CS772: Deep Learning for Natural Language Processing (DL-NLP)

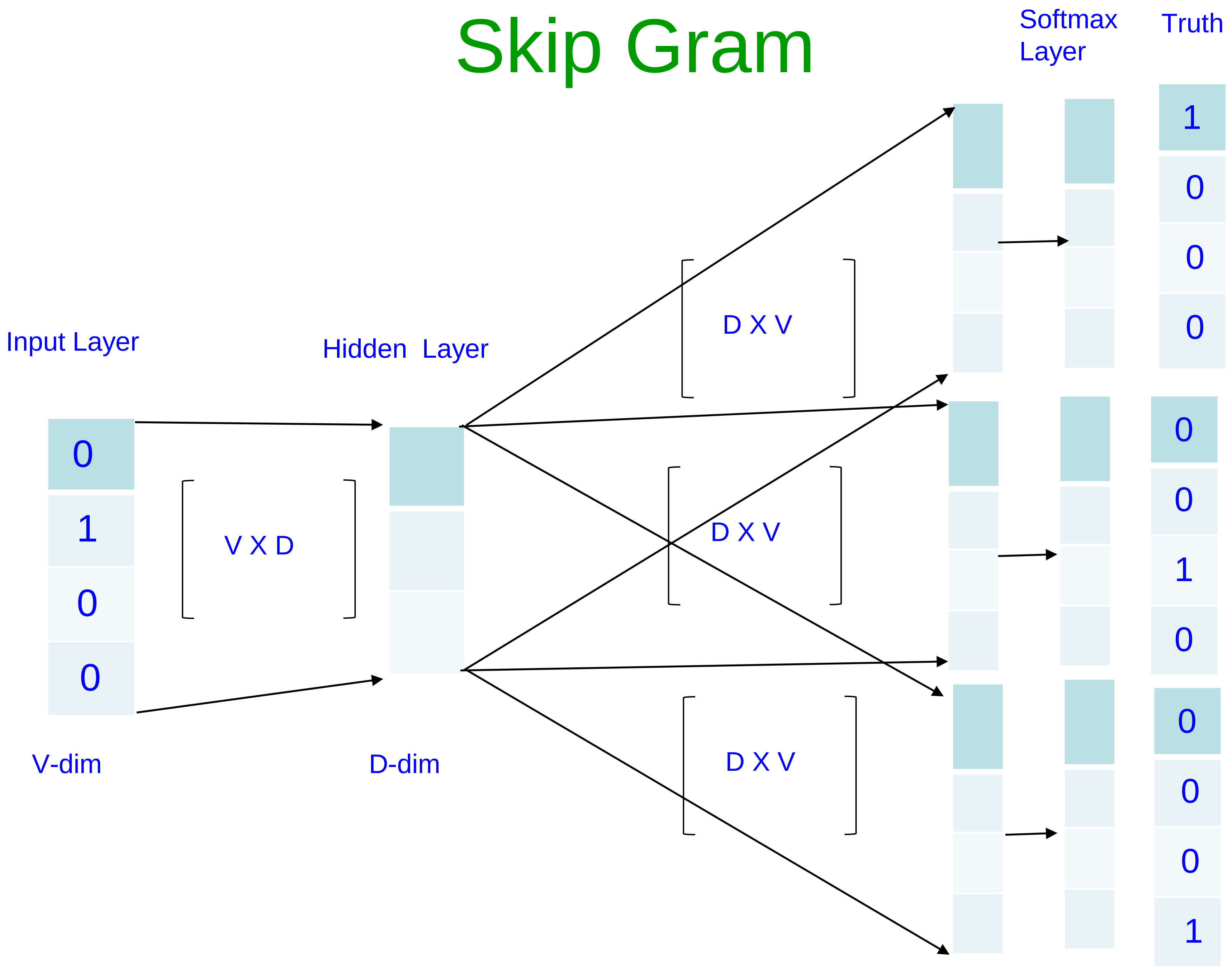
Skip Gram, Perceptron

Pushpak Bhattacharyya
Computer Science and Engineering Department
IIT Bombay

Week 3 of 17th Jan, 2022

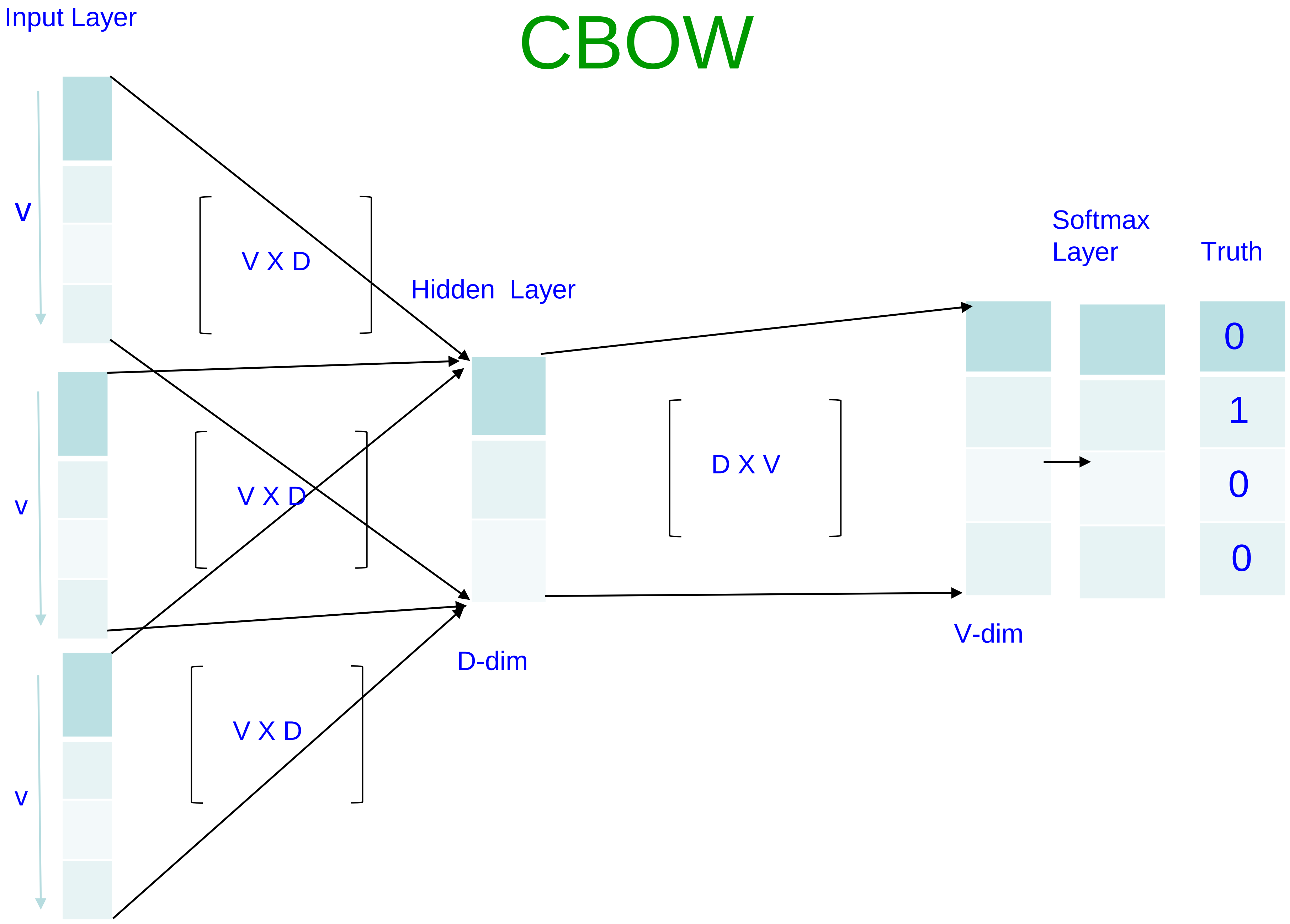
Skip Gram

Skip Gram



Softmax Layer
Truth
Input Layer
Hidden Layer
0
1
0
0
V X D
D X V
D X V
D X V
V-dim
D-dim
1
0
0
0
0
0
1
0
0
0
0
1

CBOW


Input Layer
V
V X D
v
V X D
v
V X D
Hidden Layer
D-dim
D X V
Softmax Layer
Truth
0
1
0
0
V-dim

Learning objective (skip gram)

Center word whose Representation is to be learnt

Parameters To be learnt

$$J'(\theta) = \frac{1}{T}\prod_{t=1}^{T}\prod_{\substack{-m\le j\le m\\ j\ne 0}} p(w_{t+j} \mid w_t;\theta)$$

Context

$$J(\theta) = -\frac{1}{T}\prod_{t=1}^{T}\prod_{\substack{-m\le j\le m\\ j\ne 0}} p(w_{t+j} \mid w_t;\theta)$$

$$Minimize \quad L = -\sum_{t=1}^{T}\sum_{\substack{-m\le j\le m\\ j\ne 0}} \log[p(w_{t+j} \mid w_t;\theta)]$$

Modelling *P(context word|input word)* *(1/2)*

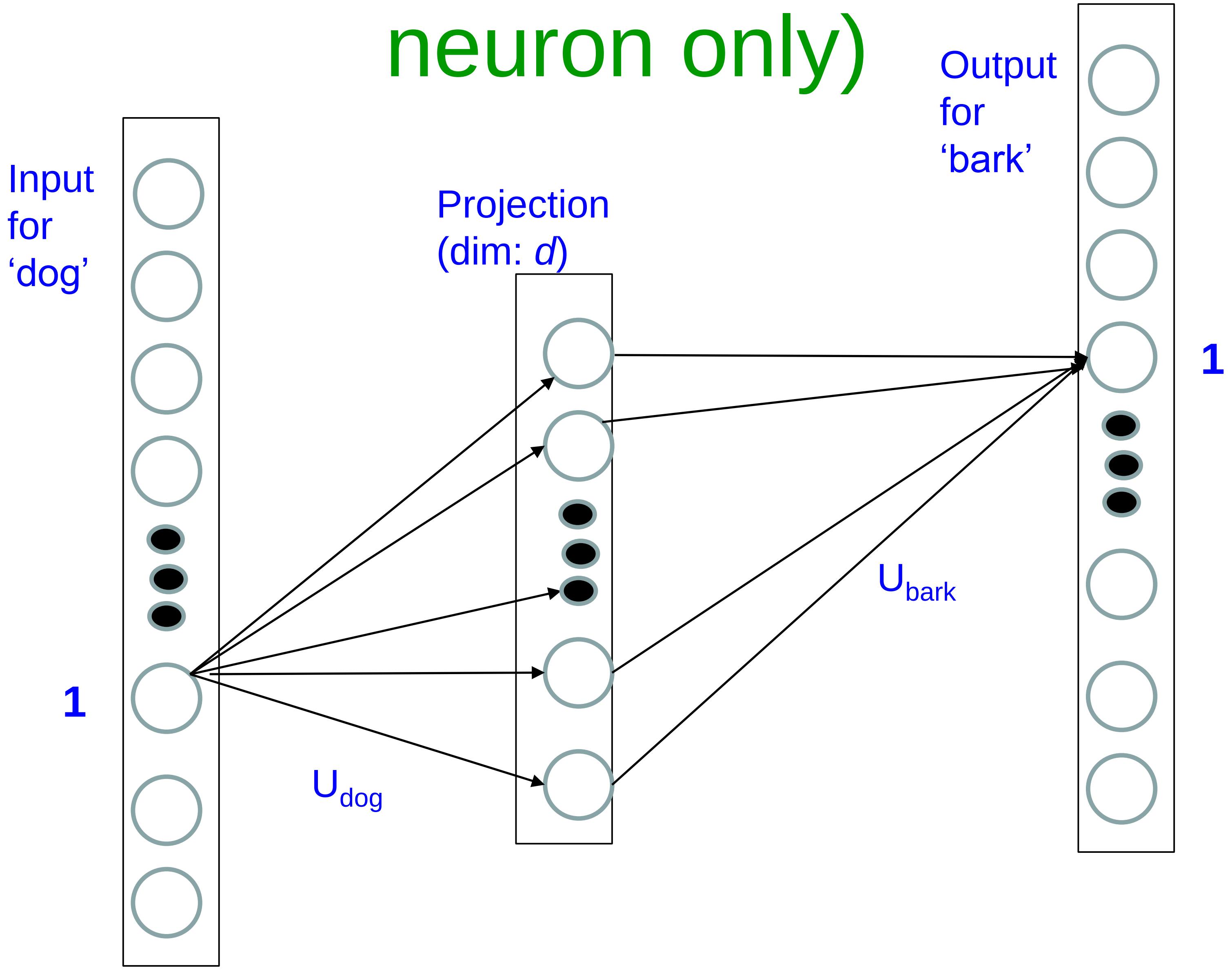
- We want, say, *P('bark'|'dog')*
- Take the weight vector **FROM** 'dog' neuron **TO** projection layer (call this U_{dog})
- Take the weight vector **TO** 'bark' neuron **FROM** projection layer (call this U_{bark})
- When initialized, U_{dog} and U_{bark} give the initial estimates of word vectors of 'dog' and 'bark'
- The weights and therefore the word vectors get fixed by back propagation

Modelling *P(context word|input word)* *(2/2)*

- To model the probability, first compute dot product of u_{dog} and v_{bark}
- Exponentiate the dot product
- Take softmax over all dot products over the whole vocabulary

$$P('bark'|'dog') = \frac{\exp(U_{dog}^{T} U_{bark})}{\sum_{R \varepsilon Vocabulary} \exp(U_{dog}^{T} U_{R})}$$

Input to Projection (shown for one neuron only)



P('bark'|'dog') (1/2)

$$P('bark'|'dog') = \frac{\exp(U_{dog}^{T} U_{bark})}{\sum_{R \varepsilon Vocabulary} \exp(U_{dog}^{T} U_{R})}$$

$$\log(P('bark'|'dog')) = U_{dog}^{T} U_{bark} - \log\left(\sum_{R \varepsilon Vocabulary} \exp(U_{dog}^{T} U_{R})\right)$$

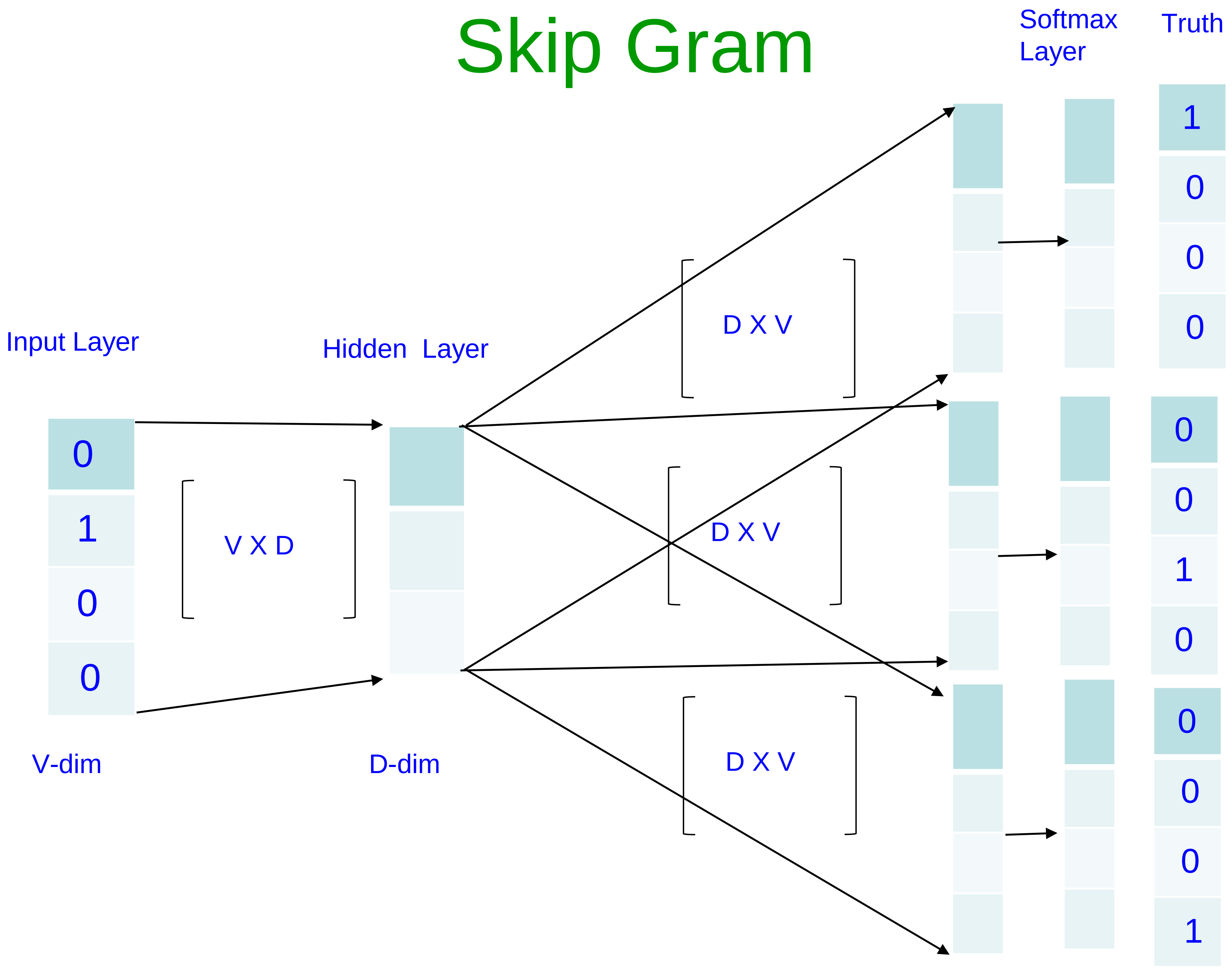
P('bark'|'dog') (2/2)

Let u^k_{dog} be the jth component of the weight vector from the '1' neuron of input to the projection layer

$$U^T_{dog}U_{bark} = (u^1_{dog}u^1_{bark} + u^2_{dog}u^2_{bark} + \ldots + u^D_{dog}u^D_{bark})$$
$$= \sum_{k=1,D} u^k_{dog}u^k_{bark}$$

$$\log(P('bark'|'dog'))$$
$$= \sum_{k=1,D} u^k_{dog}u^k_{bark} - \log(\sum_{R \in vocab} \exp(\sum_{k=1,D} u^k_{dog}u^k_R))$$

Skip Gram



Softmax Layer
Truth
Input Layer
Hidden Layer
0
1
0
0
V X D
D X V
D X V
D X V
V-dim
D-dim
1
0
0
0
0
0
1
0
0
0
0
1

Back to Loss Function (skip gram)

word

$$Minimize \quad L = -\sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \ne 0}} \log[p(w_{t+j} \mid w_t; \theta)]$$

$$L = -\sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \ne 0}} \left[\sum_{k=1,D} u_t^k u_{t+j}^k - \log\left(\sum_{R \in vocab} \exp\left(\sum_{k=1,D} u_t^k u_R^k \right)\right) \right]$$

t goes over the whole corpus,
j goes over the context words
k goes over the weight vector

Apply Gradient Descent

Change of weight is proportional to negative gradient of Loss wrt to that particular weight

$$\Delta u_t^j \propto -\frac{\partial L}{\partial u_t^j}$$

$$L = -\sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \ne 0}} \left[\sum_{k=1,D} u_t^k u_{t+j}^k - \log\left(\sum_{R \in vocab} \exp\left(\sum_{k=1,D} u_t^k u_R^k \right)\right) \right]$$

Exercise

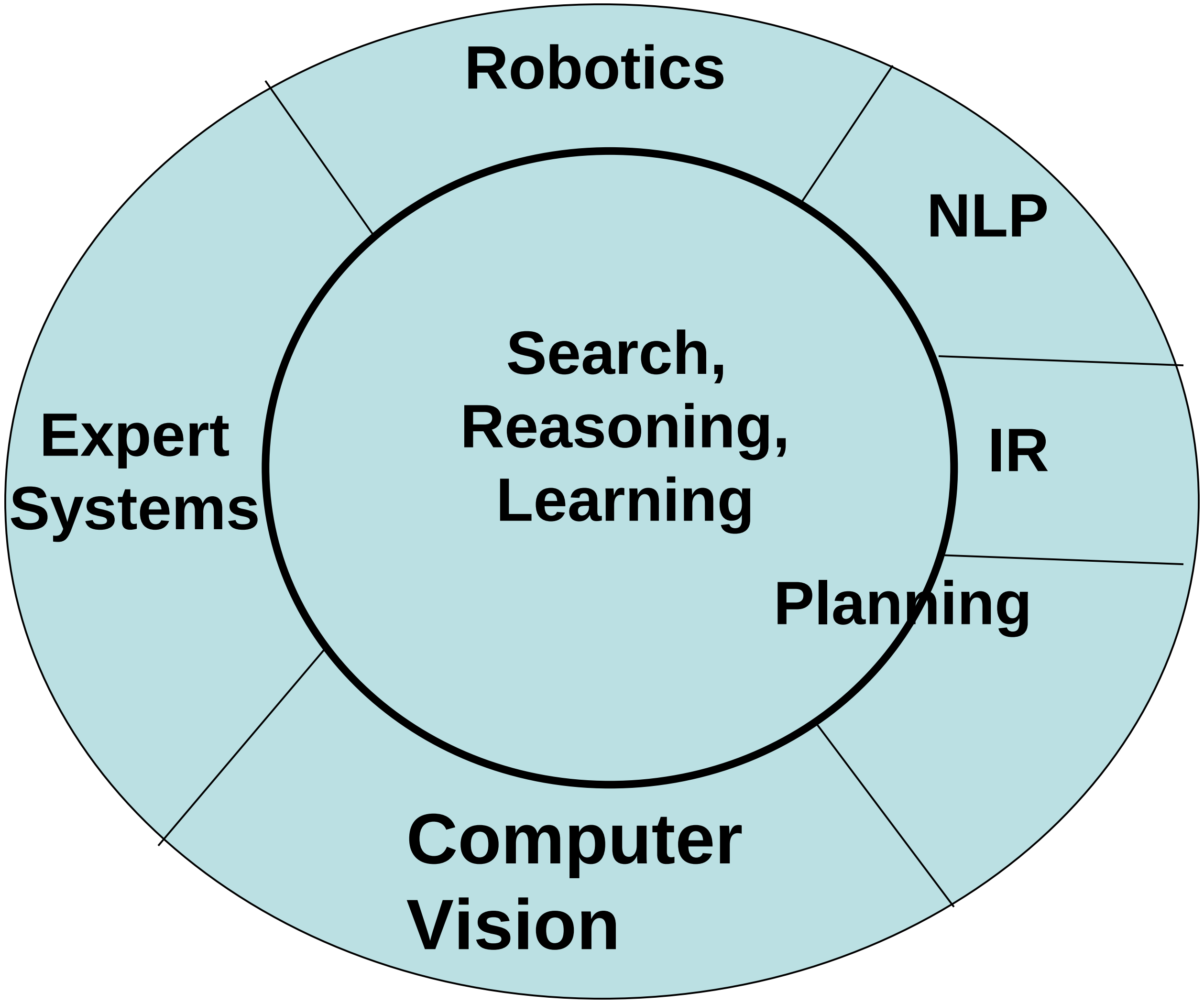
- Derive the weight change rule for Skip Gram

Assignment

- Implement skip gram and study word vectors; corpus and the exact statement for the assignment will be specified.

Neurons

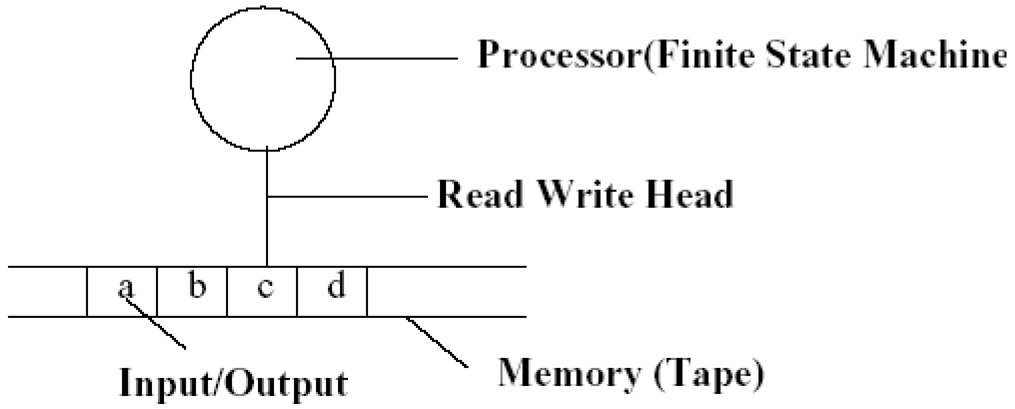
AI Perspective (post-web)



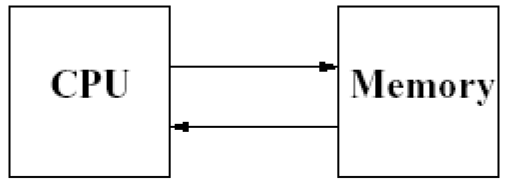
Symbolic AI

- Connectionist AI is contrasted with Symbolic AI
- Symbolic AI - Physical Symbol System Hypothesis
- Every intelligent system can be constructed by storing and processing symbols and nothing more is necessary.
- Symbolic AI has a bearing on models of computation such as

Turing Machine & Von Neumann



Turing machine



VonNeumann Machine

Challenges to Symbolic AI

- Motivation for challenging Symbolic AI
- A large number of computations and information process tasks that living beings are comfortable with, are not performed well by computers!
- The Differences

• Brain computation in living beings	TM computation in computers
• Pattern Recognition	Numerical Processing
• Learning oriented	Programming oriented
• Distributed & parallel processing processing	Centralized & serial
• Content addressable	Location addressable

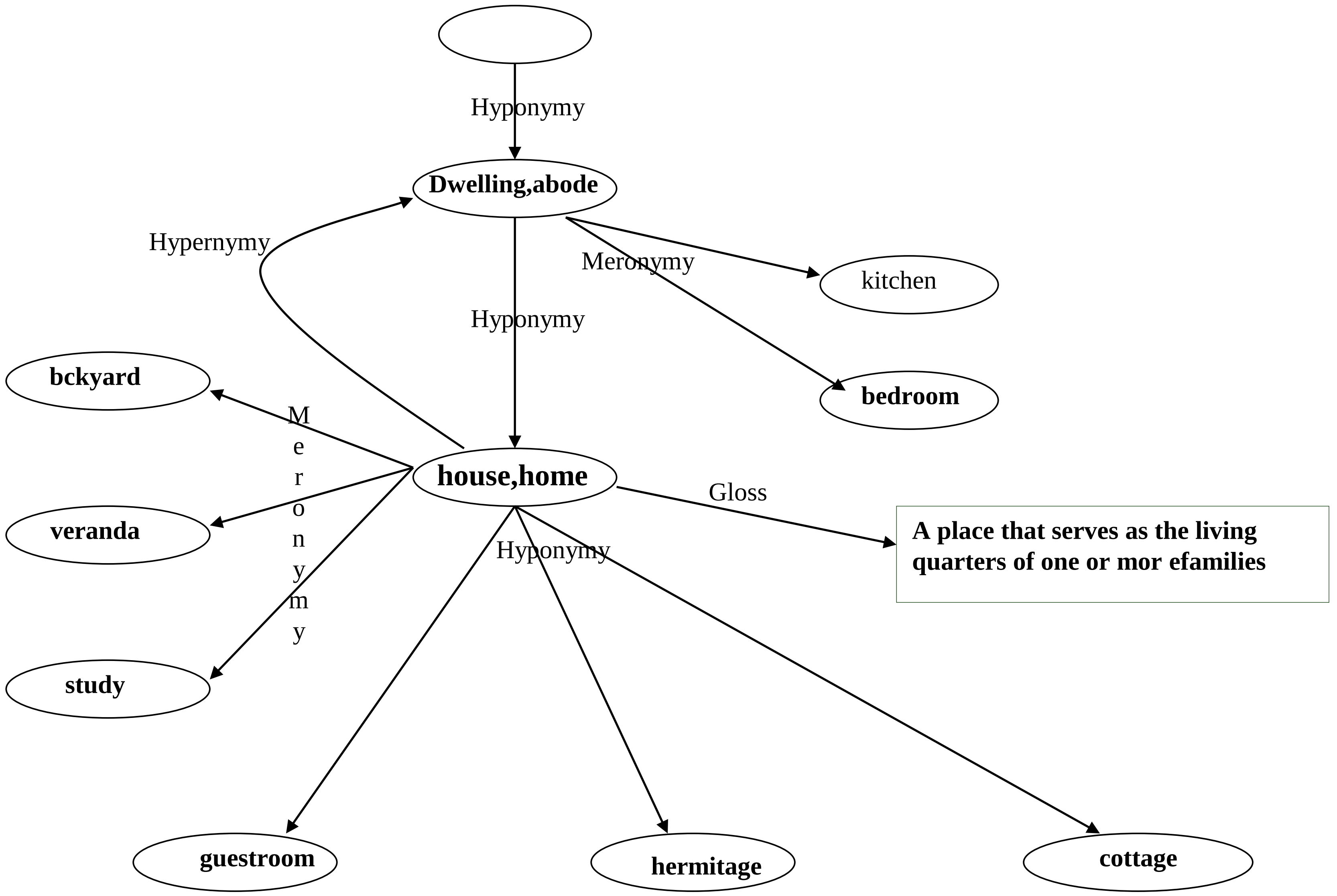
Two aims of advancing computers

- More speed- pushes frontiers in hardware, architecture, systems, programming languages
- More Intelligence- pushes frontiers in endowing computers with human like abilities, e.g., language processing
- Synergistic aims: faster helps in taking up more complex tasks; more complex tasks demand faster machines

Symbolic and connectionist representation of words

- A snapshot of wordnet subgraph is a symbolic representation of words
- A word vector on the other hand is a connectionist representation of words

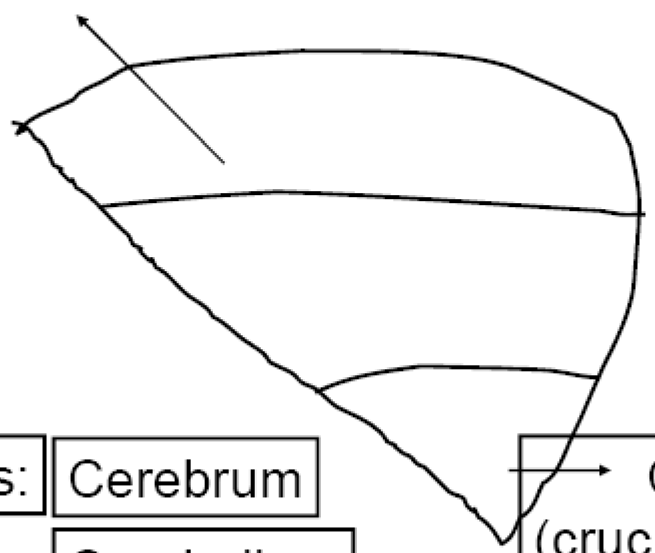
WordNet Sub-Graph



A place that serves as the living quarters of one or mor efamilies

- The human brain

- Seat of consciousness and cognition
- Perhaps the most complex information processing machine in nature

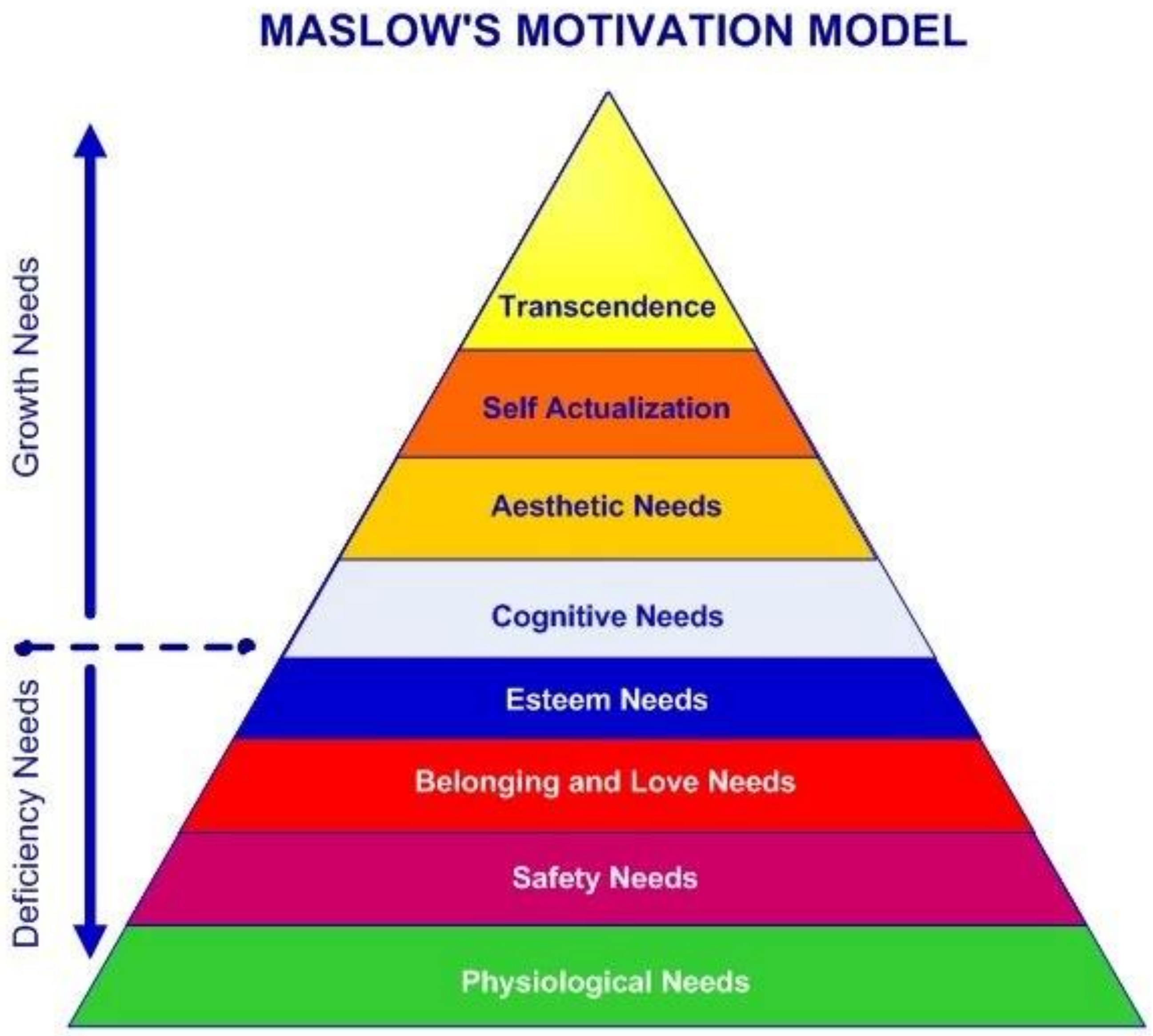


3- Layers: Cerebrum

Cerebellum

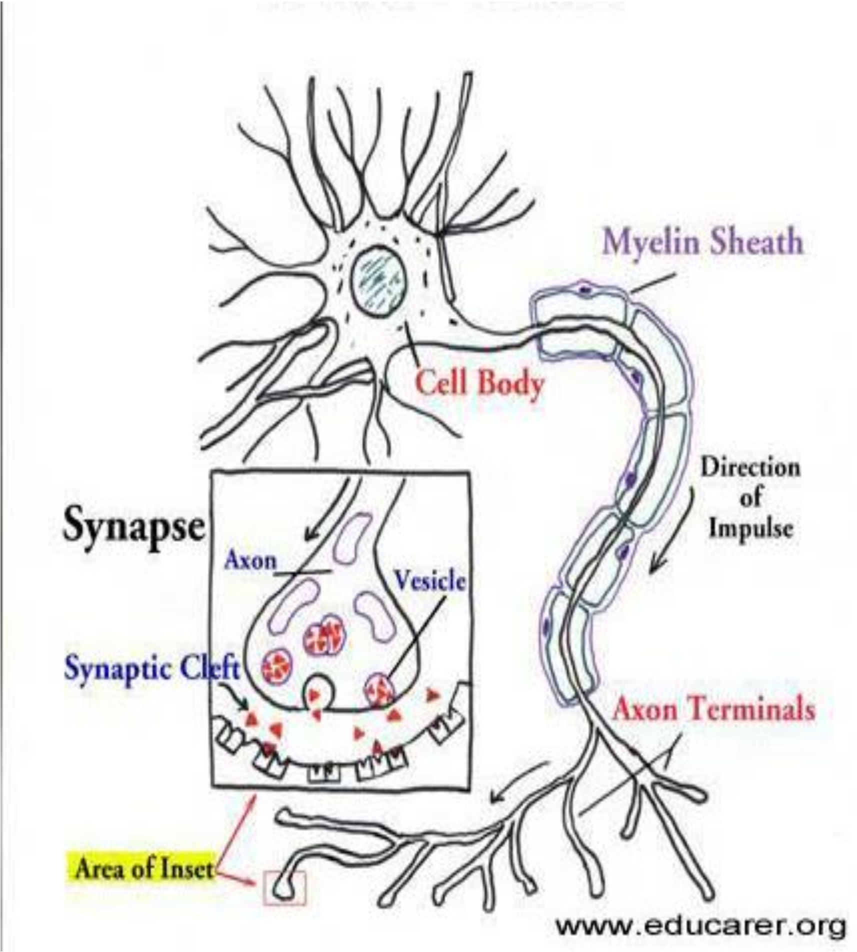
Higher brain

Maslow's Hierarchy



Neuron - “classical”

- Dendrites
  - Receiving stations of neurons
  - Don't generate action potentials
- Cell body
  - Site at which information received is integrated
- Axon
  - Generate and relay action potential
  - Terminal
    - Relays information to next neuron in the pathway

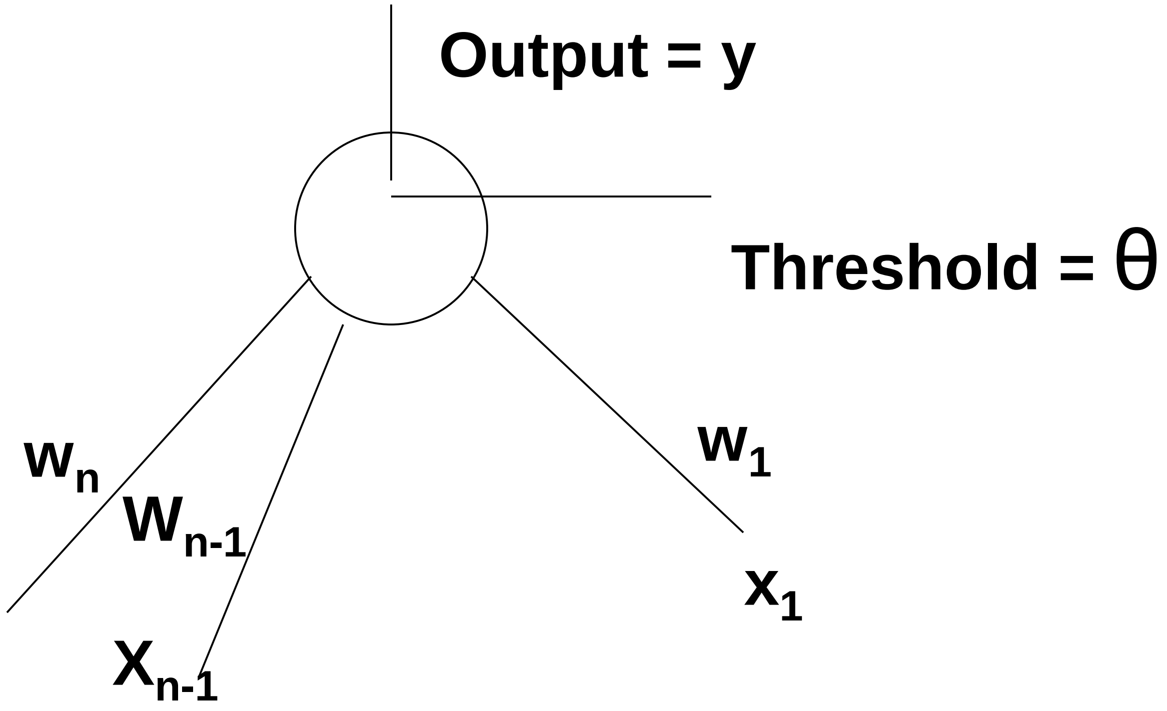


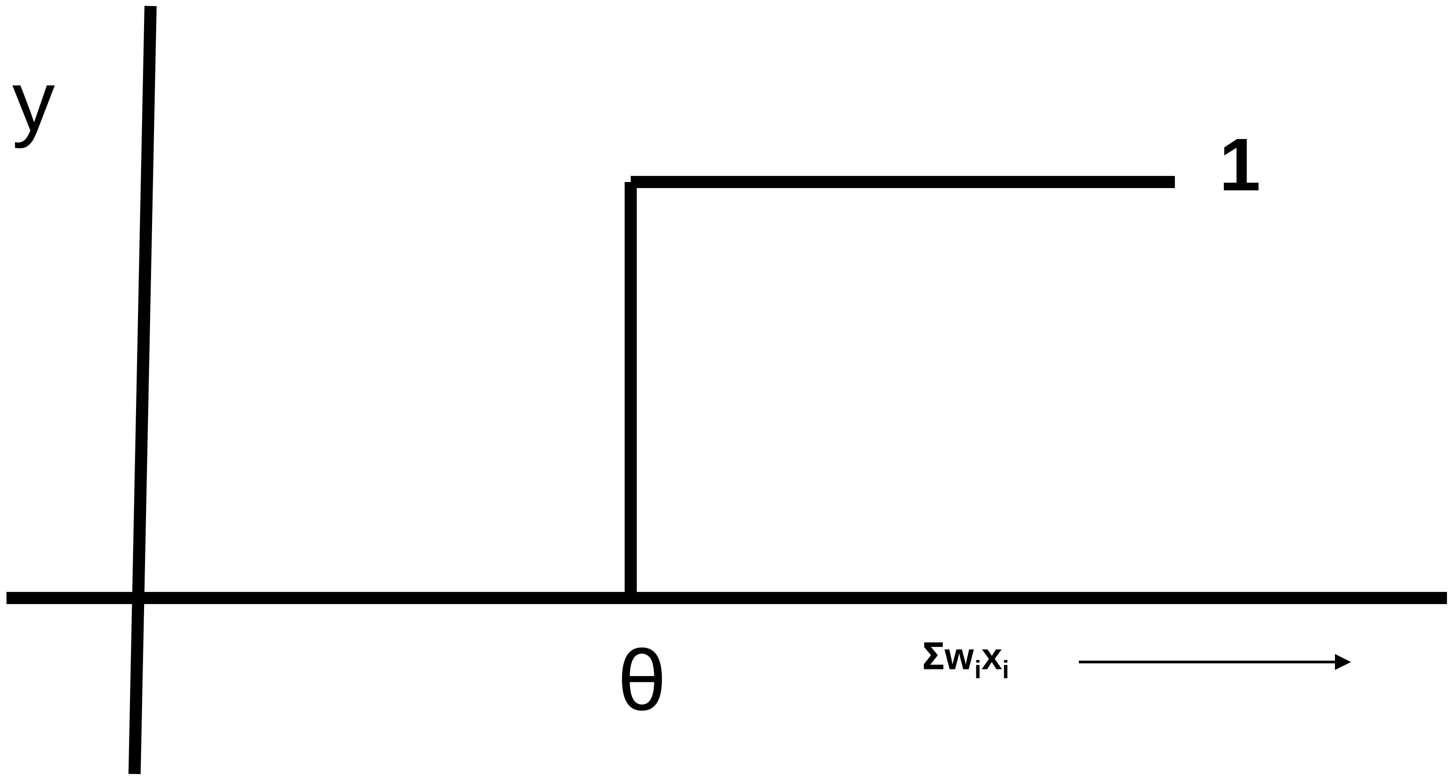
http://www.educarer.com/images/brain-nerve-axon.jpg

Perceptron

The Perceptron Model

A perceptron is a computing element with input lines having associated weights and the cell having a threshold value. The perceptron model is motivated by the biological neuron.





Step function / Threshold function

y = 1 for Σwixi >=θ

=0 otherwise

Features of Perceptron

- **Input output behavior is discontinuous and the derivative does not exist at $\Sigma w_i x_i = \theta$**

- **$\Sigma w_i x_i - \theta$ is the net input denoted as net**

- **Referred to as a linear threshold element - linearity because of x appearing with power 1**

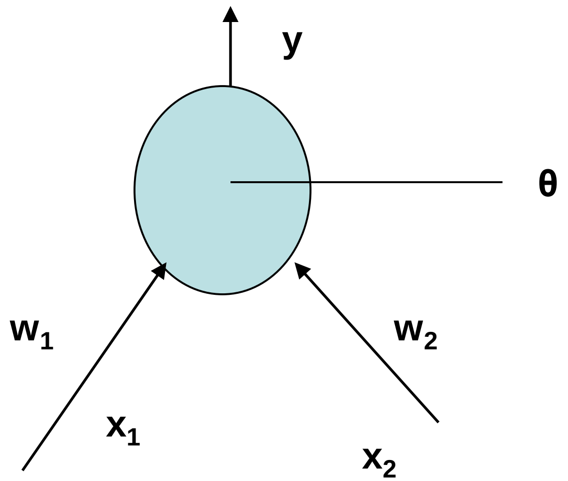
- **y= f(net): Relation between y and net is non-linear**

Computation of Boolean functions

AND of 2 inputs

X1	**x2**	**y**
0	0	0
0	1	0
1	0	0
1	1	1

The parameter values (weights & thresholds) need to be found.



Computing parameter values

$w1 * 0 + w2 * 0 <= \theta \rightarrow \theta >= 0$; since $y=0$

$w1 * 0 + w2 * 1 <= \theta \rightarrow w2 <= \theta$; since $y=0$

$w1 * 1 + w2 * 0 <= \theta \rightarrow w1 <= \theta$; since $y=0$

$w1 * 1 + w2 *1 > \theta \rightarrow w1 + w2 > \theta$; since $y=1$
$w1 = w2 = \quad = 0.5$

satisfy these inequalities and find parameters to be used for computing AND function.

Other Boolean functions

- **OR can be computed using values of w1 = w2 = 1 and = 0.5**
- **XOR function gives rise to the following inequalities:**

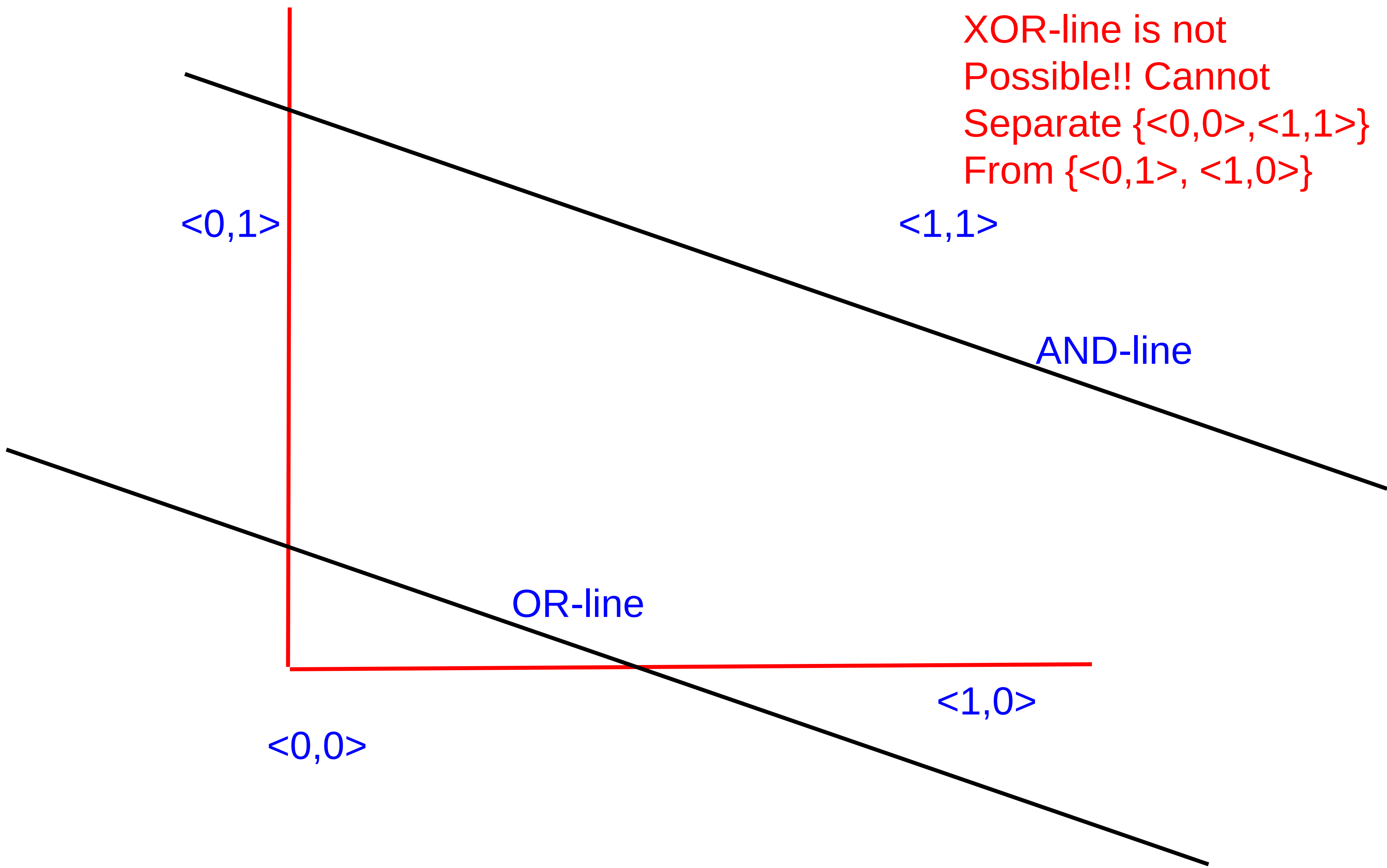
$w1 * 0 + w2 * 0 <= \theta \Rightarrow \theta >= 0$

$w1 * 0 + w2 * 1 > \theta \Rightarrow w2 > \theta$

$w1 * 1 + w2 * 0 > \theta \Rightarrow w1 > \theta$

$w1 * 1 + w2 * 1 <= \theta \Rightarrow w1 + w2 <= \theta$

No set of parameter values satisfy these inequalities.


XOR-line is not Possible!! Cannot Separate {<0,0>,<1,1>} From {<0,1>, <1,0>}
<0,1>
<1,1>
AND-line
OR-line
<1,0>
<0,0>

Threshold functions

n	# Boolean functions (2^2^n)	#Threshold Functions (2^{n^2})
1	4	4
2	16	14
3	256	128
4	64K	1008

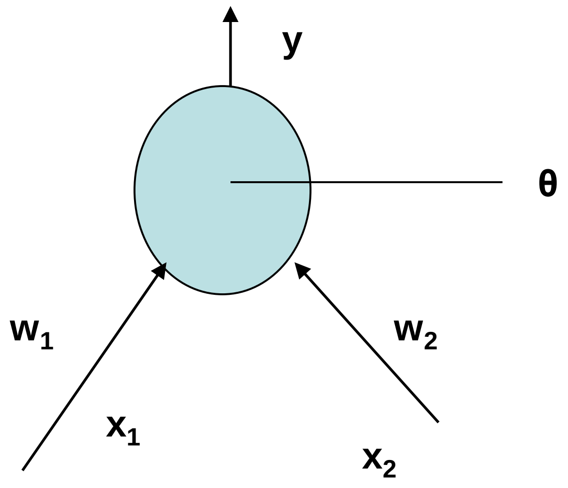
- **Functions computable by perceptrons - threshold functions**
- **#TF becomes negligibly small for larger values of #BF.**
- **For n=2, all functions except XOR and XNOR are computable.**

Muroga.S, Threshold Logic and its Applications, John Wiley, 1972

AND of 2 inputs

X1	x2	y
0	0	0
0	1	0
1	0	0
1	1	1

The parameter values (weights & thresholds) need to be found.



Constraints on w1, w2 and θ

$w1 * 0 + w2 * 0 <= \theta \Rightarrow \theta >= 0$; since $y=0$

$w1 * 0 + w2 * 1 <= \theta \Rightarrow w2 <= \theta$; since $y=0$

$w1 * 1 + w2 * 0 <= \theta \Rightarrow w1 <= \theta$; since $y=0$

$w1 * 1 + w2 *1 > \theta \Rightarrow w1 + w2 > \theta$; since $y=1$

$w1 = w2 = \ = 0.5$

These inequalities are satisfied by ONE particular region

Perceptron training

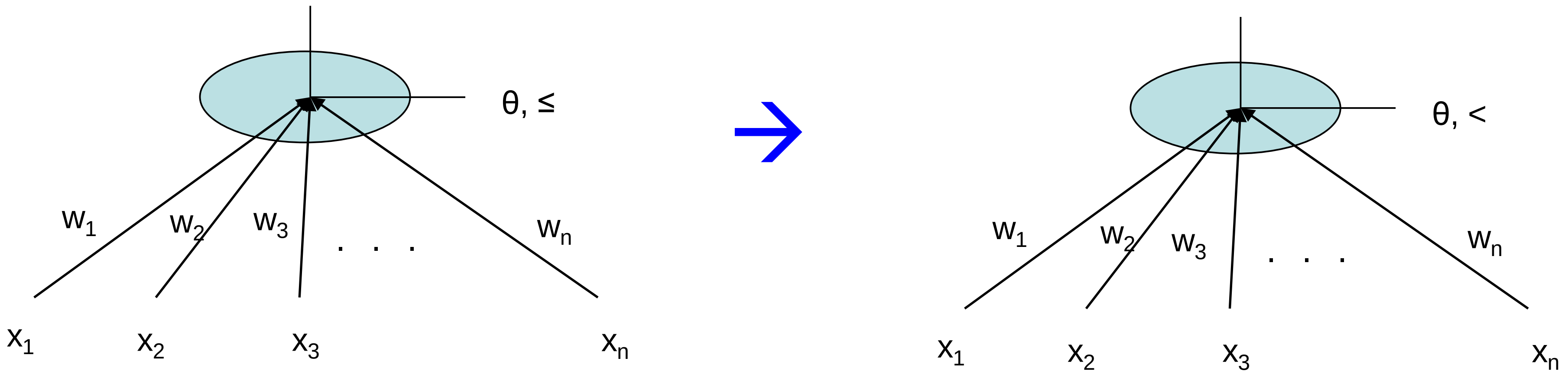
Perceptron Training Algorithm (PTA)

Preprocessing:

1. The computation law is modified to

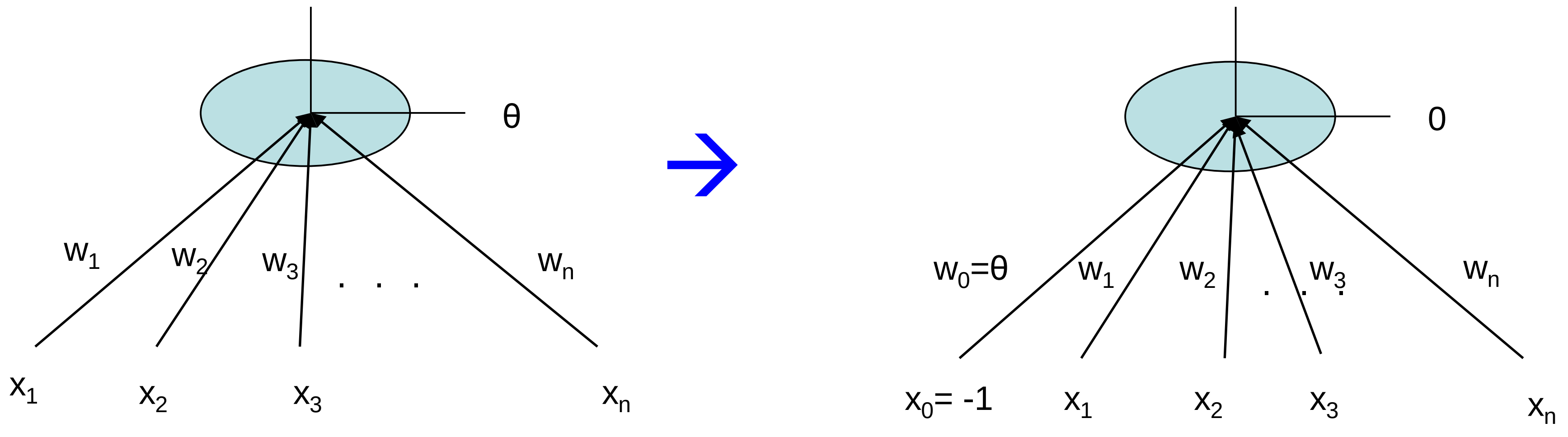
$$y = 1 \text{ if } \sum w_i x_i > \theta$$

$$y = o \text{ if } \sum w_i x_i < \theta$$



PTA – preprocessing cont…

2. Absorb θ as a weight. Comparing W.X with θ is eqv to comparing (W.X- θ) with 0



3. Negate all the zero-class examples

Example to demonstrate preprocessing

- **OR perceptron**

1-class	<1,1> , <1,0> , <0,1>
0-class	<0,0>

Augmented x vectors:-

1-class	<-1,1,1> , <-1,1,0> , <-1,0,1>
0-class	<-1,0,0>

Negate 0-class:- <1,0,0>

Example to demonstrate preprocessing cont..

Now the vectors are

$$\begin{array}{cccc} & x_0 & x_1 & x_2 \\ X_1 & -1 & 0 & 1 \\ X_2 & -1 & 1 & 0 \\ X_3 & -1 & 1 & 1 \\ X_4 & 1 & 0 & 0 \end{array}$$

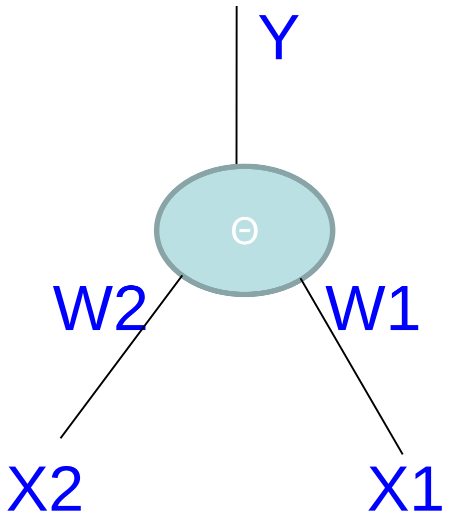
Perceptron Training Algorithm

1. Start with a random value of w
   ex: <0,0,0…>
2. Test for $wx_i > 0$
   If the test succeeds for i=1,2,…n
   then return w
3. Modify w, $w_{next} = w_{prev} + x_{fail}$

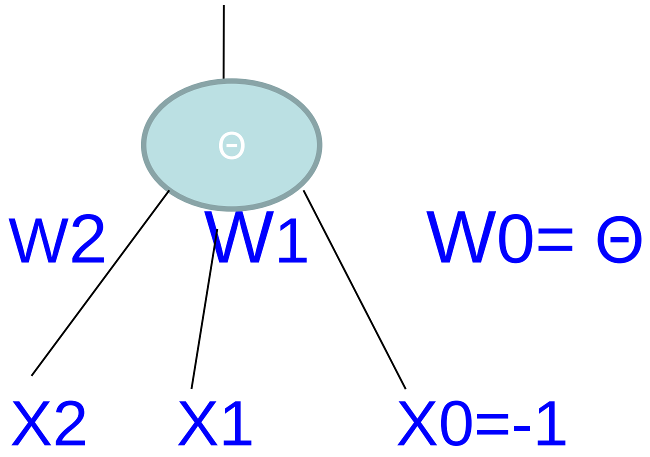
PTA on NAND

NAND:

X2	X1	Y
0	0	1
0	1	1
1	0	1
1	1	0



Converted To



Preprocessing

NAND Augmented:

X2	X1	X0	Y
0	0	-1	1
0	1	-1	1
1	0	-1	1
1	1	-1	0

NAND-0 class Negated

	X2	X1	X0
V0:	0	0	-1
V1:	0	1	-1
V2:	1	0	-1
V3:	-1	-1	1

Vectors for which W=<W2 W1 W0> has to be found such that W. Vi > 0

PTA Algo steps

Algorithm:

1. Initialize and Keep adding the failed vectors until W. Vi > 0 is true.

Step 0: $W = \langle 0, 0, 0 \rangle$

$W_1 = \langle 0, 0, 0 \rangle + \langle 0, 0, -1 \rangle$ {V_0 Fails}

$= \langle 0, 0, -1 \rangle$

$W_2 = \langle 0, 0, -1 \rangle + \langle -1, -1, 1 \rangle$ {V_3 Fails}

$= \langle -1, -1, 0 \rangle$

$W_3 = \langle -1, -1, 0 \rangle + \langle 0, 0, -1 \rangle$ {V_0 Fails}

$= \langle -1, -1, -1 \rangle$

$W_4 = \langle -1, -1, -1 \rangle + \langle 0, 1, -1 \rangle$ {V_1 Fails}

$= \langle -1, 0, -2 \rangle$

Trying convergence

$W_5 = \langle -1, 0, -2\rangle + \langle -1, -1, 1\rangle$ {V_3 Fails}

$= \langle -2, -1, -1\rangle$

$W_6 = \langle -2, -1, -1\rangle + \langle 0, 1, -1\rangle$ {V_1 Fails}

$= \langle -2, 0, -2\rangle$

$W_7 = \langle -2, 0, -2\rangle + \langle 1, 0, -1\rangle$ {V_0 Fails}

$= \langle -1, 0, -3\rangle$

$W_8 = \langle -1, 0, -3\rangle + \langle -1, -1, 1\rangle$ {V_3 Fails}

$= \langle -2, -1, -2\rangle$

$W_9 = \langle -2, -1, -2\rangle + \langle 1, 0, -1\rangle$ {V_2 Fails}

$= \langle -1, -1, -3\rangle$

Trying convergence

$W_{10} = <-1, -1, -3> + <-1, -1, 1>$ {V_3 Fails}

$= <-2, -2, -2>$

$W_{11} = <-2, -2, -2> + <0, 1, -1>$ {V_1 Fails}

$= <-2, -1, -3>$

$W_{12} = <-2, -1, -3> + <-1, -1, 1>$ {V_3 Fails}

$= <-3, -2, -2>$

$W_{13} = <-3, -2, -2> + <0, 1, -1>$ {V_1 Fails}

$= <-3, -1, -3>$

$W_{14} = <-3, -1, -3> + <0, 1, -1>$ {V_2 Fails}

$= <-2, -1, -4>$

W15 = <-2, -1, -4> + <-1, -1, 1> {V3 Fails}

= <-3, -2, -3>

W16 = <-3, -2, -3> + <1, 0, -1> {V2 Fails}

= <-2, -2, -4>

W17 = <-2, -2, -4> + <-1, -1, 1> {V3 Fails}

= <-3, -3, -3>

W18 = <-3, -3, -3> + <0, 1, -1> {V1 Fails}

= <-3, -2, -4>

W2 = -3, W1 = -2, W0 = Θ = -4

Succeeds for all vectors

Where probability can come in (1/2)

- (1) in initialization: if we are lucky to start close to the correct weight, then the number of iterations will be small
- Question- is it possible to have this kind of serendipitous initialization
- Answer: yes sometimes, if we have domain knowledge and exploit this knowledge along with task properties

Where probability can come in (2/2)

- (2) choosing the correct test vector: if we go sequentially choosing the vectors to test for W.X>0, then the failed vector may come towards the end; this can slow down the process if the input size is large
- Question: can we test only a sample of the input examples?
- Answer: yes, this is done for example in stochastic gradient descent algo (to be discussed later)

Assignment on skip gram

Steps (1/2)

- 1. Create a cluster of “animal words”: *cow, dog, bullock, cat* … (10 words)
- 2. Create a cluster of “bird words”: cuckoo, parrot, crow, sparrow,… (10 words)
- 3. Run a **concordance** for obtaining the neighboring words of these words (learn what a concordancer is)

Steps (2/2)

- *(These animal words, bird words and concordance supplied syntagmatic words form the universe of your words)*
- 4. Train a skip gram model with these words
- 5. Collect the word representations
- 6. Ensure that “animal” words are close to another words and so are “bird” words; Inter cluster distance should be large compared to intra cluster distance; use cosine similarity

IMP: skip gram code has to be your OWN

PTA convergence

Statement of Convergence of PTA

- Statement:

  Whatever be the initial choice of weights and whatever be the vector chosen for testing, PTA converges if the vectors are from a linearly separable function.

Proof of Convergence of PTA

- Suppose W_n is the weight vector at the n^{th} step of the algorithm.
- At the beginning, the weight vector is w_0
- Go from W_i to W_{i+1} when a vector X_j fails the test $w_i X_j > 0$ and update w_i as

$$W_{i+1} = W_i + X_j$$

- Since *Xj*s form a linearly separable function,

For all, W,* $W^* X_j > 0$ for all j

Proof of Convergence of PTA (cntd.)

(for notational simplicity we will not use transpose of vectors)

Consider the expression

$$G(W_n) = \frac{W_n . W^*}{|W_n|}$$

$$= \frac{|W_n| . |W^*| \cos\theta}{|W_n|}$$

$$= |W^*| \cos\theta$$

$|G(w_n)| \leq |w^*|$, as $-1 \leq \cos\theta \leq 1$

Behavior of Numerator of *G*(*W*$_n$)

$$W_n.W^* = (W_{n-1} + X_{fail}^{n-1}).W^*$$

$$= W_{n-1}.W^* + X_{fail}^{n-1}.W^*$$

$$= (W_{n-2} + X_{fail}^{n-2}).W^* + X_{fail}^{n-1}.W^*$$

$$= \ldots$$

$$= W_0.W^* + X_{fail}^{0}.W^* + X_{fail}^{1}.W^* + \ldots X_{fail}^{n-1}.W^*$$

$$= K_1 + \delta_0 + \delta_1 + \ldots.\delta_{n-1}$$

$$\geq K_1 + n\delta_{\min}, \quad \forall i,\ \delta_i > 0$$

Since W* is the separating vector, its dot product with any X_j is >0

So, numerator of *G* grows with n.

Behavior of Denominator of *G(W_n)*

$$|W_n| = \sqrt{W_n^T . W_n}$$

$$= \sqrt{(W_{n-1} + X_{fail}^{n-1})^T (W_{n-1} + X_{fail}^{n-1})}$$

$$= \sqrt{W_{n-1}^2 + 2.W_{n-1}.X_{fail}^{n-1} + (X_{fail}^{n-1})^2}$$

Dot product of weight Vector with failed vector must be non positive

$$\leq \sqrt{W_{n-1}^2 + (X_{fail}^{n-1})^2}$$

$$= \sqrt{W_0^2 + (X_{fail}^0)^2 + (X_{fail}^1)^2 + \ldots (X_{fail}^{n-1})^2}$$

$$\leq \sqrt{K_2 + n\alpha_{\max}^2}$$

$|X_j| \leq \alpha_{max}$ (max magnitude); So, Denom grows as *sqrt(n)*

Some Observations

- Numerator of *G* grows as n
- Denominator of *G* grows as *sqrt(n)*

  => Numerator grows faster than denominator
- If PTA does not terminate, $G(w_n)$ values will become unbounded.

Some Observations contd.

- But, as $|G(w_n)| \leq |w^*|$ which is finite, n becoming infinite is impossible!
- Hence, PTA has to converge.
- Proof is due to Marvin Minsky.

A Problem that can be solved using the proof of PTA

Problem: *If a weight repeats while training the perceptron, then the function is not linearly separable.*

Proof

Let us prove first $w_n . w^*$ is an increasing function.

From the proof of convergence of PTA, we can write

$$w_n . w^* = (w_{n-1} + X^{n-1}_{fail}) . w^*$$
$$= w_{n-1} . w^* + w^* . X^{n-1}_{fail}$$

Since w^* is optimal weight vector therefore:

$$w^* . X^{n-1}_{fail} > 0$$

Proof cntd.

Because in every iteration we are adding +ve number $W^*. X^{n-1}_{fail}$

Therefore:

$$W_n .W^* > W_{n-1} .W^* \qquad (1)$$

Hence $W_n .W^*$ is an increasing function.

If the weight repeats then the weight W_i at a given iteration no. i, will be equal to the weight W_{i+k} at the iteration no. $(i+k)$ where k is a +ve number. So

$$W_{i=} W_{i+k}$$

Proof cntd.

Therefore:

$$W_i . W^* = W_{i+k} . W^* \qquad (2)$$

(2) contradicts the (1)

Hence no W^* exists

So function is not linearly separable.

Some plan for the course: going forward

Plan

- Task Front
  - Language Model
  - Build up to skip gram/cbow
    - Auto-encoder, predicting the next word, predicting context words
- Technique front
  - Perceptron
  - Feedforward NN with backpropagation
  - Recurrent n/w
  - Masked Models