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

Cross Entropy Loss and Softmax, Start of RNN

Pushpak Bhattacharyya
Computer Science and Engineering Department
IIT Bombay

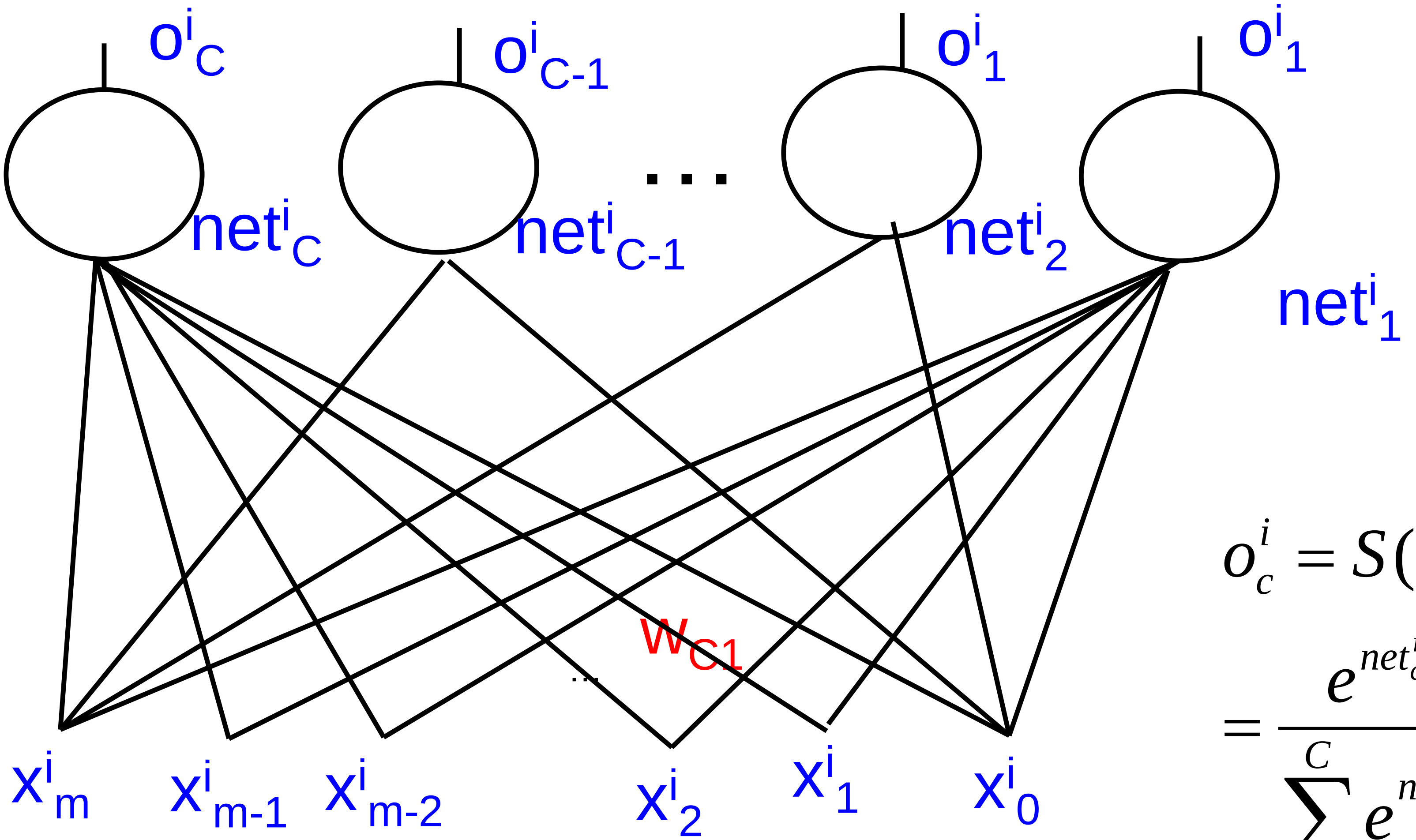
Week 6 of 7th Feb, 2022

multiclass: SOFTMAX

$$o_c^i = S(\bar{NET}_i)_c = \frac{e^{net_c^i}}{\sum_{k=1}^{C} e^{net_k^i}},$$

- 2-class → multi-class (C classes)
- Sigmoid → softmax
- *i*th input, *c*th class (small c), *k* varies over classes

Softmax Neuron



$$o_c^i = S(\overline{NET}_i)_c = \frac{e^{net_c^i}}{\sum_{k=1}^{C} e^{net_k^i}},$$

Target Vector, T_i: $<t^i_C\ t^i_{C-1} \ldots t^i_2\ t^i_1>$, $i \rightarrow$ for i^{th} input.

Only one of these C componets is 1, rest are 0.

Compare and contrast Sigmoid and Softmax

$$sigmoid : o_i = \frac{1}{1+e^{-net_i}},\ for\ i^{th}\ input$$

$$soft\max : o_c^i = \frac{e^{net_c^i}}{\sum_{k=1}^{C} e^{net_k^i}},$$

ith input, *c*th class (small c), *k* varies over classes 1 to *C*

Interpreting o^i_c

- o^i_c value is between 0 and 1
- Interpreted as probability
- Multi-class situation
- o^i_c value is the probability of the class being *'c'* for the i^{th} input

- That is,

  $$P(\text{Class of } i^{th} \text{ input}=c)=o^i_c$$

Derivatives

Derivative of Softmax

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

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

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

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

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

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

Derivative of Softmax: Case-2, class *c'* in $net^i_{c'}$ different from class c of O

$$\ln o^i_c = net^i_c - \ln\left(\sum_{k=1}^{C} e^{net^i_k}\right)$$

$$\frac{1}{o^i_c}\frac{\partial o^i_c}{\partial net^i_{c'}} = 0 - \frac{1}{\sum_{k=1}^{C} e^{net^i_k}}.e^{net^i_{c'}} = -o^i_{c'}$$

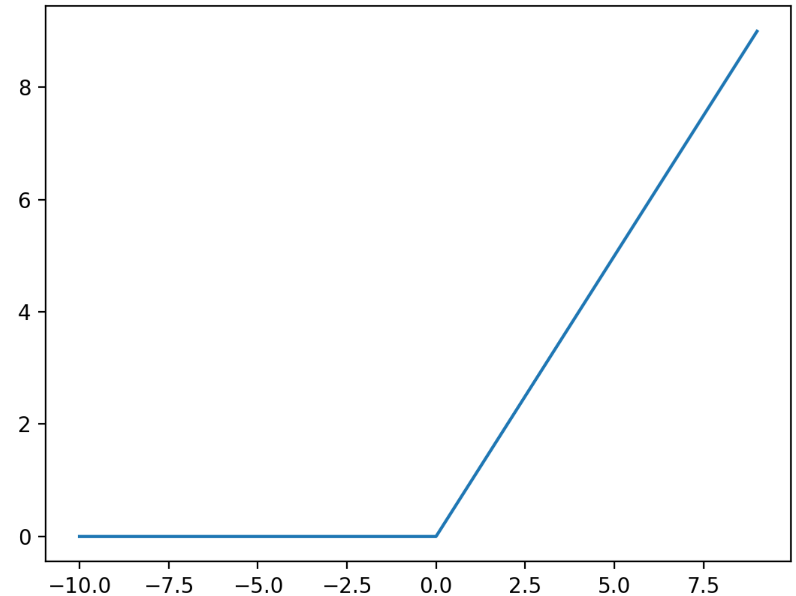
$$\Rightarrow \frac{\partial O^c_k}{\partial net^i_{c'}} = -o^i_c o^i_{c'}$$

FFNN: Working with RELU

Rectifier Linear Unit

What is RELU

y=relu(x)=max(0,x)



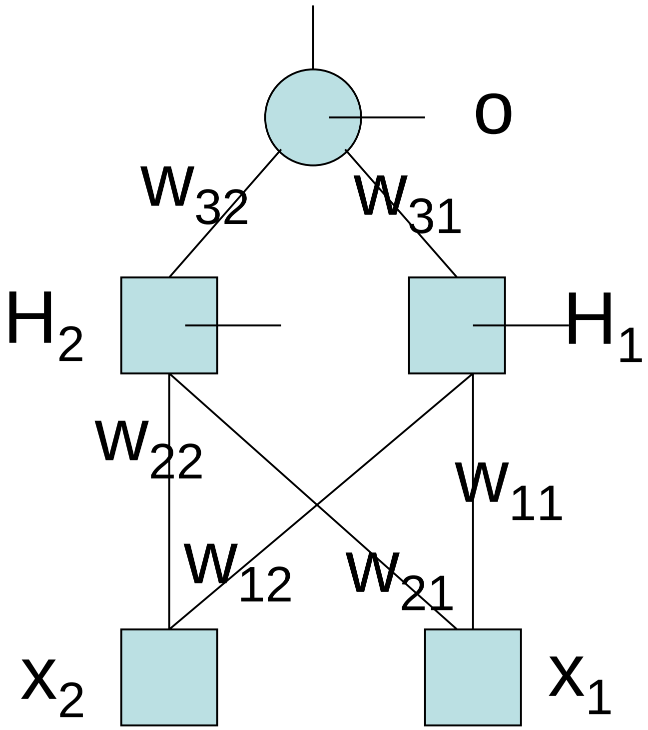
dy/dx

= *0 for x<o*

= *1 for x>0*

= *0* (forced to be 0 at x=0, though does not exit)

Output sigmod and hidden neurons as RELU



$$\Delta w_{ji} = -\eta \frac{\delta E}{\delta w_{ji}}$$

$$\eta = \text{learning rate}, 0 \le \eta \le 1$$

$$\frac{\delta E}{\delta w_{ji}} = \frac{\delta E}{\delta net_j} \times \frac{\delta net_j}{\delta w_{ji}}$$

$$net_j = \text{input at the j}^{th} \text{ neuron)}$$

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

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

Backpropagation – for outermost layer

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

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

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

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

Backpropagation – for hidden layers

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

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

$$= -\frac{\delta E}{\delta o_j} \times (1\, or\ 0)$$

$$= -\sum_{k \in \text{next layer}} \left(\frac{\delta E}{\delta net_k} \times \frac{\delta net_k}{\delta o_j}\right) \times (1\, or\ 0)$$

This recursion can give rise to vanishing and exploding Gradient problem

$$\text{Hence, } \delta_j = -\sum_{k \in \text{next layer}} (-\delta_k \times w_{kj}) \times (1\, or\ 0)$$

$$= \sum_{k \in \text{next layer}} (w_{kj} \delta_k)\ or\ 0$$

Backpropagation Rule for weight change with RELU, Sigmoid and TSS

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

$$\delta_j = (t_j - o_j) o_j (1 - o_j)$$ for outermost layer

$$= \sum_{k \in \text{next layer}} (w_{kj} \delta_k) \; or \; 0$$ for hidden layers

Softmax, Cross Entropy and RELU

Cross Entropy Function

$$H(P,Q) = -\sum_{x} P(x) \log_2 Q(x)$$

P is target distribution, *Q* is observed distribution

e.g., Positive, Negative, Neutral Sentiment

x: input sentence: *The movie was excellent*

P(x): <1,0,0>, Q(x): <0.9,0.02,0.08>, (say)

H(P,Q)=-log0.9=log(10/9)

Deriving weight change rules

Cross Entropy Softmax combination

A very ubiquitous combination in neural combination

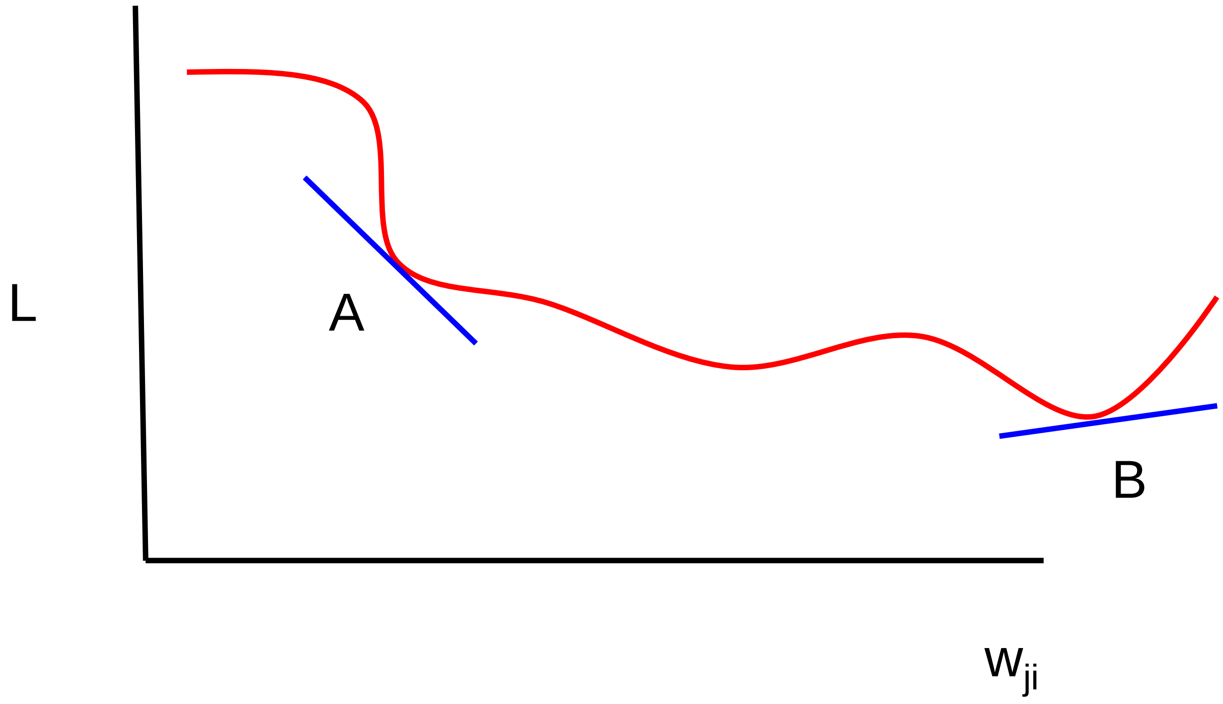
Foundation: Gradient descent

Change is weight $\Delta w_{ji} = -\eta \delta L / \delta w_{ji}$

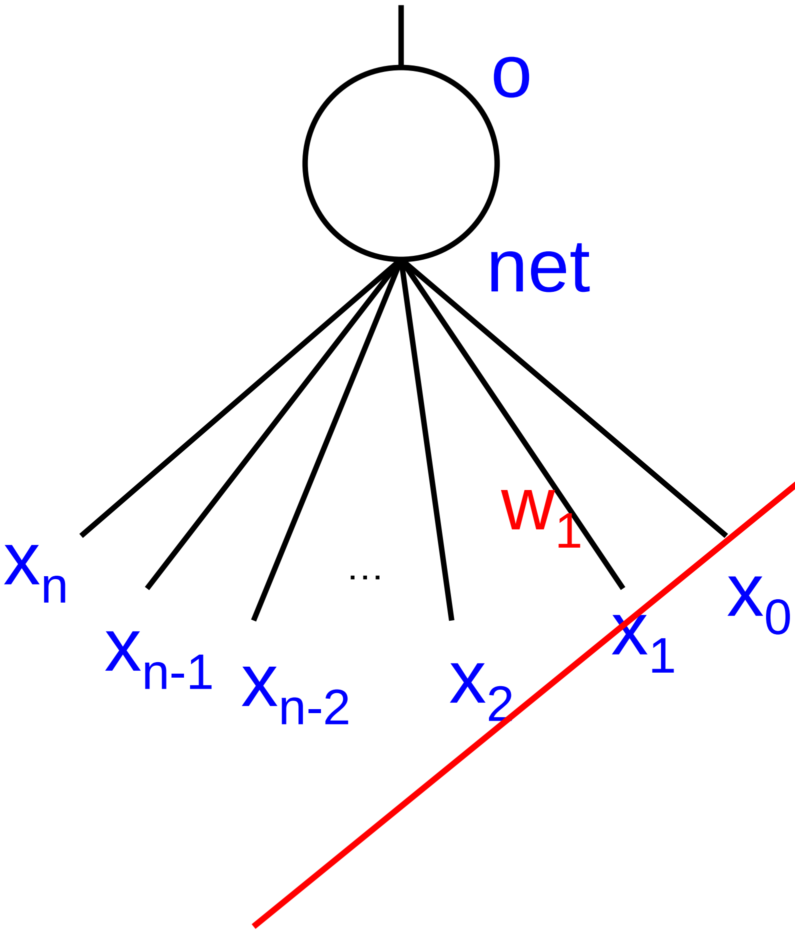
η= *learning rate, L=loss,*

w_{ji}= *weight of connection from the i^{th} neuron to j^{th}*

At A, $\delta L / \delta w_{ji}$ *is negative, so* Δw_{ji} *is positive. At B,* $\delta L / \delta w_{ji}$ *Is positive, so so* Δw_{ji} *is negative. L *always decreases. Greedy algo.*



Single neuron: *sigmoid+cross entropy* loss



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

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

$$\Rightarrow \frac{\partial L}{\partial o} = -\frac{t}{o} + \frac{1-t}{1-o} = -\frac{t-o}{o(1-o)} \quad (1)$$

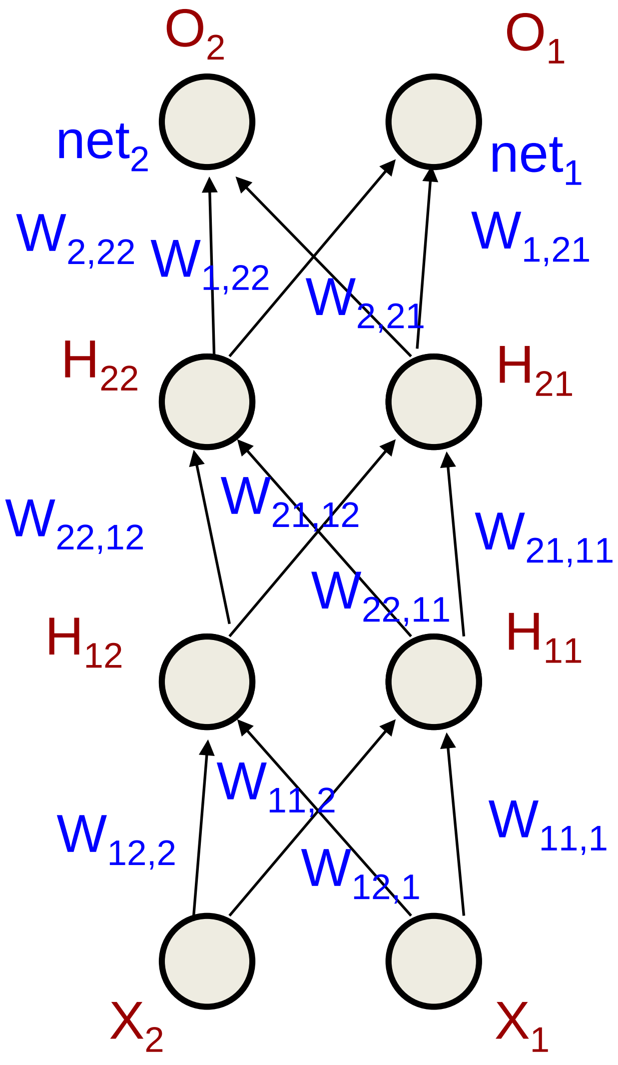
$$o = \frac{1}{1+e^{-net}} \, (sigmoid) \Rightarrow \frac{\partial o}{\partial net} = o(1-o) \; (2)$$

$$net = \sum_{i=0}^{n} w_i x_i \quad \Rightarrow \frac{\partial net}{\partial w_1} = x_1 \quad (3)$$

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

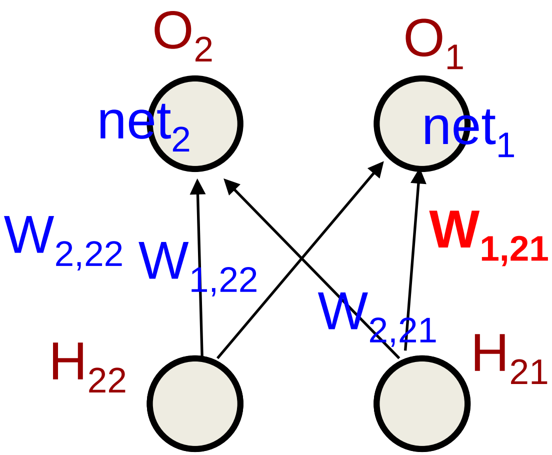
$$\Delta w_1 = \eta(t\text{-}o)x_1$$

FFNN with O_1-O_2 softmax, all hidden neurons RELU, Cross Entropy Loss



We will apply the $\Delta w_{ji}=\eta\delta_j o_i$ rule

Gradient Descent Rule and the General Weight Change Equation



$$\Delta w_{1,21} = \eta \delta_{o_1} h_{21}$$

$$\delta_{o_1} = -\frac{\delta E}{\delta net_1}$$

$$E = -t_2 \log o_2 - t_1 \log o_1$$

$$\frac{\partial E}{\partial net_1} = \frac{\partial E}{\partial o_1} \cdot \frac{\partial o_1}{\partial net_1} + \frac{\partial E}{\partial o_2} \cdot \frac{\partial o_2}{\partial net_1}$$

$$= -\frac{t_1}{o_1} o_1 (1 - o_1) + (-\frac{t_2}{o_2})(-o_1 o_2)$$

$$= -t_1 (1 - o_1) + t_2 o_1$$

$$= -t_1 o_2 + t_2 o_1 = -(t_1 - o_1)$$

$$\Rightarrow \delta_{O_1} = (t_1 - o_1)$$

$$Similarly, \delta_{O_2} = (t_2 - o_2)$$

$$\boxed{\Delta W_{1,21} = \eta (t_1 - o_1) h_{21}}$$

Weight Change for Hidden Layer, $W_{21,11}$

$$\Delta w_{21,11} = -\eta \frac{\partial E}{\partial w_{21,11}} = \eta \delta_{H_{21}} h_{11}$$

$$\delta_{H_{21}} = -\frac{\partial E}{\partial net_{H_{21}}}$$

$$\frac{\partial E}{\partial net_{H_{21}}} = \frac{\partial E}{\partial h_{21}} . \frac{\partial h_{21}}{\partial net_{H_{21}}} ; h_{21} = output(H_{21})$$

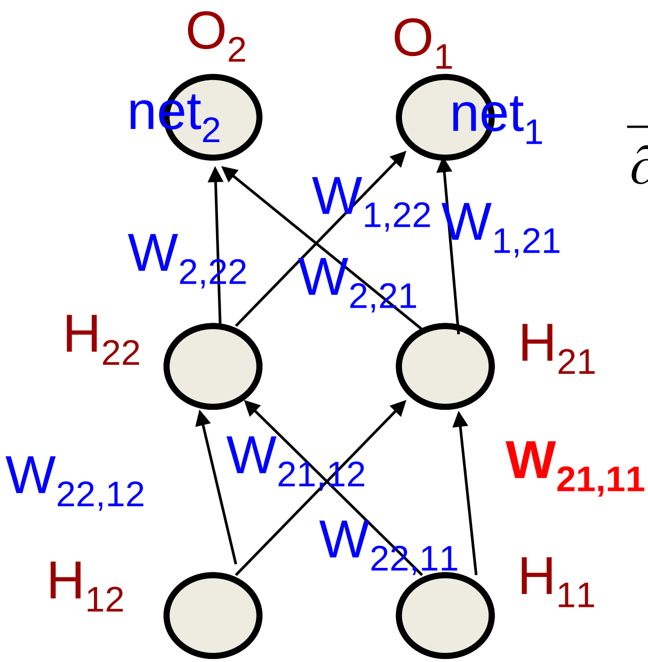
$$= \frac{\partial E}{\partial h_{21}} .r'(H_{21}); \quad r' = derivative\_RELU(H_{21})$$

$$\frac{\partial E}{\partial h_{21}} = \frac{\partial E}{\partial net_1} . \frac{\partial net_1}{\partial h_{21}} + \frac{\partial E}{\partial net_2} . \frac{\partial net_2}{\partial h_{21}}$$

$$= (-\delta_{o_1}).W_{1,21} + (-\delta_{o2}).W_{2,21}$$

$$\Rightarrow \delta_{H_{21}} = (\delta_{o_1}.W_{1,21} + \delta_{o2}.W_{2,21}).r'(H_{21})$$

$$= backpropagated\_delta.RELU\_derivative$$

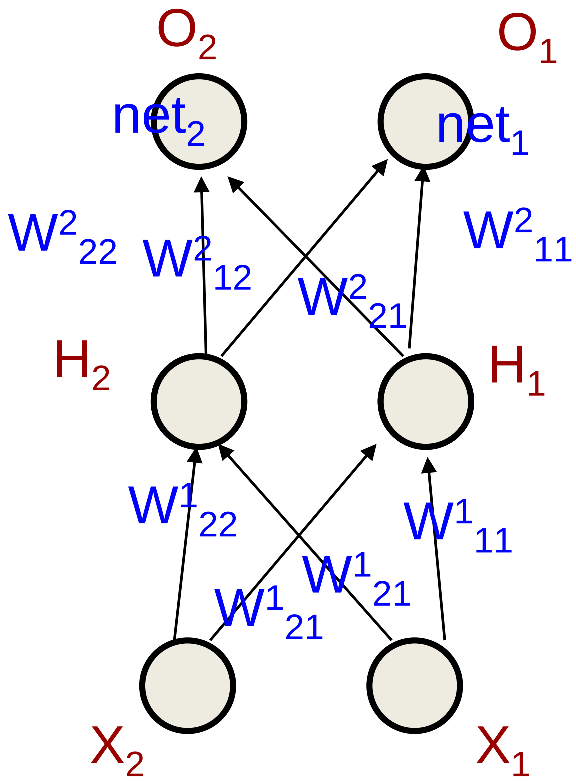


$$\boxed{\Delta W_{21,11} = \eta[(t_2 - o_2)W_{2,21} + (t_1 - o_1)W_{1,21}].r'(H_{21}).h_{11}}$$

2021: Midsem questions on FFNN (10, 11, 12)

There is a pure feedforward network 2-2-2 (2 input, 2 hidden and 2 output neurons). Input neurons are called X_1 and X_2 (right to left when drawn on paper, X_1 to the right of X_2). Similarly hidden neurons are H_1 and H_2 (right to left) and output neurons are O_1 and O_2 (right to left). H_1 and H_2 are RELU neurons. O_1 and O_2 form a softmax layer.

Remember: weight change rules



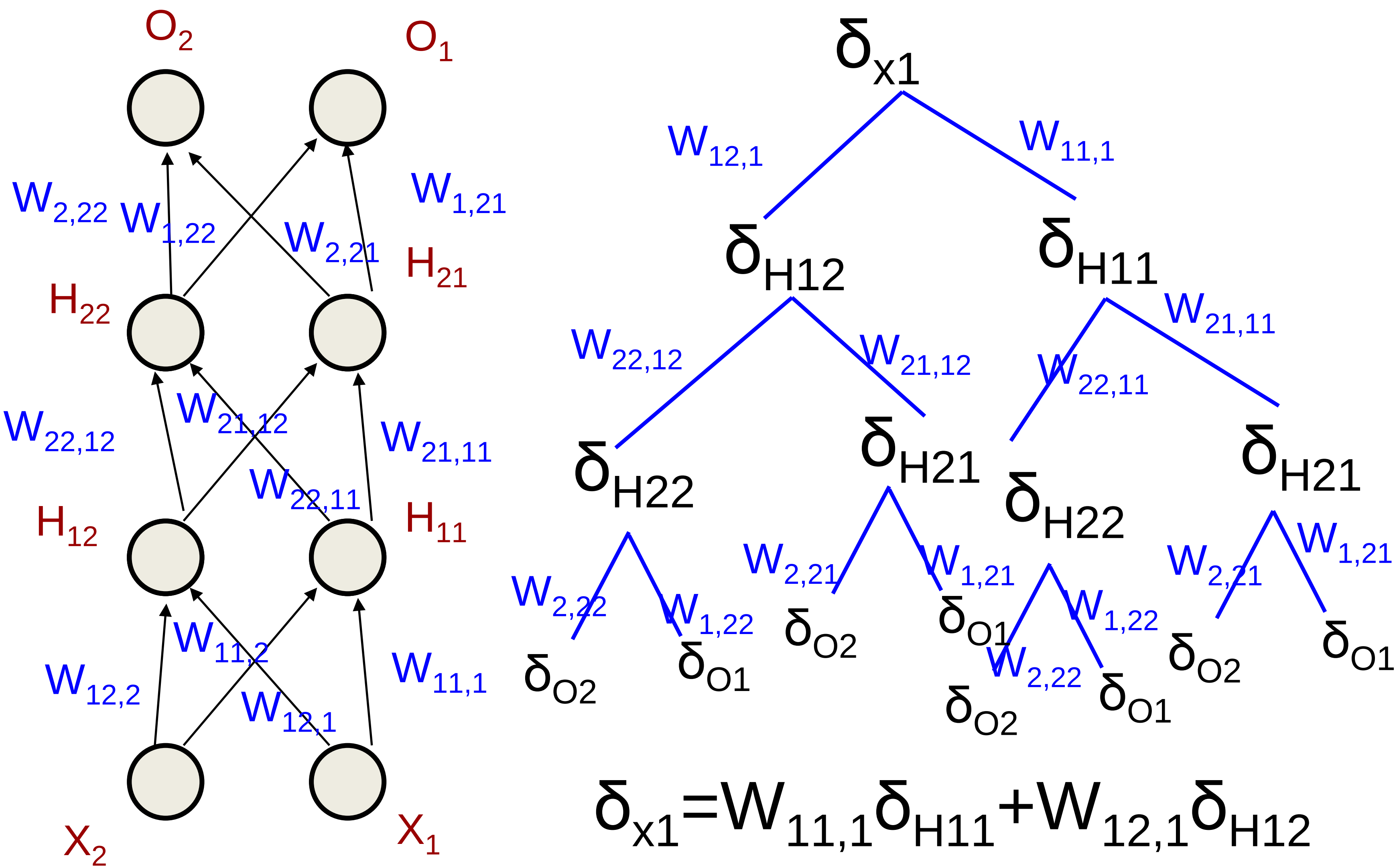
$$E = -t_2 \log o_2 - t_1 \log o_1$$

$\Delta W^2_{11} = \eta(t_1 - o_1)h_1$

$\Delta W^1_{11} = \eta[(t_2 - o_2)W^2_{21} + (t_1 - o_1)W^1_{11}].r'(H_1).h_1$

Why is RELU a solution for vanishing or exploding gradient?

Vanishing/Exploding Gradient



$$\delta_{x1}=W_{11,1}\delta_{H11}+W_{12,1}\delta_{H12}$$

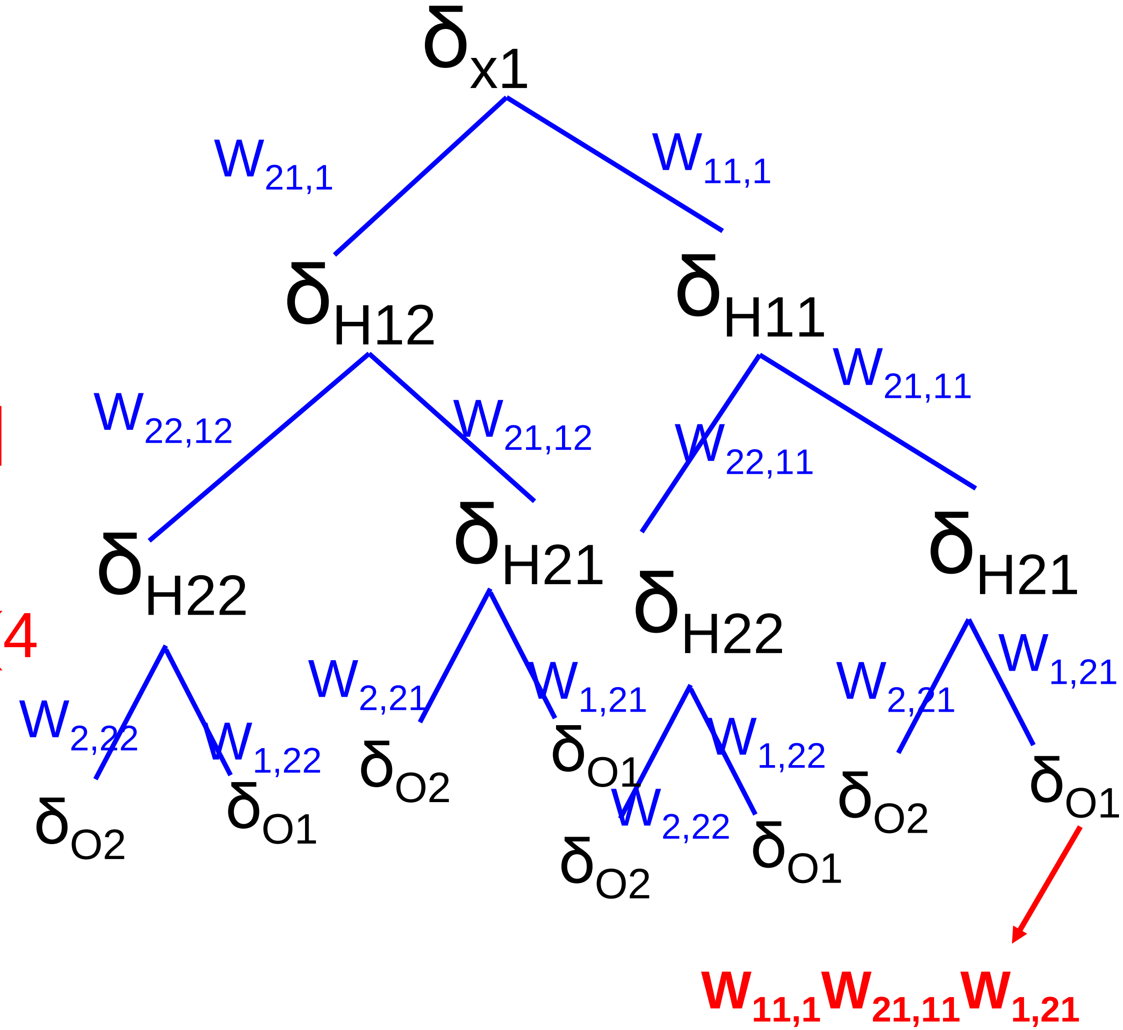
Vanishing/Exploding Gradient

$\delta_{x1}=W_{11,1}\delta_{H11}+W_{21,1}\delta_{H12}$ [2 terms]

$=W_{11,1}(W_{21,11}\delta_{H21}+ W_{22,11}\delta_{H22}).r'(H_{11})+ W_{21,1}(W_{21,12}\delta_{H21}+ W_{22,12}\delta_{H22}).\ r'(H_{12})$ [4 terms]

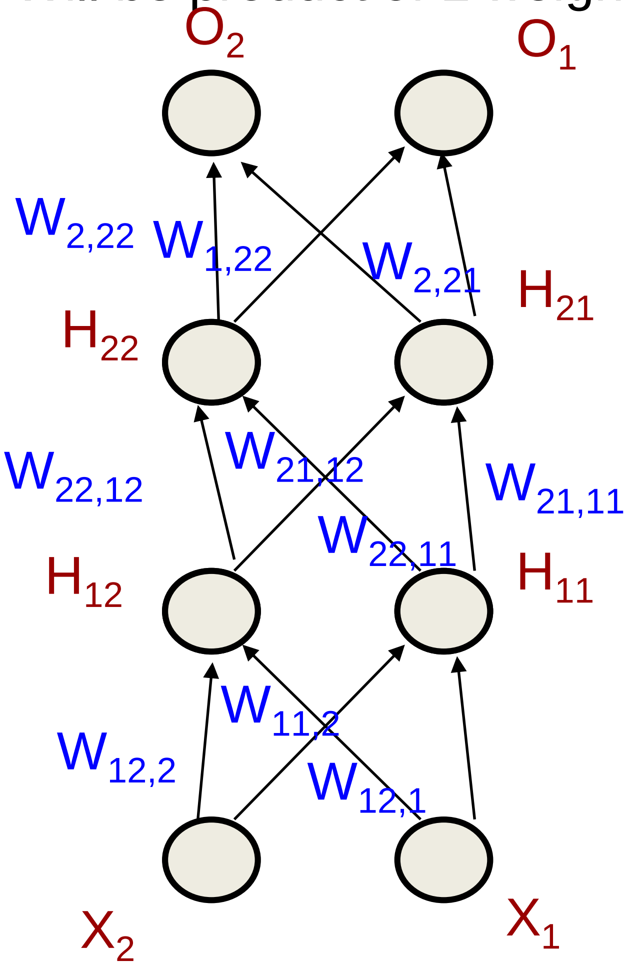
= (4 terms involving δ_{o1}) + (4 terms involving δ_{o2})

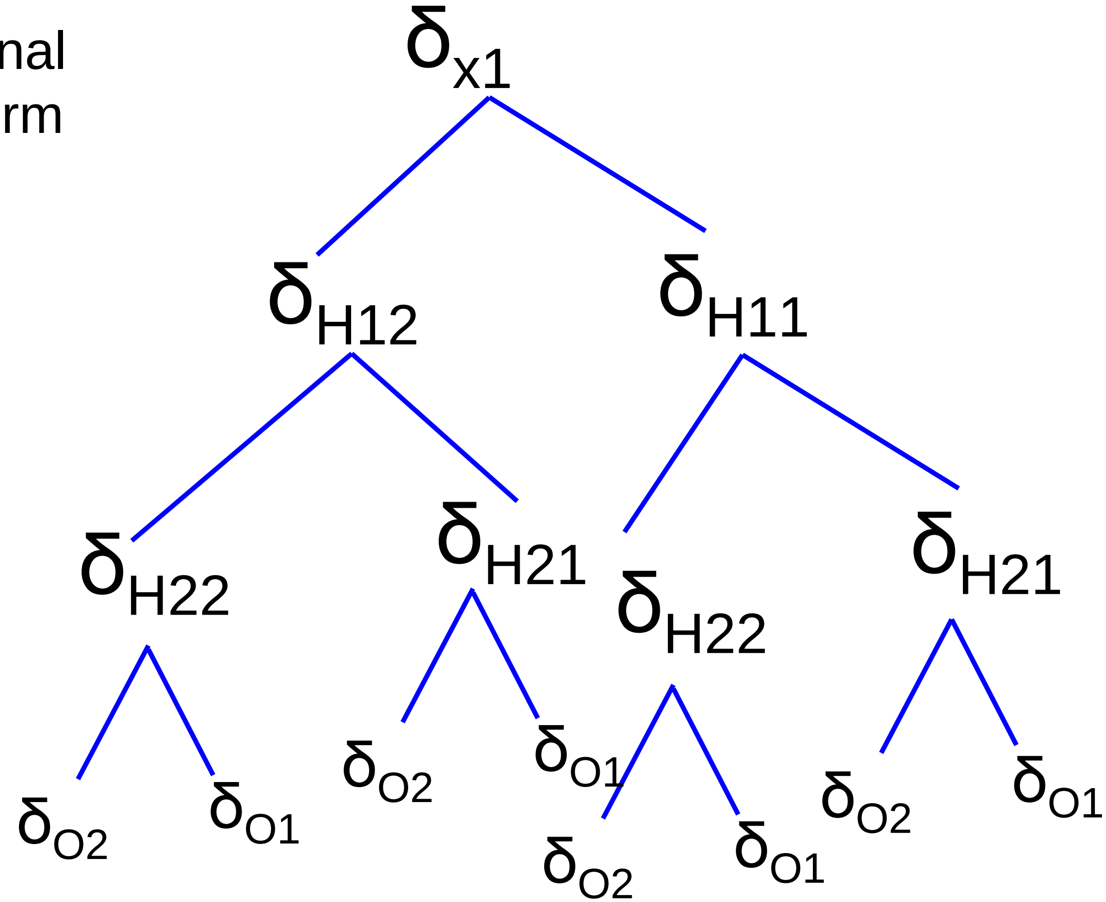
δs get multiplied by derivatives of RELU which are 1 or 0; hence δs from the output layer pass as such or as 0



Vanishing/Exploding Gradient

With '*B*' as branching factor and
'*L*' as number of levels,
There will be B^L terms in the final
Expansion of δ_{x1}. Also each term
Will be product of L weights





How can gradients explode

- Station derivatives multiply
- If <0, progressive attenuation of product
- Now the sigmoid function can be in the form of $y=K[1/(1+e^{-x})]$
- Derivative= $K.y.(1-y)$
- If K is more than 1, the product of gradients can become larger and larger, leading to explosion of gradient
- K needs to be >1, to avoid saturation of neurons

Can happen for *tanh* too

- Tanh: $y=[(e^x-e^{-x})/(e^x+e^{-x})]$
- Derivative= (1-y)(1+y)
- If we take a neuron with *K.tanh*, we can again have explosion of gradient if K>1
- Why *K* needs to be >1?
- To take care of situations where #inputs and individual components of input are large
- This is to avoid saturation of the neuron

Recurrent Neural Network

Acknowledgement:

1. http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

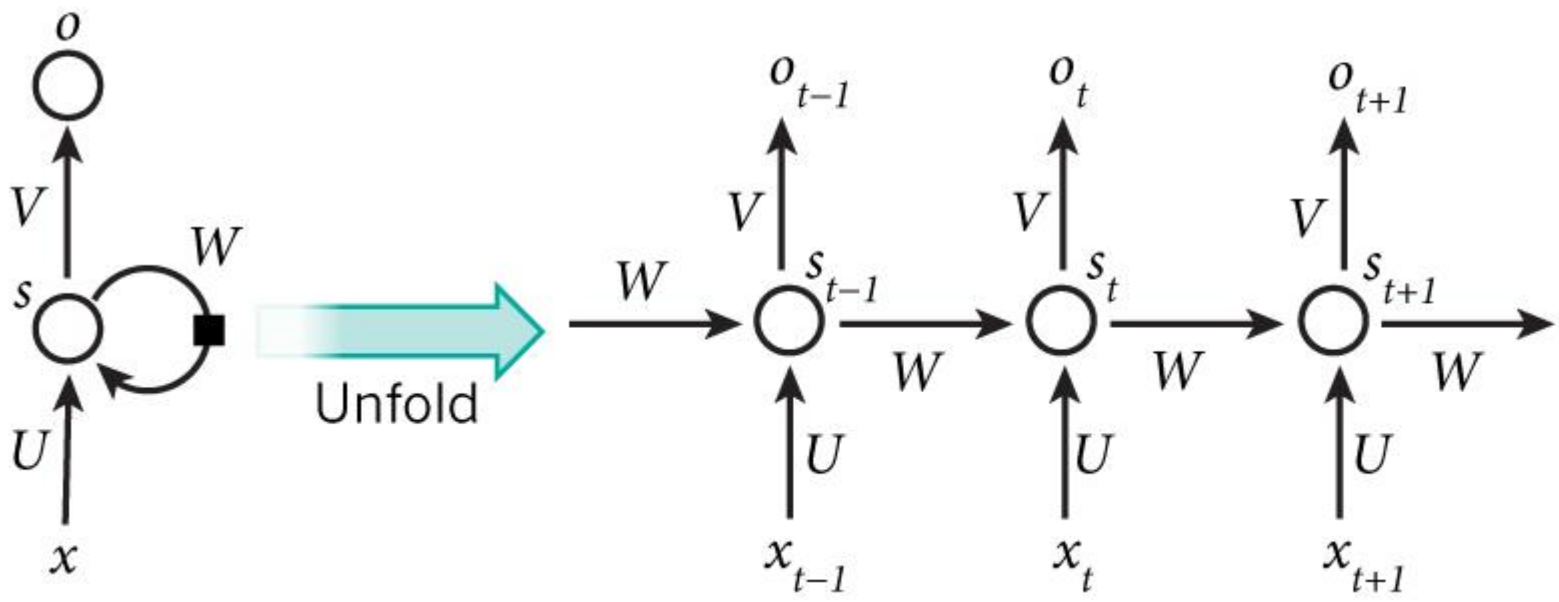
   By Denny Britz

2. Introduction to RNN by Jeffrey Hinton

http://www.cs.toronto.edu/~hinton/csc2535/lectures.html

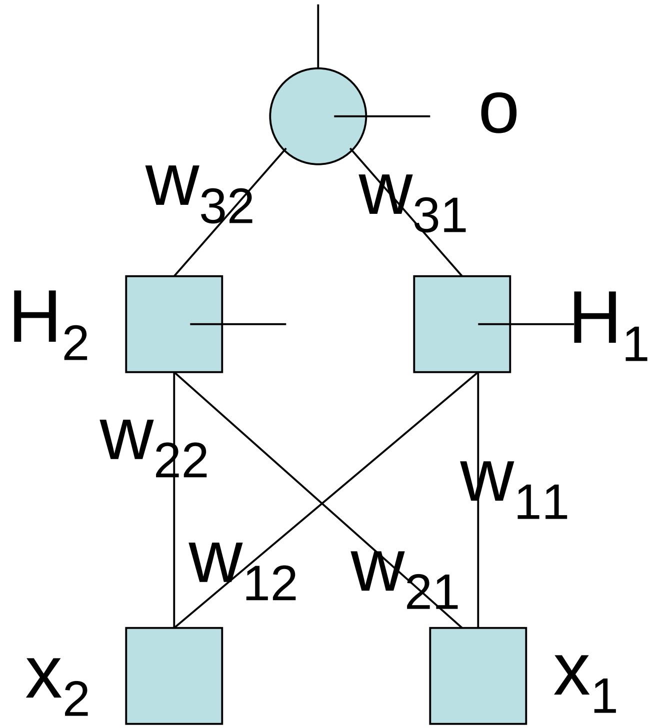
3. Dr. Anoop Kunchukuttan, Microsoft and ex-CFILT

Sequence processing m/c

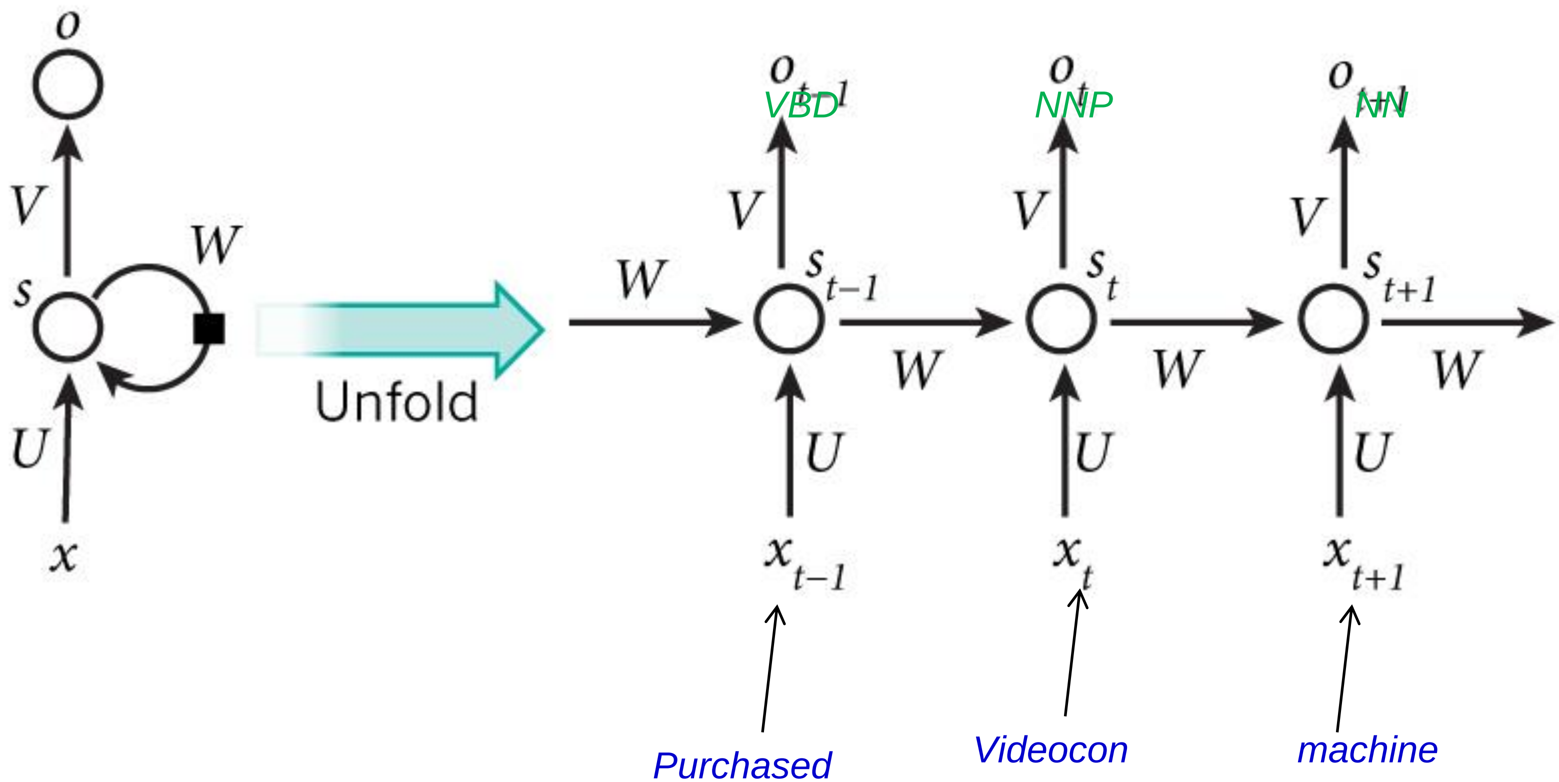


Meaning of state

- State vector → constituted of states of neurons
- State of a neuron → activation, i.e., output of the neuron corresponding to an input
- E.g., state vector for the XOR n/w is <*h*$_1$, *h*$_2$, *o*>



E.g. POS Tagging



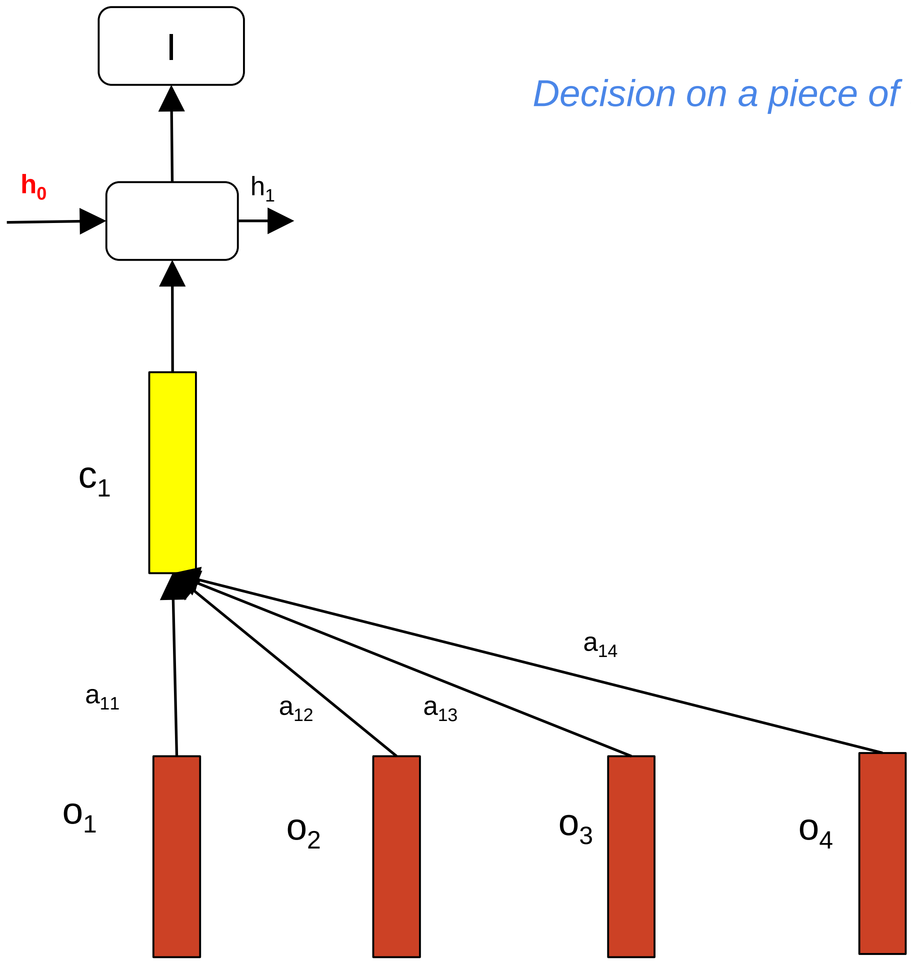
Note that POS of “purchased” is ambiguous

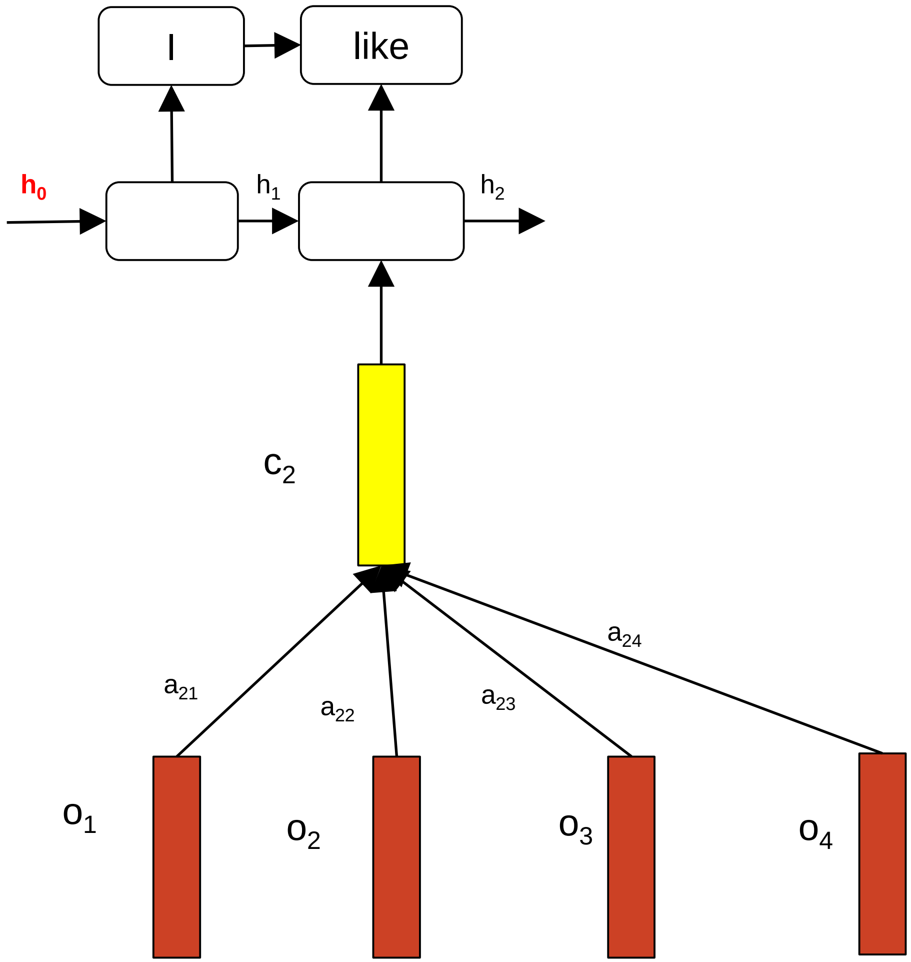
With possibilities as VBD or VBN or JJ (“I purchased Videocon machine” vs.

“my purchased Videocon machine is running well”

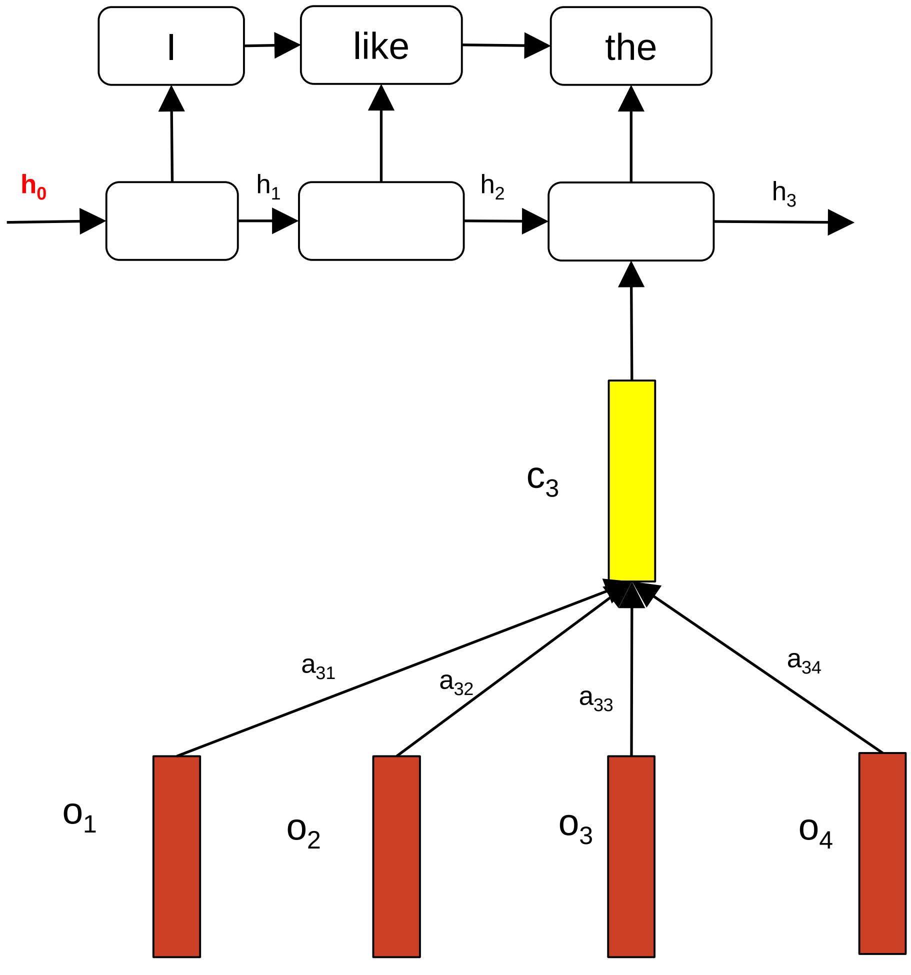
E.g. Sentiment Analysis

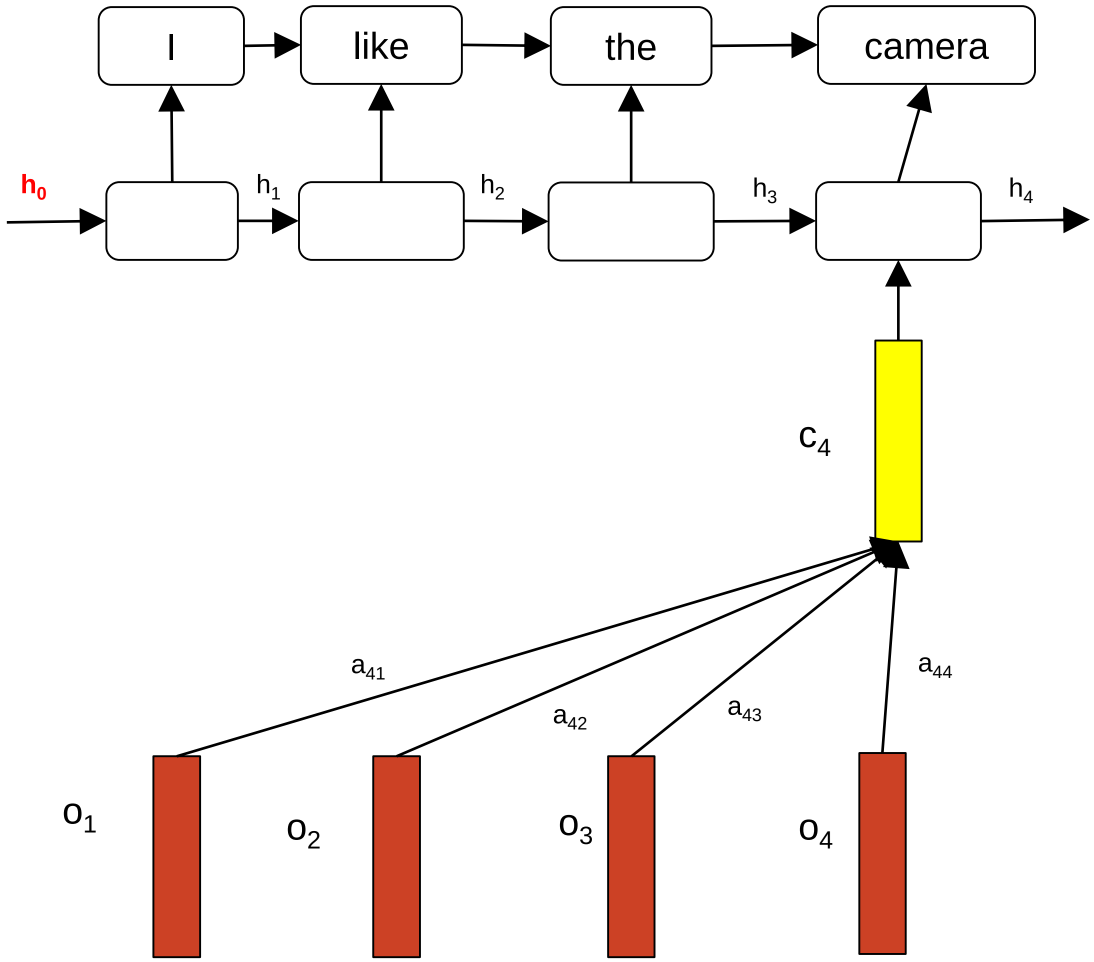
Decision on a piece of text

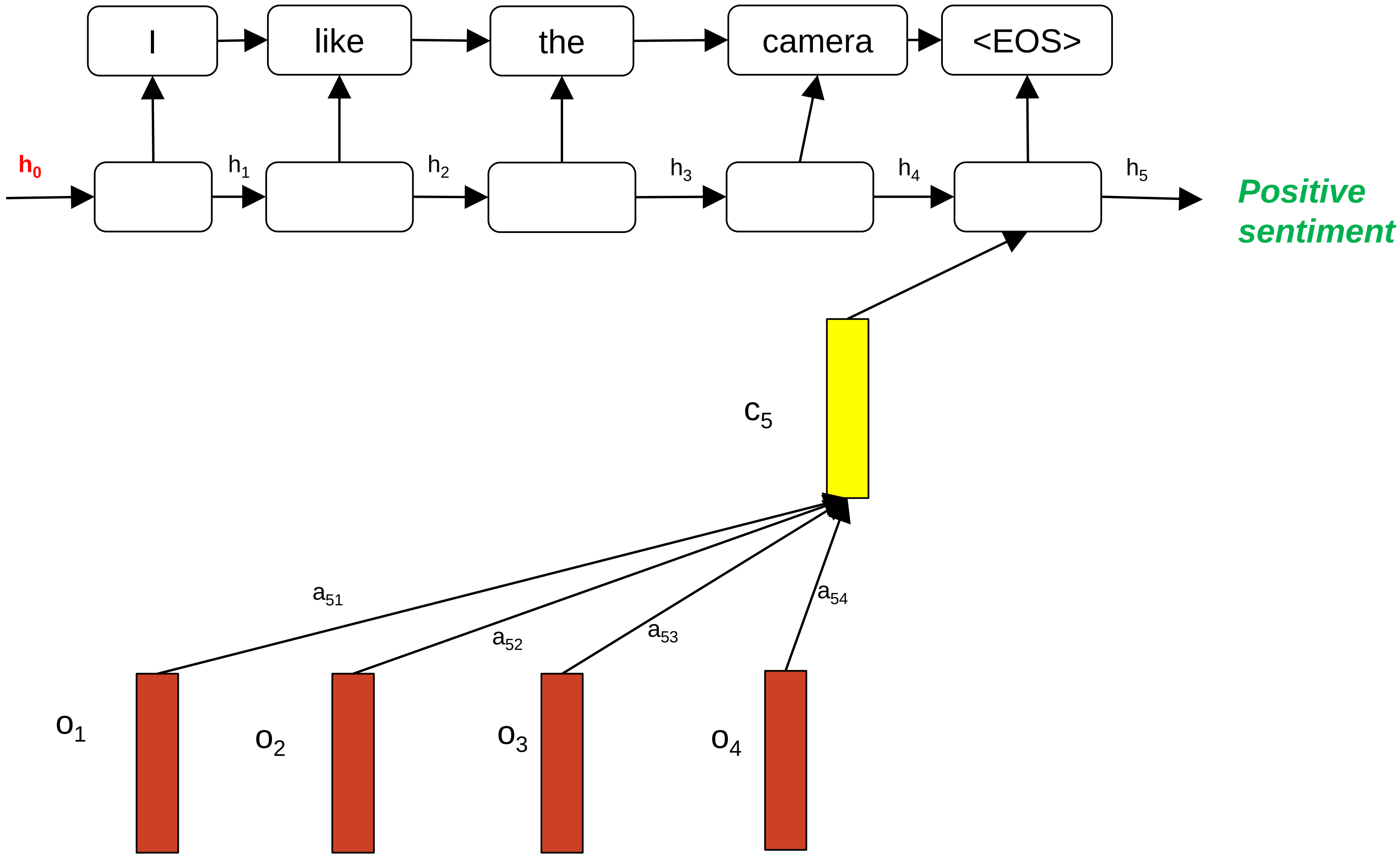




I
like
h0
h1
h2
c2
a21
a22
a23
a24
o1
o2
o3
o4


I
like
the
h0
h1
h2
h3
c3
a31
a32
a33
a34
o1
o2
o3
o4

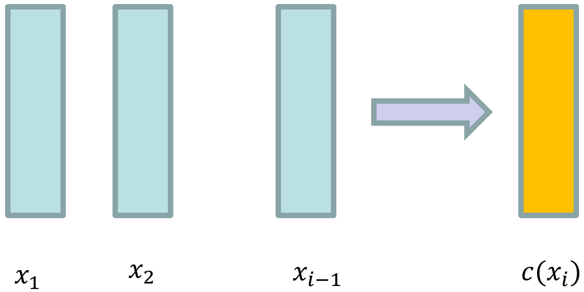

I
like
the
camera
h0
h1
h2
h3
h4
c4
a41
a42
a43
a44
o1
o2
o3
o4



I
like
the
camera
<EOS>
h_0
h_1
h_2
h_3
h_4
h_5
Positive sentiment
c_5
a_{51}
a_{52}
a_{53}
a_{54}
o_1
o_2
o_3
o_4

Recurrent Neural Networks: two key Ideas

1. Summarize context information into a single vector



$$c(x_i) = F(x_1, x_2, \dots, x_{i-1})$$

$$P(x_i | c(x_i))$$

Function G requires all context inputs at once

How does RNN address this problem?

Nature of $\boldsymbol{P}(.)$

n-gram LM: look-up table

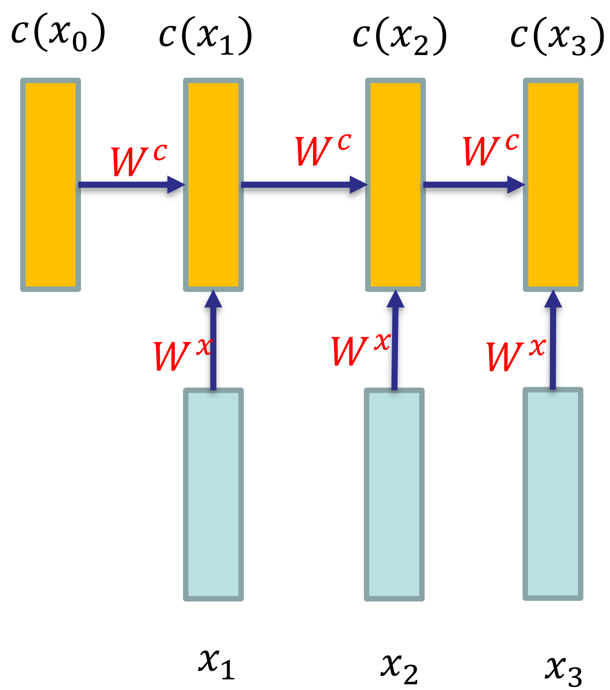
FF LM: $c(x_i) = G(x_{i-1}, x_{i-2})$ (trigram LM)

RNN LM: $c(x_i) = F(x_1, x_2, \dots, x_{i-1})$ (unbounded context)

Two Key Ideas (cntd)

2. Recursively construct the context

$$c(x_i) = F(c(x_{i-1}), x_i)$$



We just need two inputs to construct the context vector:

- Context vector of previous timestep
- Current input

The context vector ➔ state/hidden state/contextual representati

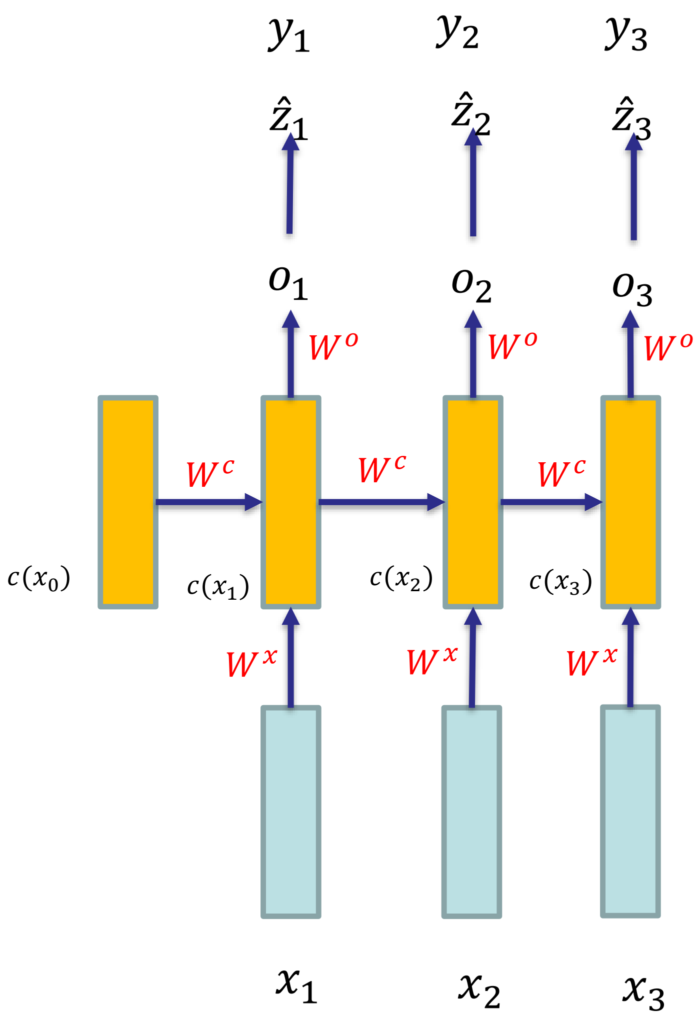
$F(.)$ can be implemented as

$$c(x_i) = \sigma(W^c c(x_{i-1}) + W^x x_i + b_1)$$

Like a feed-forward network

Generate output give the current input and state/context

W^o=wt. for output layer;
W^c= wt. for generating next state (context);
W^x= wt. for the input layer



$$o(x_i) = W^o c(x_i) + b_2$$

We are generally interested in categorical outputs

$$\hat{z}_i = softmax(o(x_i)) \\ = P(y_i | ctx(x_i))$$

$$\widehat{z_i^w} = P(y_i = w | ctx(x_i))$$

The same parameters are used at each time-step

Model size does not depend on sequence length

Long range context is modeled

Sequence Labelling Task

Input Sequence: $(x_1 \;\; x_2 \;\; x_3 \;\; x_4 \ldots\ldots x_i \ldots\ldots x_N)$

Output Sequence: $(y_1 \;\; y_2 \;\; y_3 \;\; y_4 \ldots\ldots y_i \ldots\ldots y_N)$

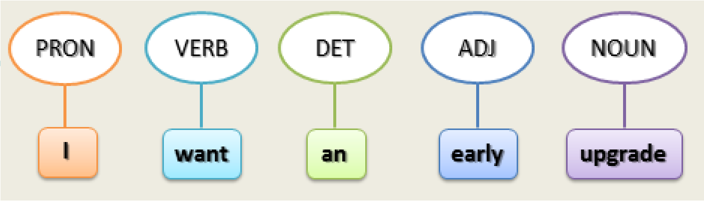
Input and output sequences have the same length

Variable length input

Output contains categorical labels

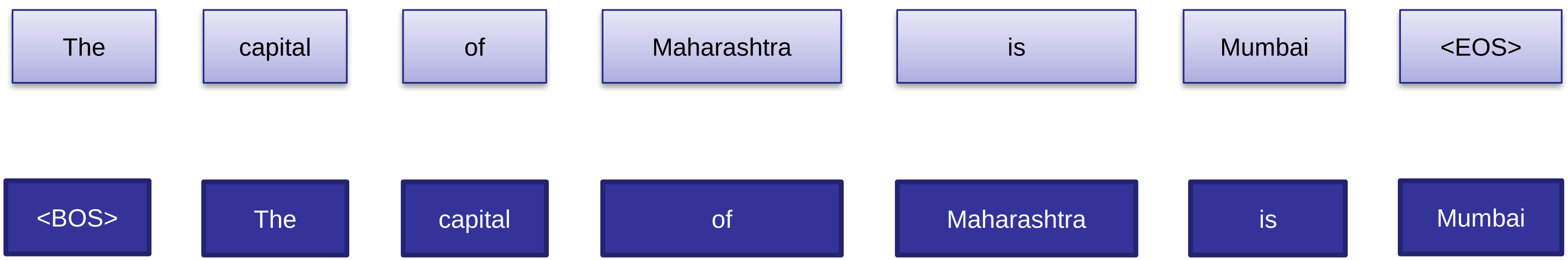
Output at any time-step typically depends on neighbouring output labels and input elements

Part-of-speech tagging



Recurrent Neural Network is a powerful model to learn sequence labelling tasks

How do we model language modeling as a sequence labeling task?



The output sequence is one-time step ahead of the input sequence

Training Language Models

Input: large monolingual corpus

- Each example is a tokenized sentence (sequence of words)
- At each time step, predict the distribution of the next word given all previous words
- Loss Function:
  - Minimize cross-entropy between actual distribution and predicted distribution
  - Equivalently, maximize the likelihood

At a single time-step:

$$J_i(\theta) = CE(z_i, \hat{z}_i) = -\sum_{w \in V} z_i^w \log \widehat{z_i^w} = -\log \widehat{z_i^L}$$

where $y_i = L$

Average over time steps for example n:

$$J^n(\theta) = \frac{1}{T}\sum_{i=1}^{T} J_i(\theta)$$

How do we learn model parameters?
More on that later!

Average over entire corpus:

$$J(\theta) = \frac{1}{N}\sum_{k=1}^{N} J^n(\theta)$$

Evaluating Language Models

How do we evaluate quality of language models?

Evaluate the ability to predict the next word given a context

Evaluate the probability of a testset of sentences

Standard testsets exist for evaluating language models: Penn Treebank, Billion Word Corpus, WikiText

Evaluating LM (cntd.)

- Ram likes to play -----
  - Cricket: high probability, low entropy, low perplexity (relatively very high frequency for ‘like to play cricket’)
  - violin: -do- (relatively high frequency possibility for ‘like to play violin’
  - Politics: moderate probability, moderate entropy, moderate perplexity (relatively moderate frequency ‘like to play politics’
  - milk: almost 0 probability, very high entropy, very high perplexity (relatively very low possibility for ‘like to play milk’

So an LM that predicts ‘milk’ is bad!

Language Model Perplexity

Perplexity: $\exp\left(J(\theta)\right)$

$J(\theta)$ is cross-entropy on the test set

Cross-entropy is measure of difference between actual and predicted distribution

Lower perplexity and cross-entropy is better

Training objective matches evaluation metric *n-gram*

RNN variants

Model	Perplexity
Interpolated Kneser-Ney 5-gram (Chelba et al., 2013)	67.6
RNN-1024 + MaxEnt 9-gram (Chelba et al., 2013)	51.3
RNN-2048 + BlackOut sampling (Ji et al., 2015)	68.3
Sparse Non-negative Matrix factorization (Shazeer et al., 2015)	52.9
LSTM-2048 (Jozefowicz et al., 2016)	43.7
2-layer LSTM-8192 (Jozefowicz et al., 2016)	30
Ours small (LSTM-2048)	43.9
Ours large (2-layer LSTM-2048)	39.8

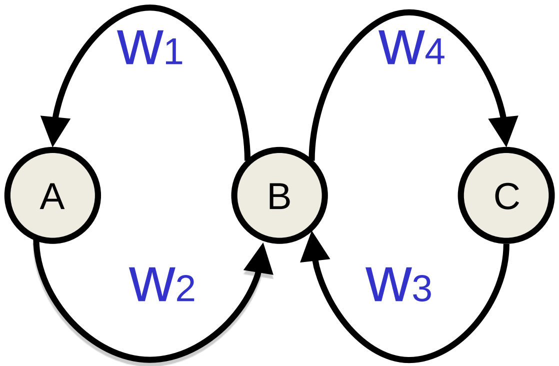
https://research.fb.com/building-an-efficient-neural-language-model-over-a-billion-words/

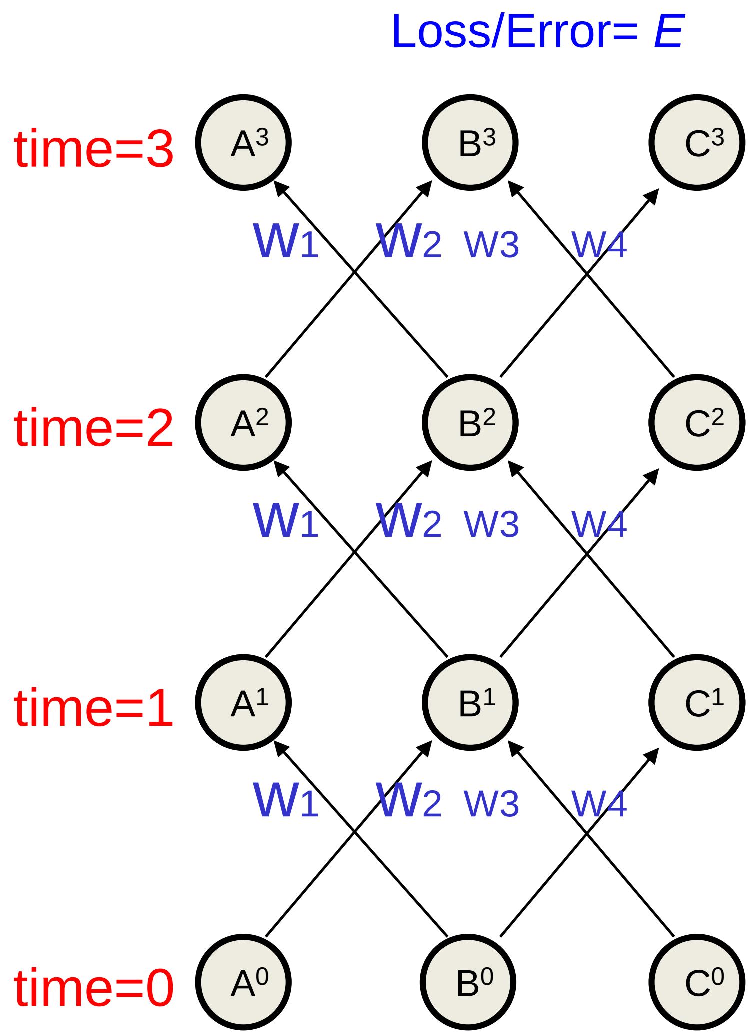
RNN models outperform n-gram models

A special kind of RNN network – LSTM- does even later ➔ we will see that soon

BPTT

The equivalence between feedforward nets and recurrent nets





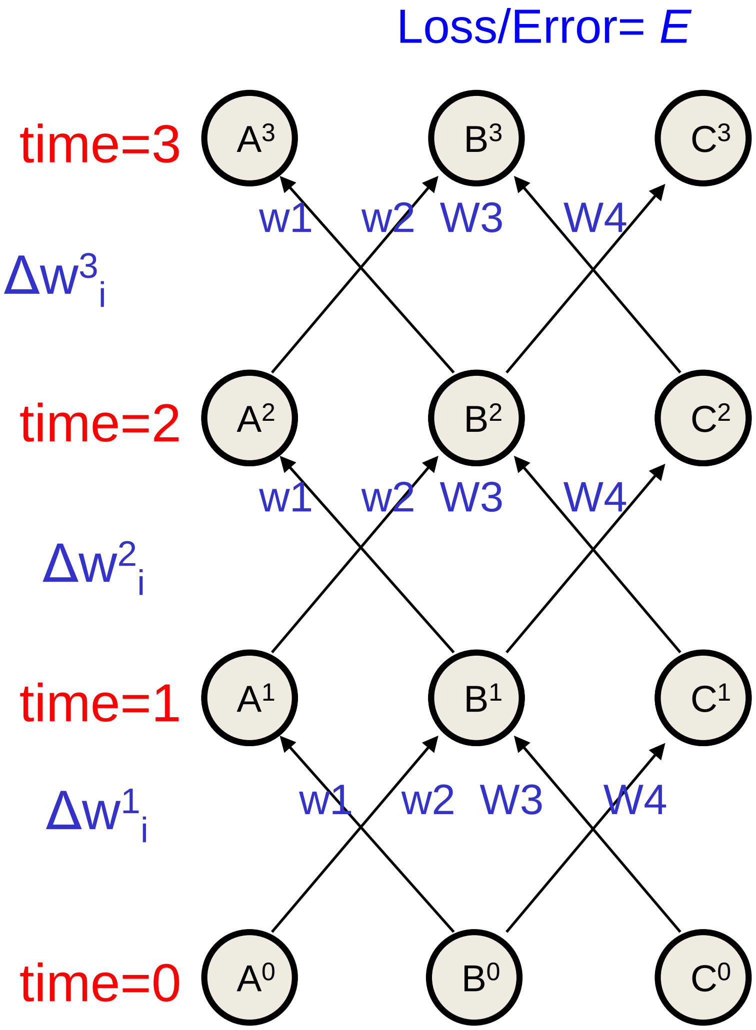
Assume that there is a time delay of 1 in using each connection.

The recurrent net is just a layered net that keeps reusing the same weights.

BPTT illustration

$\Delta w_i = \Delta w^3_i + \Delta w^2_i + \Delta w^1_i$

Vanishing/Exploding Gradient can strike!!!

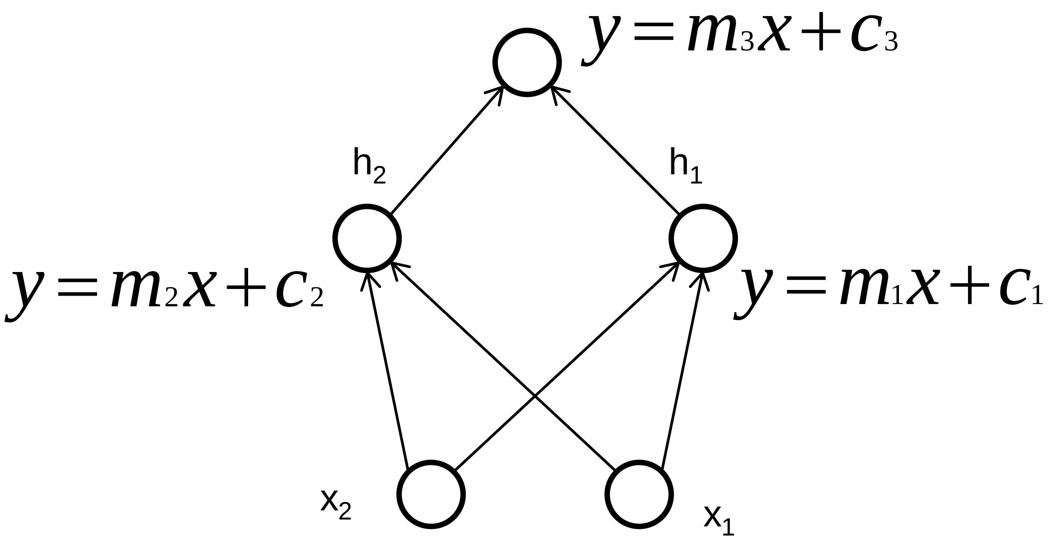


BPTT important points

- The forward pass at each time step.
- The backward pass computes the error derivatives at each time step.
- After the backward pass we add together the derivatives at all the different times for each weight.

A few points about FFNN BP

Can Linear Neurons Work?



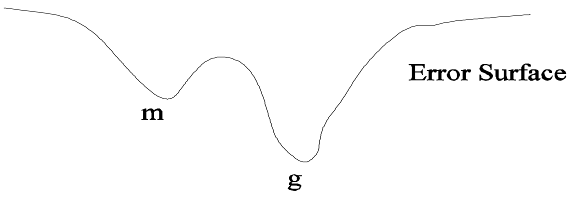
$$h_1 = m_1(w_1 x_1 + w_2 x_2) + c_1$$

$$h_1 = m_2(w_1 x_1 + w_2 x_2) + c_2$$

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

Local Minima

Due to the Greedy nature of BP, it can get stuck in local minimum *m* and will never be able to reach the global minimum *g* as the error can only decrease by weight change.



m- local minima, g- global minima

Figure- Getting Stuck in local minimum

Momentum factor

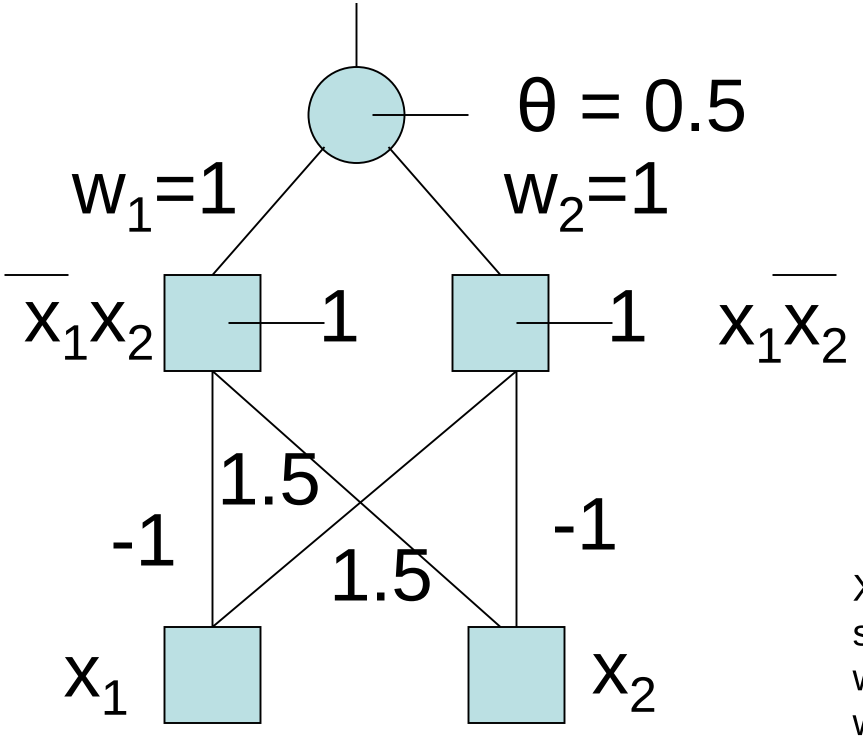
1. Introduce momentum factor.

$$(\Delta w_{ji})_{nth-iteration} = \eta \delta_j O_i + \beta (\Delta w_{ji})_{(n-1)th-iteration}$$

- Accelerates the movement out of the trough.
- Dampens oscillation inside the trough.
- Choosing β : If β is large, we may jump over the minimum.

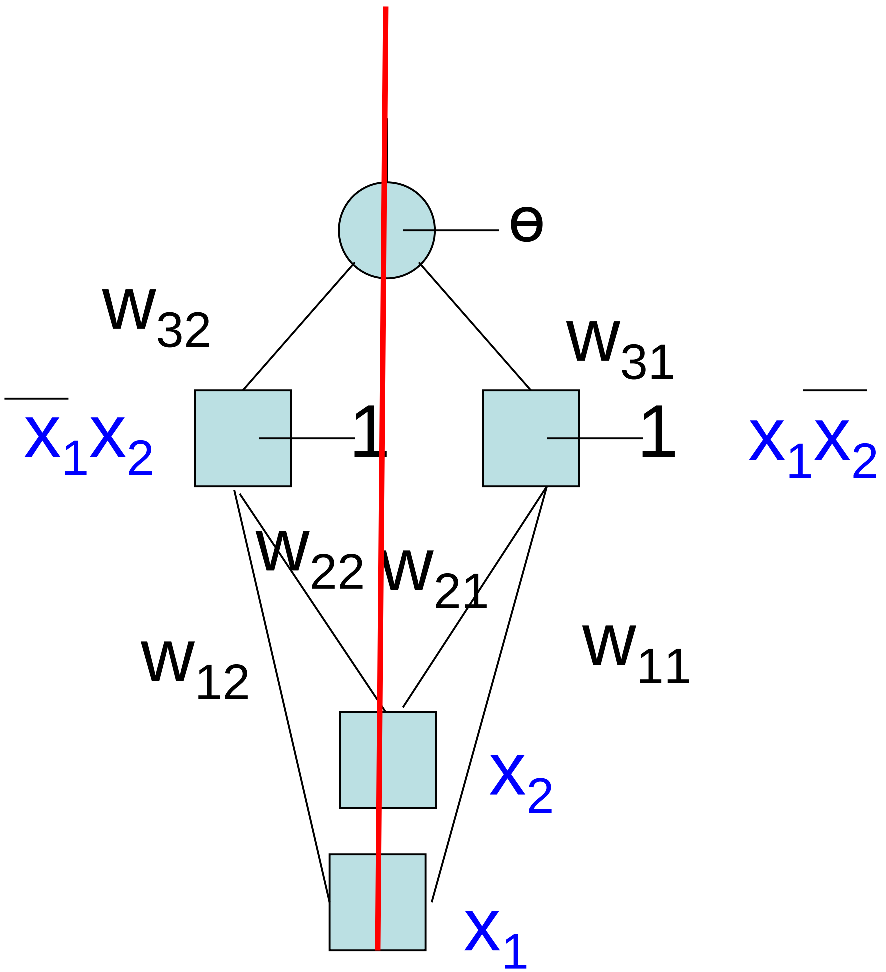
Symmetry breaking

- If mapping demands different weights, but we start with the same weights everywhere, then BP will never converge.



XOR n/w: if we s started with identical weight everywhere, BP will not converge

Symmetry breaking: understanding with proper diagram



Symmetry About The red Line should Be broken

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:

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

For (1,0), One class: $m.w_2 - m.\theta + c > 0.9$

For (1,1), Zero class: $m.w_1 - m_2.\theta + c < 0.1$

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.

An application in Medical Domain

Expert Systems also called Knowledge Based Systems (KBS)

- Expert Systems aim to mimic experts: e.g., doctors, lawyers, metallurgists and so on
- Expert knowledge is encoded in either of the two forms
  - An elaborate set of rules, or
  - Neural net
- The latter is called a connectionist expert system

Expert System for Skin Diseases Diagnosis

- Bumpiness and scaliness of skin
- Mostly for symptom gathering and for developing diagnosis skills
- Not replacing doctor’s diagnosis

Architecture of the FF NN

- 96-20-10
- 96 input neurons, 20 hidden layer neurons, 10 output neurons
- Inputs: skin disease symptoms and their parameters
  - *Location, distribution, shape, arrangement, pattern, number of lesions, presence of an active norder, amount of scale, elevation of papuls, color, altered pigmentation, itching, pustules, lymphadenopathy, palmer thickening, results of microscopic examination, presence of herald pathc, result of dermatology test called KOH*

Output

- 10 neurons indicative of the diseases:
  - *psoriasis, pityriasis rubra pilaris, lichen planus, pityriasis rosea, tinea versicolor, dermatophytosis, cutaneous T-cell lymphoma, secondery syphilis, chronic contact dermatitis, soberrheic dermatitis*

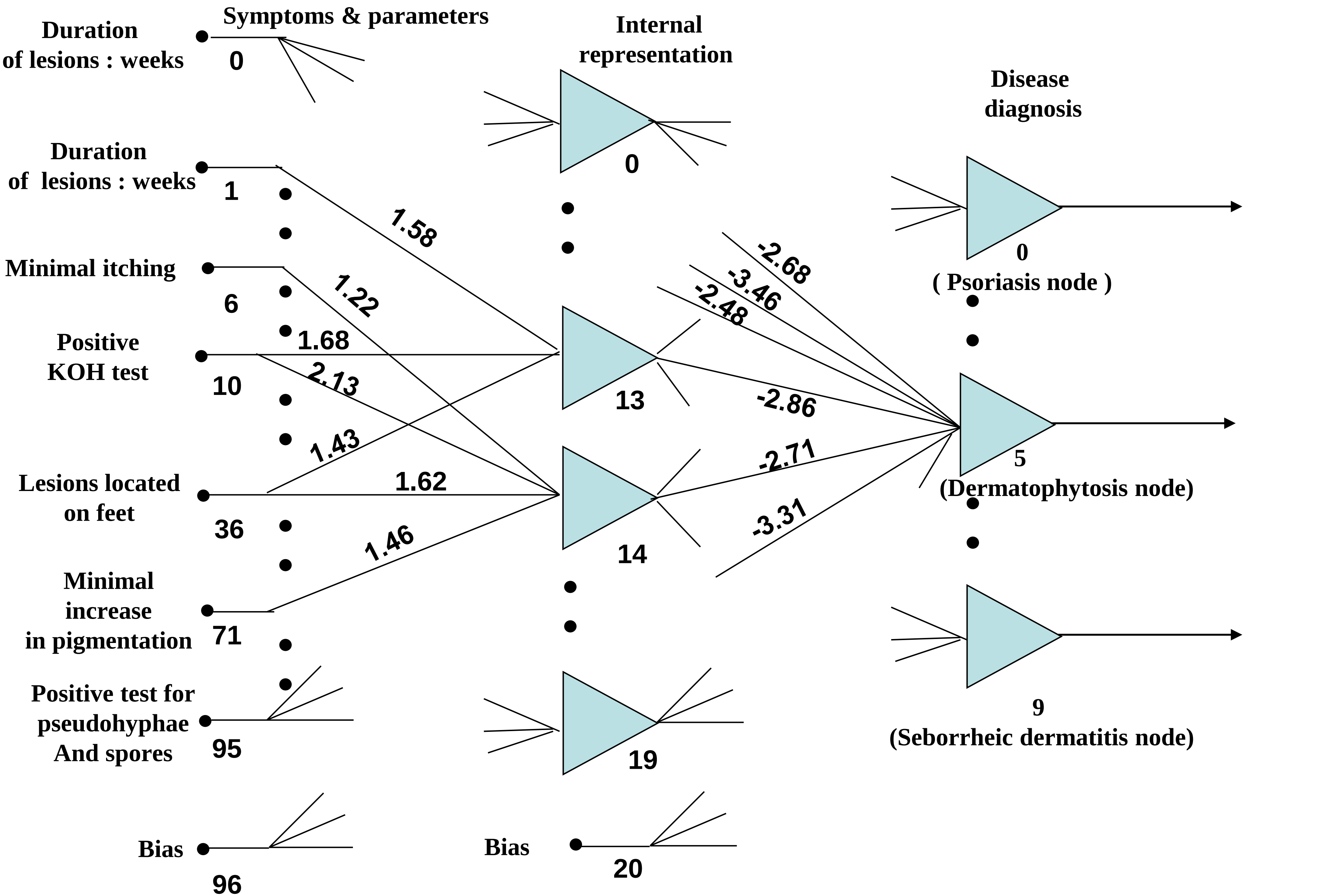


Figure : Explanation of dermatophytosis diagnosis using the DESKNET expert system.

Training data

- Input specs of 10 model diseases from 250 patients
- 0.5 is some specific symptom value is not known
- Trained using standard error backpropagation algorithm

Testing

- Previously unused symptom and disease data of 99 patients
- Result:
- Correct diagnosis achieved for 70% of papulosquamous group skin diseases
- Success rate above 80% for the remaining diseases except for psoriasis
- psoriasis diagnosed correctly only in 30% of the cases
- Psoriasis resembles other diseases within the papulosquamous group of diseases, and is somewhat difficult even for specialists to recognise.

Explanation capability

- Rule based systems reveal the explicit path of reasoning through the textual statements
- Connectionist expert systems reach conclusions through complex, non linear and simultaneous interaction of many units
- Analysing the effect of a single input or a single group of inputs would be difficult and would yield incorrect results

Explanation contd.

- The hidden layer re-represents the data
- Outputs of hidden neurons are neither symtoms nor decisions

Discussion

- Symptoms and parameters contributing to the diagnosis found from the n/w
- Standard deviation, mean and other tests of significance used to arrive at the importance of contributing parameters
- The n/w acts as apprentice to the expert

CS772 midsem questions in 2021 on FFNN (10, 11, 12) (cntd.)

The input→output patterns are $\langle X_2, X_1 \rangle \rightarrow \langle O_2, O_1 \rangle$:

$\langle 0,0 \rangle \rightarrow \langle 1,0 \rangle$

$\langle 0,1 \rangle \rightarrow \langle 0,1 \rangle$

$\langle 1,0 \rangle \rightarrow \langle 0,1 \rangle$

$\langle 1,1 \rangle \rightarrow \langle 1,0 \rangle$

- I.e., o_1 computes XOR and o_2 computes XNOR
- The values of all weights are initialized to 1; also there are no bias terms.

Q10

Q10. For the input <0,1>, the outputs from H_1, H_2, O_1, O_2 are respectively

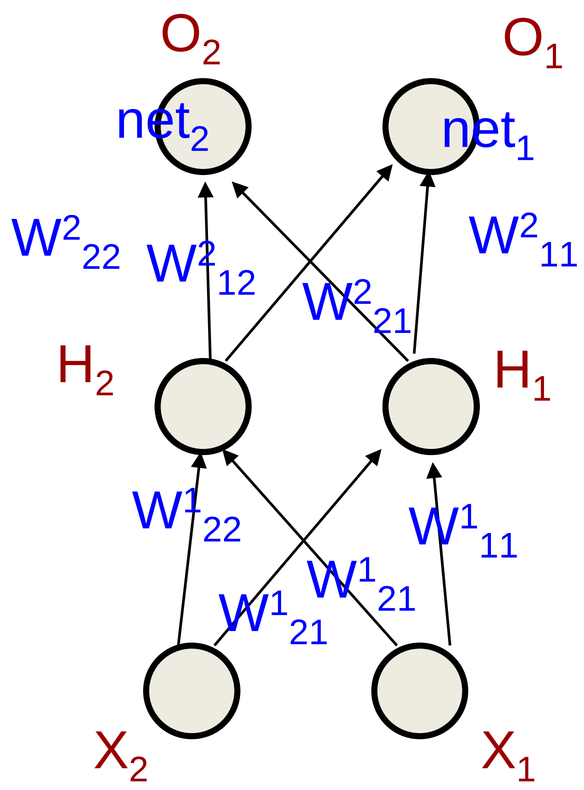
(a) 1, 1, 0.5, 0.5

(b) 1,0.5, 1, 0.5

(c) 0.5,0.5, 1, 1

(d) 0.5, 1, 0.5, 1

Ans: **(a)**



Elaboration:

$X_1=1, X_2=0 \rightarrow net_{H1}=1, net_{H2}=1$

$\rightarrow h_1=relu(net_{H1})=1, h_2=relu(net_{H2})=1$

$\rightarrow net_1=2, net_2=2$

$\rightarrow o_1=e^{net1}/(e^{net1}+e^{net2})=0.5, o_2=e^{net2}/(e^{net1}+e^{net2})=0.5$

Q11

Q11. The cross entropy error value for the input <1,1> (X_2, X_1) with respect to Q10 is (assume weight changes are posted only after a complete epoch, and initial weights are all 1):

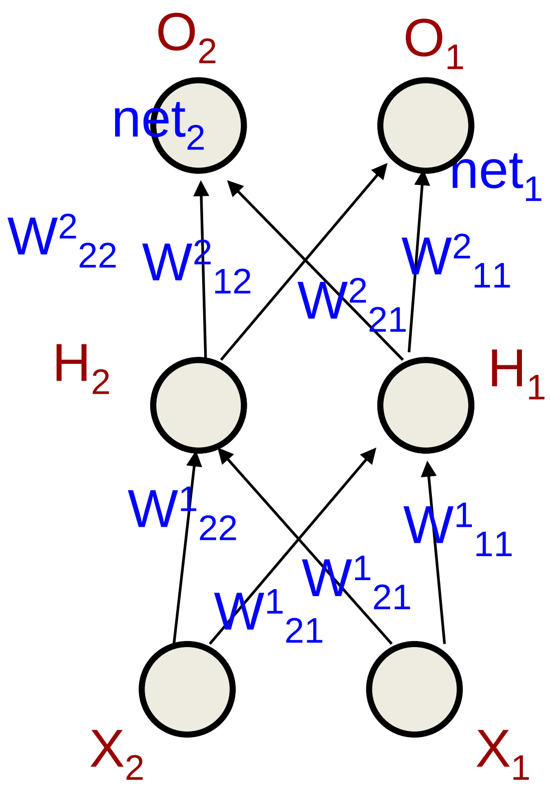
(a) $1/\log_e 2$

(b) $\log_e 2$

(c) $2\log_e(0.5)$

(d) None of the above

Ans: (b)



Elaboration:

$X_1=1$, $X_2=1$ ➔ $net_{H1}=2$, $net_{H2}=2$ ➔ $h_1=2$, $h_2=2$ ➔ $net_1=4$, $net_2=4$ ➔ $o_1=e^{net1}/(e^{net1}+e^{net2})=0.5$, $o_2=e^{net2}/(e^{net1}+e^{net2})=0.5$, $t_1=0$, $t_2=1$

➔ $E= -1\log_e(0.5)-0\log_e 0= \log_e 2$

Q12

Q12. Again consider the network of Q10, with same input-output patterns and initial values of weights (all 1). Assume the learning rate to be 0.5. The weight change values for the connections X_1-to-H_1 and H_1-to-O_1 given the input <0,1> (X_2, X_1) are:

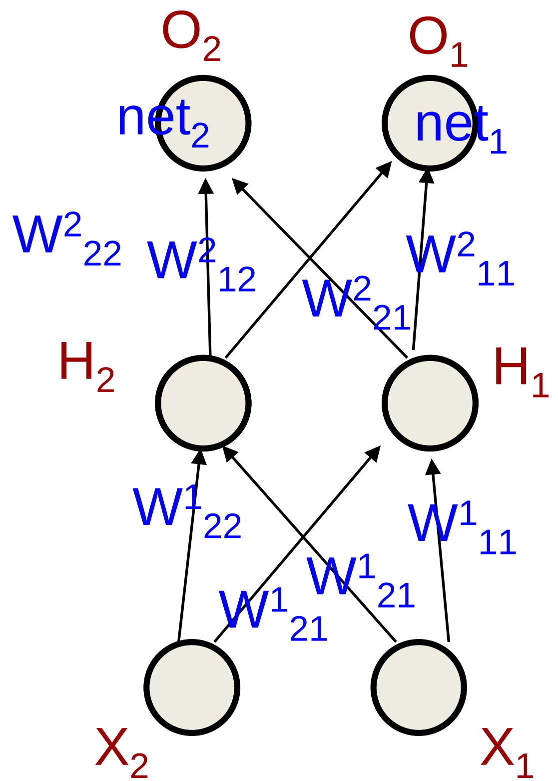
(a) 0, 0.5

(b) -0.5, 1

(c) 0.25, 0

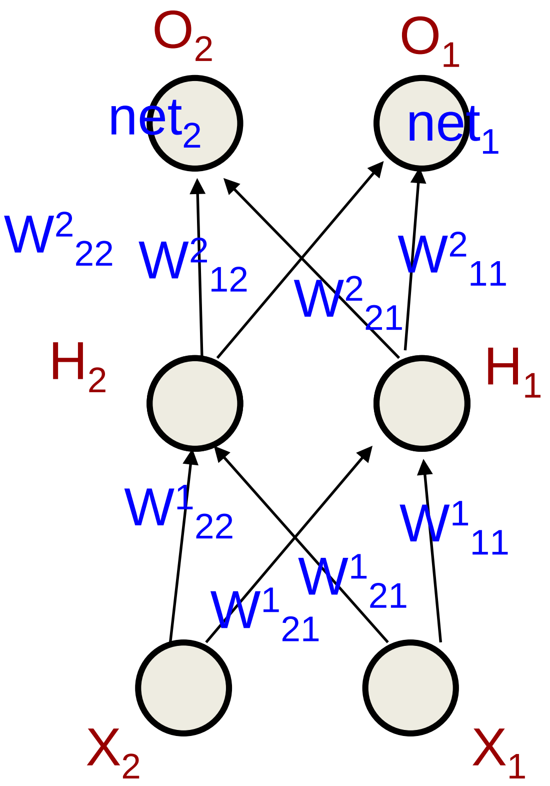
(d) None of the above

Ans: (d)



Elaboration: next slide

Q12 (cntd.)



Elaboration:

$X_1=1, X_2=0 \rightarrow o_1= 0.5, o_2= 0.5, t_1=1, t_2=0 \rightarrow \delta_{o1}=t_1-o_1=0.5, \delta_{o2}=t_2-o_2=-0.5$

$\rightarrow \Delta W_{o1H1}=\eta\delta_{o1}h_1=0.5.0.5.1=0.25,$

$\delta_{H1}=(W^2{}_{11}\delta_{o1}+W^2{}_{21}\delta_{o2}).r'(H_1)$

$= (1X0.5+1X\text{-}0.5).1=0$

$\rightarrow \Delta W_{H1X1}=\eta\delta_{H1}X_1=0.5X0X1=0$