

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

Perceptron, Sigmoid, Softmax, POS
tagging assignment

Pushpak Bhattacharyya

Computer Science and Engineering
Department

IIT Bombay

Week 2 of 11th Aug, 2025

1-slide recap, Lecture 1

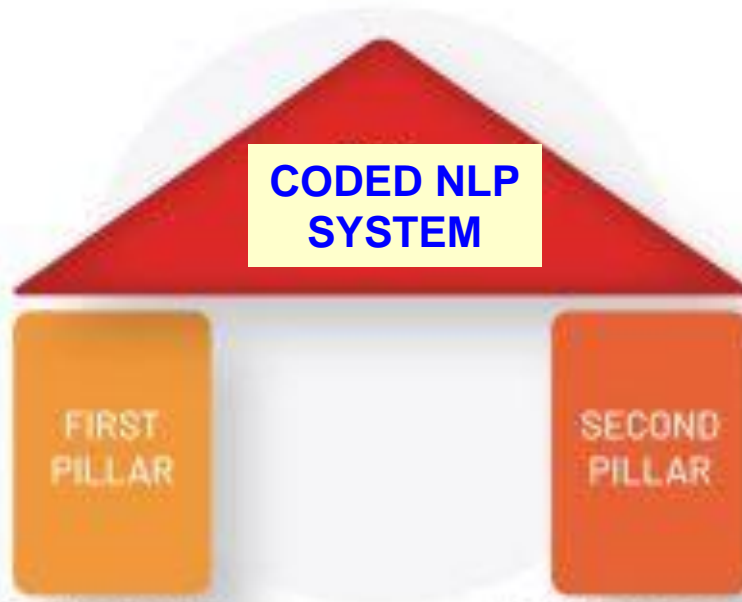
- Nature of language- displacement, recursion etc.
- Nature of NLP: NLP stack;
NLP=linguistics+probability; 3 gens of NLP
- Meaning of Language Modeling
- Main Challenge: Ambiguity
- ChatGPT's (an LLM) amazing capability- "Buffalo" sentence
- Course info- evaluation, references
- A glimpse of CFILT research on rare language phenomena and low resource setting

Perspective 2: NLP

Natural Language Processing: NLU and NLG

- 3 Generations
 - Gen1- Rule based NLP is also called Model Driven NLP
 - Gen2- Statistical ML based NLP (*Hidden Markov Model, Support Vector Machine*)
 - Gen3- Neural (Deep Learning) based NLP
 - Gen3.5- LLM and GenAI

Two Pillars of NLP



BEFORE the
model operates:
Annotation
AFTER: Error
Analysis

**Central Limit
Theorem**

**Law of Large
Numbers**

**LINGUISTICS +
= NLP**

PROBABILITY

Main challenge is Ambiguity!: an
extreme example

“Buffalo buffaloes Buffalo buffaloes buffalo
buffalo Buffalo buffaloes”

Prompt to chatGPT: what do you
understand by the above sentence

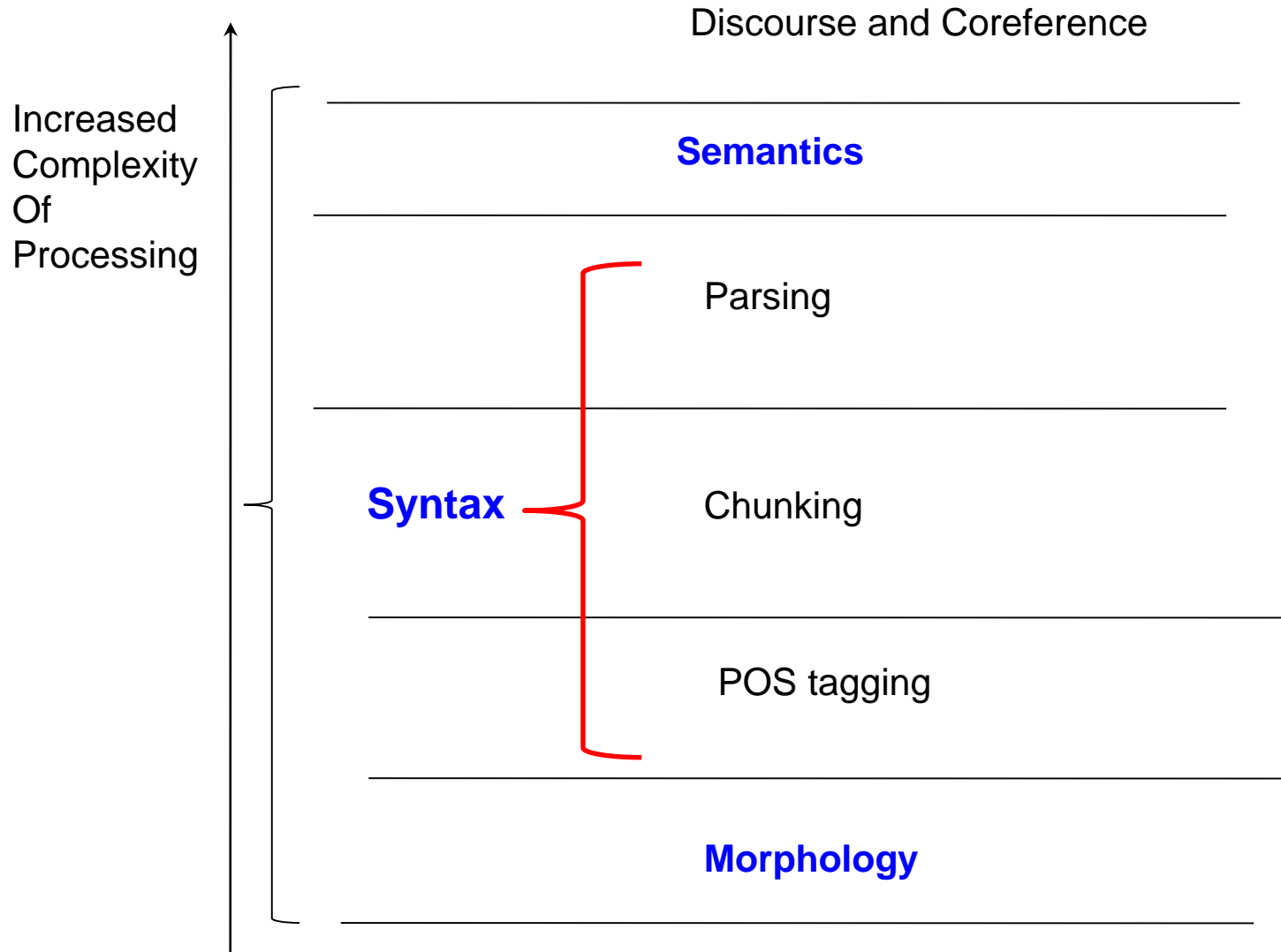
chatGPT response

- Buffalo (noun): Refers to the city of Buffalo, which is in the state of New York in the United States.
- buffaloes (verb): Means to bully, confuse, or baffle.
- Buffalo (noun): Refers to the animal, specifically the American bison.

describes a scenario where bison from Buffalo are being intimidated or confused by other bison from the same place, who, in turn, are intimidating or confusing other bison. It's a playful sentence that plays on the multiple meanings of the word "Buffalo" as a place

CORRECT!!

NLP Layers



Tools for Language Computation

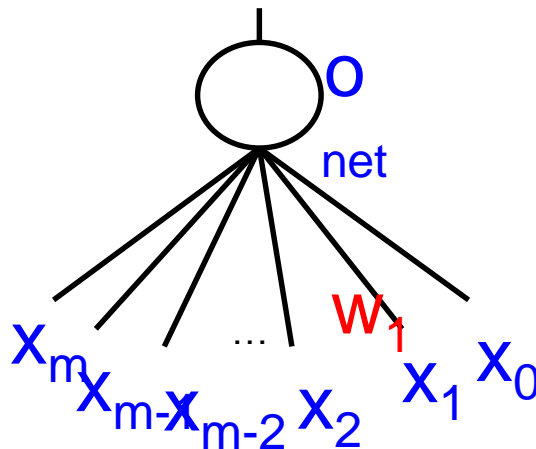
Softmax

$$\sigma(\bar{Z})_i = \frac{e^{Z_i}}{\sum_{j=1}^K e^{Z_j}}$$

Cross Entropy

$$H(P, Q) = - \sum_{x=1, N} \sum_{k=1, C} P(x, k) \log_2 Q(x, k)$$

BP



$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial o} \cdot \frac{\partial o}{\partial net} \cdot \frac{\partial net}{\partial w_1}$$

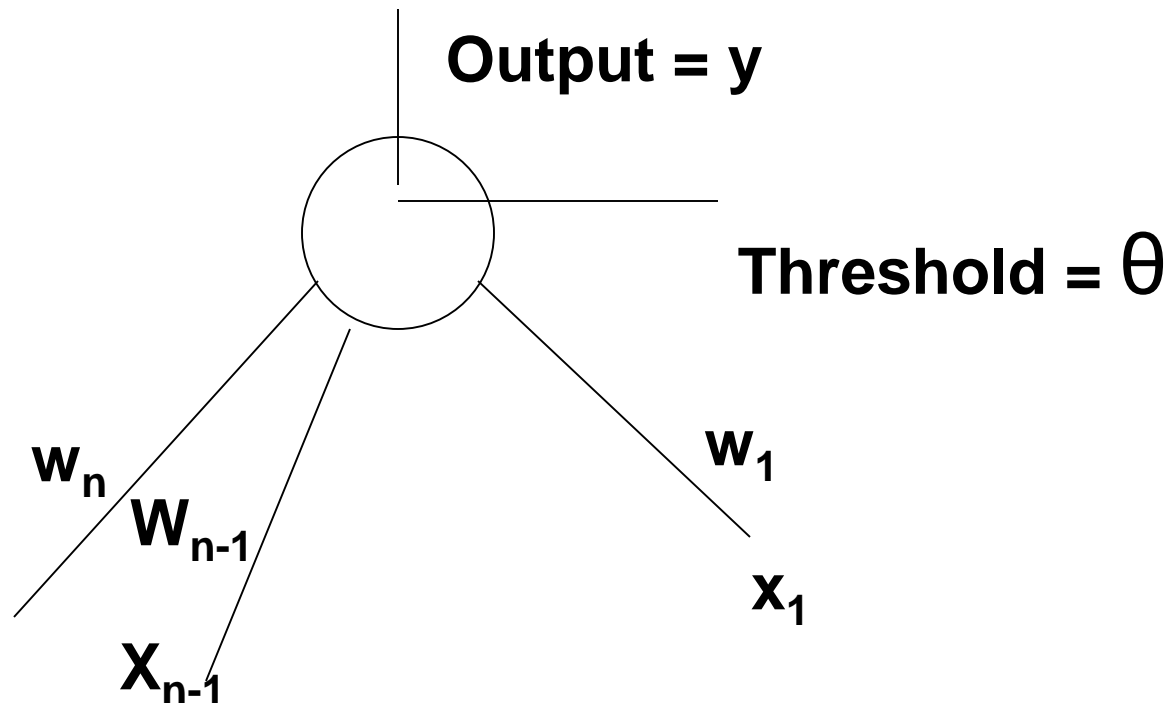
$$L = -t \log o - (1-t) \log(1-o)$$

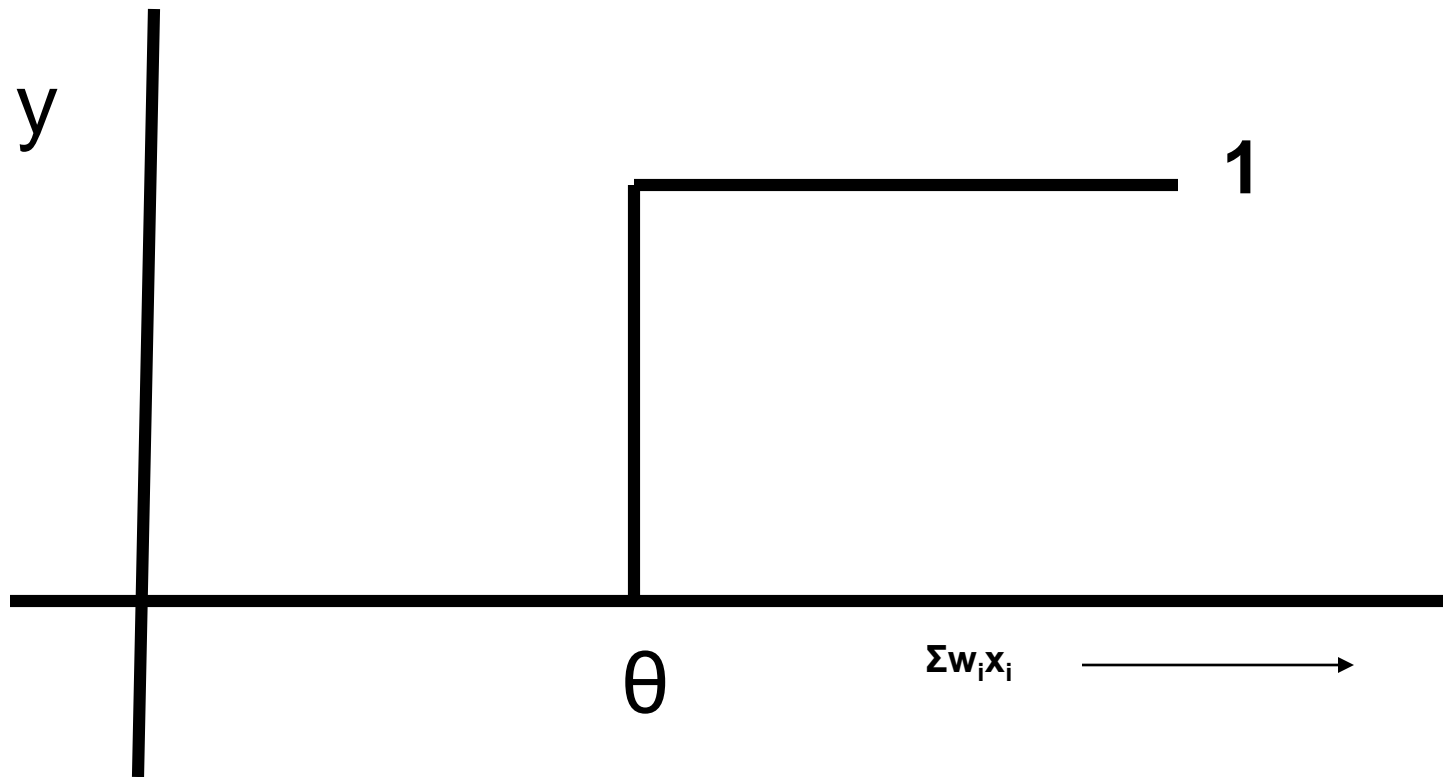
$$\Rightarrow \Delta w_1 = \eta \frac{\partial L}{\partial w_1} = \eta(t-o)x_1$$

The 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

$$y = 1 \text{ for } \sum_i w_i x_i \geq \theta$$
$$= 0 \text{ otherwise}$$

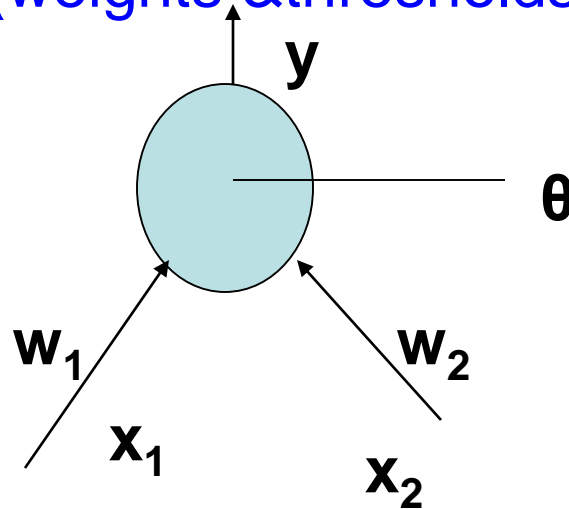
Features of Perceptron

- Input output behavior is discontinuous and the derivative does not exist at $\sum_i w_i x_i = \theta$
- $\sum_i w_i x_i - \theta$ is the net input denoted as *net*
- Referred to as a linear threshold element - linearity because of x appearing with power 1
- $y = f(\text{net})$: Relation between y and net is non-linear

Computation of Boolean functions: AND

x_2	x_1	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

- $w_1 * 0 + w_2 * 0 < \theta \Rightarrow \theta > 0$; since $y=0$
- $w_1 * 0 + w_2 * 1 < \theta \Rightarrow w_2 < \theta$; since $y=0$
- $w_1 * 1 + w_2 * 0 < \theta \Rightarrow w_1 < \theta$; since $y=0$
- $w_1 * 1 + w_2 * 1 \geq \theta \Rightarrow w_1 + w_2 \geq \theta$; since $y=1$
- $w_1=w_2=0.5, \theta=0.9$ is a possibility

Other Boolean functions

OR can be computed using values of

$$w_1=w_2=1 \text{ and } \theta=0.5$$

XOR cannot be computed:

$$w_1 * 0 + w_2 * 0 < \theta \rightarrow \theta > 0$$

$$w_1 * 0 + w_2 * 1 \geq \theta \rightarrow w_2 \geq \theta$$

$$w_1 * 1 + w_2 * 0 \geq \theta \rightarrow w_1 \geq \theta$$

$$w_1 * 1 + w_2 * 1 < \theta \rightarrow w_1 + w_2 < \theta$$

No set of parameter values satisfy these inequalities.

Threshold functions

- N variables: # 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.

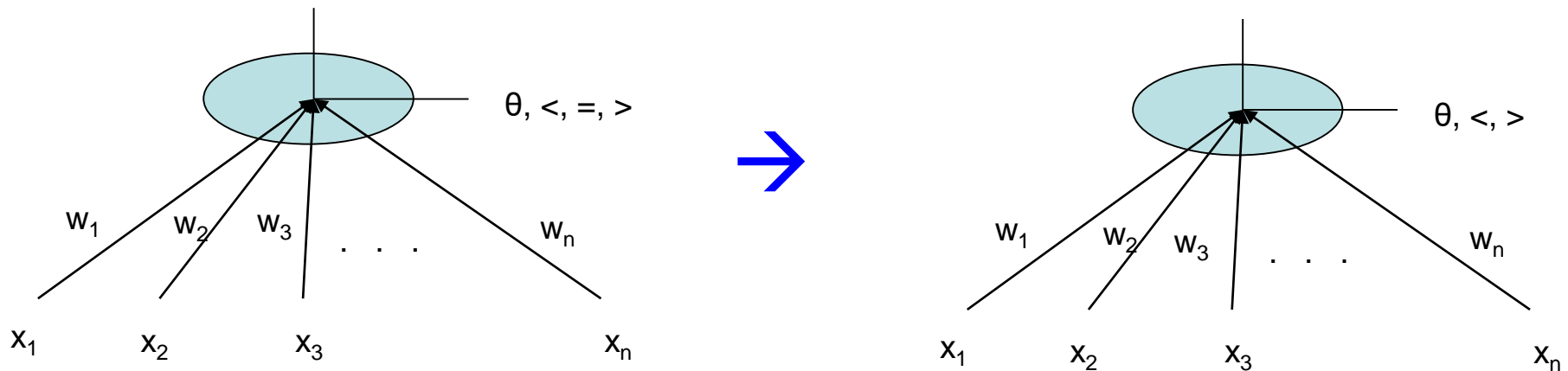
Perceptron Training Algorithm (PTA)

Preprocessing:

1. The computation law is modified to

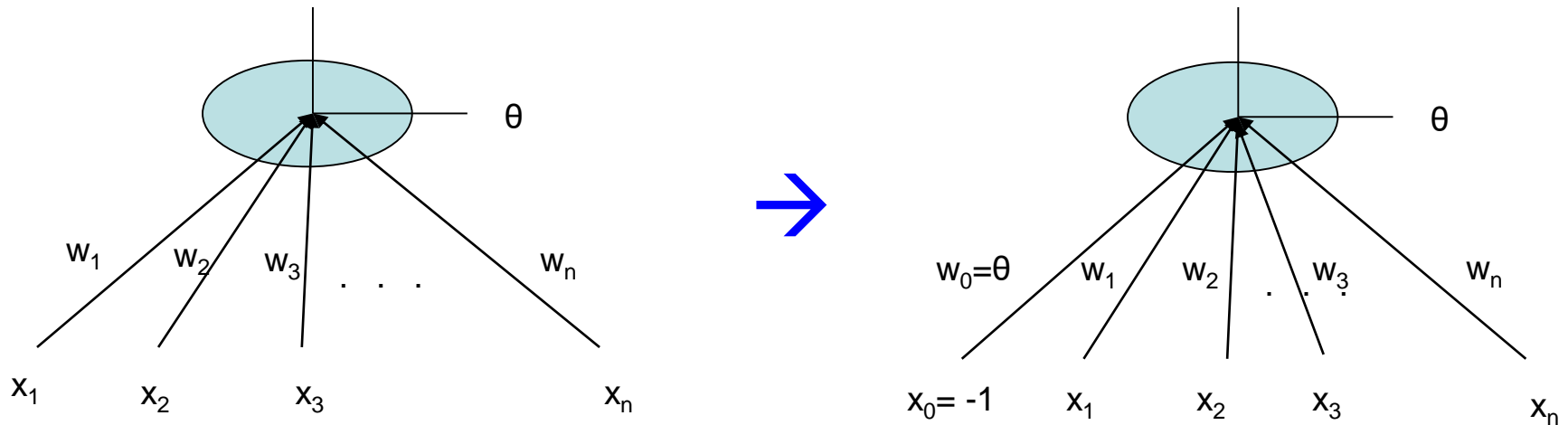
$$y=1 \text{ if } \sum w_i x_i > \theta$$

$$y=0 \text{ if } \sum w_i x_i < \theta$$



PTA – preprocessing cont...

2. Absorb θ as a weight



3. Negate all the zero-class examples

Example to demonstrate preprocessing

- **OR perceptron**

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

0-class $\langle 0, 0 \rangle$

Augmented x vectors:-

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

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

Negate 0-class:- $\langle 1, 0, 0 \rangle$

Example to demonstrate preprocessing cont..

Now the vectors
are

	x_2	x_1	x_0
X_0	0	0	-1
X_1	0	1	-1
X_2	1	0	-1
X_3	1	1	-1

After negating the 0-
class

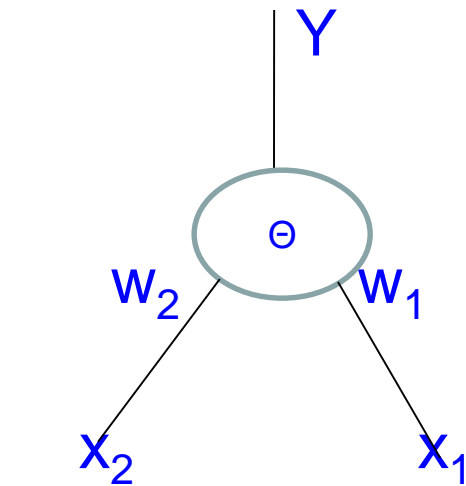
	x_2	x_1	x_0
X_0	0	0	1
X_1	0	1	-1
X_2	1	0	-1
X_3	1	1	-1

Perceptron Training Algorithm

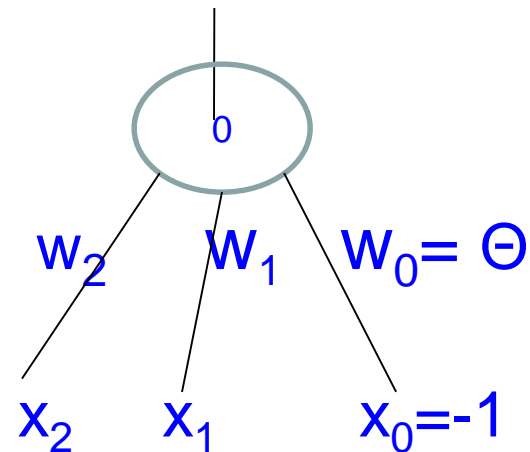
1. Start with a random value of w
ex: $\langle 0, 0, 0 \dots \rangle$
2. Test for $WX_i > 0$
If the test succeeds for $i=1, 2, \dots, n$
then return W
3. Modify W , $W_{\text{next}} = W_{\text{prev}} + X_{\text{fail}}$

PTA on NAND

NAND:		
X2	X1	Y
0	0	1
0	1	1
1	0	1
1	1	0



Converted To



Preprocessing

NAND Augmented:

NAND-0 class Negated

x_2	x_1	x_0	Y
0	0	-1	1
0	1	-1	1
1	0	-1	1
1	1	-1	0

	x_2	x_1	x_0
$X_0:$	0	0	-1
$X_1:$	0	1	-1
$X_2:$	1	0	-1
$X_3:$	-1	-1	1

Vectors for which $W = \langle w_2 \ w_1 \ w_0 \rangle$ has to be found such that $W \cdot X_i > 0$

PTA Algo steps

Step-0: $W_0 = \langle 0, 0, 0 \rangle$

$W_1 = \langle 0, 0, 0 \rangle + \langle 0, 0, -1 \rangle \quad \{X_0 \text{ Fails}\}$
 $= \langle 0, 0, -1 \rangle$

$W_2 = \langle 0, 0, -1 \rangle + \langle -1, -1, 1 \rangle \quad \{X_3 \text{ Fails}\}$
 $= \langle -1, -1, 0 \rangle$

$W_3 = \langle -1, -1, 0 \rangle + \langle 0, 0, -1 \rangle \quad \{X_0 \text{ Fails}\}$
 $= \langle -1, -1, -1 \rangle$

$W_4 = \langle -1, -1, -1 \rangle + \langle 0, 1, -1 \rangle \quad \{X_1 \text{ Fails}\}$
 $= \langle -1, 0, -2 \rangle$

X_0 :	0	0	-1
X_1 :	0	1	-1
X_2 :	1	0	-1
X_3 :	-1	-1	1

Trying convergence

$$\begin{aligned} W_5 &= \langle -1, 0, -2 \rangle + \langle -1, -1, 1 \rangle && \{X_3 \text{ Fails}\} \\ &= \langle -2, -1, -1 \rangle \end{aligned}$$

$$\begin{aligned} W_6 &= \langle -2, -1, -1 \rangle + \langle 0, 1, -1 \rangle && \{X_1 \text{ Fails}\} \\ &= \langle -2, 0, -2 \rangle \end{aligned}$$

$$\begin{aligned} W_7 &= \langle -2, 0, -2 \rangle + \langle 1, 0, -1 \rangle && \{X_0 \text{ Fails}\} \\ &= \langle -1, 0, -3 \rangle \end{aligned}$$

$$\begin{aligned} W_8 &= \langle -1, 0, -3 \rangle + \langle -1, -1, 1 \rangle && \{X_3 \text{ Fails}\} \\ &= \langle -2, -1, -2 \rangle \end{aligned}$$

$$\begin{aligned} W_9 &= \langle -2, -1, -2 \rangle + \langle 1, 0, -1 \rangle && \{X_2 \text{ Fails}\} \\ &= \langle -1, -1, -3 \rangle \end{aligned}$$

X_0 :	0	0	-1
X_1 :	0	1	-1
X_2 :	1	0	-1
X_3 :	-1	-1	1

Trying convergence

$$\begin{aligned} W_{10} &= \langle -1, -1, -3 \rangle + \langle -1, -1, 1 \rangle \quad \{X_3 \text{ Fails}\} \\ &= \langle -2, -2, -2 \rangle \end{aligned}$$

$$\begin{aligned} W_{11} &= \langle -2, -2, -2 \rangle + \langle 0, 1, -1 \rangle \quad \{X_1 \text{ Fails}\} \\ &= \langle -2, -1, -3 \rangle \end{aligned}$$

$$\begin{aligned} W_{12} &= \langle -2, -1, -3 \rangle + \langle -1, -1, 1 \rangle \quad \{X_3 \text{ Fails}\} \\ &= \langle -3, -2, -2 \rangle \end{aligned}$$

$$\begin{aligned} W_{13} &= \langle -3, -2, -2 \rangle + \langle 0, 1, -1 \rangle \quad \{X_1 \text{ Fails}\} \\ &= \langle -3, -1, -3 \rangle \end{aligned}$$

$$\begin{aligned} W_{14} &= \langle -3, -1, -3 \rangle + \langle 0, 1, -1 \rangle \quad \{X_2 \text{ Fails}\} \\ &= \langle -2, -1, -4 \rangle \end{aligned}$$

X_0 :	0	0	-1
X_1 :	0	1	-1
X_2 :	1	0	-1
X_3 :	-1	-1	1

$$W15 = \langle -2, -1, -4 \rangle + \langle -1, -1, 1 \rangle \quad \{X_3 \text{ Fails}\}$$

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

$$W16 = \langle -3, -2, -3 \rangle + \langle 1, 0, -1 \rangle \quad \{X_2 \text{ Fails}\}$$

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

$$W17 = \langle -2, -2, -4 \rangle + \langle -1, -1, 1 \rangle \quad \{X_3 \text{ Fails}\}$$

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

$$W18 = \langle -3, -3, -3 \rangle + \langle 0, 1, -1 \rangle \quad \{X_1 \text{ Fails}\}$$

$$= \langle -3, -2, -4 \rangle$$

$$W2 = -3, \quad W1 = -2, \quad W0 = \Theta = -4$$

X_0 :	0	0	-1
X_1 :	0	1	-1
X_2 :	1	0	-1
X_3 :	-1	-1	1

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 X_j s form a linearly separable function,
- there exists w^* s.t. $w^* X_j > 0$ for all j

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^*

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

Behavior of Numerator of G

$$\begin{aligned}w_n \cdot w^* &= (w_{n-1} + X_{\text{fail}}^{n-1}) \cdot w^* \\&= w_{n-1} \cdot w^* + X_{\text{fail}}^{n-1} \cdot w^* \\&= (w_{n-2} + X_{\text{fail}}^{n-2}) \cdot w^* + X_{\text{fail}}^{n-1} \cdot w^* \dots \\&= w_0 \cdot w^* + (X_{\text{fail}}^0 + X_{\text{fail}}^1 + \dots + X_{\text{fail}}^{n-1}) \cdot w^*\end{aligned}$$

$w^* \cdot X_{\text{fail}}^i$ is always positive: note carefully

- Suppose $w^* \cdot X_{\text{fail}}^i \geq \delta_{\min}$, where δ_{\min} is a positive quantity
- Num of G $\geq |w_0 \cdot w^*| + n \delta_{\min}$
- So, numerator of G grows with n.

Behavior of Denominator of G

- $|w_n| = (w_n \cdot w_n)^{1/2}$
 $= [(w_{n-1} + X_{fail}^{n-1})^2]^{1/2}$
 $= [(w_{n-1})^2 + 2 \cdot w_{n-1} \cdot X_{fail}^{n-1} + (X_{fail}^{n-1})^2]^{1/2}$
 $\leq [(w_{n-1})^2 + (X_{fail}^{n-1})^2]^{1/2} \quad (\text{as } w_{n-1} \cdot X_{fail}^{n-1} \leq 0)$
 $\leq [(w_0)^2 + (X_{fail}^0)^2 + (X_{fail}^1)^2 + \dots + (X_{fail}^{n-1})^2]^{1/2}$
- $|X_j| \leq \delta_{\max}$ (max magnitude)
- So, Denom $\leq [(w_0)^2 + n \delta_{\max}^2]^{1/2}$
- Denom grows as $n^{1/2}$

Some Observations

- Numerator of G grows as n
- Denominator of G grows as $n^{1/2}$
=> Numerator grows faster than denominator
- If PTA does not terminate, $G(w_n)$ values will become unbounded.

Some Observations contd.

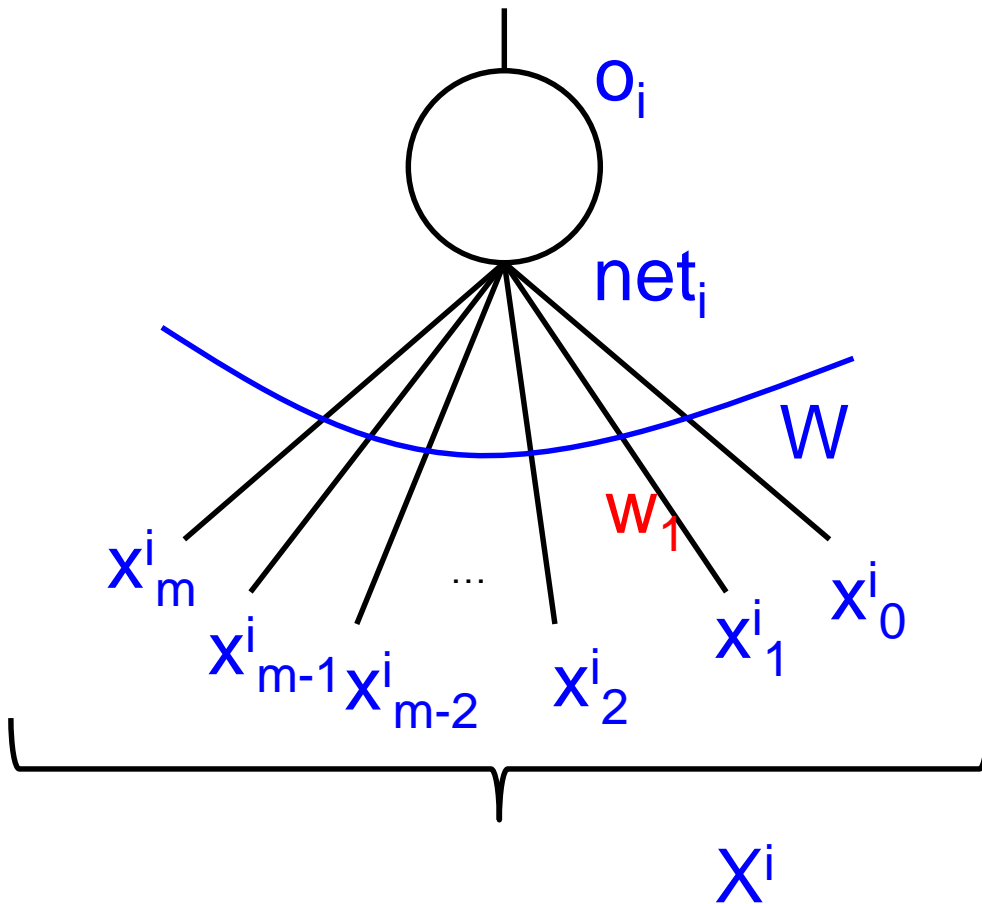
- But, as $|G(w_n)| \leq |w^*|$ which is finite, this is impossible!
- Hence, PTA has to converge.
- Proof is due to Marvin Minsky.

Convergence of PTA proved

- *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.*

Sigmoid

Sigmoid neuron



$$o^i = \frac{1}{1 + e^{-net^i}}$$

$$net_i = W.X^i = \sum_{j=0}^m w_j x_j^i$$

Sigmoid function: can saturate

- Brain saving itself from itself, in case of extreme agitation, emotion etc.



Definition: Sigmoid or Logit function

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

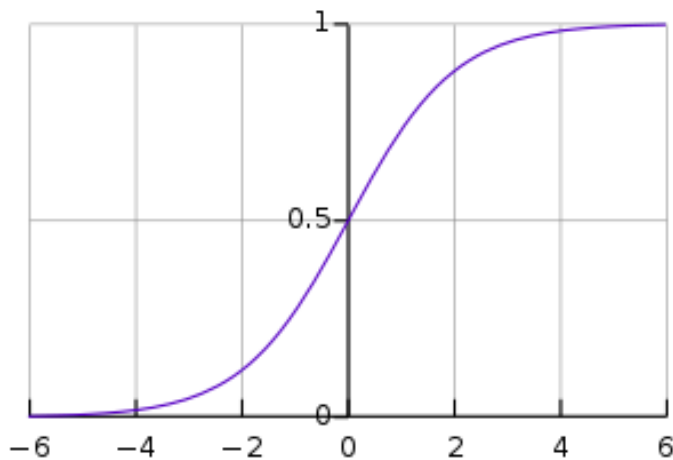
$$y = \frac{1}{1 + e^{-kx}}$$

$$\frac{dy}{dx} = y(1 - y)$$

$$\frac{dy}{dx} = ky(1 - y)$$

If k tends to infinity, sigmoid tends to the step function

Sigmoid function



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

$$\begin{aligned} f(x) &= \frac{1}{1+e^{-x}} \\ \frac{df(x)}{dx} &= \frac{d}{dx} \left(\frac{1}{1+e^{-x}} \right) \\ &= \frac{e^{-x}}{(1+e^{-x})^2} \\ &= \frac{1}{1+e^{-x}} \left(1 - \frac{1}{1+e^{-x}} \right) \\ &= f(x) \cdot (1 - f(x)) \end{aligned}$$

Decision making under sigmoid

- Output of sigmoid is between 0-1
- Look upon this value as probability of Class-1 (C_1)
- $1-\text{sigmoid}(x)$ is the probability of Class-2 (C_2)
- Decide C_1 , if $P(C_1) > P(C_2)$, else C_2

Sigmoid function and multiclass classification

- Why can't we use sigmoid for n-class classification? Have segments on the curve devoted to different classes, just like $-\infty$ to 0.5 is for class 2 and 0.5 to plus infinity is class 2.
- Think about it!!

multiclass: SOFTMAX

- 2-class \rightarrow multi-class (C classes)
- Sigmoid \rightarrow softmax
- i^{th} input, c^{th} class (small c), c varies over classes
- In softmax, decide for that class which has the highest probability

What is softmax

- Turns a vector of K real values into a vector of K real values that sum to 1
- Input values can be positive, negative, zero, or greater than one
- But softmax transforms them into values between 0 and 1
- so that they can be interpreted as probabilities.

Mathematical form

$$\sigma(\bar{Z})_i = \frac{e^{Z_i}}{\sum_{j=1}^K e^{Z_j}}$$

- σ is the **softmax** function
- Z is the input vector of size K
- The RHS gives the i^{th} component of the output vector
- Input to softmax and output of softmax are of the same dimension

Example

$$\bar{Z} = \langle 1, 2, 3 \rangle$$

$$Z_1 = 1, Z_2 = 2, Z_3 = 3$$

$$e^1 = 2.72, e^2 = 7.39, e^3 = 20.09$$

$$\sigma(\bar{Z}) = \left\langle \frac{2.72}{2.72 + 7.39 + 20.09}, \frac{7.39}{2.72 + 7.39 + 20.09}, \frac{20.09}{2.72 + 7.39 + 20.09} \right\rangle$$
$$= \langle .09, 0.24, 0.67 \rangle$$

Softmax and Cross Entropy

- Intimate connection between softmax and cross entropy
- Softmax gives a vector of probabilities
- Winner-take-all strategy will give a classification decision

Winner-take-all with softmax

- Consider the softmax vector obtained from the example where the softmax vector is $\langle 0.09, 0.24, 0.65 \rangle$
- These values correspond to 3 classes
 - For example, - *positive (+), negative (-) and neutral (0)* sentiments, given an input sentence like
 - (a) *I like the story line of the movie (+).* (b) *However the acting is weak (-).* (c) *The protagonist is a sports coach (0)*

Sentence vs. Sentiment

Sentence vs. Sentiment	Positive	Negative	Neutral
	(a) <i>I like the story line of the movie (+).</i> (b) <i>However the acting is weak (-).</i> (c) <i>The protagonist is a sports coach (0)</i>		
Sent (a)	1 <i>(P_{max} from softmax)</i>	0	0
Sentence (b)	0	1 <i>(P_{max} from softmax)</i>	0
Sentence (C)	0	0`	1 <i>(P_{max} from softmax)</i>

Training data

- *(a) I like the story line of the movie (+).*
- *(b) However the acting is weak (-).*
- *(c) The protagonist is a sports coach (0)*

Input

Output

(a)

$\langle 1, 0, 0 \rangle$

(b)

$\langle 0, 1, 0 \rangle$

(c)

$\langle 0, 0, 1 \rangle$

Finding the error

- Difference between target (T) and obtained (Y)
- Difference is called **LOSS**
- Options:
 - Total Sum Square Loss (TSS)
 - Cross Entropy (*measures difference between two probability distributions*)
- Softmax goes with cross entropy

Cross Entropy Function

$$H(P, Q) = - \sum_{x=1, N} \sum_{k=1, C} P(x, k) \log_2 Q(x, k)$$

x varies over N data instances, c varies over C classes
 P is target distribution; Q is observed distribution

Cross Entropy Loss

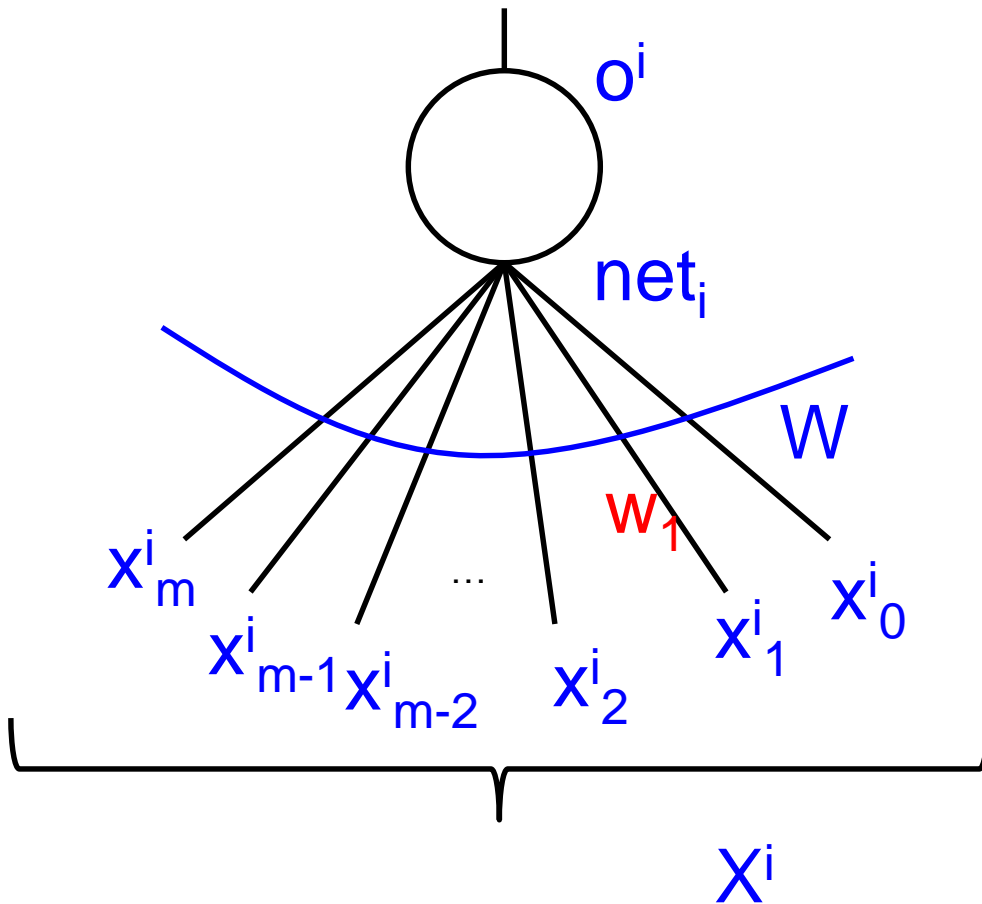
- Can we sum up cross entropies over the instances?
Is it allowed?
- Yes, summing up cross entropies (i.e. the total cross entropy loss) is equivalent to multiplying probabilities.
- Minimizing the total cross entropy loss is equivalent to maximizing the likelihood of observed data.

How to minimize loss

- Gradient descent approach
- Backpropagation Algorithm
- Involves derivative of the input-output function for each neuron
- FFNN with BP is the most important **TECHNIQUE** for us in the course

Sigmoid and Softmax neurons

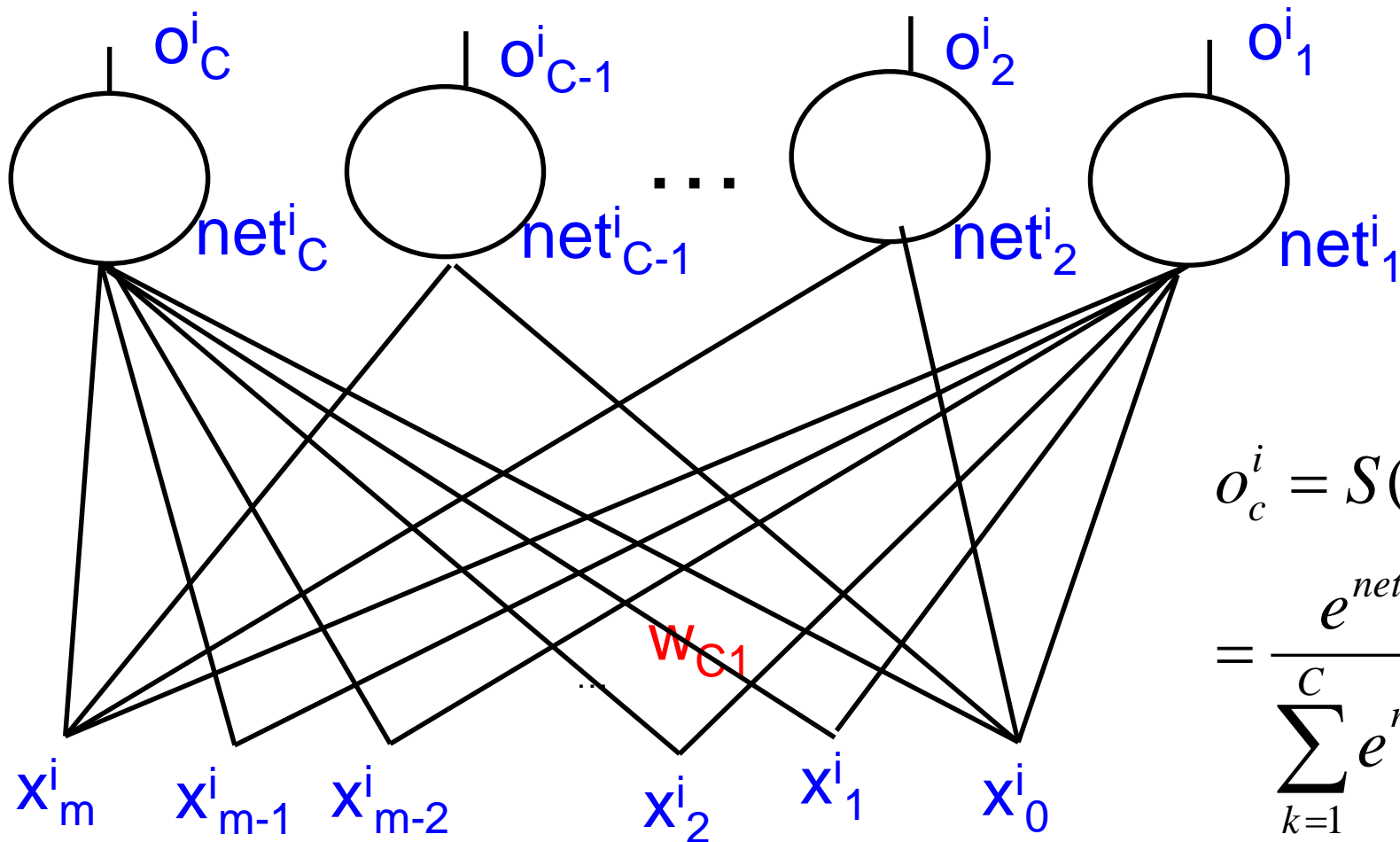
Sigmoid neuron



$$o^i = \frac{1}{1 + e^{-net^i}}$$

$$net_i = W \cdot X^i = \sum_{j=0}^m w_j x_j^i$$

Softmax Neuron



$$o_c^i = S(NET^i)_c$$

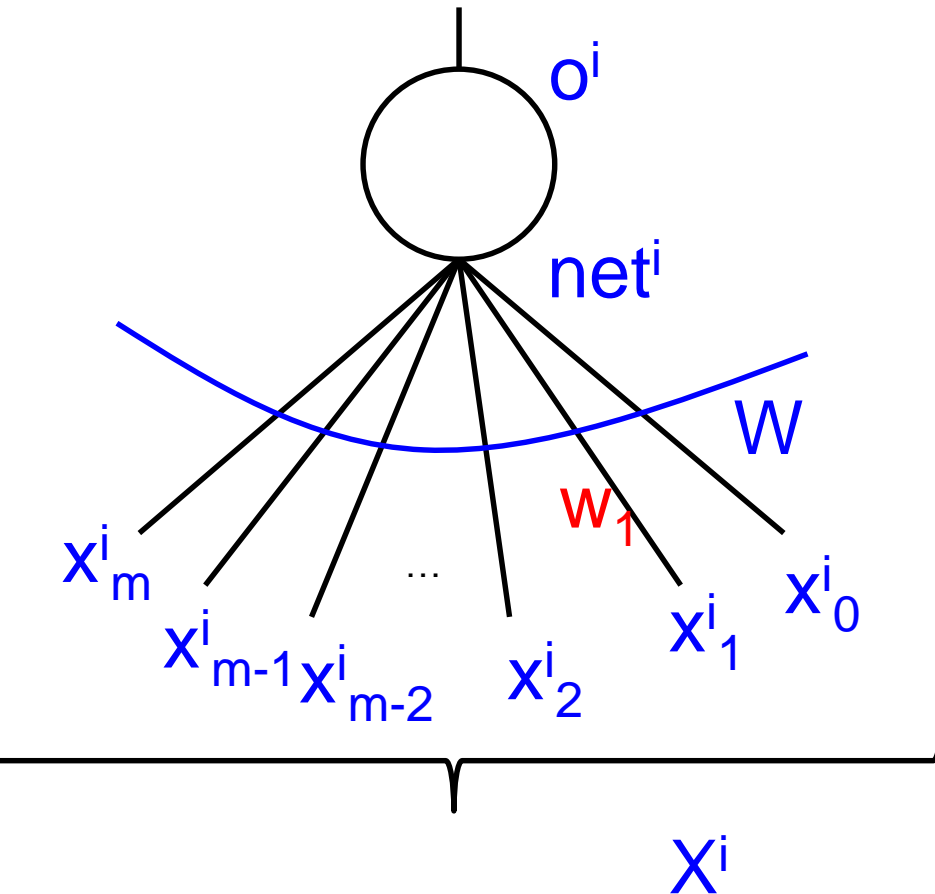
$$= \frac{e^{net_c^i}}{\sum_{k=1}^C e^{net_k^i}}$$

Output for class c (small c), c:1 to C

Notation

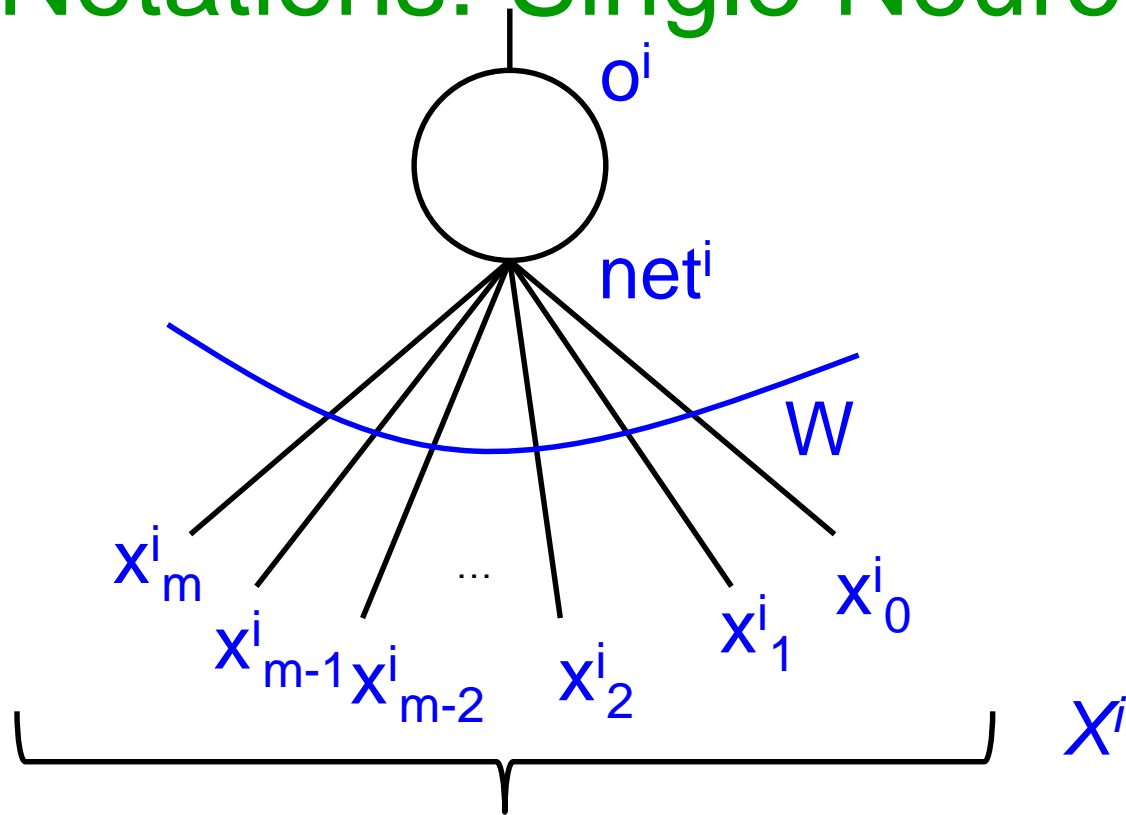
- $i=1..N$
- N i-o pairs, i runs over the training data
- $j=0...m$, m components in the input vector, j runs over the input dimension (also weight vector dimension)
- $k=1...C$, C classes (C components in the output vector)

Fix Notations: Single Neuron (1/2)



- Capital letter for vectors
- Small letter for scalars (therefore for vector components)
- X^i : i^{th} input vector
- o_i : output (scalar)
- W : weight vector
- net_i : $W \cdot X^i$
- There are n input-output observations

Fix Notations: Single Neuron (2/2)



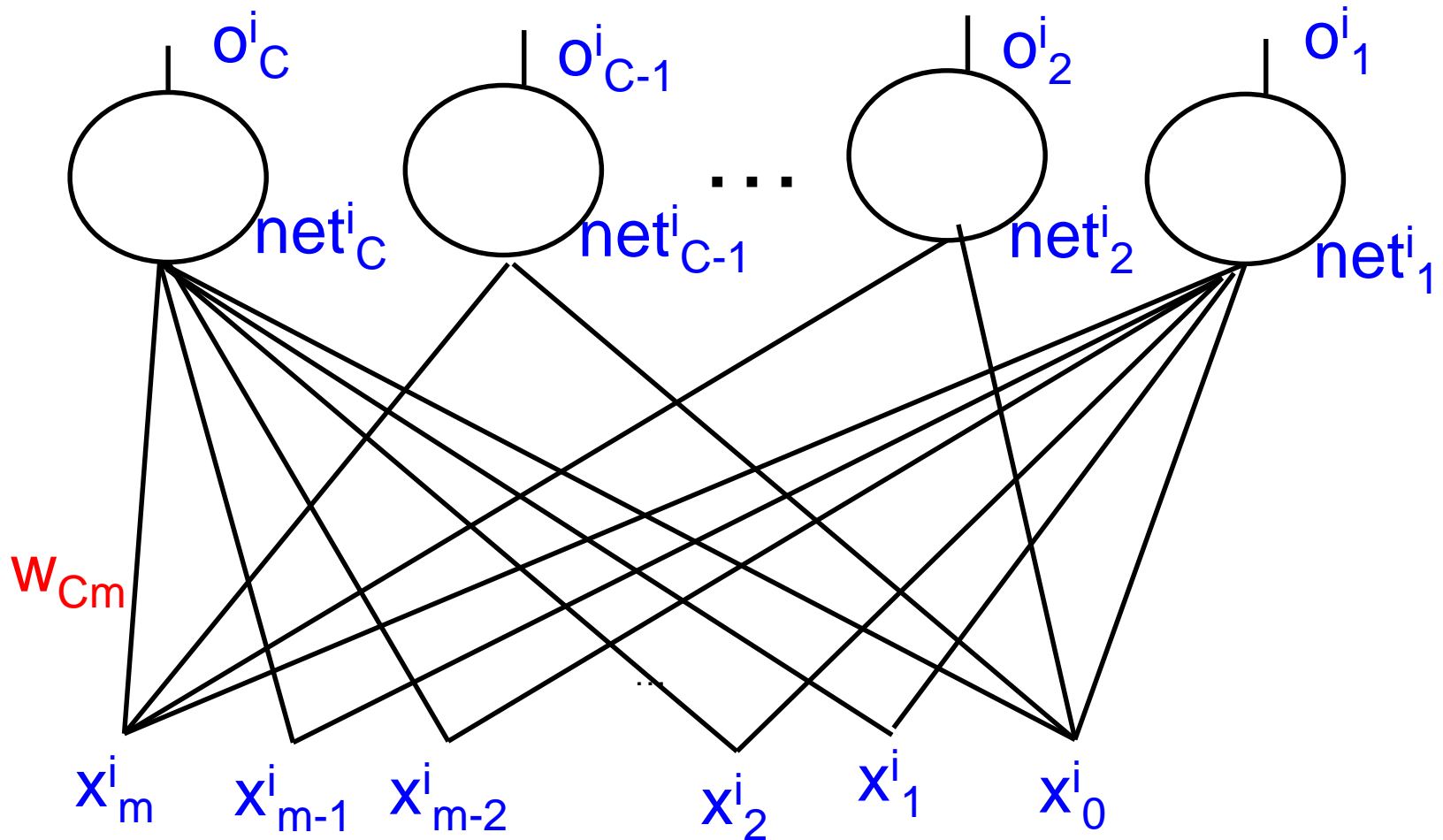
W and each X^i has m components

$W: \langle w_m, w_{m-1}, \dots, w_2, w_0 \rangle$

$X^i: \langle x_m^i, x_{m-1}^i, \dots, x_2^i, x_0^i \rangle$

Upper suffix i indicates i^{th} input

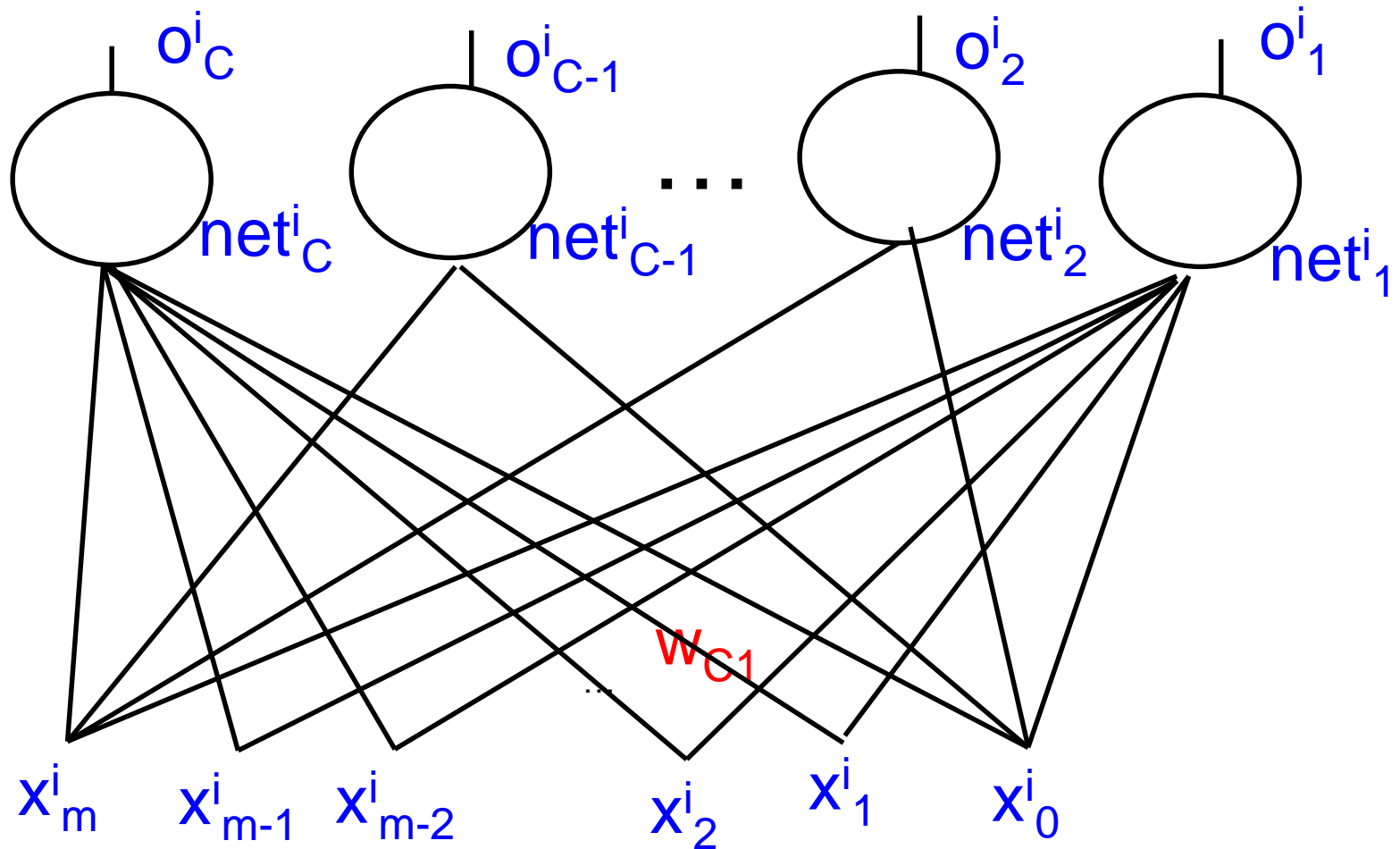
Fixing Notations: Multiple neurons in o/p layer



Now, O^i and NET^i are vectors for i^{th} input

W_k is the weight vector for c^{th} output neuron, $c=1..C$

Fixing Notations



Target Vector, $T^i: \langle t^i_c \ t^i_{c-1} \dots t^i_2 \ t^i_1 \rangle$, $i \rightarrow$ for i^{th} input. Only one of these C componets is 1, rest are 0

Derivatives

Derivative of sigmoid

$$o^i = \frac{1}{1 + e^{-net^i}}, \text{ for } i^{th} \text{ input}$$

$$\ln o^i = -\ln(1 + e^{-net^i})$$

$$\frac{1}{o^i} \frac{\partial o^i}{\partial net^i} = -\frac{1}{1 + e^{-net^i}} \cdot -e^{-net^i} = \frac{e^{-net^i}}{1 + e^{-net^i}} = (1 - o^i)$$

$$\Rightarrow \frac{\partial o^i}{\partial net^i} = o^i (1 - o^i)$$

Derivative of Softmax

$$o_c^i = \frac{e^{net_c^i}}{\sum_{k=1}^C e^{net_k^i}}, \text{ } i^{th} \text{ input pattern}$$

Derivative of Softmax: Case-1, class c for O and NET same

$$\ln o_c^i = net_c^i - \ln\left(\sum_{k=1}^C e^{net_k^i}\right)$$

$$\frac{1}{o_c^i} \frac{\partial o_c^i}{\partial net_c^i} = 1 - \frac{1}{\sum_{k=1}^C e^{net_k^i}} \cdot e^{net_c^i} = 1 - o_c^i$$

$$\Rightarrow \frac{\partial o_c^i}{\partial net_c^i} = o_c^i (1 - o_c^i)$$

Derivative of Softmax: Case-2, class c' in net_c^i , different from class c of O

$$\ln o_c^i = net_c^i - \ln\left(\sum_{k=1}^C e^{net_k^i}\right)$$

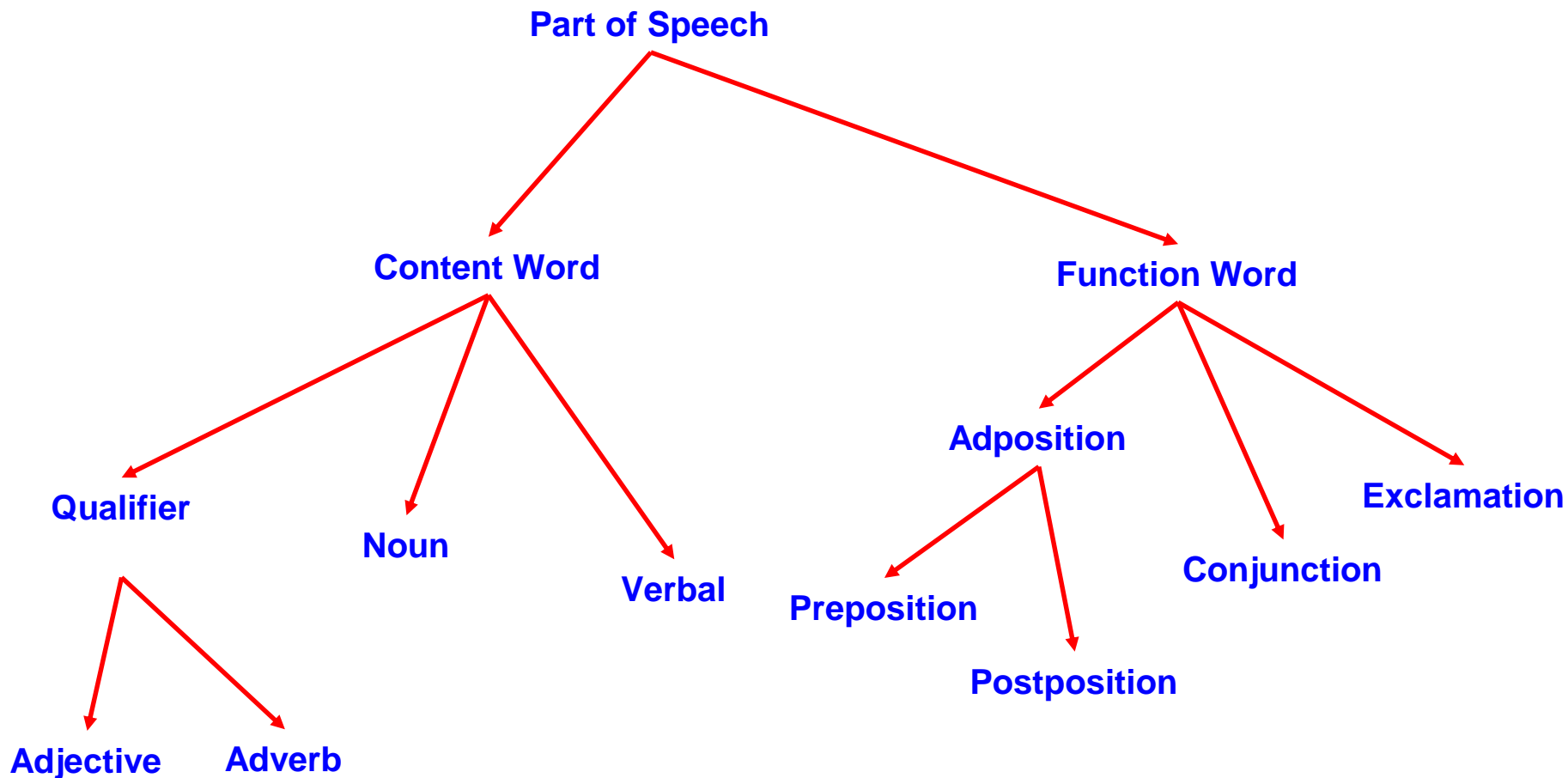
$$\frac{1}{o_c^i} \frac{\partial o_c^i}{\partial net_c^i} = 0 - \frac{1}{\sum_{k=1}^C e^{net_k^i}} \cdot e^{net_c^i} = -o_c^i$$

$$\Rightarrow \frac{\partial O_c^i}{\partial net_c^i} = -o_c^i o_c^i$$

Your first assignment

Compare and contrast HMM based, EnCo-DeCo based and LLM based POS tagging

POS hierarchy



Problem Statement

- Input: a sequence of words
- Output: a sequence of labels of these words

POS ambiguity instances

best ADJ ADV NP V

better ADJ ADV V DET

close ADV ADJ V N (*running close to the competitor, close escape, close the door, towards the close of the play*)

cut V N VN VD

even ADV DET ADJ V

grant NP N V –

hit V VD VN N

lay ADJ V NP VD

left VD ADJ N VN

like CNJ V ADJ P –

near P ADV ADJ DET

open ADJ V N ADV

past N ADJ DET P

present ADJ ADV V N

read V VN VD NP

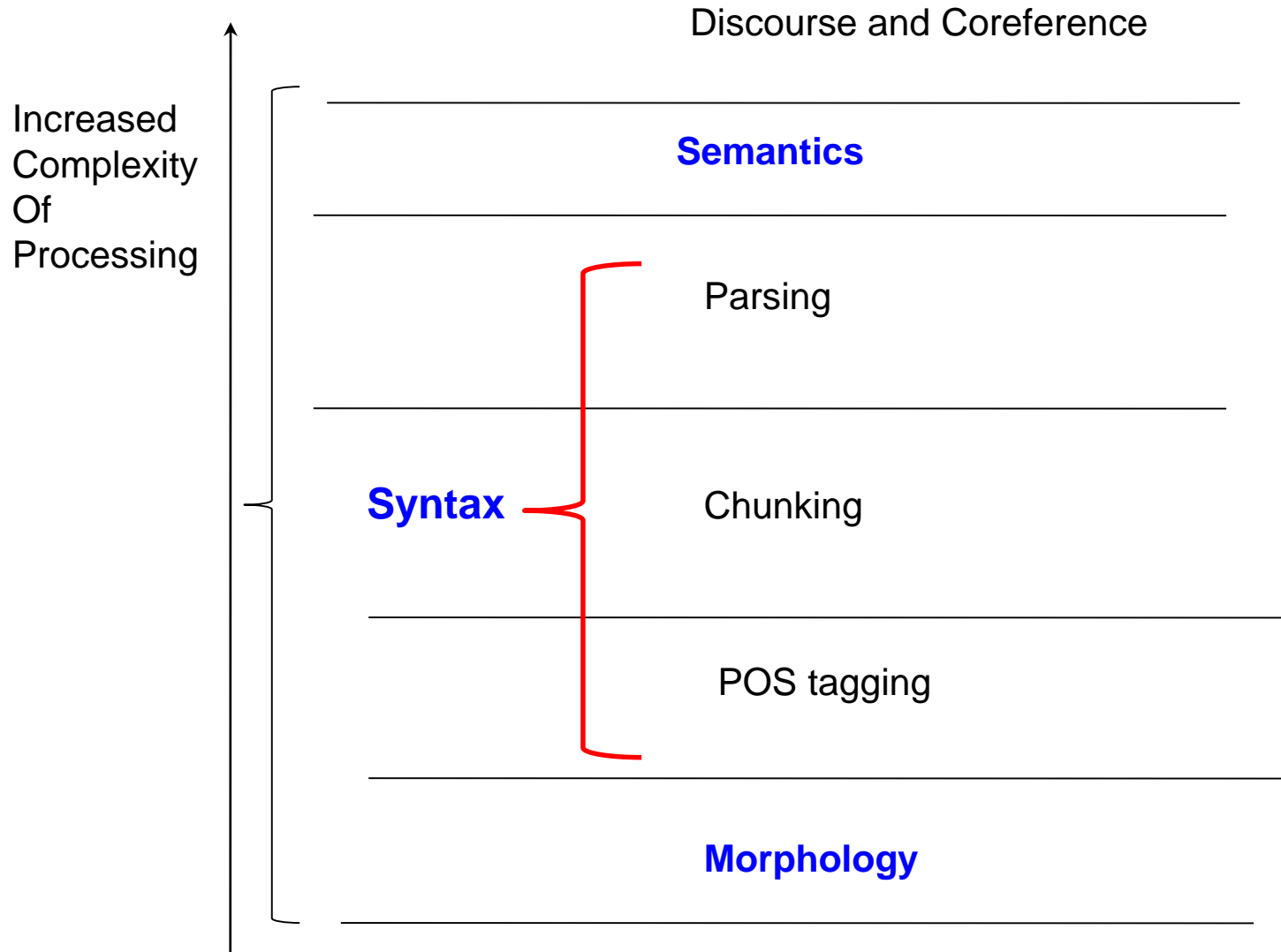
right ADJ N DET ADV

second NUM ADV DET N

set VN V VD N –

that CNJ V WH DET

NLP Layers



POS Tags- "To bank, I bank on the bank on the river bank"

- To (IN - Preposition)
- bank (VB - Verb, base form)
- , (PUNCT - Punctuation)
- I (PRP - Pronoun)
- bank (VBP - Verb, non-3rd person singular present)
- on (IN - Preposition)
- the (DT - Determiner)
- bank (NN - Noun, singular)
- on (IN - Preposition)
- the (DT - Determiner)
- river (NN - Noun, singular)
- bank (NN - Noun, singular)

Constituency Parse- "To bank, I bank on the bank on the river bank"

ChatGPT ▾



Here is the constituency parse tree for the sentence "To bank, I bank on the bank on the river bank":

```
SCPS
Copy code
(S
  (PP (TO To) (NP (NN bank)))
  (, ,)
  (S
    (NP (PRP I))
    (VP (VBP bank)
      (PP (IN on)
        (NP
          (NP (DT the) (NN bank))
          (PP (IN on)
            (NP (DT the) (NN river) (NN bank)))))))
  )
)
```

ML Based POS Tagging

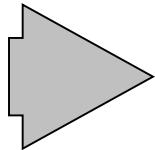
Noisy Channel Model



$(w_n, w_{n-1}, \dots, w_1)$

$(t_m, t_{m-1}, \dots, t_1)$

**Sequence W is transformed into
sequence T**



$$T^* = \underset{T}{\operatorname{argmax}} (P(T|W))$$

$$W^* = \underset{W}{\operatorname{argmax}} (P(W|T))$$

Bayes Theorem

- $P(B|A)=[P(B).P(A|B)]/P(A)$
- $P(B|A)$: Posterior Probability
- $P(B)$: Prior
- $P(A|B)$: Likelihood
- Should we work with the LHS or the RHS?

Mathematics of POS tagging

Argmax computation (1/2)

Best tag sequence

$$= T^*$$

$$= \operatorname{argmax} P(T|W)$$

$$= \operatorname{argmax} P(T)P(W|T) \quad (\text{by Baye's Theorem})$$

$$P(T) = P(t_0 = \wedge t_1 t_2 \dots t_{n+1} = .)$$

$$= P(t_0)P(t_1|t_0)P(t_2|t_1 t_0)P(t_3|t_2 t_1 t_0) \dots$$

$$P(t_n|t_{n-1} t_{n-2} \dots t_0)P(t_{n+1}|t_n t_{n-1} \dots t_0)$$

$$= P(t_0)P(t_1|t_0)P(t_2|t_1) \dots P(t_n|t_{n-1})P(t_{n+1}|t_n)$$

$$\prod$$

$$= \prod_{i=0}^{N+1} P(t_i|t_{i-1})$$

Bigram Assumption

Argmax computation (2/2)

$$P(W|T) = P(w_0|t_0-t_{n+1})P(w_1|w_0t_0-t_{n+1})P(w_2|w_1w_0t_0-t_{n+1}) \dots \\ P(w_n|w_0-w_{n-1}t_0-t_{n+1})P(w_{n+1}|w_0-w_nt_0-t_{n+1})$$

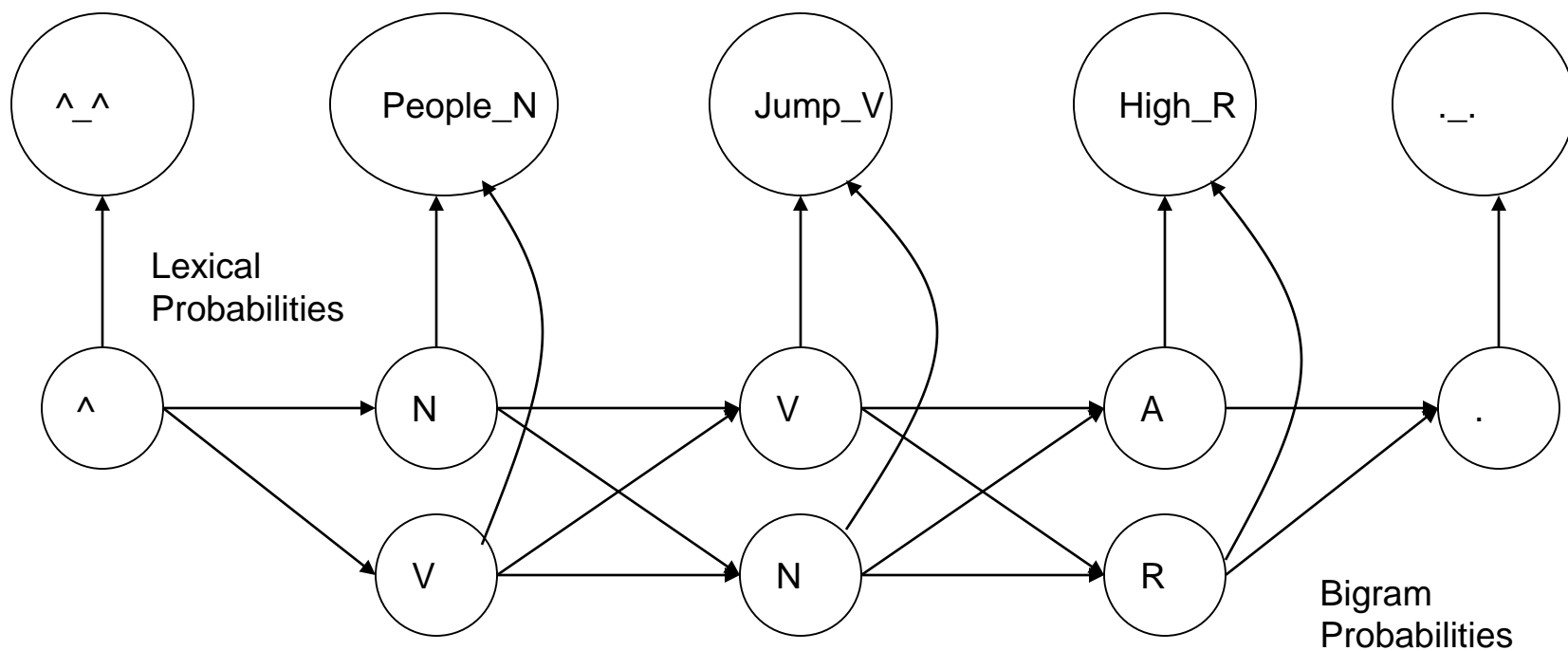
Assumption: A word is determined completely by its tag. This is inspired by speech recognition

$$= P(w_0|t_0)P(w_1|t_1) \dots P(w_{n+1}|t_{n+1})$$

$$\prod_{i=0}^{n+1} P(w_i|t_i)$$

$$\prod_{i=1}^{n+1} P(w_i|t_i) \quad (\text{Lexical Probability Assumption})$$

Generative Model

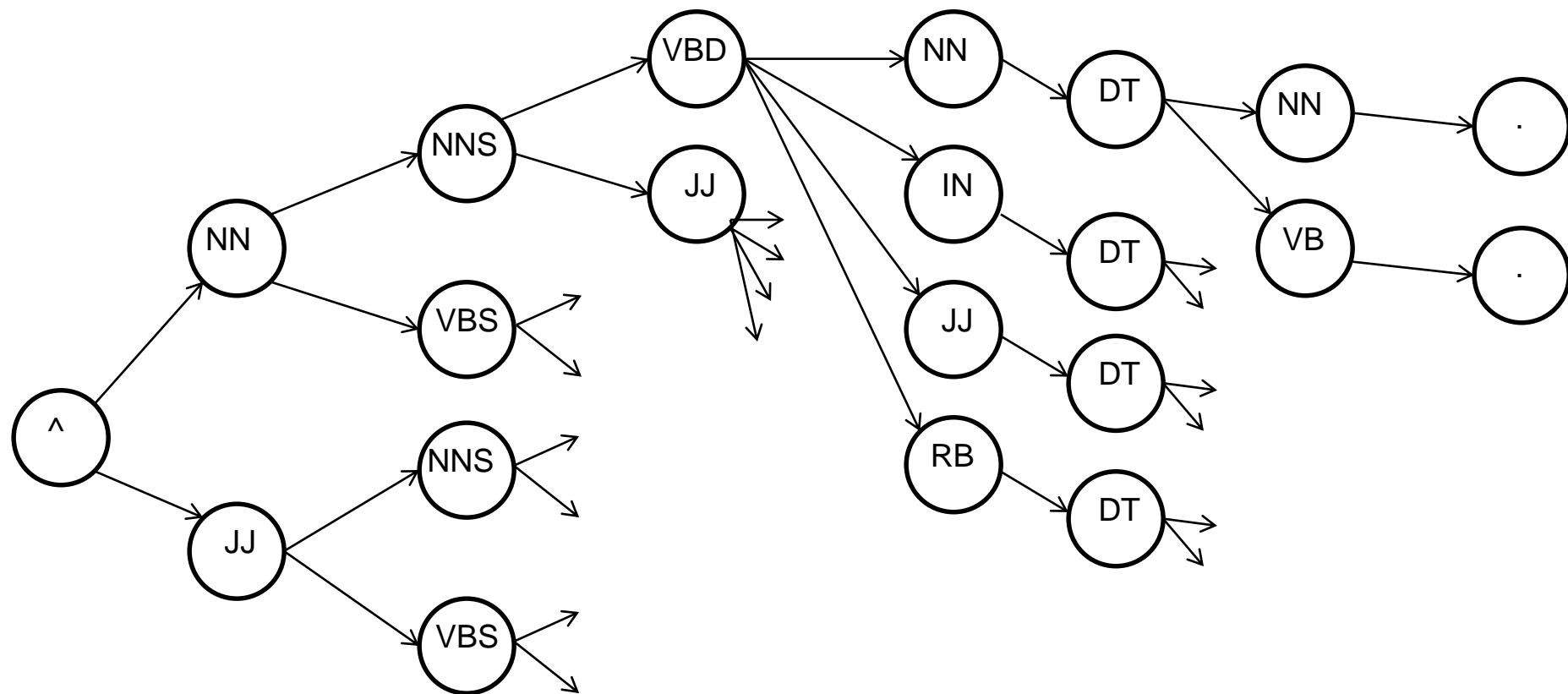


This model is called Generative model.
Here words are observed from tags as states.
This is similar to HMM.

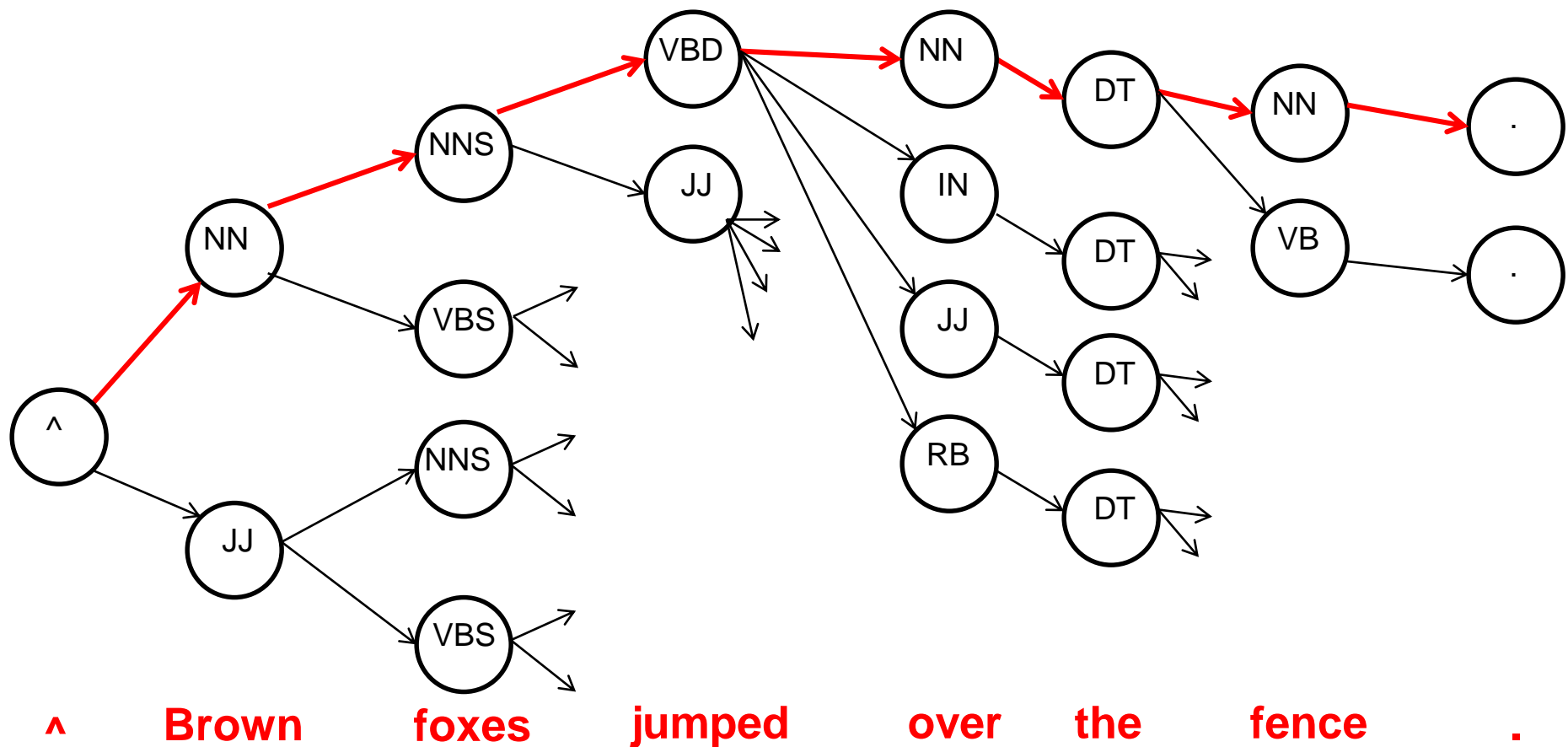
Computation of POS tags

DECODING

W:	^	Brown	foxes	jumped	over	the	fence	.
T:	^	JJ	NNS	VBD	NN	DT	NN	.
		NN	VBS	JJ	IN		VB	
					JJ			
					RB			



^ Brown foxes jumped over the fence .



Probability of a path (e.g. Top most path) = $P(T) * P(W|T)$

$$P(^) . P(NN|^) . P(NNS|NN) . P(VBD|NNS) . P(NN|VBD) . \\ P(DT|NN) . P(NN|DT) . P(.|NN) . P(.)$$

*

$$P(^|^) . P(brown|NN) . P(foxes|NNS) . P(jumped|VBD) . \\ P(over|NN) . P(the|DT) . P(fence|NN) . P(.|.)$$

Questions?

- Where do tags come from?
 - Tag set
- How to get probability values i.e. $P(.)$?
 - Annotated corpora

After modeling of the problem,
emphasis should be on the corpus

Computing P(.) values

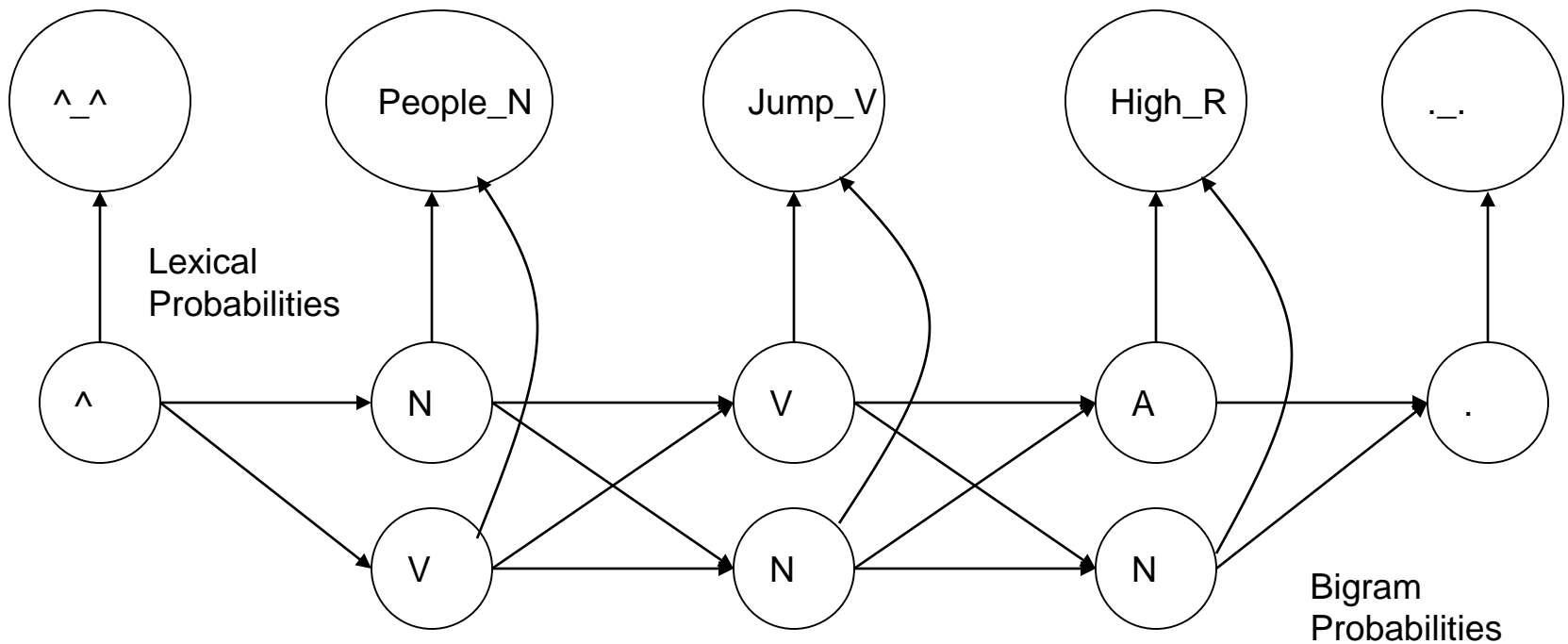
Let us suppose annotated corpus has the following sentence

I	have	a	brown	bag	.
PRN	VB	DT	JJ	NN	.

$$P(NN | JJ) = \frac{\text{Number_of_times_JJ_followed_by_NN}}{\text{Number_of_times_JJ_appeared}}$$

$$P(\text{Brown} | JJ) = \frac{\text{Number_of_times_Brown_tagged_as_JJ}}{\text{Number_of_times_JJ_appeared}}$$

HMM: Generative Model: Decode by Viterbi



This model is called Generative model.
Here words are observed from tags as states.
This is similar to HMM.

Your Assignment- due 30aug25

- Take POS tag data from NLTK
<https://www.nltk.org/>
- Use HMM, EnCo-DeCo and any LLM of your choice to compare performance
- You can use pre-written/available/LLM-generated code, but you will have to explain what the code is doing
- Later you will be asked to implement something innovative on/based-on POS tagging which will require you to code