hebb rule as a Hebb net

Algorithm

Step 0 Initialize all weights: w_i = **0** (i = 1 to n).
Step 1. For each input training vector and target output pair, s : ***t*** **,** do steps 2-4.
Step 2. Set activations for input units: = (i = 1 to n).
Step 3. Set activation for output unit: **y= t** .
Step 4. Adjust the weights for
w_i(new) = wi(old) + (i = 1 to n).
Adjust the bias:

$$b(\text{new}) = b(\text{old}) + y.$$

Note that the bias is adjusted exactly like a weight from a "unit" whose output signal is always 1. The weight update can also be expressed in vector form as

$$\mathbf{w}(\text{new}) = \mathbf{w}(\text{old}) + \mathbf{x}y.$$

This is often written in terms of the weight change, $\Delta\mathbf{w}$,

$$\Delta\mathbf{w} = \mathbf{x}y$$

and

$$\mathbf{w}(\text{new}) = \mathbf{w}(\text{old}) + \Delta\mathbf{w}.$$

Application of Hebb Net

Logic gates: AND, NOR,ANDNOT,OR.NAND,

1. Binary input and target
2. Bipolar input and target
3. Binary input and Bipolar target
4. Bipolar input and Binary target

Example 1: Training a **Hebb learning rule** with AND function binary input and target, initialization weights (w1=w2=b=0)

Solution:

For Hebb rule, AND function

X_1	X_2	1	Target(t)
1	1	1	1
1	0	1	0
0	1	1	0
0	0	1	0

Set activation function for input

Xi=S_i

Set activation function for output

y= ***t***

Adjust the weights for

$w_i(new) = w_i(old)+x_iy$ ***(i = 1 to n).***

Adjust the bias:

b(new)=b(old)+y

$\Delta w_1=x_1t$, $\Delta w_2=x_2t$, $\Delta b= t$

Input			Target	Weight changes			Weights		
X_1	X_2	1		Δw_1	Δw_2	Δb	W_1	W_2	b
							0	0	0
1	1	1		1	1	1	1	1	1
1	0	1		0	0	0	1	1	1
0	1	1		0	0	0	1	1	1
0	0	1		0	0	0	1	1	1

Example 2: Training a **Hebb learning r**ule with AND function binary input and bipolar targets target, initialization weights (w_1=w_2=b=0)

Solution:

INPUT (x_1 x_2 1)	TARGET
(1 1 1)	1
(1 0 1)	−1
(0 1 1)	−1
(0 0 1)	−1

Presenting the first input, including a value of 1 for the third component, yields the following:

INPUT (x_1 x_2 1)	TARGET	WEIGHT CHANGES (Δw_1 Δw_2 Δb)	WEIGHTS (w_1 w_2 b)
			(0 0 0)
(1 1 1)	1	(1 1 1)	(1 1 1)

INPUT (x_1 x_2 1)	TARGET	WEIGHT CHANGES (Δw_1 Δw_2 Δb)	WEIGHTS (w_1 w_2 b)
(1 0 1)	−1	(−1 0 −1)	(0 1 0)
(0 1 1)	−1	(0 −1 −1)	(0 0 −1)
(0 0 1)	−1	(0 0 −1)	(0 0 −2)

However, these weights do not provide the correct response for the first input pattern.

2. Perceptron

Perceptrons had perhaps the most far-reaching impact of any of the early neural nets. The perceptron learning rule is a more powerful learning rule than the Hebb rule. Under suitable assumptions, its iterative learning procedure can be proved to converge to the correct weights, i.e., the weights that allow the net to produce the correct output value for each of the training input patterns. Not too surprisingly, one of the necessary assumptions is that such weights exist.

The activation function for each associate unit was the binary step function with an arbitrary, but fixed, threshold. Thus, the signal sent from the associator units to the output unit was a binary (0 or 1) signal. The output of the perceptron is y= f(y-in) where the activation function is

$$f(y\_in) = \begin{cases} 1 & \text{if } y\_in > \theta \\ 0 & \text{if } -0 \leq y\_in \leq \theta \\ -1 & \text{if } y\_in < -\theta \end{cases}$$

If an error occurred for a particular training input pattern, the weights would be changed according to the formula

$$w_i(\text{new}) = w_i(\text{old}) + \alpha t x_i,$$

Algorithm

Q: Explain the Algorithm of Perceptron Rule learning and draw the network configuration for this rule

Step 0. Initialize weights and bias.
(For simplicity, set weights and bias to zero.)
Set learning rate a ($0 < a \leq 1$).
(For simplicity, a can be set to 1.)

Step 1. While stopping condition is false, do Steps 2–6.

Step 2. For each training pair s:t, do Steps 3–5.

Step 3. Set activations of input units:

$$x_i = s_i.$$

Step 4. Compute response of output unit:

$$y\,in = b + \sum_i x_i w_i;$$

$$y = \begin{cases} 1 & \text{if } y\_in > \theta \\ 0 & \text{if } -\theta \leq y\_in \leq \theta \\ -1 & \text{if } y\_in < -\theta \end{cases}$$

Step 5. Update weights and bias if an error occurred for this pattern.

If $y \neq t$,

$$w_i(\text{new}) = w_i(\text{old}) + \alpha t x_i,$$

$$b(\text{new}) = b(\text{old}) + \alpha t.$$

else

$$w_i(\text{new}) = w_i(\text{old}),$$

$$b(\text{new}) = b(\text{old}).$$

Step 6. Test stopping condition:
If no weights changed in Step 2, stop; else, continue.

Application of Perceptron Net

Example 3: Training a **perceptron learning rule for AND** function with two epochs, (θ =0.2) and (a =1) , binary input, and bipolar target, initialization weights (w1=w2=b=0)

Solution:

For perceptron, AND function, binary input and bipolar target

Solution:

$$y_{in} = b + \sum_i x_i w_i;$$

$$y = \begin{cases} 1 & \text{if } y\_in > \theta \\ 0 & \text{if } -\theta \le y\_in \le \theta \\ -1 & \text{if } y\_in < -\theta \end{cases}$$

If Y≠ t

W_i(new)= W_i(old)+atX_i ;

b(new)= b(old)+ at

Else (Y= t)

W_i(new)= W_i(old)

b(new)= b(old)

X_1	X_2	1	Target
1	1	1	1
1	0	1	-1
0	1	1	-1
0	0	1	-1

The first epoch

Input			Net	OUT	Target	Weight changes			Weights		
X_1	X_2	1				Δw_1	Δw_2	Δb	W_1	W_2	b
									0	0	0
1	1	1	0	0	1	1	1	1	1	1	1
1	0	1	2	1	-1	-1	0	-1	0	1	0
0	1	1	1	1	-1	0	-1	-1	0	0	-1
0	0	1	-1	-1	-1	0	0	0	0	0	-1

The second epoch

Input			Net	OUT	Target	Weight changes			Weights		
X_1	X_2	1				Δw_1	Δw_2	Δb	W_1	W_2	b
									0	0	-1
1	1	1	-1	-1	1	1	1	1	1	1	0
1	0	1	1	1	-1	-1	0	-1	0	1	-1
0	1	1	0	0	-1	0	-1	-1	0	0	-2
0	0	1	-2	-1	-1	0	0	0	0	0	-2

Example 4: Training a **perceptron learning rule for AND** function with two epochs, (θ =0.2) and (a =1) , bipolar input, and bipolar target, initialization weights (w1=w2=b=0)

INPUT			NET	OUT	TARGET	WEIGHT CHANGES			WEIGHTS		
(x_1	x_2	1)							(w_1	w_2	b)
									(0	0	0)
(1	1	1)	0	0	1	(1	1	1)	(1	1	1)
(1	−1	1)	1	1	−1	(−1	1	−1)	(0	2	0)
(−1	1	1)	2	1	−1	(1	−1	−1)	(1	1	−1)
(−1	−1	1)	−3	−1	−1	(0	0	0)	(1	1	−1)

In the second epoch of training, we have:

(1	1	1)	1	1	1	(0	0	0)	(1	1	−1)
(1	−1	1)	−1	−1	−1	(0	0	0)	(1	1	−1)
(−1	1	1)	−1	−1	−1	(0	0	0)	(1	1	−1)
(−1	−1	1)	−3	−1	−1	(0	0	0)	(1	1	−1)

Since all the Aw's are 0 in epoch 2, the system was fully trained after the first epoch.

Supplementary Problems**:**

1. Training a Hebb learning rule with AND function binary input and target, initialization weights (w1=w2=b=0)
2. Explain the Algorithm of Hebb Rule learning and draw the network configuration for this rule
3. Training a perceptron learning rule for OR function with two epochs, (θ =0.2) and (a =1) , binary input, and bipolar target, initialization weights (w1=w2=b=0)
4. Training a perceptron learning rule for ANDNOT function with two epochs, (θ =0.2) and (a =1) , bipolar input, and bipolar target, initialization weights (w1=w2=b=0)
5. Training a An adaline for the OR_GATE bipolar input and target with the initial weights (w1=w2=b=0.1) and the learning rate (0.1)
6. Explain the Architecture and algorithm of perceptron learning rule
7. Explain the Algorithm of Perceptron Rule learning and draw the network configuration for this rule