CS626: Speech, NLP and the Web

Start of Neural Network

Pushpak Bhattacharyya
Computer Science and Engineering Department
IIT Bombay

Week of 12th October, 2020

Task *vs.* Technique Matrix

Task (row) vs. Technique (col) Matrix	**Rules Based/Knowledge-Based**	**Classical ML**				**Deep Learning**		
		Perceptron	Logistic Regression	SVM	Graphical Models (HMM, MEMM, CRF)	Dense FF with BP and softmax	RNN-LSTM	CNN
Morphology								
POS								
Chunking								
Parsing								
NER, MWE								
Coref								
WSD								
Machine Translation								
Semantic Role Labeling								
Sentiment								
Question Answering								

Agenda for the week

- Introduction to Neural Network as a framework for “deep learning”
- Perceptron and Feedforward N/W
- Recurrent N/W
- NLP and Neural Net

Stages of development

- Perceptron
- Feedforward Neural N/W
- (in parallel with FFNN) Recurrent Neural Nets
- Multilayer recurrent n/w: Self Organization, Neocognitron
- (recent) LSTM, Bi-LSTM, GRU
- (recent) FFNN with softmax
- After RNN) **Transformers**
  - ***Main difference with RNN, data need not be in sequential order!***

https://huggingface.co/transformers/ (1/6)

- The library currently contains PyTorch and Tensorflow implementations, pre-trained model weights, usage scripts and conversion utilities for the following models:
- BERT (from Google) released with the paper BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding by Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova.
- GPT (from OpenAI), Improving Language Understanding by Generative Pre-Training by Radford et al.
- GPT-2 (from OpenAI), Language Models are Unsupervised Multitask Learners by Radford et al.
- Transformer-XL (from Google/CMU), released with the paper Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context by Zihang Dai et al.

Huggingface cntd. (2/6)

- XLNet (from Google/CMU), XLNet: Generalized Autoregressive Pretraining for Language Understanding by Zhilin Yang et al.
- XLM (from Facebook), Cross-lingual Language Model Pretraining by Guillaume Lample and Alexis Conneau.
- RoBERTa (from Facebook), Robustly Optimized BERT Pretraining Approach by Yinhan Liu et al.
- DistilBERT (from HuggingFace), DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter by Victor Sanh et al. The same method has been applied to compress GPT2 into DistilGPT2.
- CTRL (from Salesforce), CTRL: A Conditional Transformer Language Model for Controllable Generation by Keskar et al.

Huggingface (3/6)

- CamemBERT (from FAIR, Inria, Sorbonne Université), CamemBERT: a Tasty French Language Model by Louis Martin et al.
- ALBERT (from Google Research), ALBERT: A Lite BERT for Self-supervised Learning of Language Representations by Zhenzhong Lan et al.
- T5 (from Google), Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer by Raffel et al.
- XLM-RoBERTa (from Facebook AI), Unsupervised Cross-lingual Representation Learning at Scale by Conneau et al.

Huggingface (4/6)

- MMBT (from Facebook), Supervised Multimodal Bitransformers for Classifying Images and Text by Kiela et al.
- FlauBERT (from CNRS), FlauBERT: Unsupervised Language Model Pre-training for French by Le et al.
- BART (from Facebook), BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension by Lewis et al.
- ELECTRA (from Google Research/Stanford University), ELECTRA: Pre-training text encoders as discriminators rather than generators by Clark et al.
- DialoGPT (from Microsoft Research), DialoGPT: Large-Scale Generative Pre-training for Conversational Response Generation by Zhang et al.

Huggingface (5/6)

- Reformer (from Google Research), Reformer: The Efficient Transformer by Kitaev et al.
- MarianMT (developed by the Microsoft Translator Team) machine translation models trained using OPUS pretrained_models data by Jörg Tiedemann.
- Longformer (from AllenAI), Longformer: The Long-Document Transformer by Beltagy et al.
- DPR (from Facebook), Dense Passage Retrieval for Open-Domain Question Answering by Karpukhin et al.
- Pegasus (from Google), PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization by Jingqing Zhang, Yao Zhao, Mohammad Saleh and Peter J. Liu.

Huggingface (6/6)

- MBart (from Facebook) released with the paper Multilingual Denoising Pre-training for Neural Machine Translation by Liu et al.
- LXMERT (from UNC Chapel Hill), LXMERT: Learning Cross-Modality Encoder Representations from Transformers for Open-Domain Question Answering by Tan and Mohit Bansal.
- Funnel Transformer (from CMU/Google Brain), Funnel-Transformer: Filtering out Sequential Redundancy for Efficient Language Processing by Dai et al.
- Bert For Sequence Generation (from Google), Leveraging Pre-trained Checkpoints for Sequence Generation Tasks Rothe et al.
- LayoutLM (from Microsoft Research Asia), LayoutLM: Pre-training of Text and Layout for Document Image Understanding by Xu et al.

Using Transformers: tasks

- Sequence Classification
- Extractive Question Answering
- Language Modeling
- Named Entity Recognition
- Summarization
- Translation

Using Transformers: Models

- Autoregressive models
- Autoencoding models
- Sequence-to-sequence models
- Multimodal models
- Retrieval-based models

Difference between “Discriminative” and “Generative” Models

- Historical reason
- Binary classification problem
- Want to decide if a patient has cancer based on different “features” from the reports
- $Argmax_D(P(D|S))$
- *D* takes values ‘Y’ and ‘N’
- Decide ‘Y’ if $P(D=Y|S) > P(=N|S)$, else ‘N’

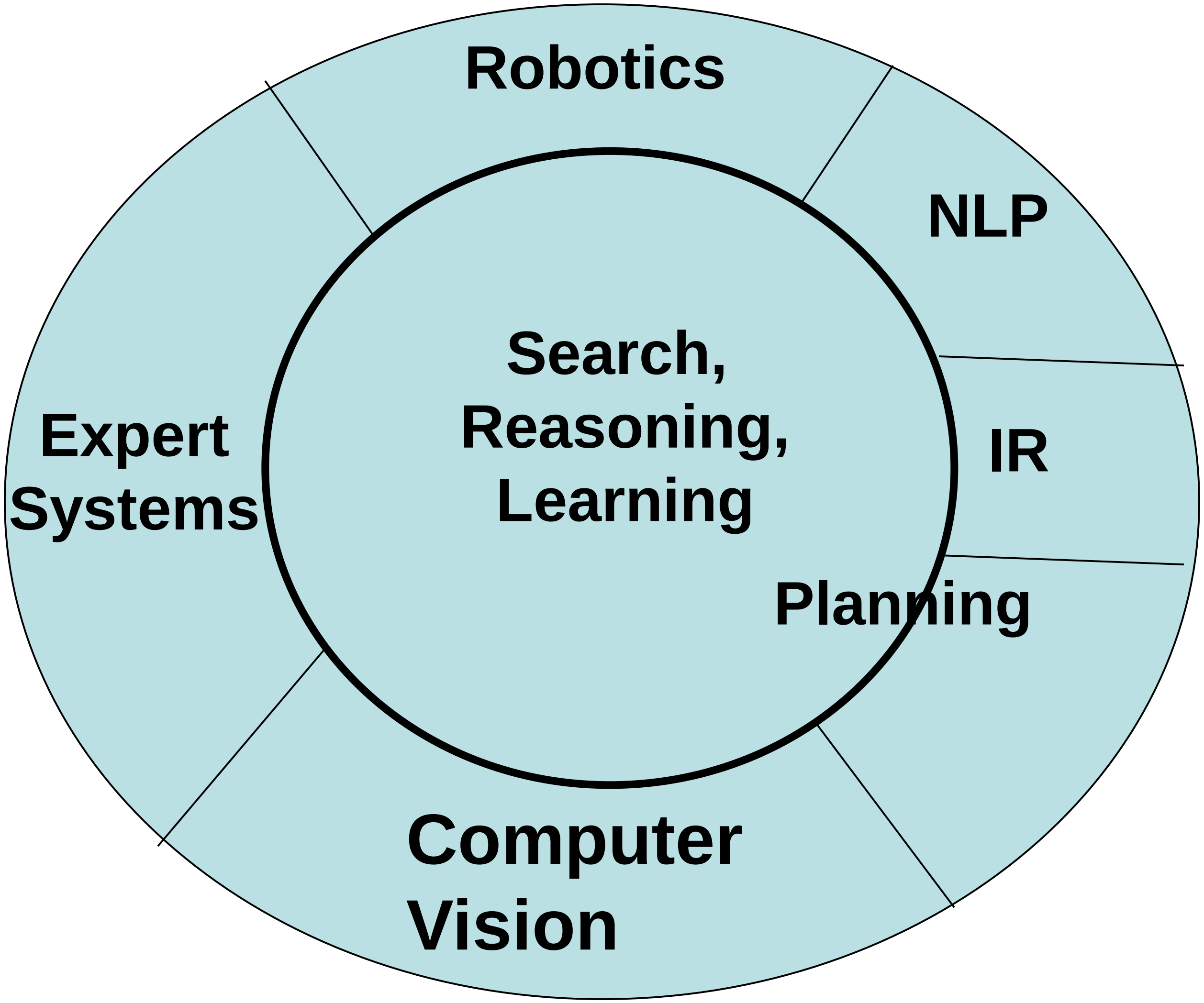
Discriminative Model

- Compute *P(D|S)* directly
- "Features" from reports, $S = \{F_1, F_2, F_3, \ldots, F_K\}$ (like, fever, weight loss, hair loss, haemoglobin level etc.)
- P(D=Y|<fever, weight loss, hair loss, haemoglobin level,…>)
- We are discriminating, i.e., differentiating wrt the features input

Generative Model

- Compute *P(D)* and *P(S|D)* and take product
- For P(D) we will need the proportion of cancer patients in the population (obtained via sampling)
- For the likelihood, we will make use of naïve Bayes assumption and require values of $P(F_i|D)$, e.g., what is the probability of a cancer patient having fever
- Hence the “discrimination” is not direct!!

AI Perspective (post-web)

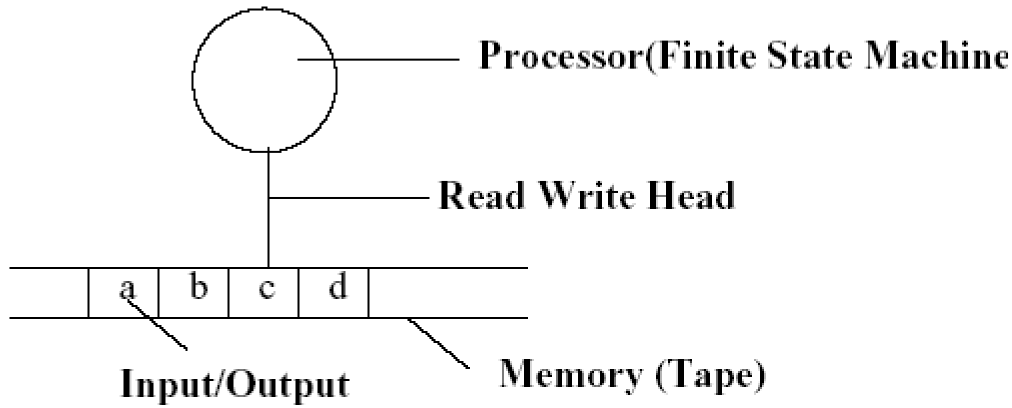


Symbolic AI

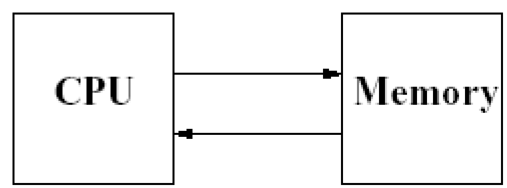
- Connectionist AI is contrasted with Symbolic AI
- Symbolic AI - Physical Symbol System Hypothesis
- Every intelligent system can be constructed by storing and processing symbols and nothing more is necessary.

- Symbolic AI has a bearing on models of computation such as

Turing Machine & Von Neumann



Turing machine



VonNeumann Machine

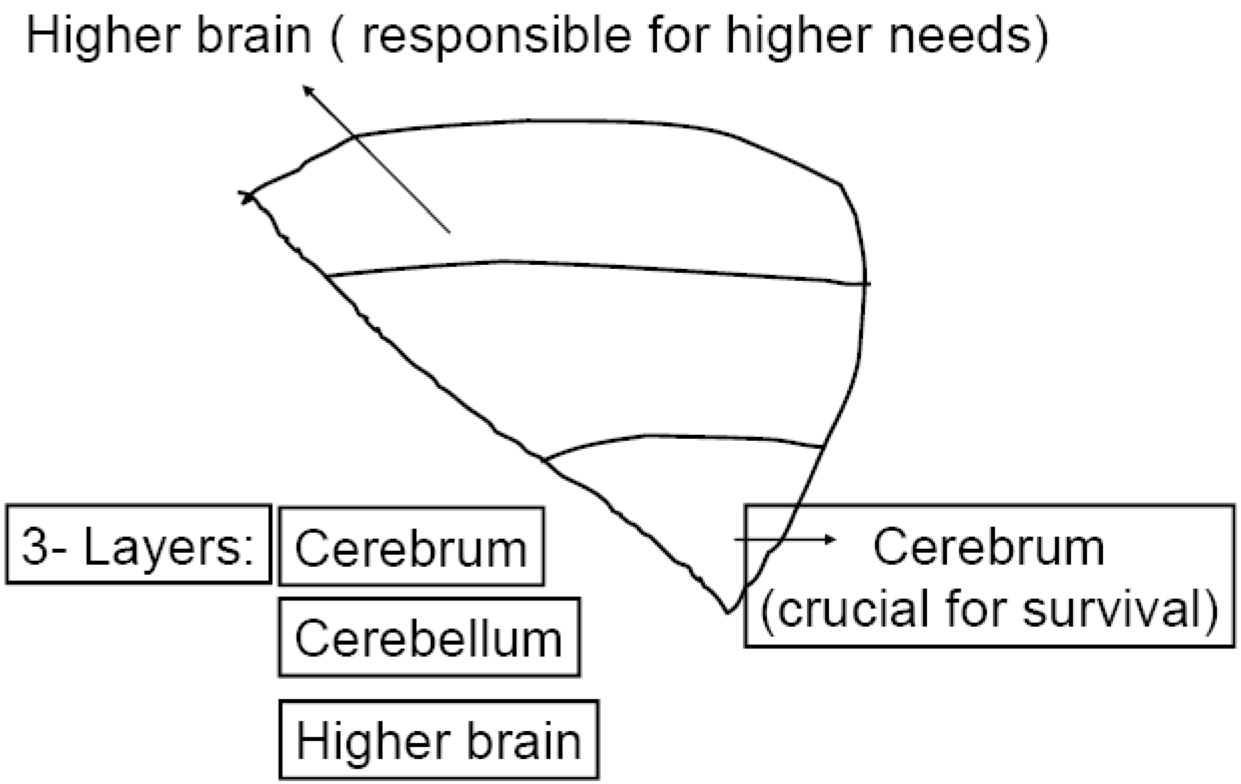
Challenges to Symbolic AI

- 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	Centralized & serial processing
• Content addressable	Location addressable

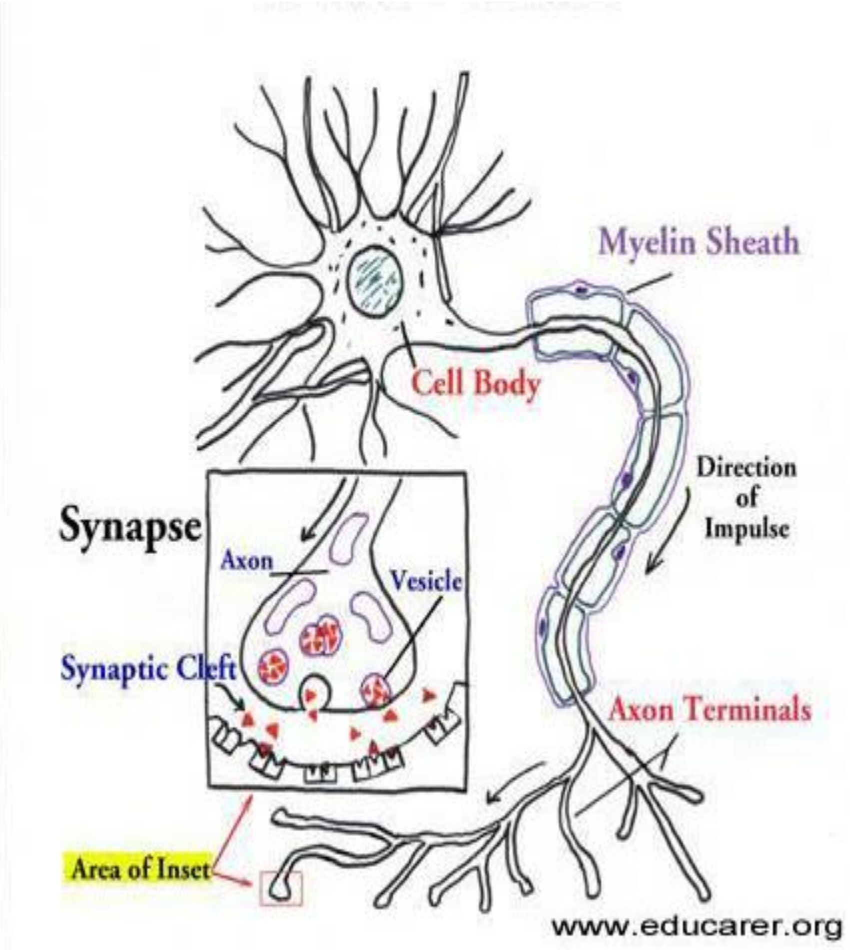
- The human brain

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


Higher brain (responsible for higher needs)
3- Layers:
Cerebrum
Cerebellum
Higher brain
Cerebrum
(crucial for survival)

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

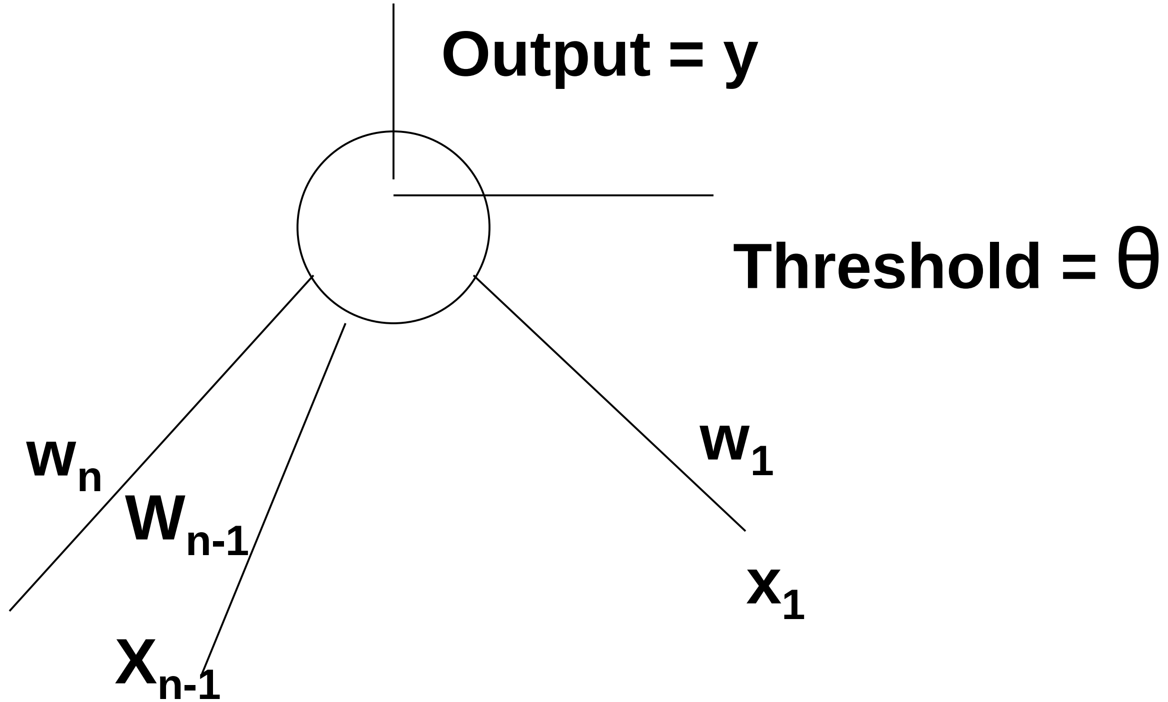


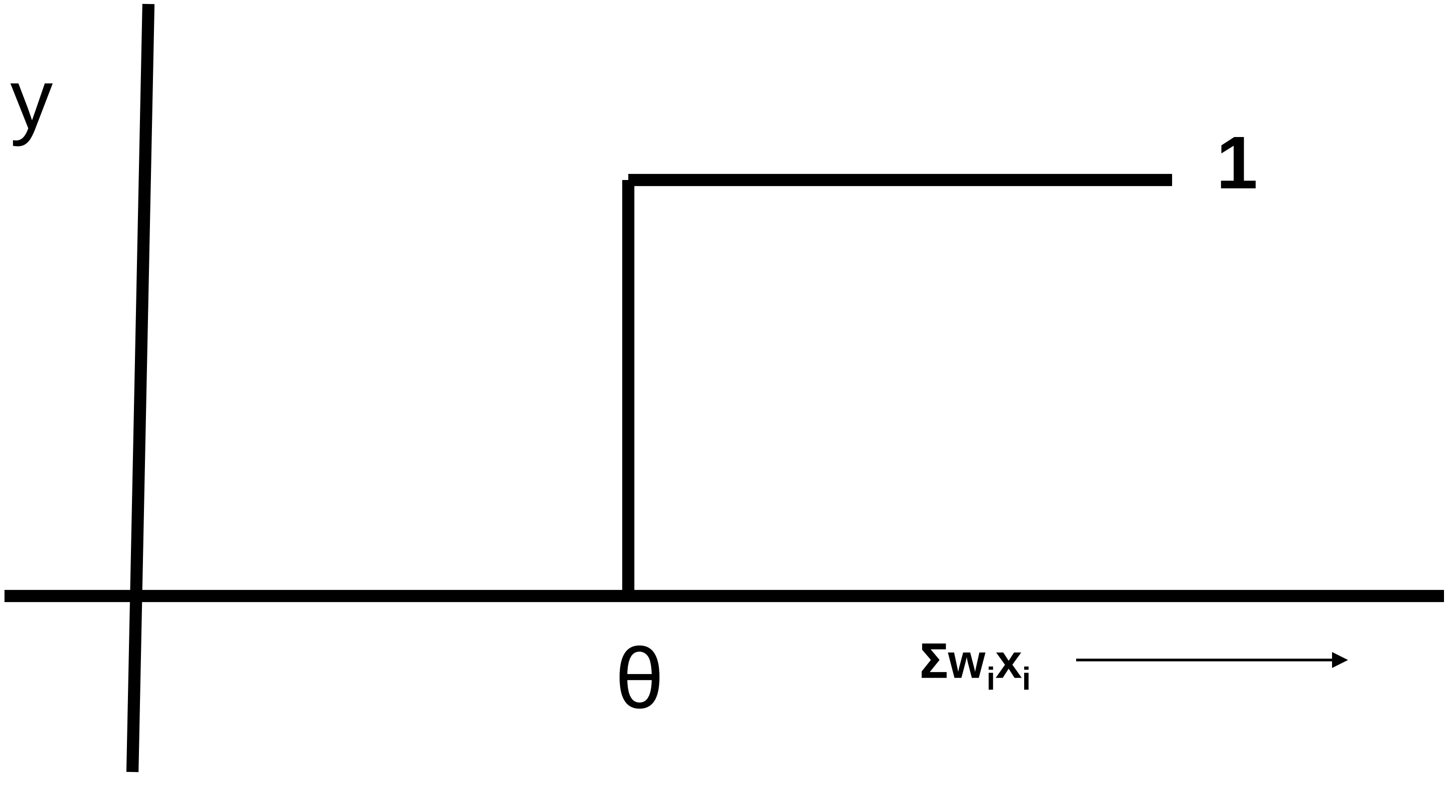
http://www.educarer.com/images/brain-nerve-axon.jpg

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**

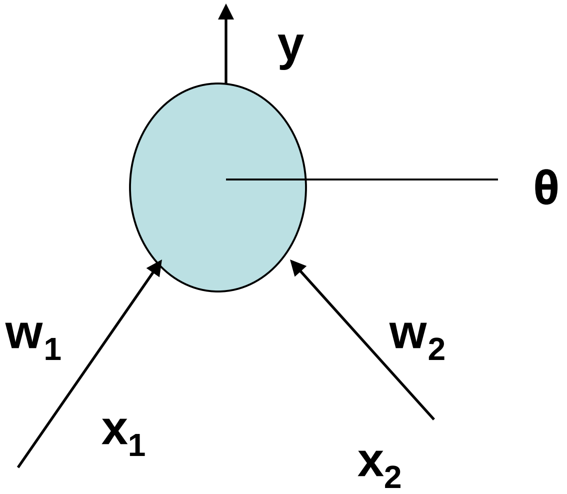
- **y= f(net): Relation between y and net is non-linear**

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 $y=1$

$w1 = w2 = \; = 0.5$

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.

Threshold functions

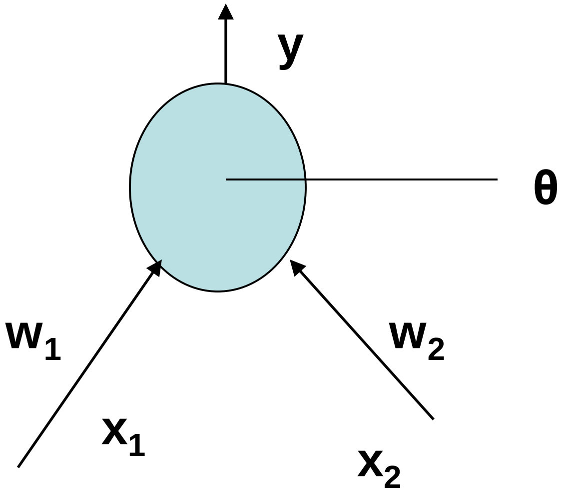
n	# Boolean functions (2^2^n)	#Threshold Functions (2^{n^2})
1	4	4
2	16	14
3	256	128
4	64K	1008

- **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.**

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 <= θ ➔ θ >= 0; since y=0

w1 * 0 + w2 * 1 <= θ ➔ w2 <= θ; since y=0

w1 * 1 + w2 * 0 <= θ ➔ w1 <= θ; since y=0

w1 * 1 + w2 *1 > θ ➔ w1 + w2 > θ; since y=1

w1 = w2 = = 0.5

These inequalities are satisfied by ONE particular region

Perceptron training

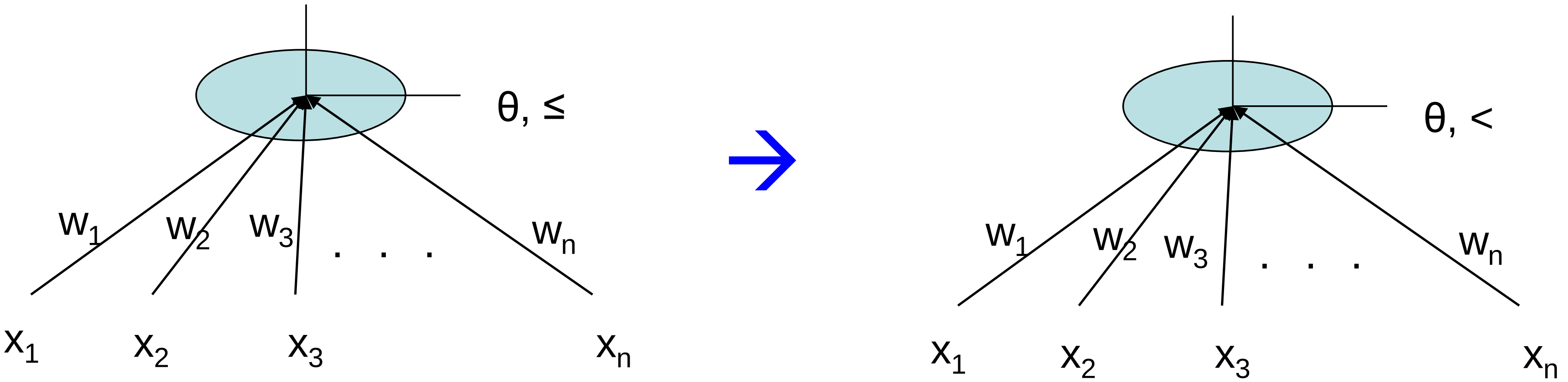
Perceptron Training Algorithm (PTA)

Preprocessing:

1. The computation law is modified to

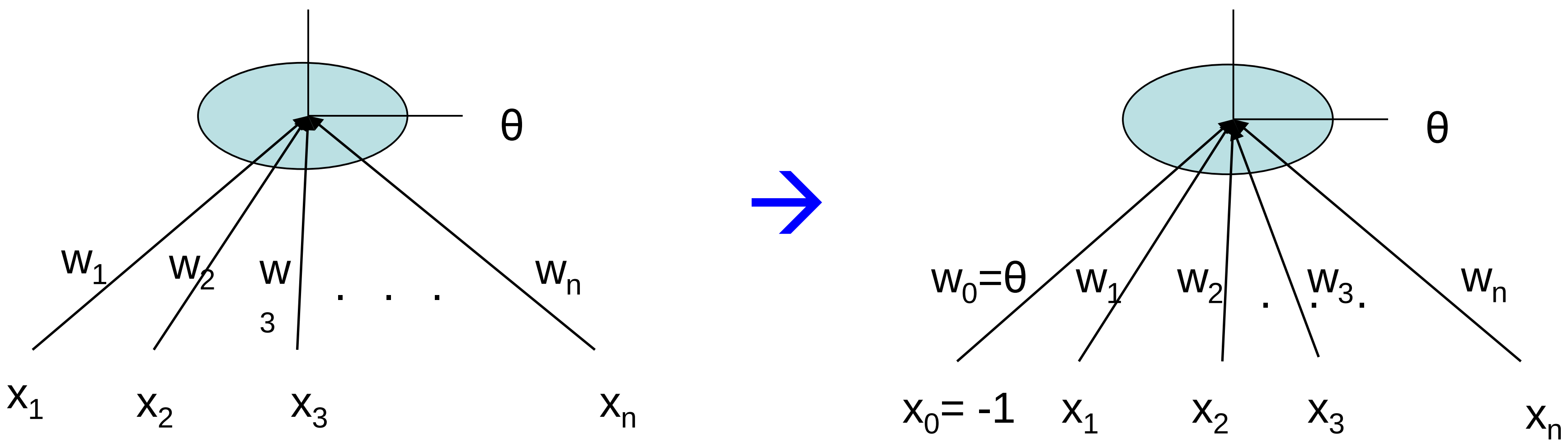
$$y = 1 \text{ if } \sum w_i x_i > \theta$$

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



PTA – preprocessing cont…

2. Absorb θ as a weight



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

	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

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

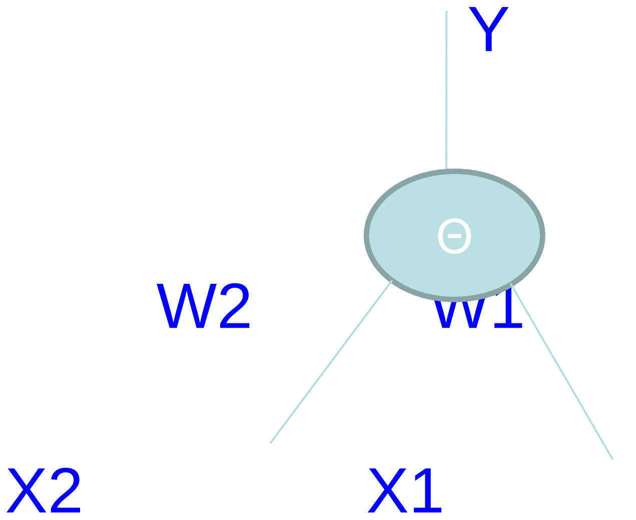
   then return w

3. Modify w, $w_{next} = w_{prev} + x_{fail}$

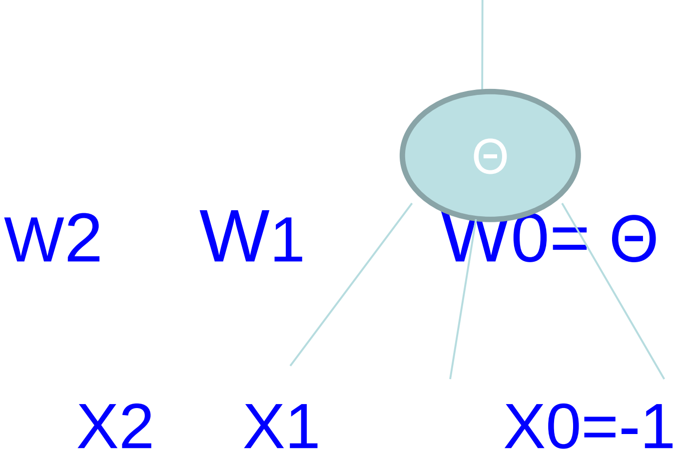
PTA on NAND

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: $W = <0, 0, 0>$

$W_1 = <0, 0, 0> + <0, 0, -1>$ {V_0 Fails}

$= <0, 0, -1>$

$W_2 = <0, 0, -1> + <-1, -1, 1>$ {V_3 Fails}

$= <-1, -1, 0>$

$W_3 = <-1, -1, 0> + <0, 0, -1>$ {V_0 Fails}

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

$W_4 = <-1, -1, -1> + <0, 1, -1>$ {V_1 Fails}

$= <-1, 0, -2>$

Trying convergence

$W_5 = <-1, 0, -2> + <-1, -1, 1>$ {V_3 Fails}

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

$W_6 = <-2, -1, -1> + <0, 1, -1>$ {V_1 Fails}

$= <-2, 0, -2>$

$W_7 = <-2, 0, -2> + <1, 0, -1>$ {V_0 Fails}

$= <-1, 0, -3>$

$W_8 = <-1, 0, -3> + <-1, -1, 1>$ {V_3 Fails}

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

$W_9 = <-2, -1, -2> + <1, 0, -1>$ {V_2 Fails}

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

Trying convergence

$W_{10} = \langle -1, -1, -3 \rangle + \langle -1, -1, 1 \rangle$ {V_3 Fails}

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

$W_{11} = \langle -2, -2, -2 \rangle + \langle 0, 1, -1 \rangle$ {V_1 Fails}

$= \langle -2, -1, -3 \rangle$

$W_{12} = \langle -2, -1, -3 \rangle + \langle -1, -1, 1 \rangle$ {V_3 Fails}

$= \langle -3, -2, -2 \rangle$

$W_{13} = \langle -3, -2, -2 \rangle + \langle 0, 1, -1 \rangle$ {V_1 Fails}

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

$W_{14} = \langle -3, -1, -3 \rangle + \langle 0, 1, -1 \rangle$ {V_2 Fails}

$= \langle -2, -1, -4 \rangle$

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

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 Xjs form a linearly separable function,

Proof of Convergence of PTA (cntd.)

- Consider the expression

$$G(w_n) = \frac{w_n \cdot w^*}{|w_n|}$$

where w_n = weight at nth iteration

- $$G(w_n) = \frac{|w_n| \cdot |w^*| \cdot \cos\theta}{|w_n|}$$

where θ = angle between w_n and w^*

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

Behavior of Numerator of G

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

$= w_{n-1} \,.\, w^* + X^{n-1}_{fail} \,.\, w^*$

$= (w_{n-2} + X^{n-2}_{fail}) \,.\, w^* + X^{n-1}_{fail} \,.\, w^* \,.....$

$= w_0 \,.\, w^* + (X^0_{fail} + X^1_{fail} + ... + X^{n-1}_{fail}) .\, w^*$

$w^*.X^i_{fail}$ is always positive: note carefully

- Suppose $|X_j| \geq \delta$, where δ is the minimum magnitude.
- Num of G $\geq |w_0 \,.\, w^*| + n\,\delta \,.\, |w^*|$

Behavior of Denominator of G

- $|w_n| = \sqrt{w_n \cdot w_n}$

$= \sqrt{(w_{n-1} + X^{n-1}_{fail})^2}$

$= \sqrt{(w_{n-1})^2 + 2.\, w_{n-1}.\, X^{n-1}_{fail} + (X^{n-1}_{fail})^2}$

$\leq \sqrt{(w_{n-1})^2 + (X^{n-1}_{fail})^2}$ (as $w_{n-1}.\, X^{n-1}_{fail} \leq 0$)

$\leq \sqrt{(w_0)^2 + (X^0_{fail})^2 + (X^1_{fail})^2 + \ldots + (X^{n-1}_{fail})^2}$

- $|X_j| \leq \rho$ (max magnitude)
- So Denom $\leq \sqrt{(w_0)^2 + n\rho^2}$

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, this 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.

According to the claim made by theorem, if weight repeat then the weight w_i at a given iteration i, will be equal to the weight w_{i+k} at a given iteration $(i+k)$ where k is a +ve number

$$w_i = w_{i+k}$$

Proof cntd.

Therefore:

$w_i \cdot w^* = w_{i+k} \cdot w^*$ (2)

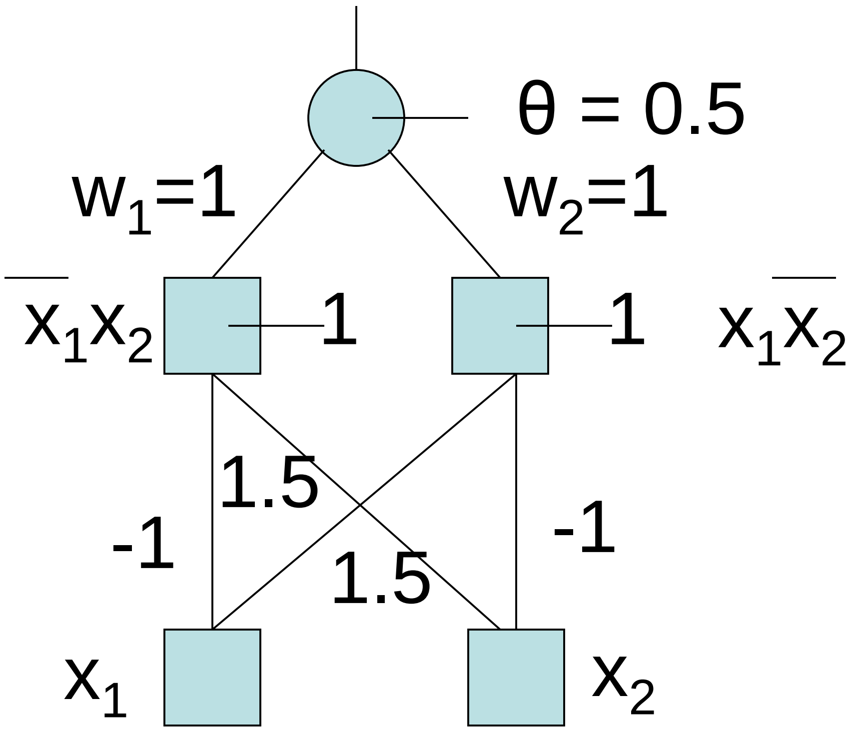
(2) contradicts the (1)

Hence no w* exists

So function is not linearly separable.

Feedforward Network and Backpropagation

Example - XOR



Can we use PTA for training FFN?

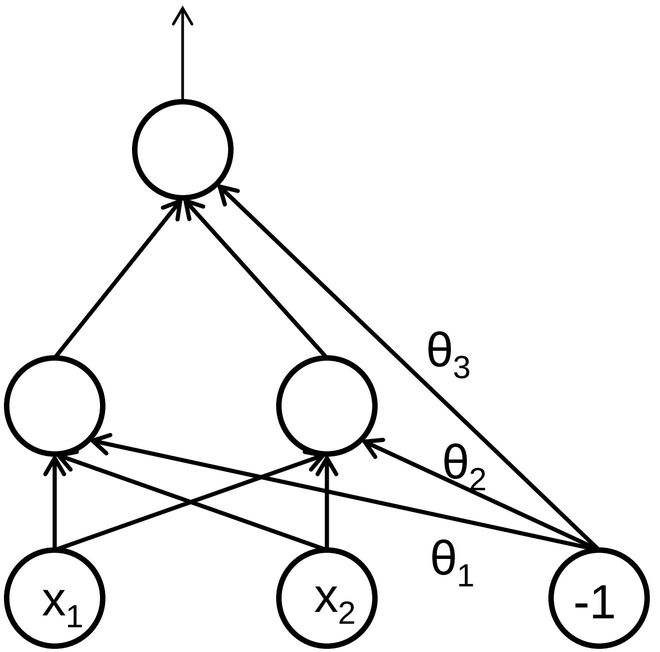
0, 0	0
0, 1	1
1, 0	1
1, 1	0

⇒

-1, 0, 0	0
-1, 0, 1	1
-1, 1, 0	1
-1, 1, 1	0

⇒

1, 0, 0	0
-1, 0, 1	1
-1, 1, 0	1
1, -1, -1	0



No, else the individual neurons are solving XOR, which is impossible.
Also, for the hidden layer neurons we do nothave the i/o behaviour.

Gradient Descent Technique

- Let E be the error at the output layer

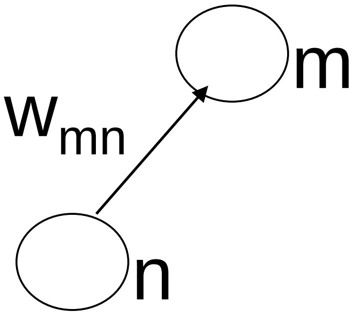
$$E = \frac{1}{2}\sum_{j=1}^{p}\sum_{i=1}^{n}(t_i - o_i)_j^2$$

- t_i = target output; o_i = observed output

- i is the index going over n neurons in the outermost layer
- j is the index going over the p patterns (1 to p)
- Ex: XOR:– p=4 and n=1

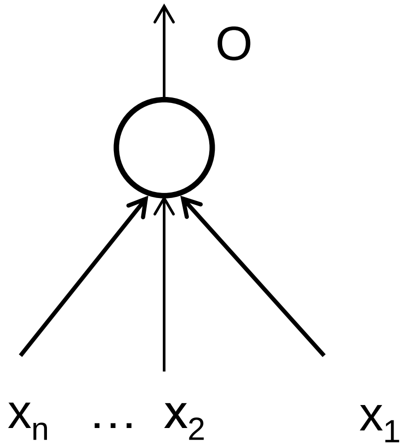
Weights in a FF NN

- w_{mn} is the weight of the connection from the n^th neuron to the m^th neuron
- E *vs* $\overline{W}$ surface is a complex surface in the space defined by the weights w_{ij}
- $-\frac{\delta E}{\delta w_{mn}}$ gives the direction in which a movement of the operating point in the w_{mn} co-ordinate space will result in maximum decrease in error



$$\Delta w_{mn} \propto -\frac{\delta E}{\delta w_{mn}}$$

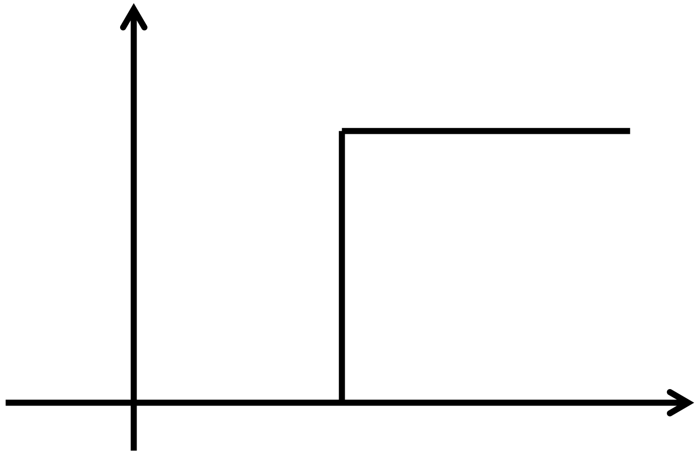
Step function v/s Sigmoid function



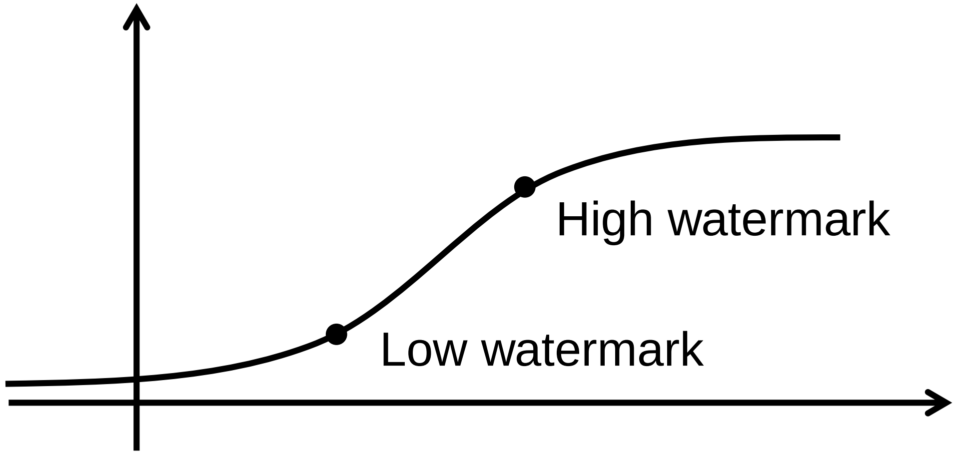
$$O = f(\sum w_i x_i)$$
$$= f(net)$$

So partial derivative of O w.r.t. net is

$$\frac{\delta O}{\delta net}$$

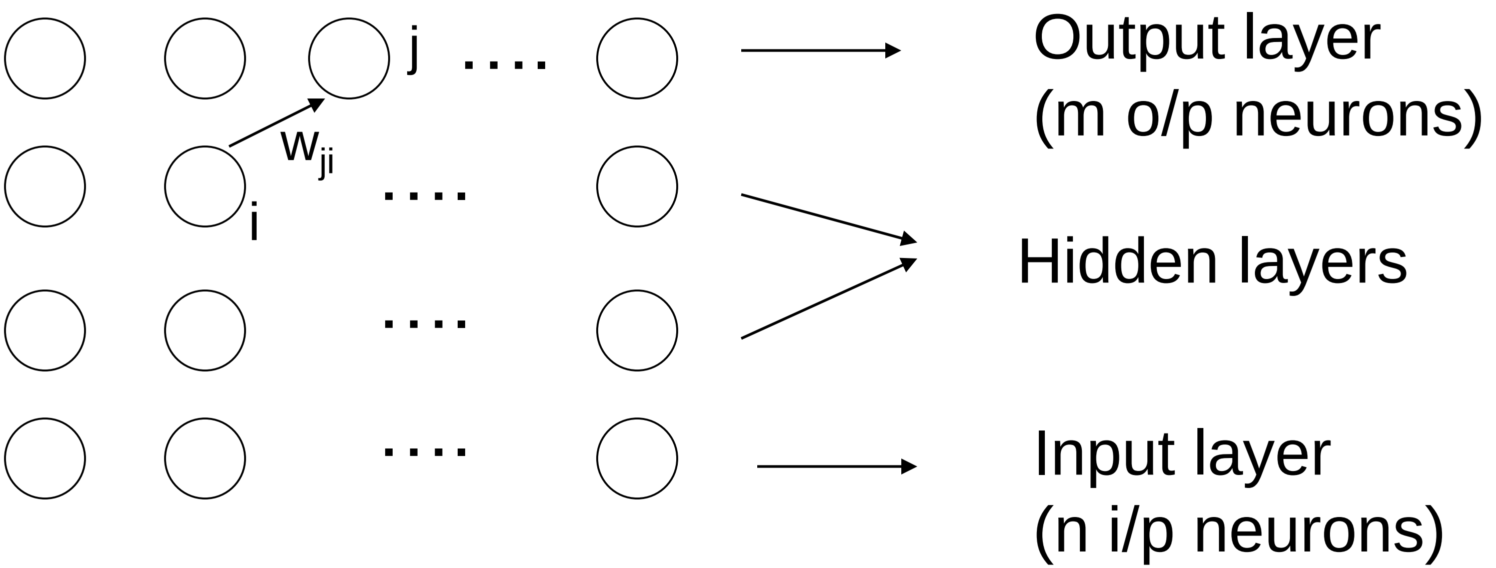


Non-differentiable



Differentiable

Backpropagation algorithm



- Fully connected feed forward network
- Pure FF network (no jumping of connections over layers)

Gradient Descent Equations

$$\Delta w_{ji} = -\eta \frac{\delta E}{\delta w_{ji}} (\eta = \text{learning rate}, 0 \le \eta \le 1)$$

$$\frac{\delta E}{\delta w_{ji}} = \frac{\delta E}{\delta net_j} \times \frac{\delta net_j}{\delta w_{ji}} (net_j = \text{input at the j}^{th} \text{ layer})$$

$$\frac{\delta E}{\delta net_j} = -\delta j$$

$$\Delta w_{ji} = \eta \delta j \frac{\delta net_j}{\delta w_{ji}} = \eta \delta j o_i$$

Backpropagation – for outermost layer

$$\delta j = -\frac{\delta E}{\delta net_j} = -\frac{\delta E}{\delta o_j} \times \frac{\delta o_j}{\delta net_j} (net_j = \text{input at the j}^{th} \text{ layer})$$

$$E = \frac{1}{2}\sum_{p=1}^{m}(t_p - o_p)^2$$

$$\text{Hence, } \delta j = -(-(t_j - o_j)o_j(1 - o_j))$$

$$\Delta w_{ji} = \eta(t_j - o_j)o_j(1 - o_j)o_i$$

Observations from Δw_{ji}

$$\Delta w_{ji} = \eta(t_j - o_j)o_j(1 - o_j)o_i$$

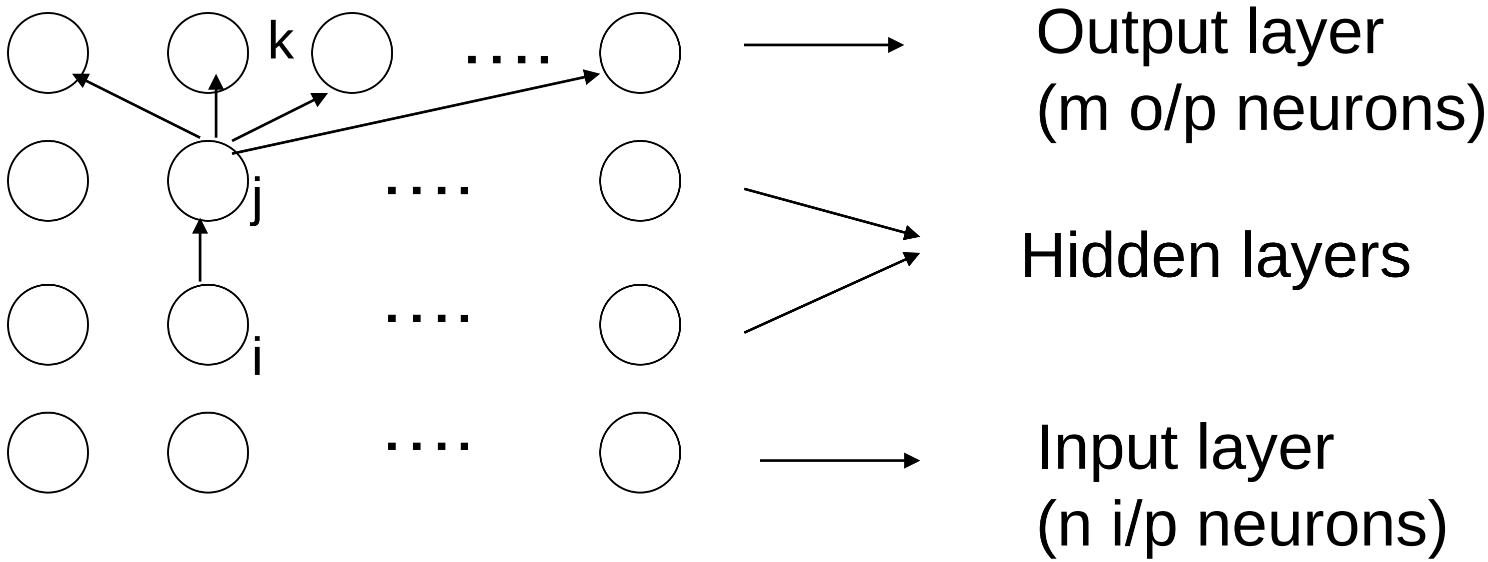
$\Delta w_{ji} \rightarrow 0$ if,

1. $O_j \rightarrow t_j$ and/or
2. $O_j \rightarrow 1$ and/or
3. $O_j \rightarrow 0$ and/or
4. $O_i \rightarrow 0$

(2 and 3): Saturation behaviour

(4): Credit/Blame assignment

Backpropagation for hidden layers



δ_k is propagated backwards to find value of δ_j

Backpropagation – for hidden layers

$$\Delta w_{ji} = \eta \delta j o_i$$

$$\delta j = -\frac{\delta E}{\delta net_j} = -\frac{\delta E}{\delta o_j} \times \frac{\delta o_j}{\delta net_j}$$

$$= -\frac{\delta E}{\delta o_j} \times o_j (1 - o_j)$$

$$= -\sum_{k \in \text{next layer}} \left(\frac{\delta E}{\delta net_k} \times \frac{\delta net_k}{\delta o_j}\right) \times o_j (1 - o_j)$$

$$\text{Hence, } \delta_j = -\sum_{k \in \text{next layer}} (-\delta_k \times w_{kj}) \times o_j (1 - o_j)$$

$$= \sum_{k \in \text{next layer}} (w_{kj} \delta_k) o_j (1 - o_j)$$

This recursion can give rise to vanishing and exploding Gradient problem

General Backpropagation Rule

- General weight updating rule:

$$\Delta w_{ji} = \eta \delta j o_i$$

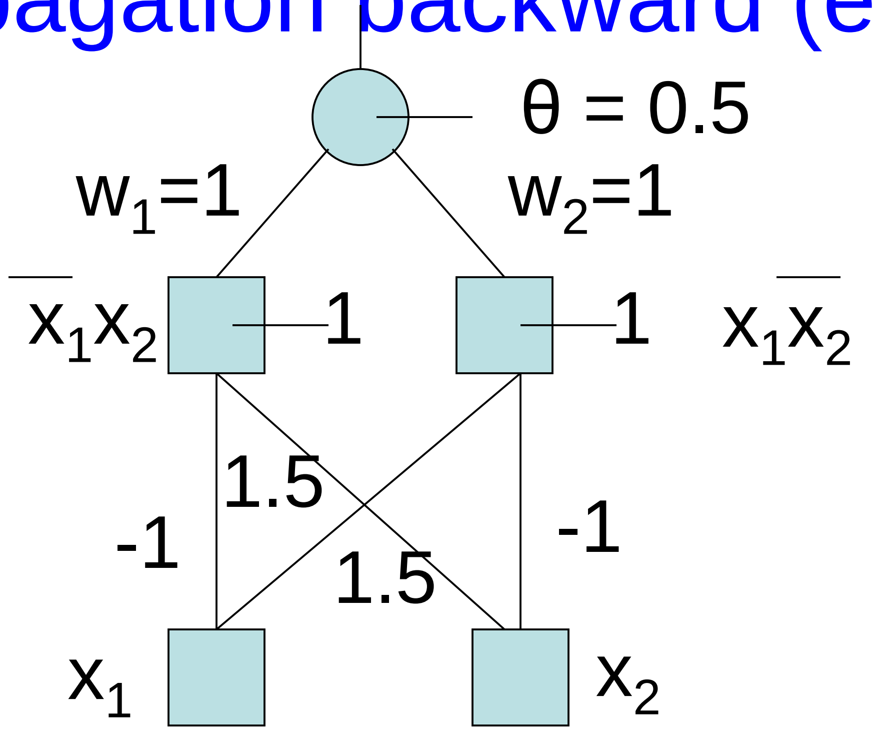
- Where

$$\delta_j = (t_j - o_j) o_j (1 - o_j) \quad \text{for outermost layer}$$

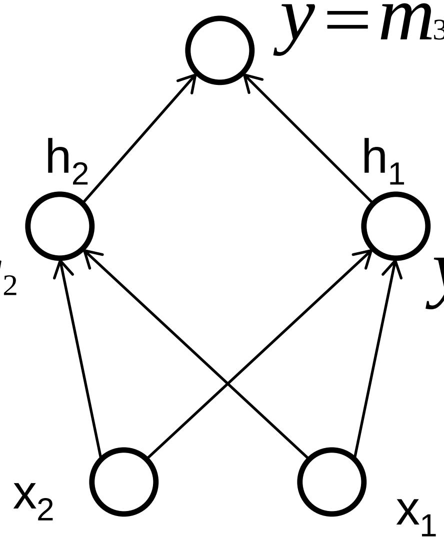
$$= \sum_{k \in \text{next layer}} (w_{kj} \delta_k) o_j (1 - o_j) o_i \quad \text{for hidden layers}$$

How does it work?

- Input propagation forward and error propagation backward (e.g. XOR)



Can Linear Neurons Work?



$$y = m_3 x + c_3$$

$$y = m_2 x + c_2$$

$$y = m_1 x + c_1$$

$$h_1 = m_1(w_1 x_1 + w_2 x_2) + c_1$$

$$h_1 = m_1(w_1 x_1 + w_2 x_2) + c_1$$

$$Out = (w_5 h_1 + w_6 h_2) + c_3$$
$$= k_1 x_1 + k_2 x_2 + k_3$$

Note: The whole structure shown in earlier slide is reducible to a single neuron with given behavior

$$Out = k_1 x_1 + k_2 x_2 + k_3$$

Claim: A neuron with linear I-O behavior can't compute X-OR.

Proof: Considering all possible cases:

[assuming 0.1 and 0.9 as the lower and upper thresholds]

For (0,0), Zero class:

$$m(w_1.0 + w_2.0 - \theta) + c < 0.1$$
$$\Rightarrow c - m.\theta < 0.1$$

For (0,1), One class:

$$m(w_2.1 + w_1.0 - \theta) + c > 0.9$$
$$\Rightarrow m.w_1 - m.\theta + c > 0.9$$

For (1,0), One class: $m.w_1 - m.\theta + c > 0.9$

For (1,1), Zero class: $m.w_1 - m.\theta + c > 0.9$

These equations are inconsistent. Hence X-OR can’t be computed.

Observations:

1. A linear neuron can’t compute X-OR.
2. A multilayer FFN with linear neurons is collapsible to a single linear neuron, hence **no a additional power due to hidden layer.**
3. Non-linearity is essential for power.