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

Glove, PCA, Word2vec weights, RNN

Pushpak Bhattacharyya
Computer Science and Engineering Department
IIT Bombay

Week 6 of 6th Feb, 2023

Re-cap

Two main models for learning word vectors

- 1) global matrix factorization methods, such as latent semantic analysis (LSA) (Deerwester et al., 1990) and
- 2) local context window methods, such as the skip-gram model of Mikolov et al. (2013)
- Currently, both families suffer significant drawbacks.

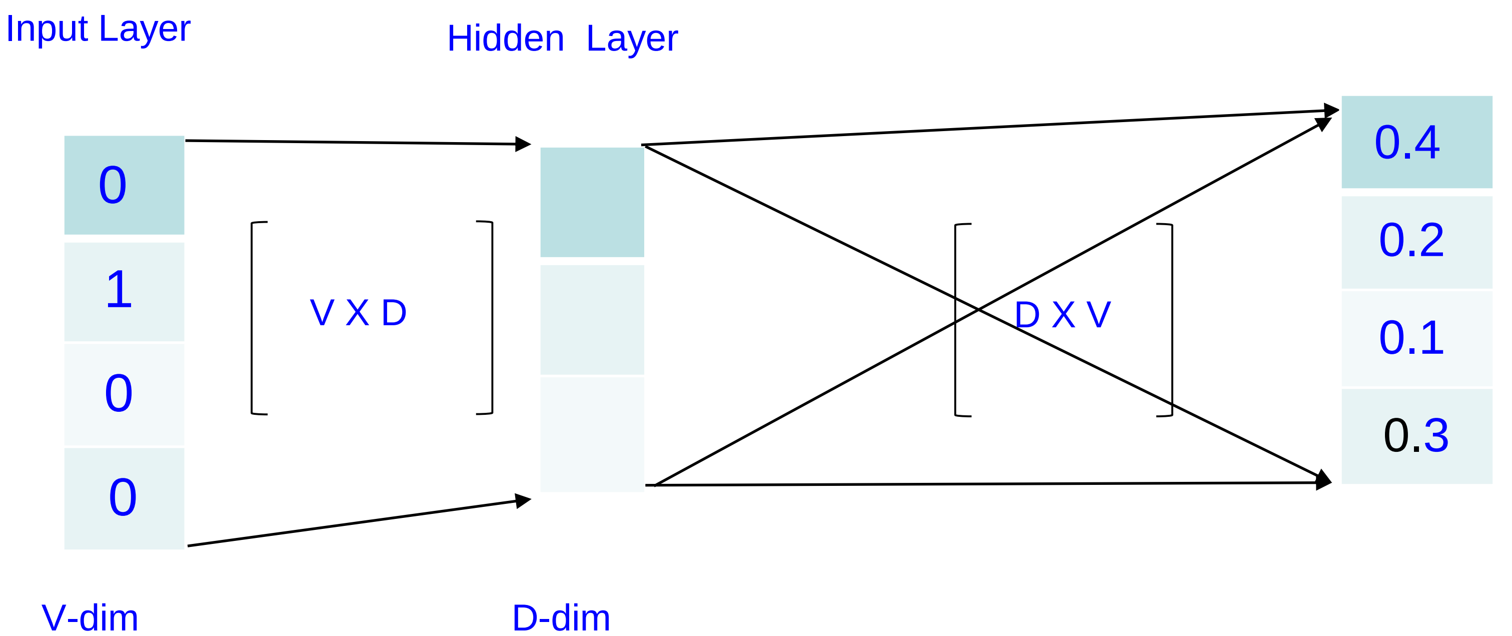
Matrix Factorization: drawback

- “most frequent words contribute a disproportionate amount to the similarity measure: the number of times two words co-occur with *the* or *and*, for example, will have a large effect on their similarity despite conveying relatively little about their semantic relatedness.”

Skip Gram & CBOW: drawback

- "shallow window-based methods suffer from the disadvantage that they do not operate directly on the co-occurrence statistics of the corpus. Instead,these models scan context windows across the entire corpus, which fails to take advantage of the vast amount of repetition in the data"

chitecture for Glove work?



Representation using syntagmatic relations: Co-occurrence Matrix

Corpora: I enjoy cricket. I like music. I like deep learning

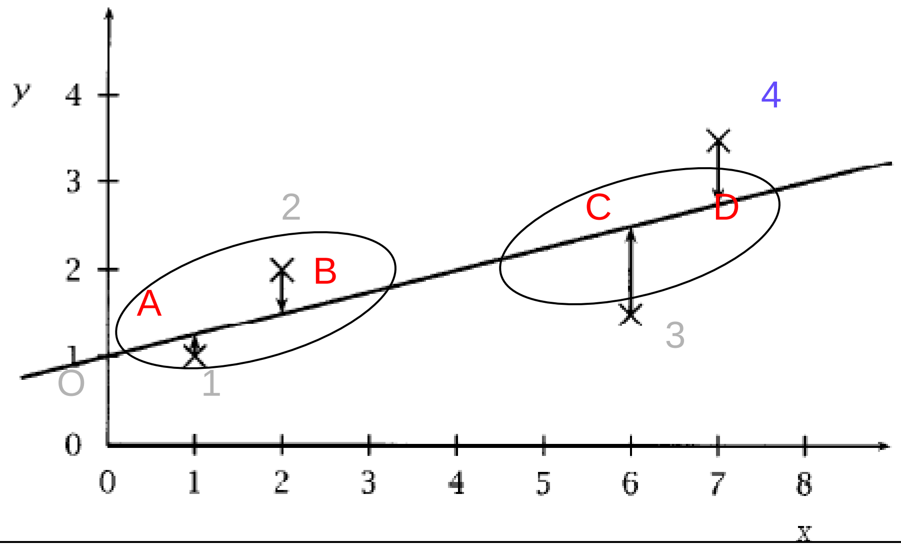
	I	**enjoy**	**cricket**	**like**	**music**	**deep**	**learning**
I	-	1	1	2	1	1	1
enjoy	1	-	1	0	0	0	0
cricket	1	1	-	0	0	0	0
like	2	0	0	-	1	1	1
music	1	0	0	1	-	0	0
deep	1	0	0	1	0	-	1
learning	1	0	0	1	0	1	-

Solution: uses co-occurences

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2$$

Dimensionality Reduction by PCA

Intuition for Dimensionality Reduction

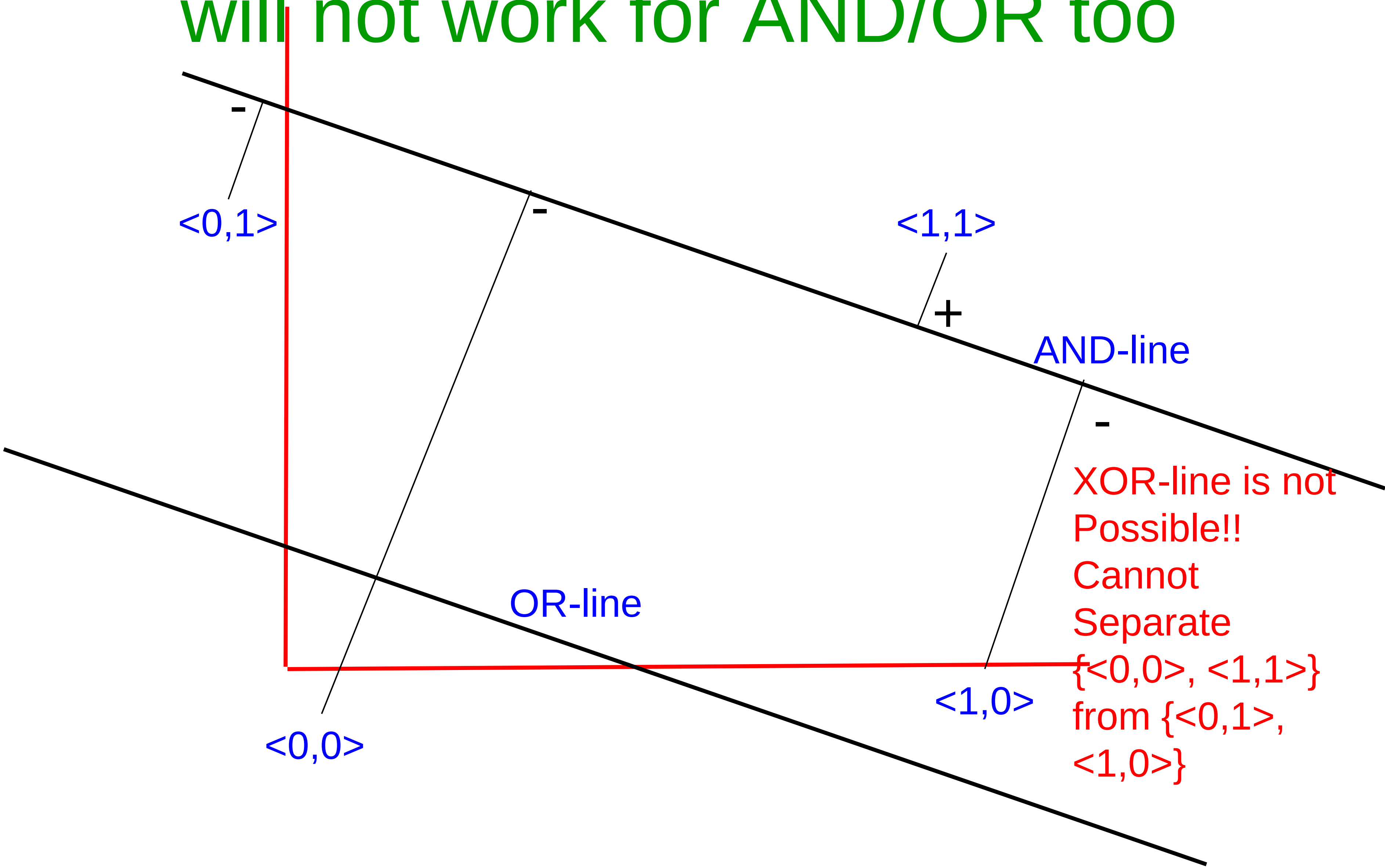


- 1, 2, 3, 4: are the points
- A, B, C, D: are their projections on the fitted line by linear regression

- Suppose 1, 2 form a class and 3, 4 another class
- Of course, it is easy to set up a hyper plane that will separate 1 and 2 from 3 and 4
- That will be classification in **2 dimension**

- But suppose we form another attribute of these points, viz., distances of their
- projections On the line from “O”
- Then the points can be classified by a threshold on these distances
- This effectively is classification in the **reduced dimension (1 dimension)**

XOR problem; Projection on regression line will not work for AND/OR too



Principal Component Analysis

Example: *IRIS Data (only 3 values out of 150)*

ID	Petal Length (a_1)	Petal Width (a_2)	Sepal Length (a_3)	Sepal Width (a_4)	Classification
001	5.1	3.5	1.4	0.2	Iris-setosa
051	7.0	3.2	4.7	1.4	Iris-versicolor
101	6.3	3.3	6.0	2.5	Iris-virginica

Training and Testing Data

- Training: 80% of the data; 40 from each class: total 120
- Testing: Remaining 30
- Do we have to consider all the 4 attributes for classification?
- Less attributes is likely to increase the generalization performance (Occam Razor Hypothesis: *A simpler hypothesis generalizes better*)

The multivariate data: *n* instances, *p* attributes

$$
\begin{array}{cccccc}
\mathbf{X_1} & \mathbf{X_2} & \mathbf{X_3} & \mathbf{X_4} & \mathbf{X_5} \cdots & \mathbf{X_p} \\
x_{11} & x_{12} & x_{13} & x_{14} & x_{15} \cdots & x_{1p} \\
x_{21} & x_{22} & x_{23} & x_{24} & x_{25} \cdots & x_{2p} \\
x_{31} & x_{32} & x_{33} & x_{34} & x_{35} \cdots & x_{3p} \\
x_{41} & x_{42} & x_{43} & x_{44} & x_{45} \cdots & x_{4p} \\
 & & & \cdots & & \\
 & & & \cdots & & \\
x_{n1} & x_{n2} & x_{n3} & x_{n4} & x_{n5} \cdots & x_{np}
\end{array}
$$

Some preliminaries

- Sample mean vector: $<\mu_1, \mu_2, \mu_3, \ldots, \mu_p>$

  For the i^{th} attribute: $\mu_i = (\Sigma^n_{j=1} x_{ij})/n$

- Variance for the i^{th} attribute:

  $$\sigma_i^{\,2} = [\Sigma^n_{j=1} (x_{ij} - \mu_i)^2] / [n-1]$$

- Sample covariance:

  $$c_{ab} = [\Sigma^n_{j=1} ((x_{aj} - \mu_a)(x_{bj} - \mu_b))] / [n-1]$$

This measures the correlation INSIDE the data

In fact, the correlation coefficient

$$r_{ab} = c_{ab} / \sigma_a \sigma_b$$

Standardize the variables

- For each variable x_{ij}

  Replace the values by

$$y_{ij} = (x_{ij} - \mu_i)/\sigma_i^{\ 2}$$

Create the Correlation Matrix

$$R = \begin{bmatrix} 1 \ r_{12} \ r_{13} \ldots \ r_{1p} \\ r_{21} \ 1 \ r_{23} \cdots \ r_{2p} \\ \vdots \\ r_{p1} \ r_{p2} \ r_{p3} \cdots 1 \end{bmatrix}$$

Short digression: Eigenvalues and Eigenvectors

$AX=\lambda X$

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots\ a_{1p}x_p = \lambda x_1$$

$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \ldots\ a_{2p}x_p = \lambda x_2$$

$$\ldots$$

$$a_{p1}x_1 + a_{p2}x_2 + a_{p3}x_3 + \ldots\ a_{pp}x_p = \lambda x_p$$

Here, λs are eigenvalues and the solution

$$\langle x_1, x_2, x_3, \ldots\ x_p \rangle$$

For each λ is the eigenvector

Short digression: To find the Eigenvalues and Eigenvectors

Solve the characteristic function

$$\det(A - \lambda I) = 0$$

Example:

$$A = \begin{pmatrix} -9 & 4 \\ 7 & -6 \end{pmatrix} \qquad \lambda I = \begin{pmatrix} \lambda & 0 \\ 0 & \lambda \end{pmatrix}$$

Characteristic equation

$$(-9-\lambda)(-6-\lambda)-28=0$$

Real eigenvalues: -13, -2

Eigenvector of eigenvalue -13:

(-1, 1)

Eigenvector of eigenvalue -2:

(4, 7)

Verify:

$$\begin{pmatrix} -9 & 4 \\ 7 & -6 \end{pmatrix} \begin{pmatrix} -1 \\ 1 \end{pmatrix} = -13 \begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

Next step in finding the PCs

$$R = \begin{bmatrix} 1 & r_{12} & r_{13} & \cdots & r_{1p} \\ r_{21} & 1 & r_{23} & \cdots & r_{2p} \\ & & \vdots & & \\ r_{p1} & r_{p2} & r_{p3} & \cdots & 1 \end{bmatrix}$$

Find the eigenvalues and eigenvectors of *R*

Example

49 birds: 21 survived in a storm and 28 died.

5 body characteristics given

X_1: body length; X_2: alar extent; X_3: beak and head length

X_4: humerus length; X_5: keel length

Could we have predicted the fate from the body characteristic

$$
\begin{array}{cccccc}
 & X_1 & X_2 & X_3 & X_4 & X_5 \\
\end{array}
$$

$$
R = \begin{bmatrix} 1.000 & & & & \\ 0.735 & 1.000 & & & \\ 0.662 & 0.674 & 1.000 & & \\ 0.645 & 0.769 & 0.763 & 1.000 & \\ 0.605 & 0.529 & 0.526 & 0.607 & 1.000 \end{bmatrix} \begin{array}{c} X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \end{array}
$$

Eigenvalues and Eigenvectors of *R*

Eigenvalues: 3.612, 0.532, 0.386, 0.302, 0.165

First Eigen-vector: V_1	**V_2**	**V_3**	**V_4**	**V_5**
0.452	**0.462**	**0.451**	**0.471**	**0.398**
-0.051	**0.300**	**0.325**	**0.185**	**-0.877**
0.691	**0.341**	**-0.455**	**-0.411**	**-0.179**
-0.420	**0.548**	**-0.606**	**0.388**	**0.069**
0.374	**-0.530**	**-0.343**	**0.652**	**-0.192**

Which principal components are important?

- Total variance in the data=

  $\lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 + \lambda_5$

  = sum of diagonals of $R = 5$
- First eigenvalue= 3.616 ≈ 72% of total variance 5
- Second ≈ 10.6%, Third ≈ 7.7%, Fourth ≈ 6.0% and Fifth ≈ 3.3%
- ***First PC is the most important and sufficient for studying the classification***

Forming the PCs

- $Z_1 = 0.451X_1 + 0.462X_2 + 0.451X_3 + 0.471X_4 + 0.398X_5$
- $Z_2 = -0.051X_1 + 0.300X_2 + 0.325X_3 + 0.185X_4 - 0.877X_5$
- For all the 49 birds find the first two principal components
- This becomes the new data
- Classify using them

For the first bird

X_1=156, X_2=245, X_3=31.6, X_4=18.5, X_5=20.5

After standardizing

Y_1=(156-157.98)/3.65=-0.54,

Y_2=(245-241.33)/5.1=0.73,

Y_3=(31.6-31.5)/0.8=0.17,

Y_4=(18.5-18.46)/0.56=0.05,

Y_5=(20.5-20.8)/0.99=-0.33

PC_1 *for the first bird=*

Z_1= 0.45X(-0.54)+ 0.46X(0.725)+0.45X(0.17)+0.47X(0.05)+0.39X(-0.33)

=0.064

Similarly, Z_2= 0.602

Reduced Classification Data

- Instead of

X_1	X_2	X_3	X_4	X_5
		49 rows		

- Use

Z_1	Z_2
49	rows

Correlation in NLP Tasks

- For PCA, correlation is the crux of the matter
- We did not have an NLP example
- Think about correlation in NLP situations:
  - How can we merge strongly related attributes to form new attributes?
    - Co-occurrence matrix; which words are very strongly correlated and why?
    - POS tagging
    - Parsing
    - Semantic graph

Difference between Explainability & Causality (1/2)

- NLP research is continuously pushing the frontiers of explainability to understand causality

- Difference can be understood with the following example -
  - A doctor knows that when body has jaundice it becomes yellowish. But why? “Yellowness is NOT an explanation of jaundice

Difference between Explainability & Causality (2/2)

- Causal explanation: Liver malfunctioning released increased amount of Bilirubin which makes the urine yellow
- Explainability as it is done today: surface signals are taken
- Deeper signals (causes)- we need to look at other segments of data
- Thus explainability needs to navigate through databases to get into causality. Explainability is a surface signal while causality is a deeper signal.

PCA of co-occurrence matrix

- Sum of eigenvalues= sum of diagonale elements

- What implication does this have for the co-occurrence matrix?

Working out a simple case of word2vec

Example (1/3)

- 4 words: *heavy, light, rain, shower*
  - *Heavy:* U_0 *<0,0,0,1>*
  - *light:* U_1*: <0,0,1,0>*
  - *rain:* U_2*: <0,1,0,0>*
  - *shower:* U_3*: <1,0,0,0>*
- We want to predict as follows:
  - *Heavy*→*rain*
  - *Light*→*shower*

Note

- Any bigram is theoretically possible, but actual probability differs

- E.g., heavy-heavy, heavy-light are possible, but unlikely to occur
- Language imposes constraints on what bigrams are possible
- Domain and corpus impose further restriction

Example (2/3)

- Input-Output
  - *Heavy: U_0 <0,0,0,1>, light: U_1: <0,0,1,0>, rain: U_2: <0,1,0,0>, shower: U_3: <1,0,0,0>*

- *Heavy: V_0 <0,0,0,1>, light: V_1: <0,0,1,0>, rain: V_2: <0,1,0,0>, shower: V_3: <1,0,0,0>*

Example (3/3)

- *heavy→ rain*
  - *heavy:* U_0 *<0,0,0,1>*

    →

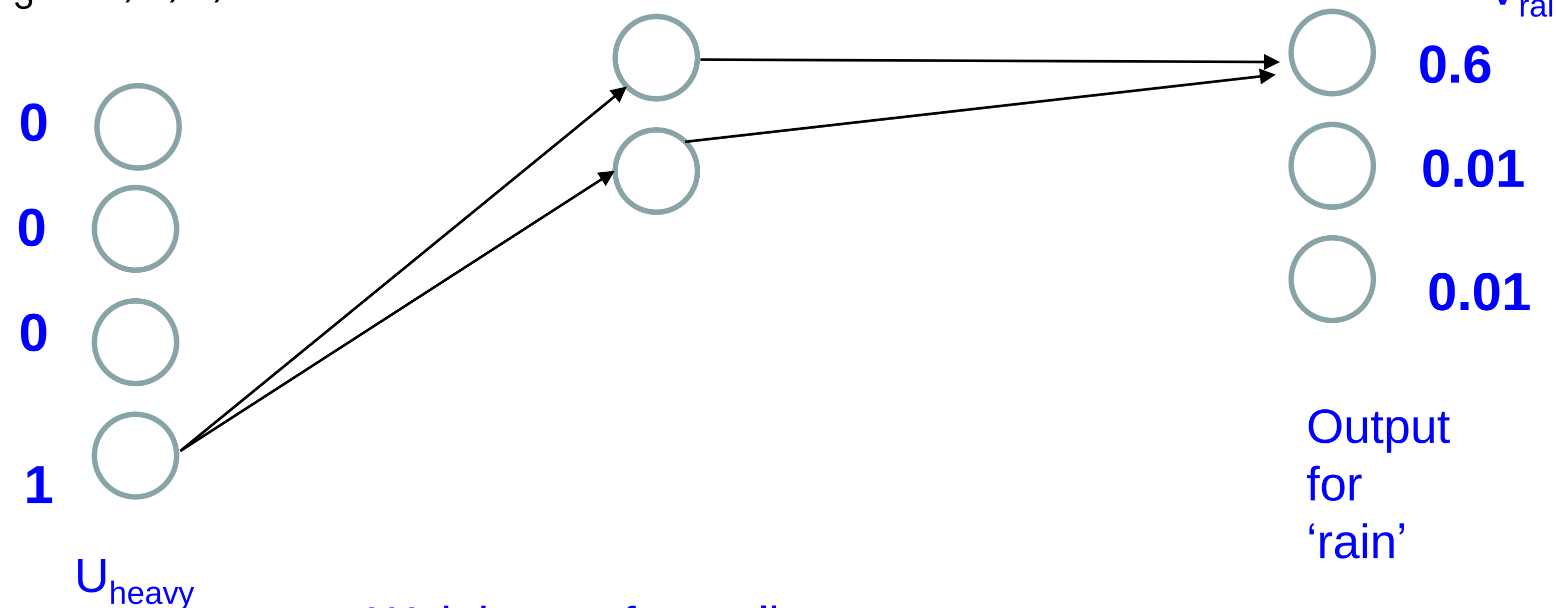
  - *rain:* V_2*: <0,1,0,0>*

- *light→ shower*
  - *light:* U_1*: <0,0,1,0>, → shower:* V_3*: <1,0,0,0>*

Word2vec n/w

Heavy: V_0 <0,0,0,1>
light: V_1: <0,0,1,0>
rain: V_2: <0,1,0,0>
shower: V_3: <1,0,0,0>

Heavy: U_0 <0,0,0,1>
light: U_1: <0,0,1,0>
rain: U_2: <0,1,0,0>
shower: U_3: <1,0,0,0>



Weights go from all neurons to
all neurons in the next layer; shown
For only one input and output

Chain of thinking

- *P(rain|heavy)* should be the highest

- So the output from V2 should be the highest because of softmax

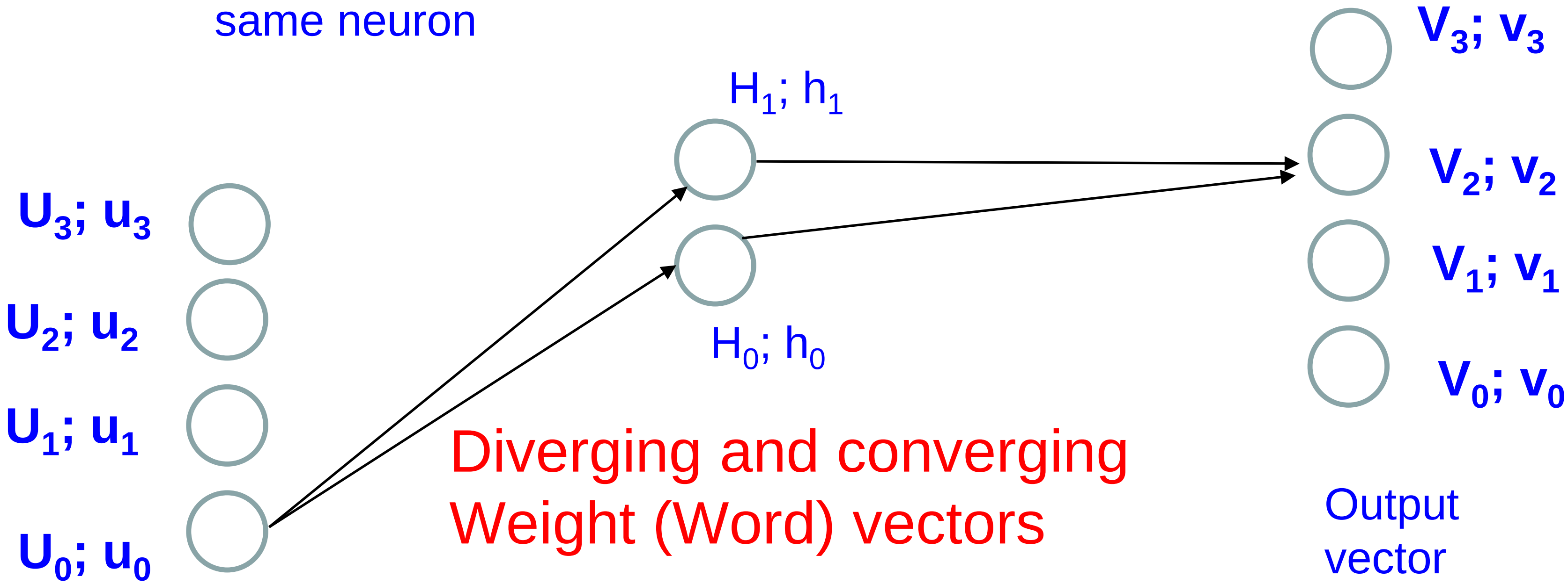
- This way of converting an English statement into probability in insightful

Developing word2vec weight change rule

Illustrated with 4 words only

Word2vec n/w

Convention: Capital letter for NAME of neuron; small letter for output from the same neuron



Diverging and converging Weight (Word) vectors

Output vector V

Input vector U

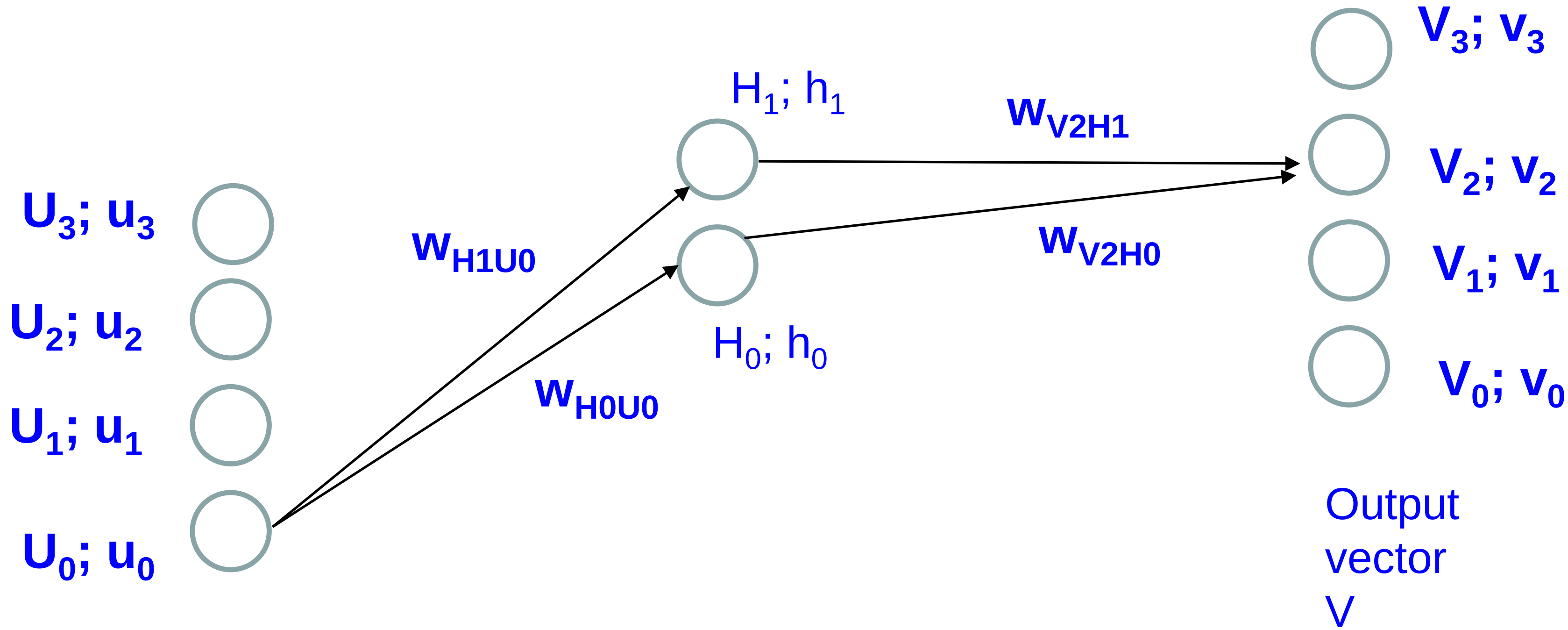
Weights go from all neurons to all neurons in the next layer; shown For only one input and output

Notation Convention

- Weights indicated by small ‘w’
- Index close to ‘w’ is for the destination neuron
- The other index is for the source neuron

Word2vec n/w

Capital letter for NAME of neuron; small letter for output from the same neuron



Input
vector
U

Weights go from all neurons to all neurons in the next layer; shown For only one input and output

More notation

- Net input to hidden and output layer neurons play an important role in BP
- Net input to hidden layer neurons: net_{H0} and net_{H1}
- Net input to output layer neurons: net_{V0}, net_{V1}, net_{V2}, net_{V3}

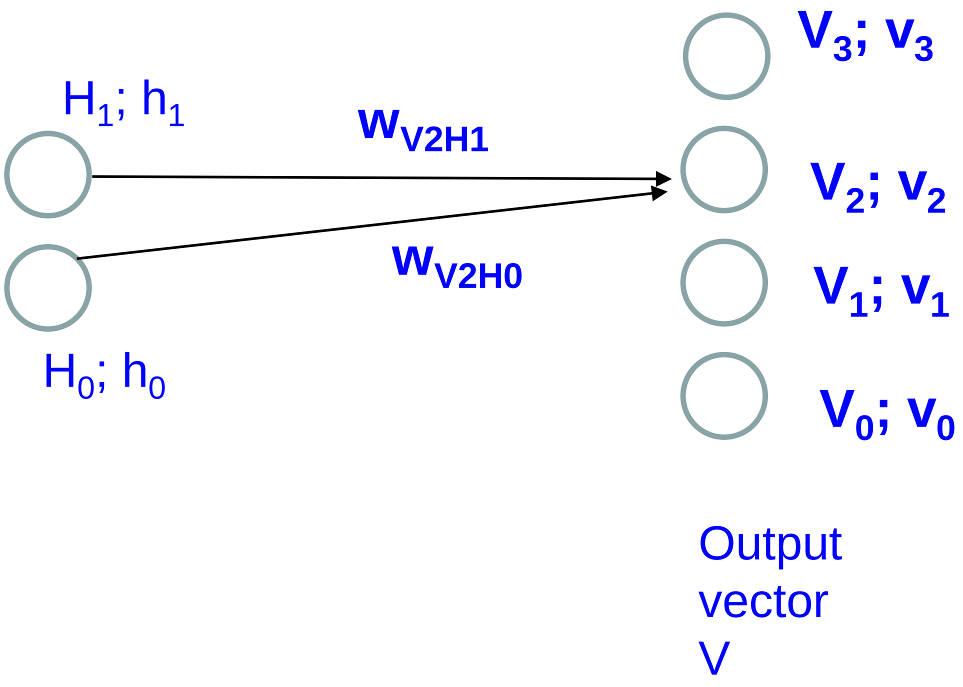
Outputs at the outermost layer

$$v_0 = \frac{e^{net_{V_0}}}{e^{net_{V_0}} + e^{net_{V_1}} + e^{net_{V_2}} + e^{net_{V_3}}}$$

$$v_1 = \frac{e^{net_{V_1}}}{e^{net_{V_0}} + e^{net_{V_1}} + e^{net_{V_2}} + e^{net_{V_3}}}$$

$$v_2 = \frac{e^{net_{V_2}}}{e^{net_{V_0}} + e^{net_{V_1}} + e^{net_{V_2}} + e^{net_{V_3}}}$$

$$v_3 = \frac{e^{net_{V_3}}}{e^{net_{V_0}} + e^{net_{V_1}} + e^{net_{V_2}} + e^{net_{V_3}}}$$



Note

- No non-linearity in the hidden layer
- Why?
- Hidden layer should do ONLY dimensionality reduction
- Can be proved: hidden layer with linearity gives the principal components (will discuss of which Matrix)

Why Dimensionality Reduction?

- The vectors of words represent their distributional similarity

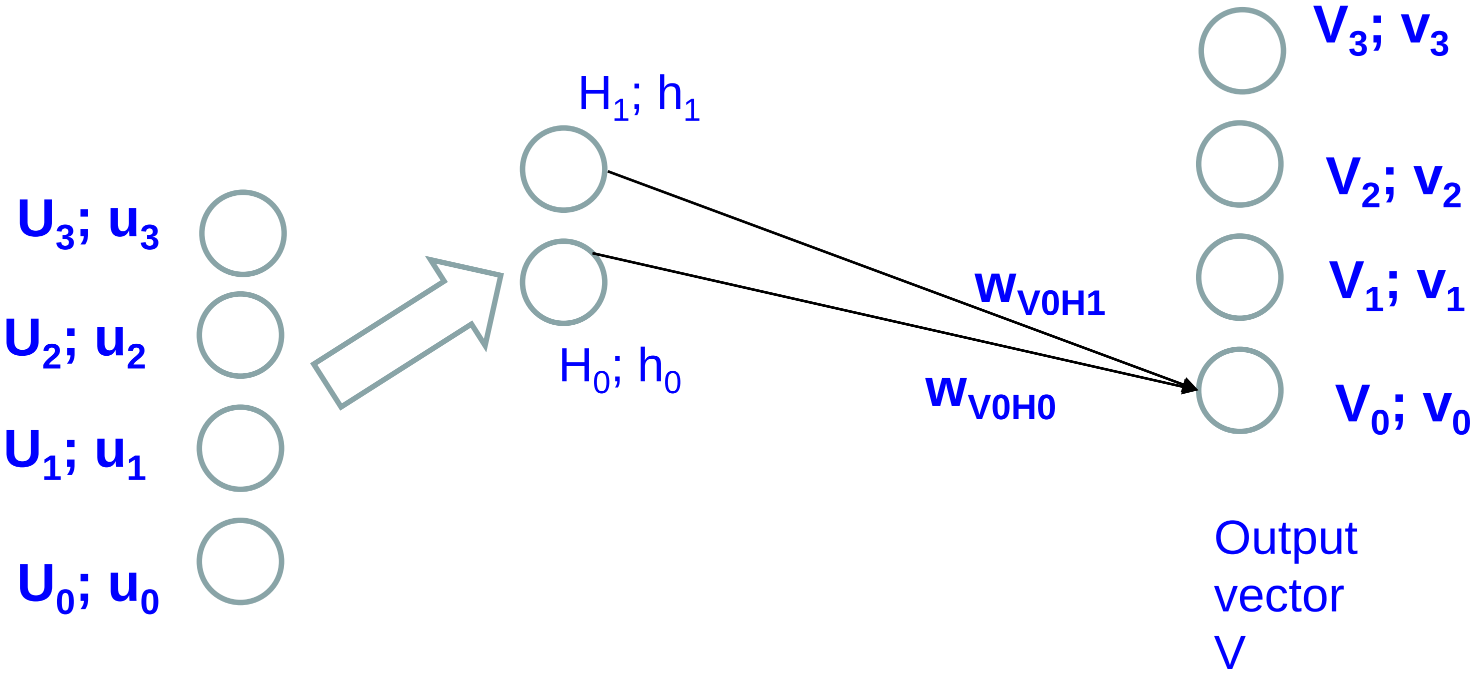
- Dimensionality reduction achieves capturing commonality of these distributional similarities across words

Developing "net_{vi}" (1/2)

$$net_{V_0} = w_{V_0H_0} h_0 + w_{V_0H_1} h_1$$

$$h_0 = w_{H_0U_0} u_0 + w_{H_0U_1} u_1 + w_{H_0U_2} u_2 + w_{H_0U_3} u_3$$

$$h_1 = w_{H_1U_0} u_0 + w_{H_1U_1} u_1 + w_{H_1U_2} u_2 + w_{H_1U_3} u_3$$



Developing "net$_{vi}$" (2/2)

- For "heavy", only u_0 is 1, u_1=u_2=u_3=0
- So,

$$h_0 = w_{H_0U_0}$$

$$h_1 = w_{H_1U_0}$$

$$net_{v_0} = w_{V_0H_0} w_{H_0U_0} + w_{V_0H_1} w_{H_1U_0}$$

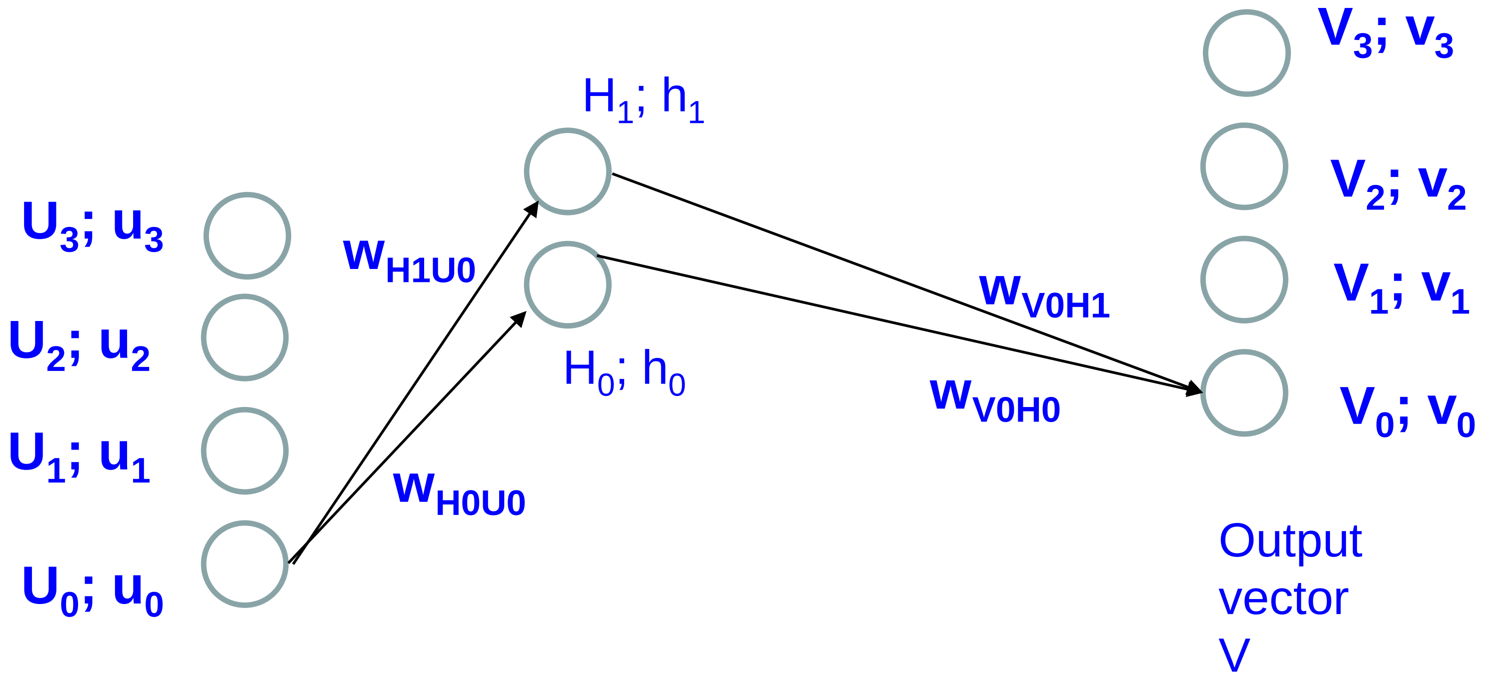
$$= \begin{bmatrix} w_{H_0U_0} & w_{H_1U_0} \end{bmatrix} \begin{bmatrix} w_{V_0H_0} \\ w_{V_0H_1} \end{bmatrix}$$

More Notation

- Weight vector FROM U_0 is called W_{U0} (capital ‘W’)
- Weight vector INTO V_0 is called W_{V0}
- Slight liberty with notation, but has intuitive advantage

For "heavy" (=U_0), the value of net_{v0}

$$net_{V_0} = W_{U_0} . W_{V_0}^T$$



For “heavy” (=U_0), values of other net_{vi}s

$$net_{V_0} = W_{U_0} . W_{V_0}^T$$

$$net_{V_1} = W_{U_0} . W_{V_1}^T$$

$$net_{V_2} = W_{U_0} . W_{V_2}^T$$

$$net_{V_3} = W_{U_0} . W_{V_3}^T$$

We want to maximize $P(\text{'rain'}=V_2|\text{'heavy'}=U_0)$

- This probability is in terms of softmax.

$$P(\text{'}rain\text{'}=V_2\,|\,\text{'}heavy\text{'}=U_0)$$

$$=v_2=\frac{e^{net_{V_2}}}{e^{net_{V_0}}+e^{net_{V_1}}+e^{net_{V_2}}+e^{net_{V_3}}}$$

Equivalent to

- minimize *-log[P('rain'=V_2|'heavy'=U_0)]*

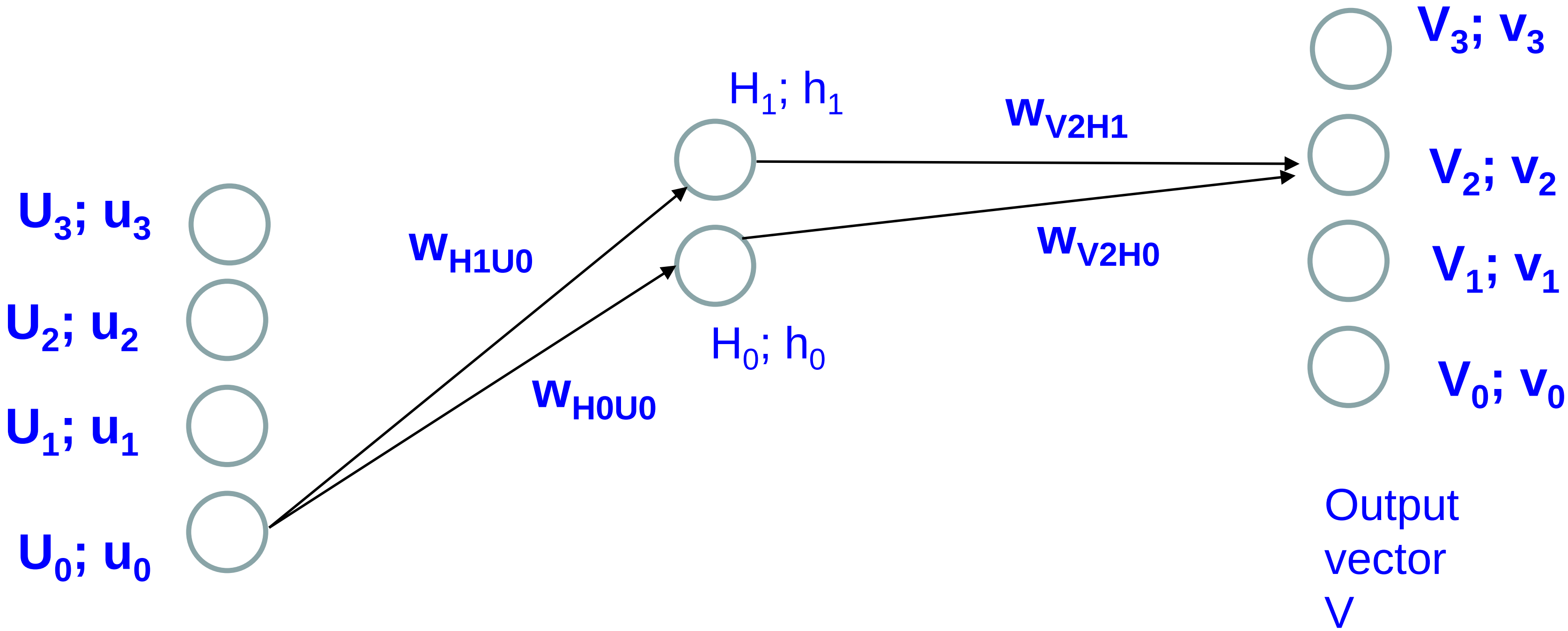
$$-\log[P('rain'=V_2 \,|\, 'heavy'=U_0)]$$

$$=-net_{V_2}+\log(e^{net_{V_0}}+e^{net_{V_1}}+e^{net_{V_2}}+e^{net_{V_3}})$$

$$=-W_{U_0}W_{V_2}^T+\log(e^{net_{V_0}}+e^{net_{V_1}}+e^{net_{V_2}}+e^{net_{V_3}})$$

Word2vec n/w

Capital letter for NAME of neuron; small letter for output from the same neuron



Input
vector
U

Weights go from all neurons to
all neurons in the next layer; shown
For only one input and output

Computing Δw_{V2H0}

$$\Delta w_{V_2H_0} = -\eta \frac{\delta E}{\delta w_{V_2H_0}}$$

$$E = -net_{V_2} + \log(e^{net_{V_0}} + e^{net_{V_1}} + e^{net_{V_2}} + e^{net_{V_3}})$$

$$= -W_{U_0} W_{V_2}^T + \log(e^{net_{V_0}} + e^{net_{V_1}} + e^{net_{V_2}} + e^{net_{V_3}})$$

$$W_{U_0} W_{V_2}^T = w_{V_2H_0} w_{H_0U_0} + w_{V_2H_1} w_{H_1U_0}$$

$$\frac{\delta E}{\delta w_{V_2H_0}} = -w_{H_0U_0} + \frac{e^{W_{V_2}.W_{U_0}}}{e^{W_{V_0}.W_{U_0}} + e^{W_{V_1}.W_{U_0}} + e^{W_{V_2}.W_{U_0}} + e^{W_{V_3}.W_{U_0}})} . w_{H_0U_0}$$

$$= -w_{H_0U_0} + v_2 . w_{H_0U_0}$$

$$\Rightarrow \Delta w_{V_2H_0} = \eta(1 - v_2).w_{H_0U_0} = \eta(1 - v_2) o_{H_0}$$

o/p of hidden neuron H_0

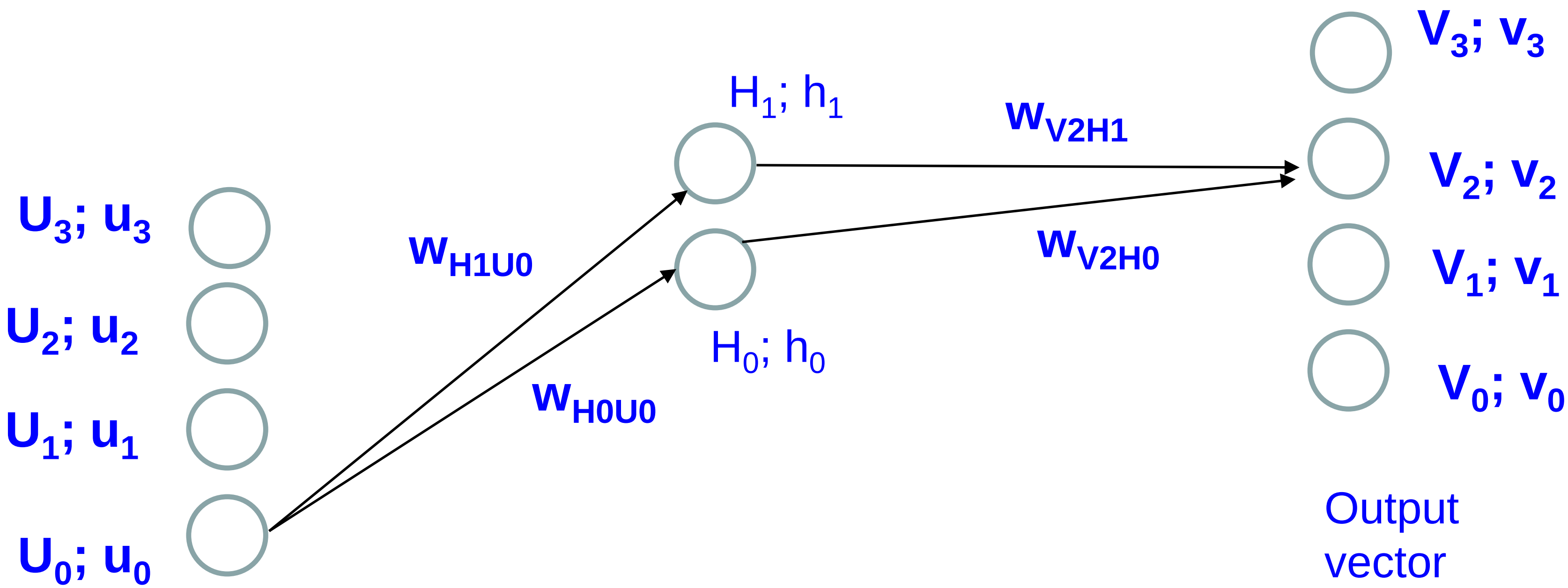
Interpretation of weight change rule for V_2

- If v_2 is close to 1, change in weight too is small

- w_{H0U0} is equal to the input to H_0 (since $u_0=1$) and to its output too, since hidden neurons simply transmit the output.

Word2vec n/w

Capital letter for NAME of neuron; small letter for output from the same neuron



Output vector V

Input vector U

Weights go from all neurons to all neurons in the next layer; shown For only one input and output

Change in other weights to output layer, say, V_1, due to input U_0

$$\Delta w_{V_1H_0} = -\eta \frac{\delta E}{\delta w_{V_1H_0}}$$

$$E = -net_{V_2} + \log(e^{net_{V0}} + e^{net_{V1}} + e^{net_{V2}} + e^{net_{V3}})$$

$$= -W_{U_0} W_{V_2}^T + \log(e^{net_{V0}} + e^{net_{V1}} + e^{net_{V2}} + e^{net_{V3}})$$

$$W_{U_0} W_{V_2}^T = w_{V_2H_0} w_{H_0U_0} + w_{V_2H_1} w_{H_1U_0}$$

$$\frac{\delta E}{\delta w_{V_1H_0}} = -0 + \frac{e^{W_{V1}.W_{U0}}}{e^{W_{V0}.W_{U0}} + e^{W_{V1}.W_{U0}} + e^{W_{V2}.W_{U0}} + e^{W_{V3}.W_{U0}})}.w_{H_0U_0}$$

$$= v_1.w_{H_0U_0}$$

$$\Rightarrow \Delta w_{V_1H_0} = -\eta v_1 w_{H_0U_0} = -\eta v_1 o_{H_0}$$

Interpretation of weight change rule for V_1

- Assume w_{H0U0} to be positive
- For training $U0 \rightarrow V2$, i.e., ‘heavy’→’rain’, if v_2 is not 1, Δw_{V2H0} is +ve
- For the same input, Δw_{V1H0} is negative
- So the two weight changes are of opposite sign.
- The effect is that while v_2 increases, v_1 decrease for the input U_0, as it should be since we want to increase *P(‘rain’|’heavy’)* and depress all other probabilities

Weight change for input to hidden layer, say, w_{H0U0}

$$\Delta w_{H_0U_0} = -\eta \frac{\delta E}{\delta w_{H_0U_0}}$$

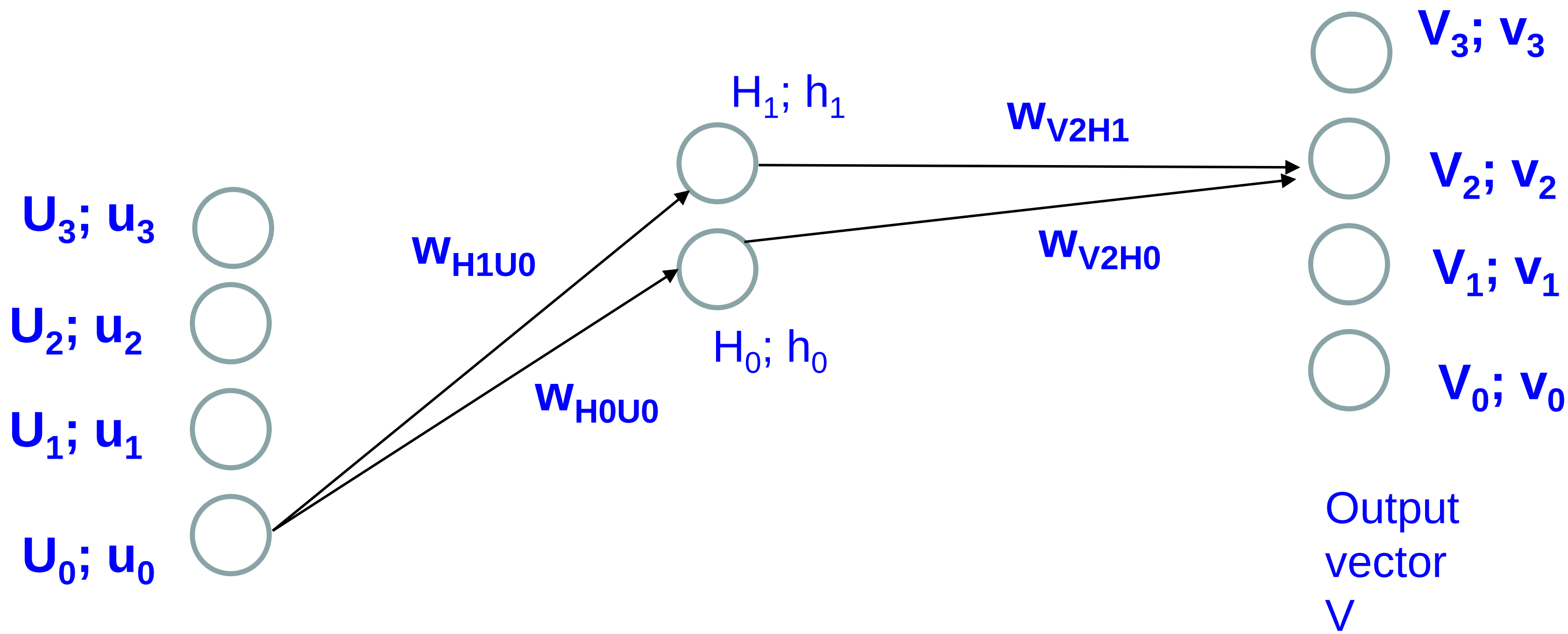
$$E = -net_{V_2} + \log(e^{net_{V0}} + e^{net_{V1}} + e^{net_{V2}} + e^{net_{V3}})$$

$$= -W_{U_0} W_{V_2}^T + \log(e^{net_{V0}} + e^{net_{V1}} + e^{net_{V2}} + e^{net_{V3}})$$

$$W_{U_0} W_{V_2}^T = w_{V_2H_0} w_{H_0U_0} + w_{V_2H_1} w_{H_1U_0}$$

Word2vec n/w

Capital letter for NAME of neuron; small letter for output from the same neuron



Input vector U

Weights go from all neurons to all neurons in the next layer; shown For only one input and output

Cntd: Weight change for input to hidden layer, say, w_{H0U0}

$$\frac{\delta E}{\delta w_{H_0U_0}}$$

$$= -w_{V_2H_0} + \frac{w_{V_0H_0} e^{W_{V_0}.W_{U_0}} + w_{V_1H_0} e^{W_{V_1}.W_{U_0}} + w_{V_2H_0} e^{W_{V_2}.W_{U_0}} + w_{V_3H_0} e^{W_{V3}.W_{U_0}}}{e^{W_{V_0}.W_{U_0}} + e^{W_{V_1}.W_{U_0}} + e^{W_{V_2}.W_{U_0}} + e^{W_{V_3}.W_{U_0}}}$$

$$= -w_{V_2H_0} + w_{V_0H_0} v_0 + w_{V_1H_0} v_1 + w_{V_2H_0} v_2 + w_{V_3H_0} v_3$$

$$\Rightarrow \Delta w_{H_0U_0} = \eta[(1 - v_2) w_{V_2H_0} - w_{V_0H_0} v_0 - w_{V_1H_0} v_1 - w_{V_3H_0} v_3]$$

Need for efficiency

- Hierarchical softmax
- Negative sampling
- We have to update |H|.|V| weights in the hidden to output layer
- |H|=dimension of hidden layer, |V|=vocab size
- For 300 dimension word vector and 100,000 words vocabulary, 30 million weights need to be updated for every input word!!
- Efficiency measures to be discussed

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*

$P(x)$: $\langle 1,0,0 \rangle$, $Q(x)$: $\langle 0.9,0.02,0.08 \rangle$, (say)

$H(P,Q) = -\log 0.9 = \log(10/9)$

Deriving weight change rules

Cross Entropy Softmax combination

A very ubiquitous combination in neural combination

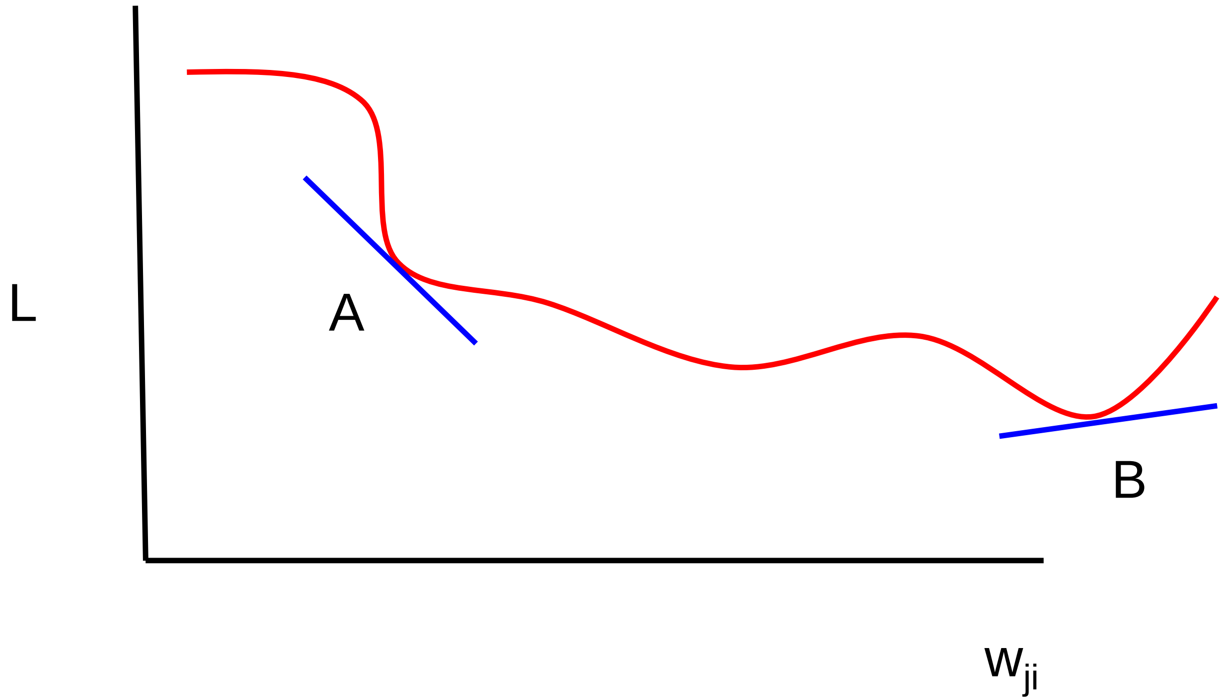
Foundation: Gradient descent

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

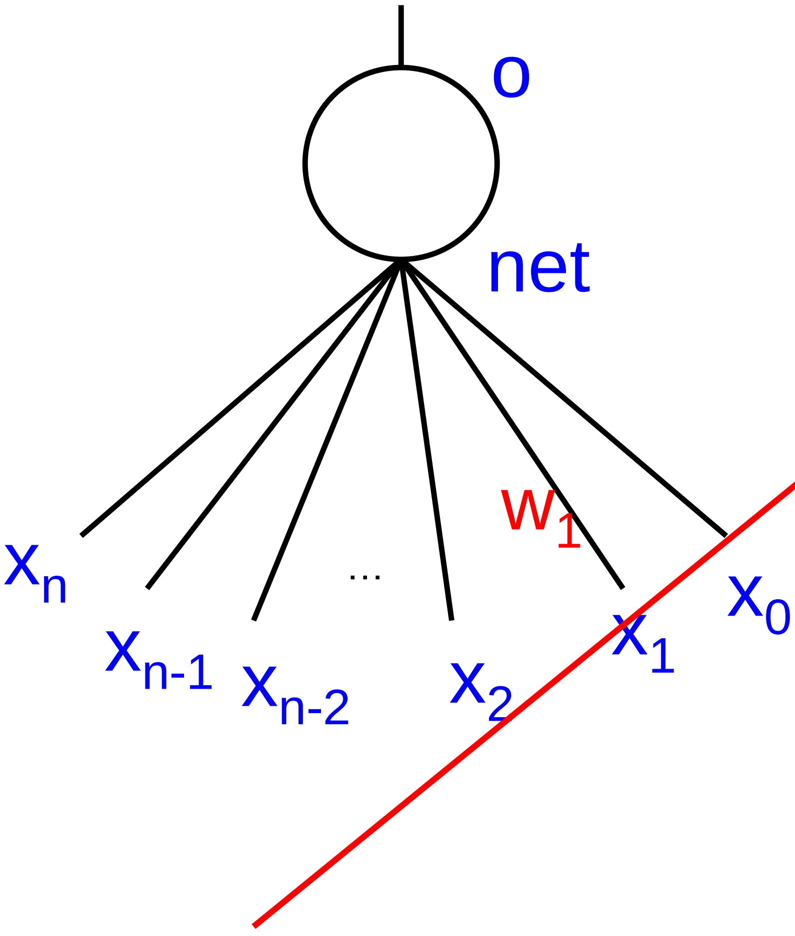
η= *learning rate, L=loss,*

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



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

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

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

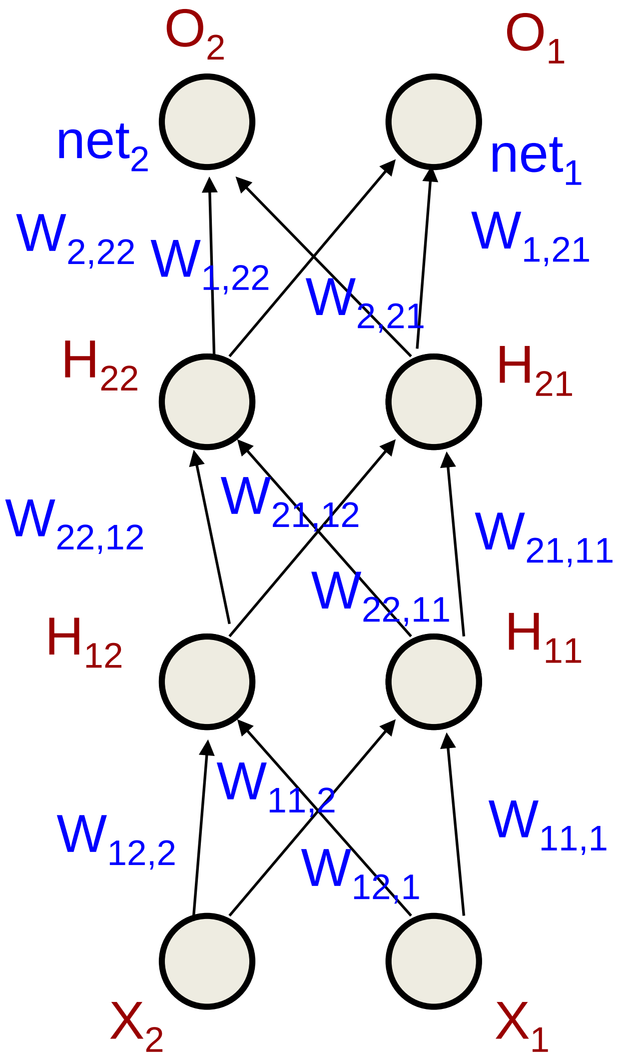
$$o = \frac{1}{1+e^{-net}} \; (sigmoid) \Rightarrow \frac{\partial o}{\partial net} = o(1-o) \; (2)$$

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

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

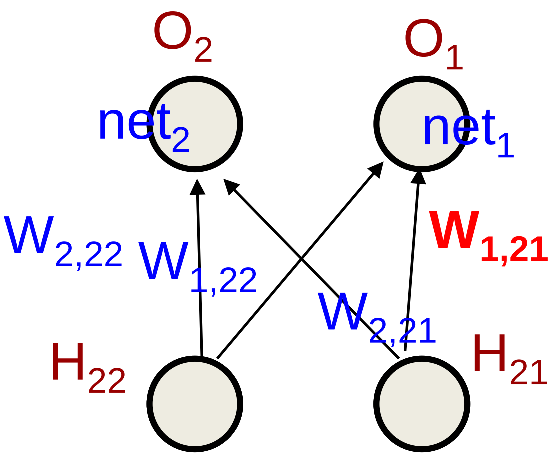
$$\Delta w_1 = \eta(t-o)x_1$$

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



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

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

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

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

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

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

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

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

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

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

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

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

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

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

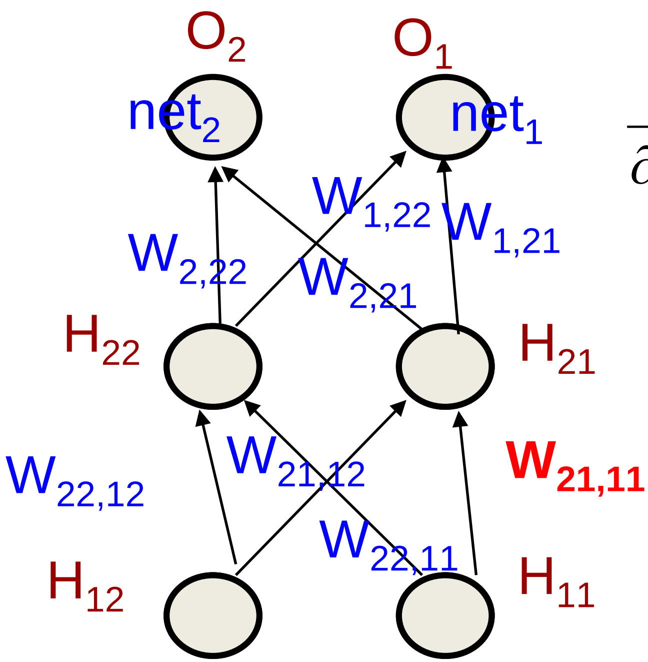
$$= \frac{\partial E}{\partial h_{21}} .r'(H_{21}); \quad r' = derivative\_RELU(H_{21})$$

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

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

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

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

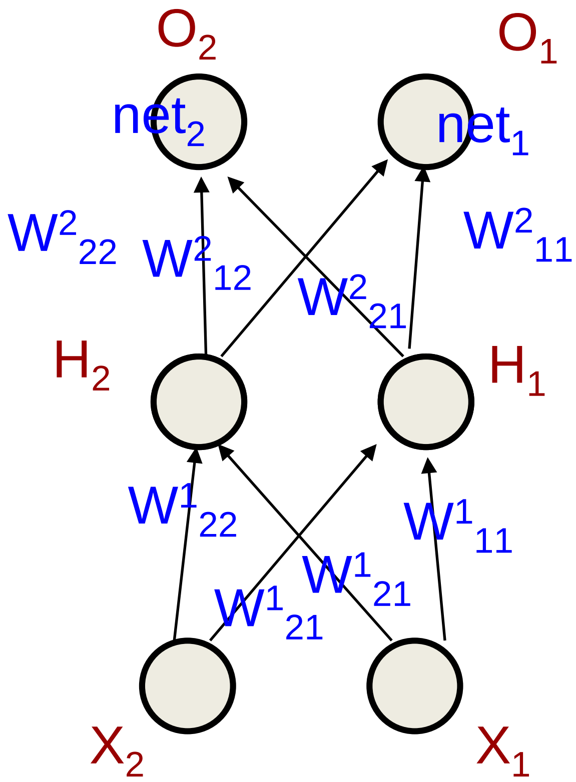


$\Delta W_{21,11} = \eta[(t_2 - o_2)W_{2,21} + (t_1 - o_1)W_{1,21}].r'(H_{21}).h_{11}$

An Example

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



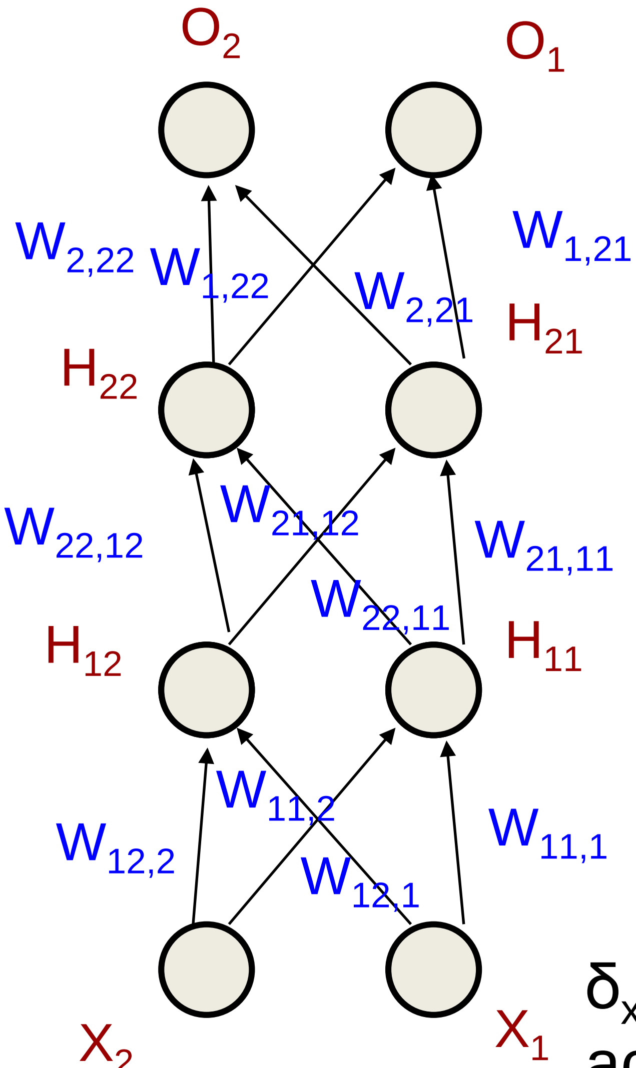
$$E = -t_2 \log o_2 - t_1 \log o_1$$

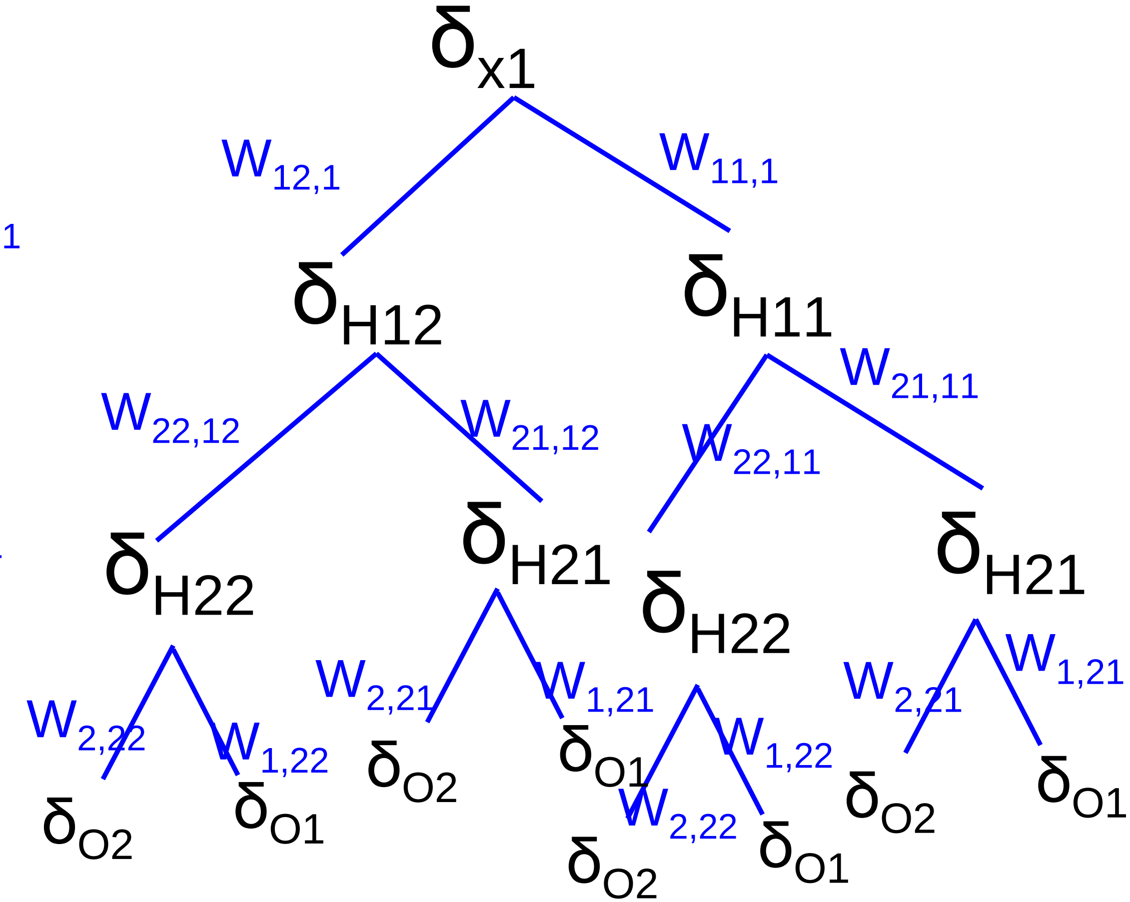
$$\Delta W^2_{11} = \eta (t_1 - o_1) h_1$$

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

Why is RELU a solution for vanishing or exploding gradient?

Vanishing/Exploding Gradient





$\delta_{x1}=[W_{11,1}\delta_{H11}+W_{12,1}\delta_{H12}]$.derivative of activation at $X_1=[W_{11,1}\delta_{H11}+W_{12,1}\delta_{H12}].1$ (convention)

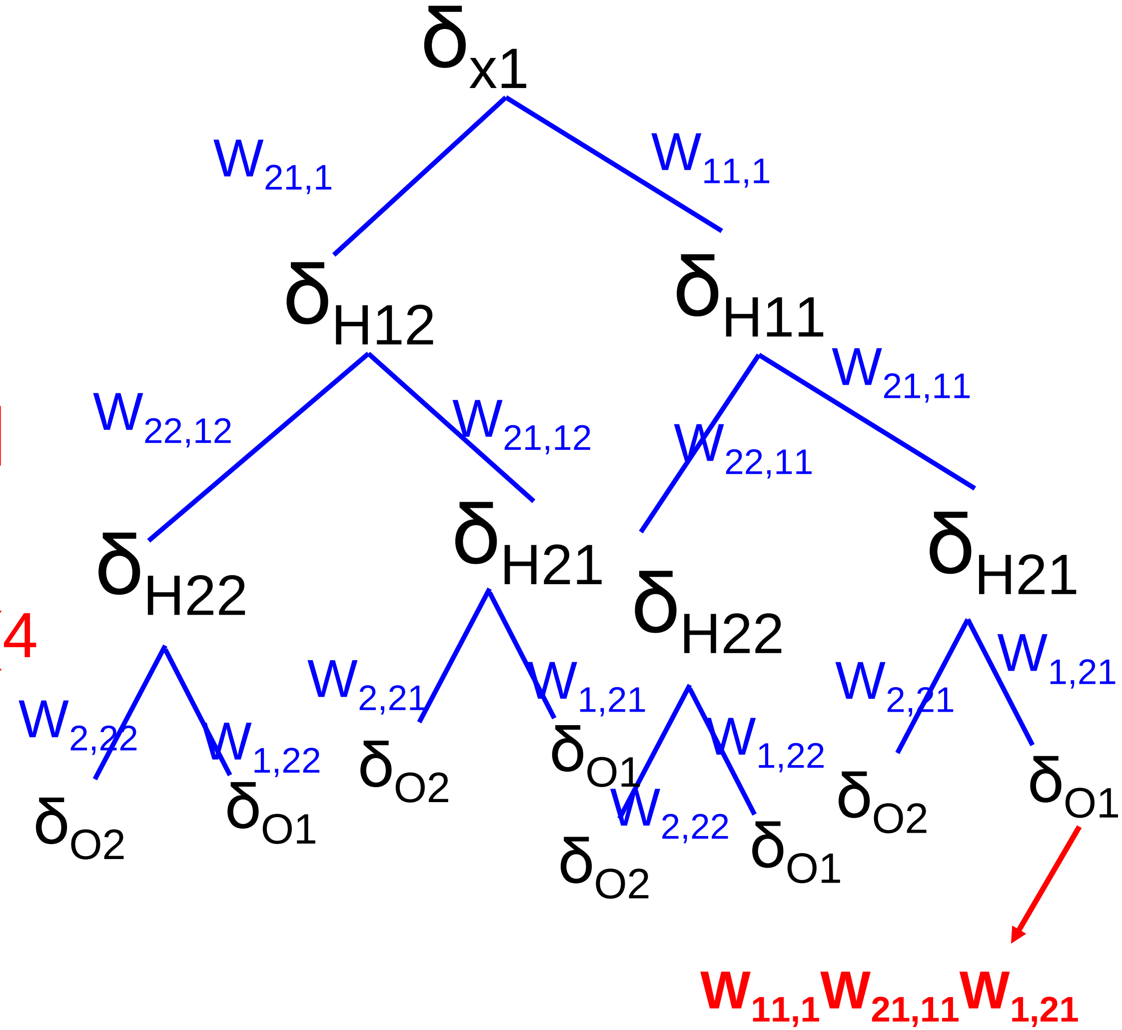
Vanishing/Exploding Gradient

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

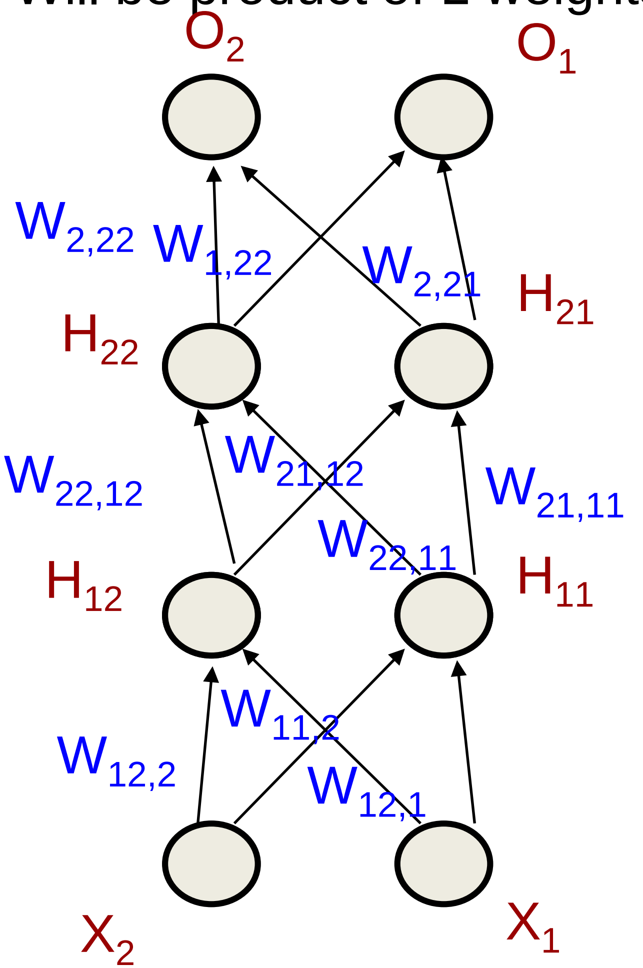
= (4 terms involving δ_{o1}) + (4 terms involving δ_{o2})

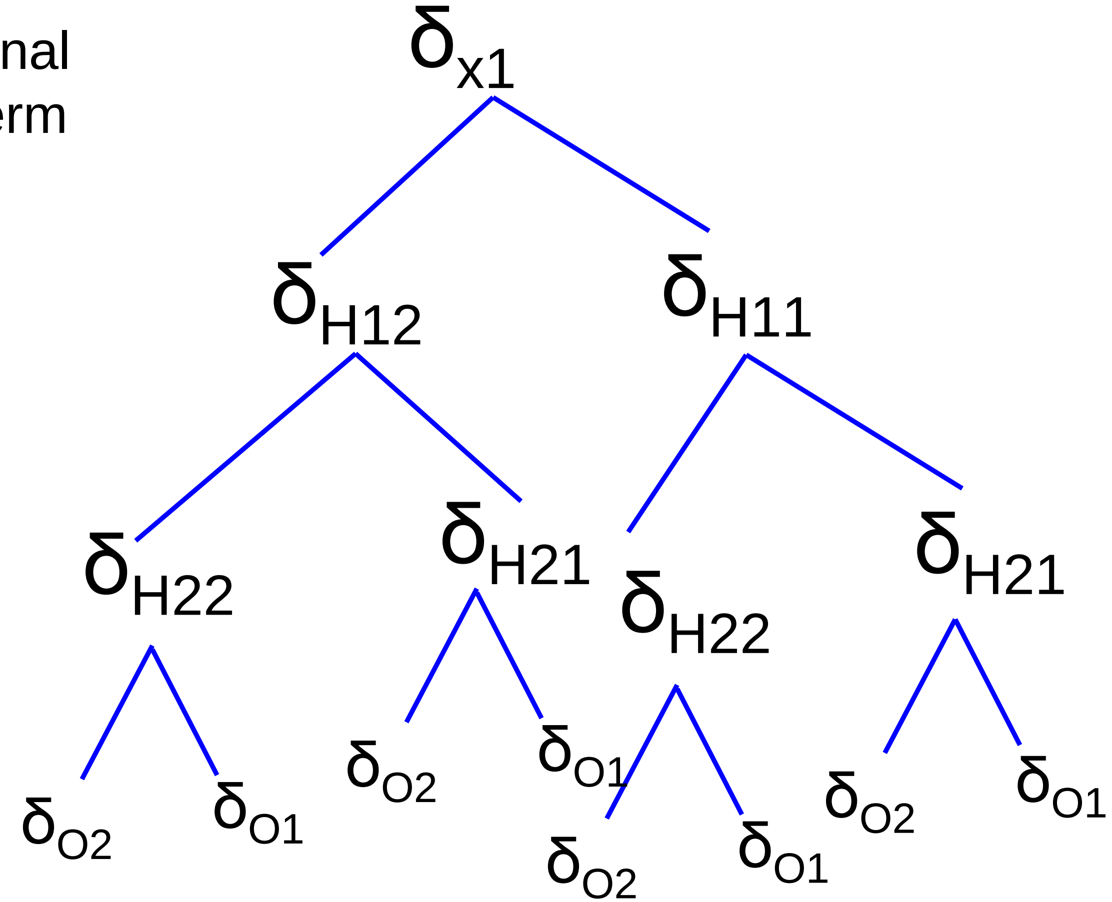
δs get multiplied by derivatives of RELU which are 1 or 0; hence δs from the output layer pass as such or as 0



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/

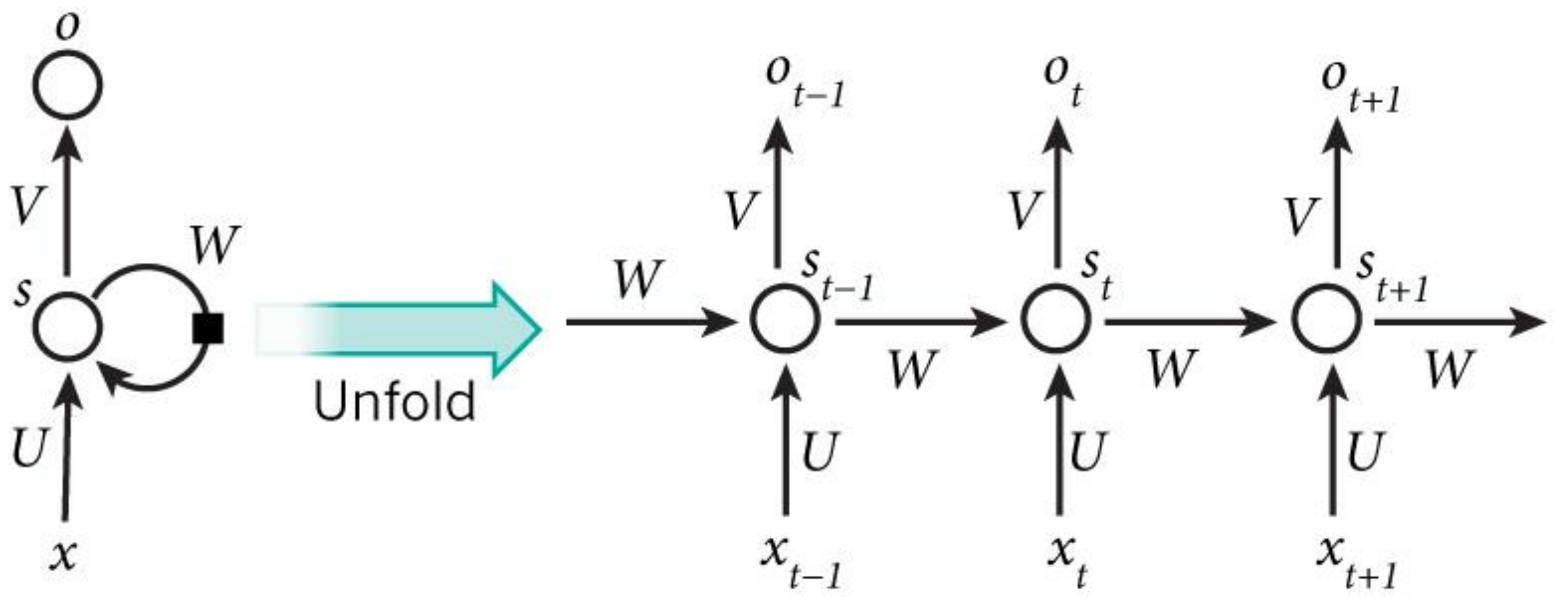
   By Denny Britz

2. Introduction to RNN by Jeffrey Hinton

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

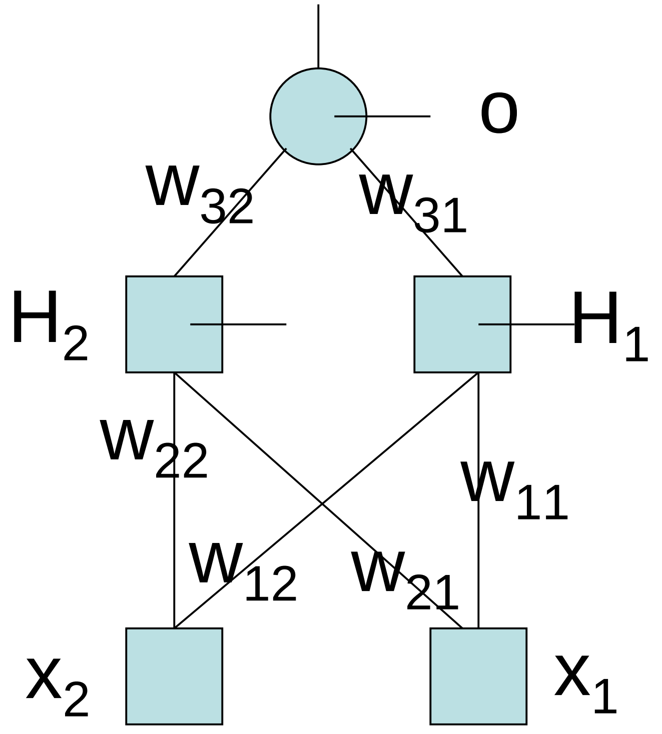
3. Dr. Anoop Kunchukuttan, Microsoft and ex-CFILT

Sequence processing m/c

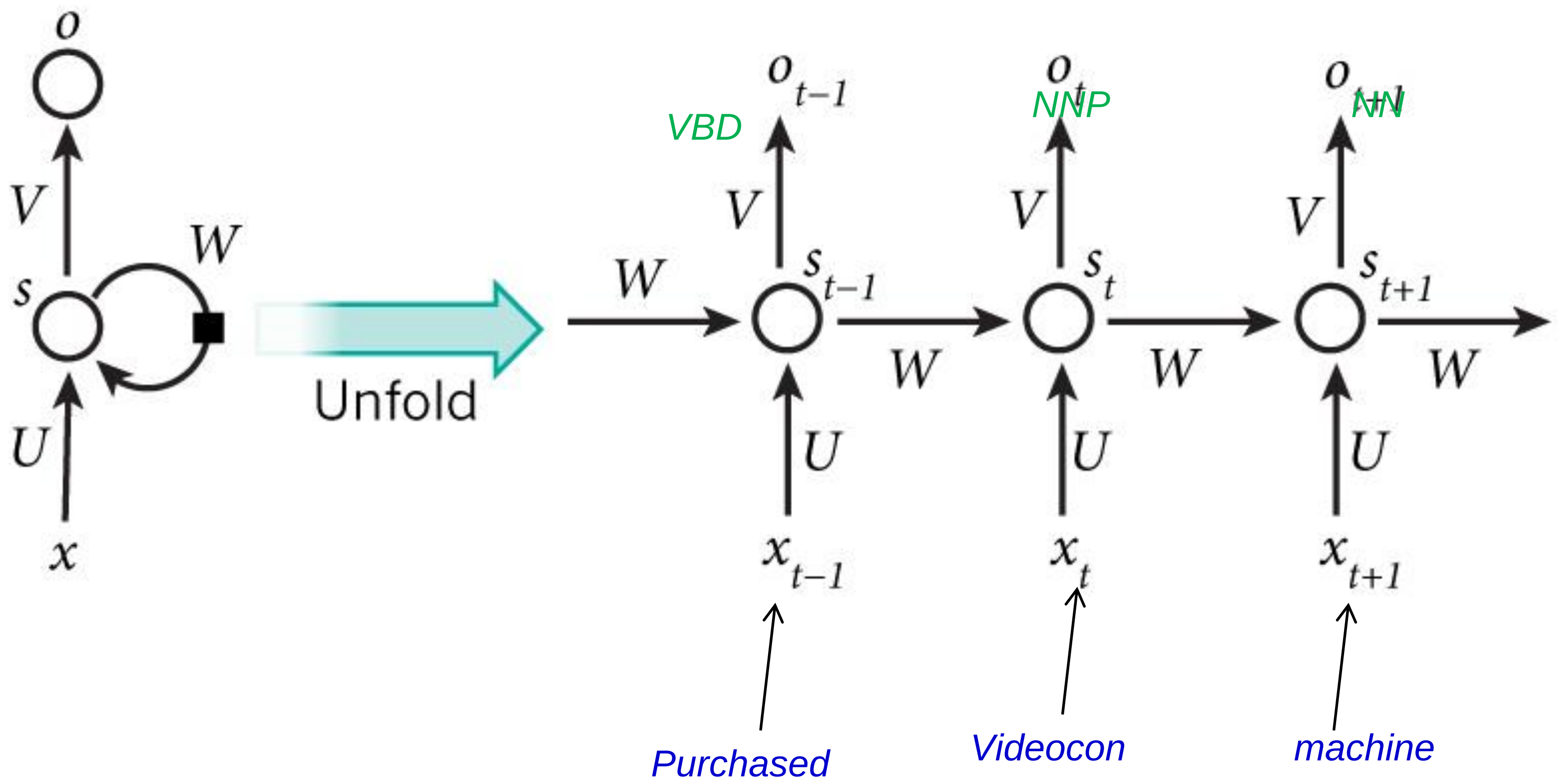


Meaning of state

- 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”

POS Annotation

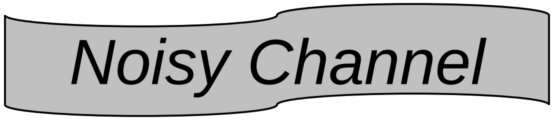
- *Who_WP is_VZ the_DT prime_JJ minister_NN of _IN India_NNP ?_PUNC*

- Becomes the training data for ML based POS tagging

3 Generations of POS tagging techniques

- Rule Based POS Tagging
  - Rule based NLP is also called Model Driven NLP
- Statistical ML based POS Tagging (*Hidden Markov Model, Support Vector Machine*)
- Neural (Deep Learning) based POS Tagging

Noisy Channel Model

W  **T**

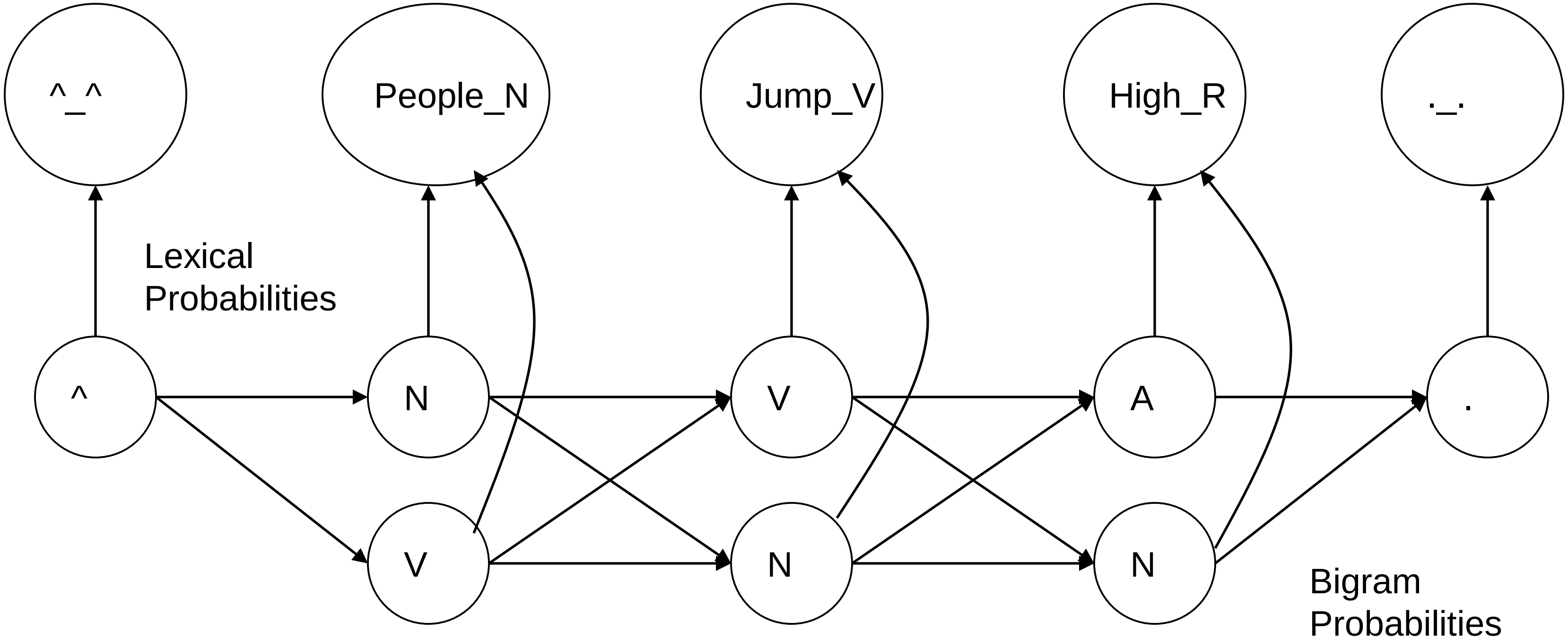
$(w_n, w_{n-1}, \ldots, w_1)$ $(t_m, t_{m-1}, \ldots, t_1)$

Sequence *W* is transformed into sequence *T*

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

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

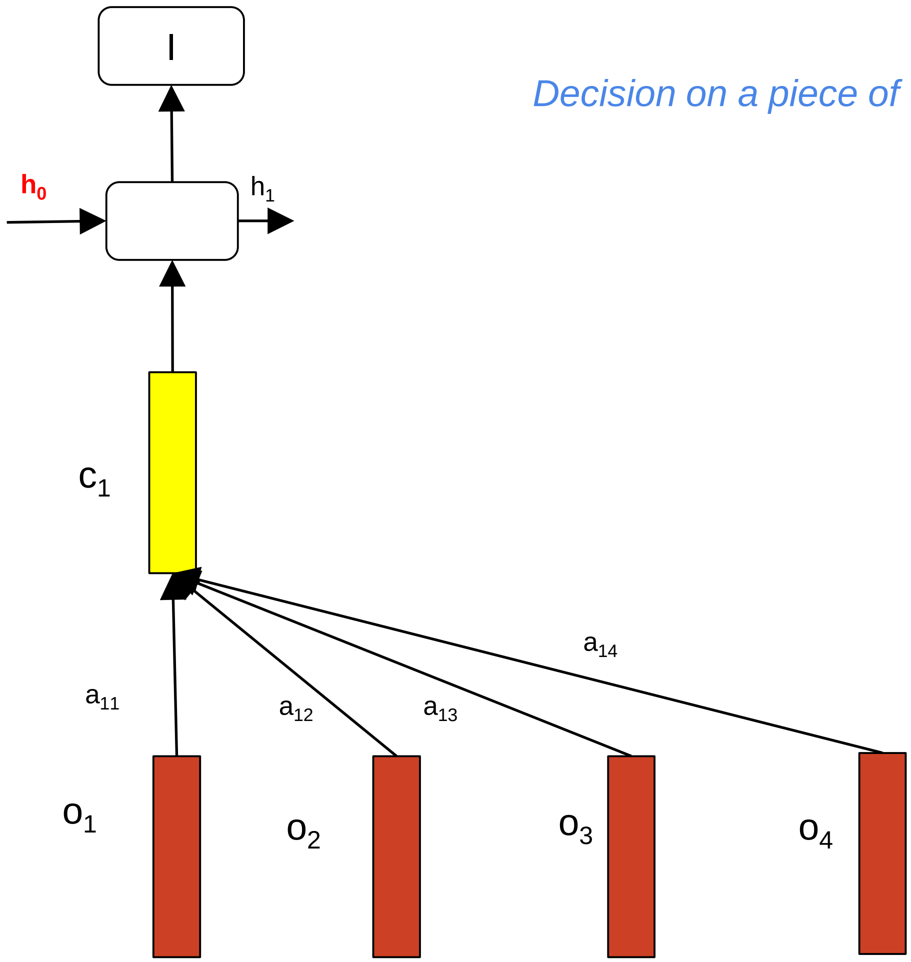
HMM: Generative Model

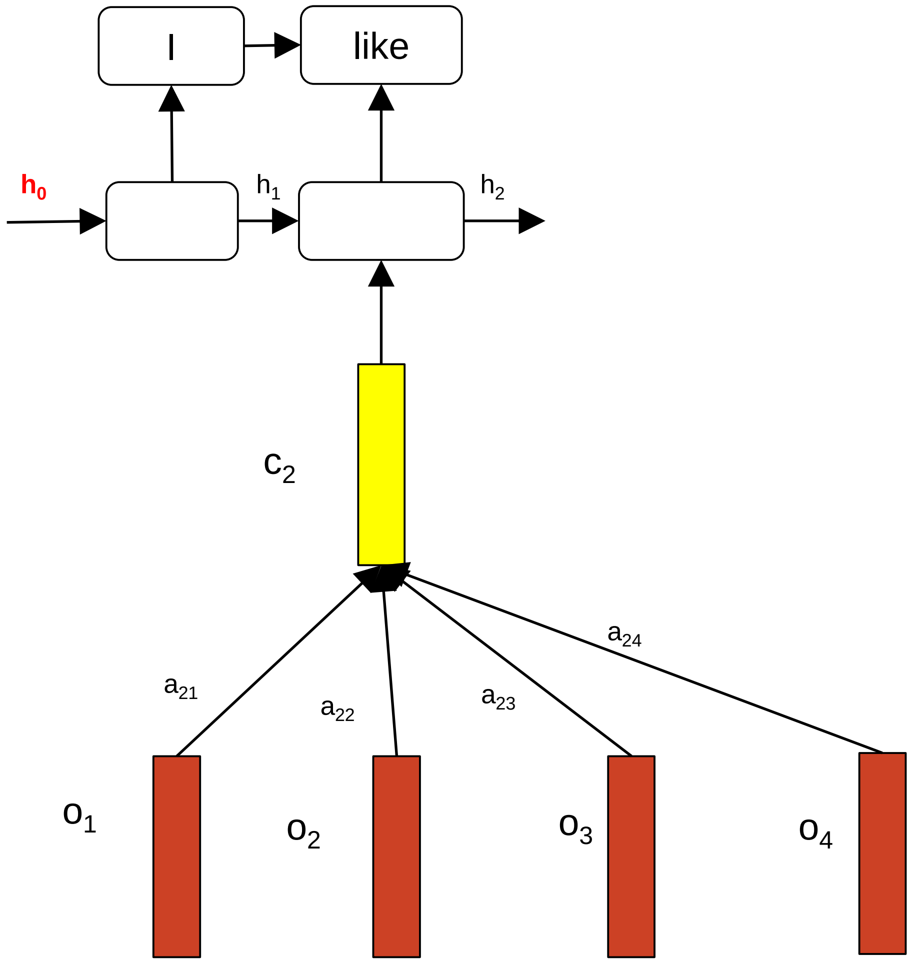


This model is called Generative model.
Here words are observed from tags as states.
This is similar to HMM.

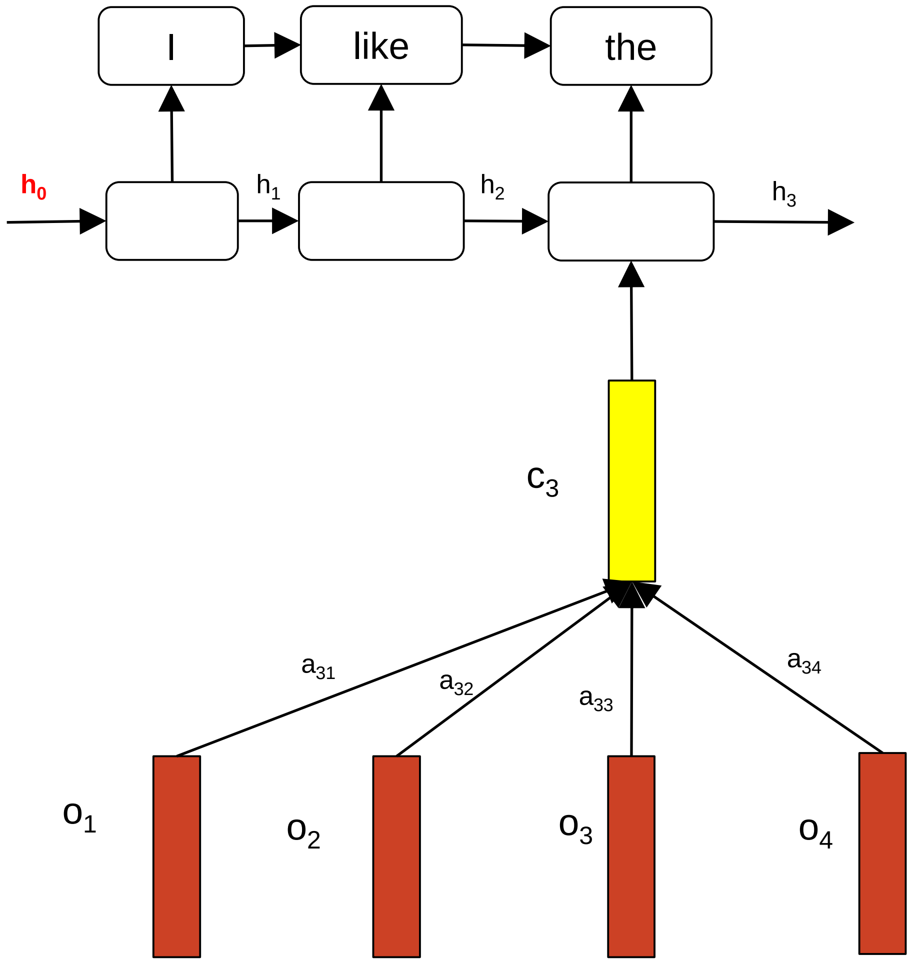
E.g. Sentiment Analysis

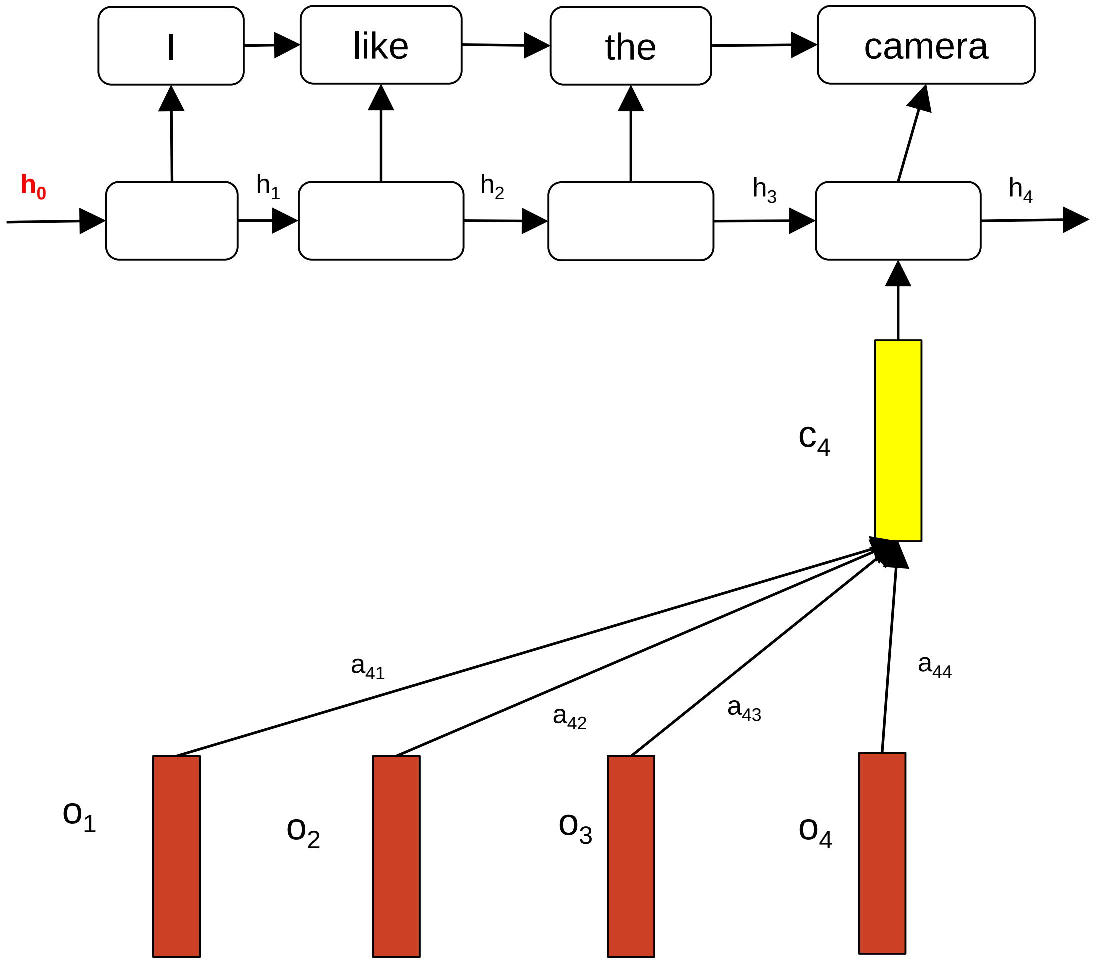
Decision on a piece of text

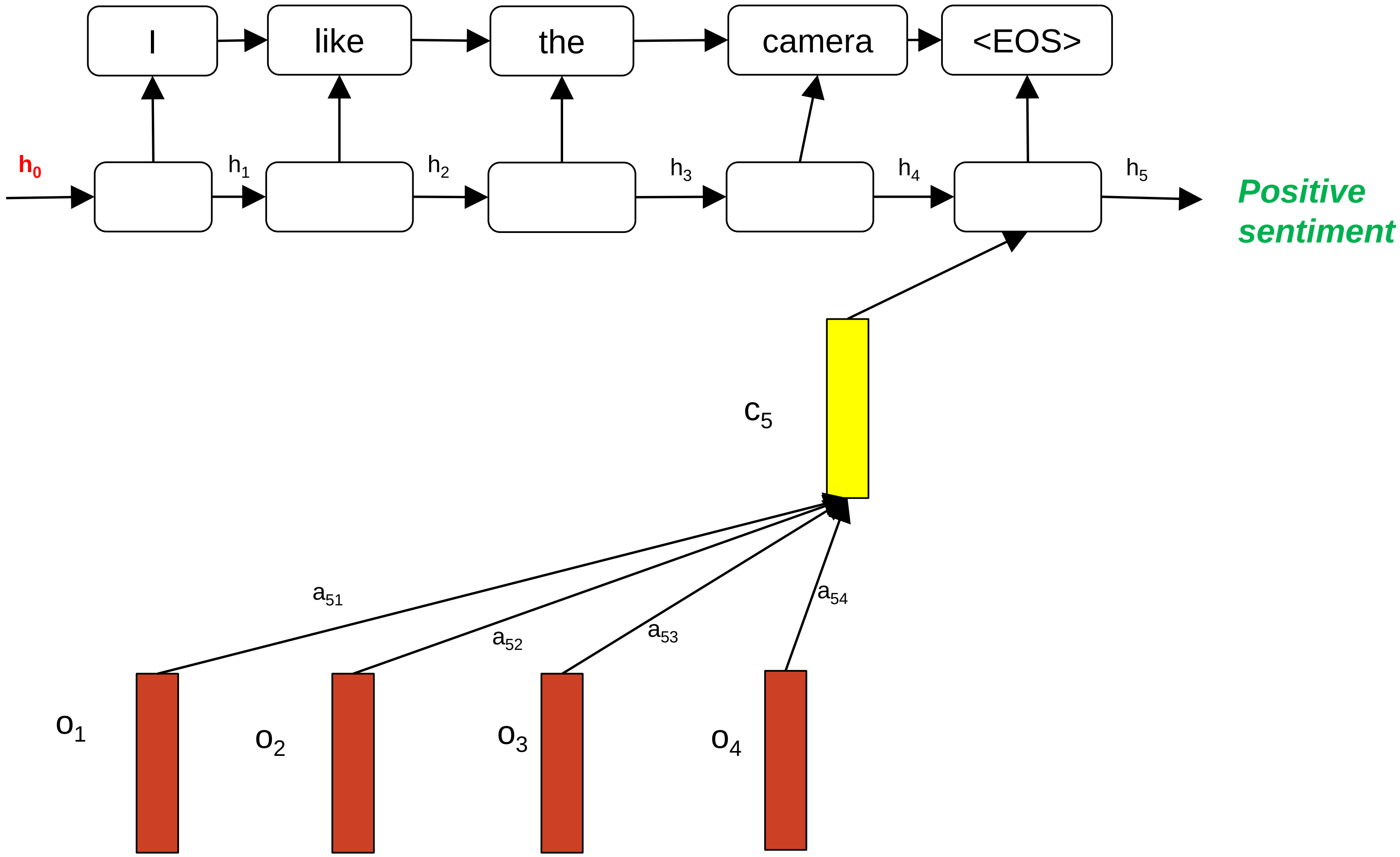




I
like
h0
h1
h2
c2
a21
a22
a23
a24
o1
o2
o3
o4

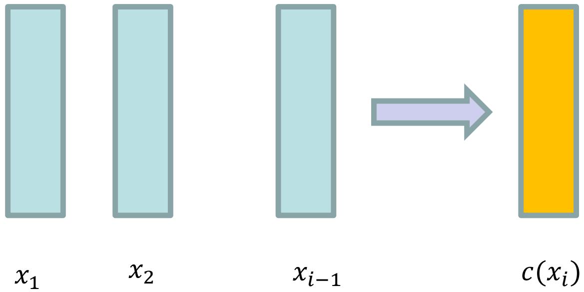

I
like
the
h0
h1
h2
h3
c3
a31
a32
a33
a34
o1
o2
o3
o4


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


I
like
the
camera
<EOS>
h0
h1
h2
h3
h4
h5
Positive sentiment
c5
a51
a52
a53
a54
o1
o2
o3
o4

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

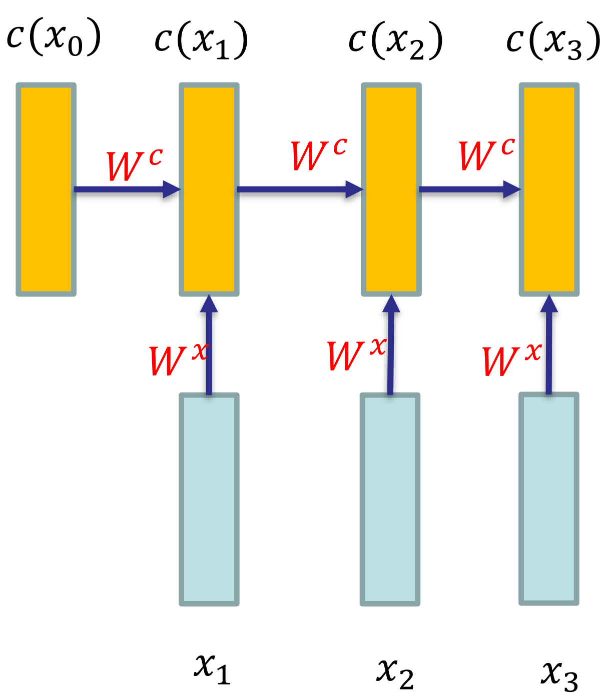
FF LM: $c(x_i) = G(x_{i-1}, x_{i-2})$ (trigram LM)

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 representation

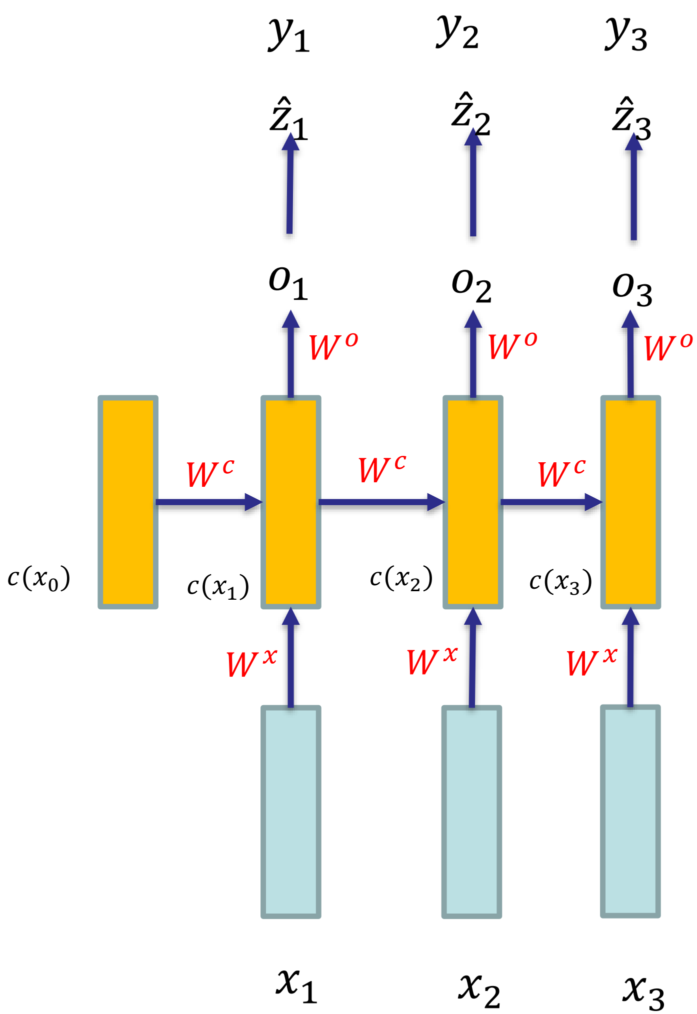
$F(.)$ can be implemented as

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

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

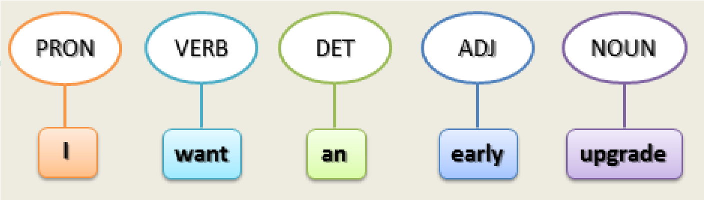
Input and output sequences have the same length

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	capital	of	Maharashtra	is	Mumbai	<EOS>
<BOS>	The	capital	of	Maharashtra	is	Mumbai

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 test sets 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

Importance of Probabilistic Language Modelling (1/2)

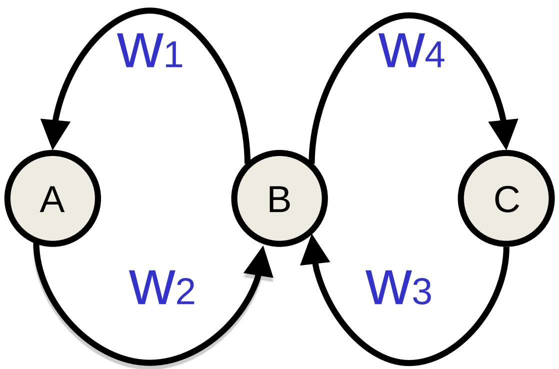
- In early days, researchers used context free grammar for language models
  - Is a given string of words in language or not
  - Example:
    - *Ram saw Shyam* (correct word order)
    - *Ram Shyam saw* (incorrect word order)
  - However, belongingness to language is not a black and white issue
  - There are no grammatical and ungrammatical sentences, only sentences with probabilities

Importance of Probabilistic Language Modelling (2/2)

- Example:
  - Indian English: *You will go to the movie, no?*
  - US/UK English: *You will go to the movie, won't you?*
- English has different forms through differences in regional dialects and even through periods of time
  - English language evolves every year, new words and their different sentence positions are introduced
- Hence we cannot assign 0/1 value to sentences
  - But we can assign probabilities to word orders
  - Equivalent to Prob (Wn | W1,W2,...Wn-1)

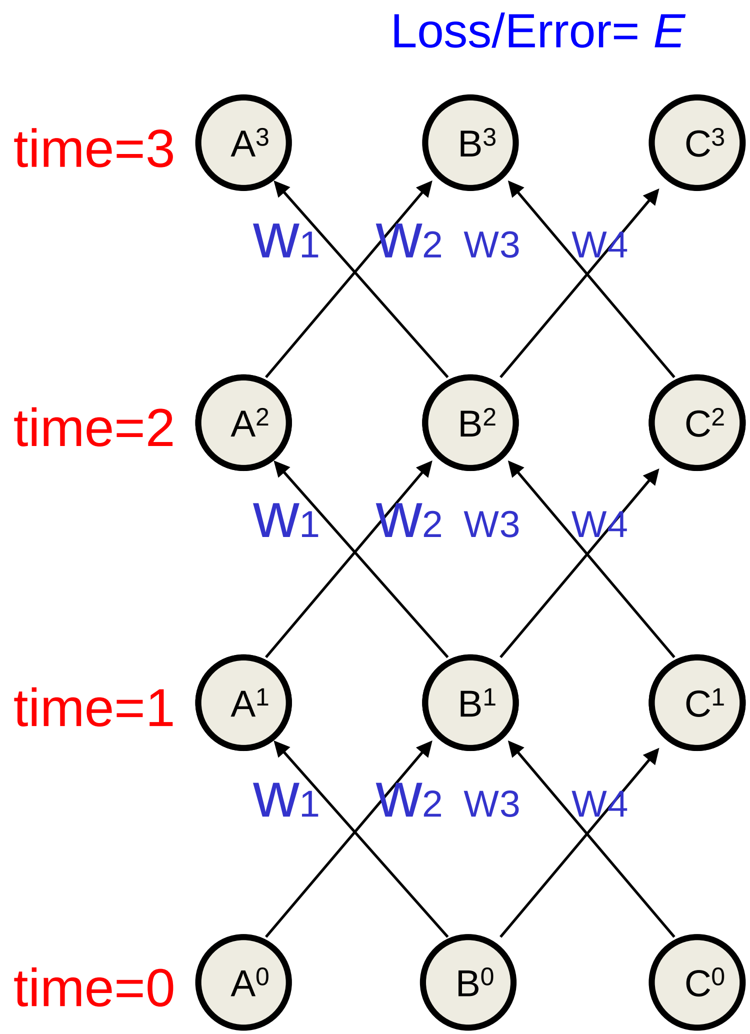
BPTT

The equivalence between feedforward nets and recurrent nets



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

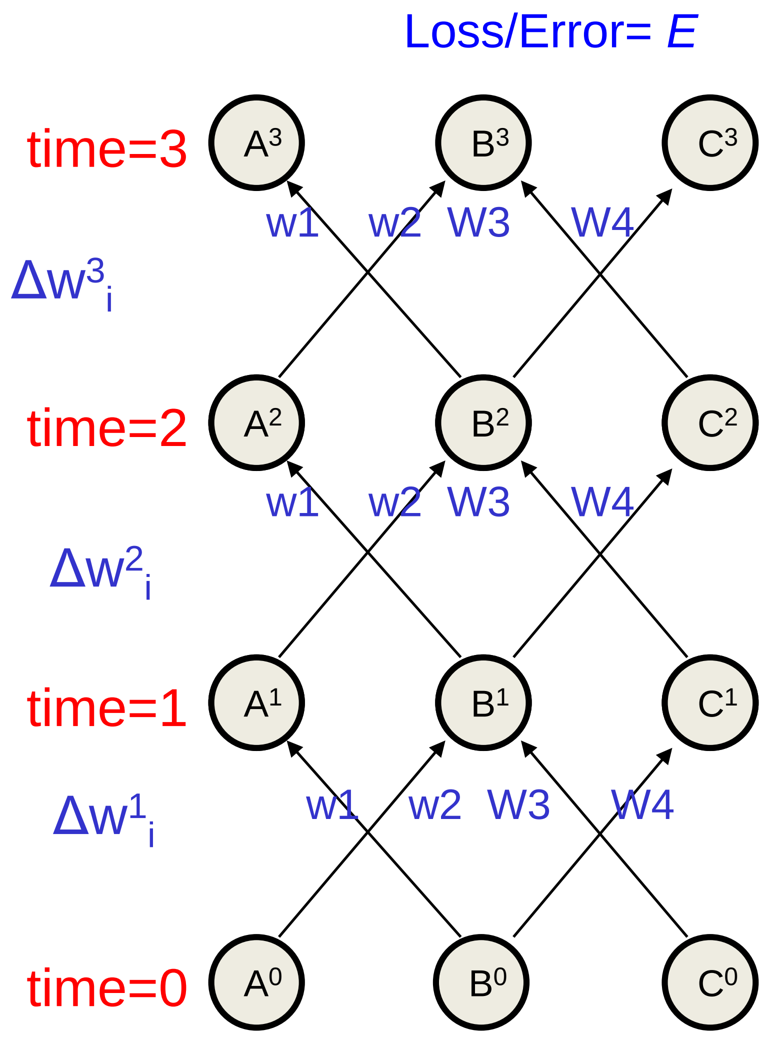
The recurrent net is just a layered net that keeps reusing the same weights.



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.

Long word sequences

- The famous book by Charles Dickens “A Tale of Two Cities” starts the book with the famous sentence “This was the best of times, this was the worst of times….”
- The sentence has 119 words
- “It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness, it was the epoch of belief, it was the epoch of incredulity, it was the season of Light, it was the season of Darkness, it was the spring of hope, it was the winter of despair, we had everything before us, we had nothing before us, we were all going direct to Heaven, we were all going direct the other way--in short, the period was so far like the present period that some of its noisiest authorities insisted on its being received, for good or for evil, in the superlative degree of comparison only.

The “best of times…” sentence

- Vanishing gradient will surely strike!!
- Exercise: give an example from NLP, where exploding gradient will strike!!

Sentence-1

- Ram who is a good student and lives in London which is a large metro, will go to the University for higher studies.
- राम जो एक अच्छा छात्र है और लंदन में रहता है जो एक बड़ी मेट्रो है, उच्च अध्ययन के लिए विश्वविद्यालय जाएगा।

Sentence-2

- Sita who is a good student and lives in London which is a large metro, will go to the University for higher studies.
- सीता जो एक अच्छी छात्रा है और लंदन में रहती है जो एक बड़ी मेट्रो है, उच्च अध्ययन के लिए विश्वविद्यालय जाएगी।

Long distance dependency: WSD

The ***bank***

Long distance dependency: WSD

The ***bank*** that Ram

Long distance dependency: WSD

The *bank* that Ram used to visit

Long distance dependency: WSD

The ***bank*** that Ram used to visit 30 years before

Long distance dependency: WSD

The ***bank*** that Ram used to visit 30 years before was closed

Long distance dependency: WSD

The ***bank*** that Ram used to visit 30 years before was closed due to

Long distance dependency: WSD

The ***bank*** that Ram used to visit 30 years before was closed due to the lockdown

Long distance dependency: WSD

The ***bank*** that Ram used to visit 30 years before was closed due to the lockdown with the Govt

Long distance dependency: WSD

The *bank* that Ram used to visit 30 years before was closed due to the lockdown with the Govt. getting worried that

Long distance dependency: WSD

The *bank* that Ram used to visit 30 years before was closed due to the lockdown with the Govt. getting worried that crowding of people

Long distance dependency: WSD

The *bank* that Ram used to visit 30 years before was closed due to the lockdown with the Govt. getting worried that crowding of people during the

Long distance dependency: WSD

The ***bank*** that Ram used to visit 30 years before was closed due to the lockdown with the Govt. getting worried that crowding of people during the immersion ceremony

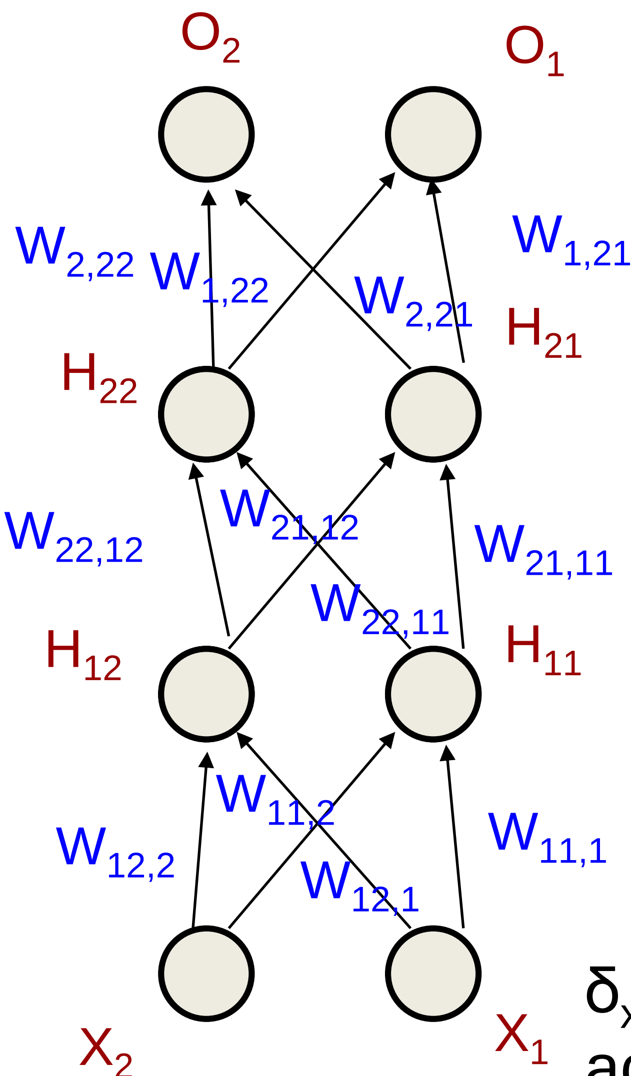
Long distance dependency: WSD

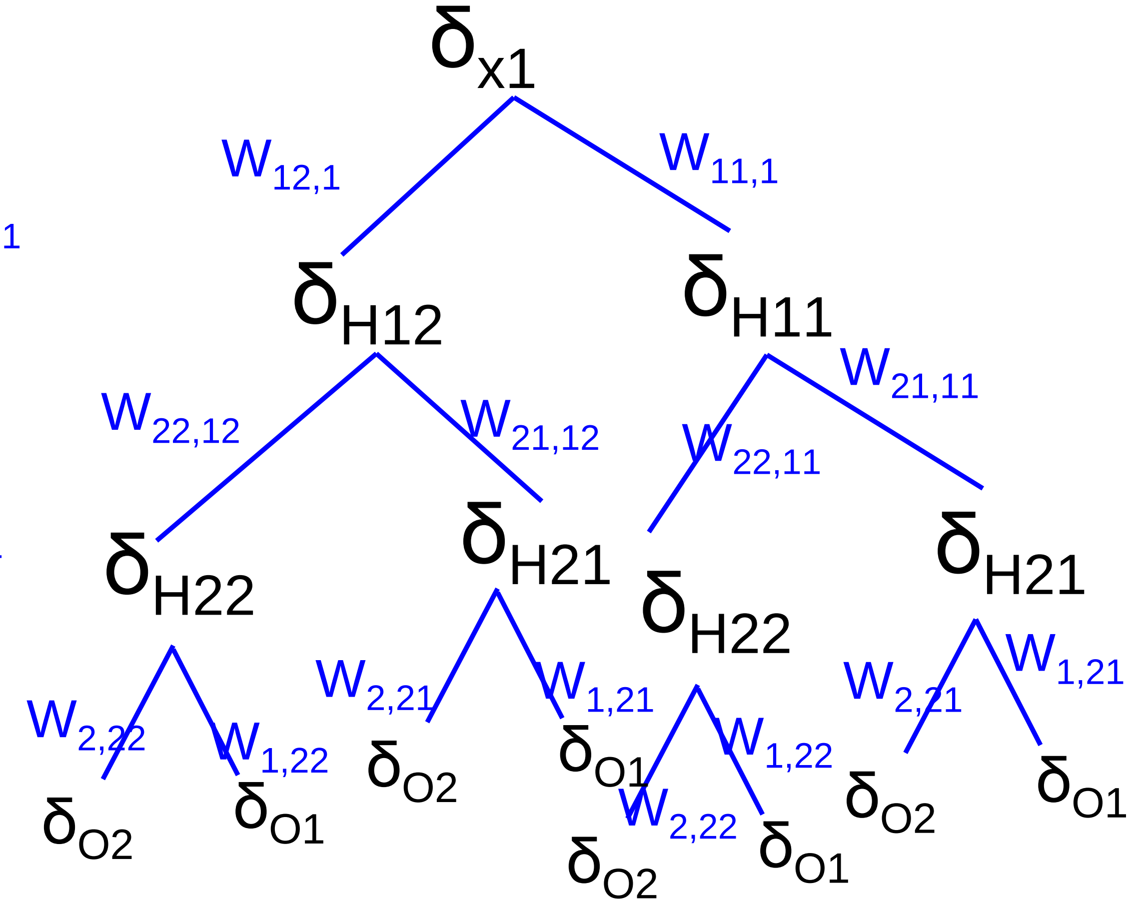
The *bank* that Ram used to visit 30 years before was closed due to the lockdown with the Govt. getting worried that crowding of people during the immersion ceremony on the river will aggravate the situation.

Movement of probability mass for “bank”

- Seeing “closed”, probability mass edges toward “financial” sense, because of strong association between “bank” and “closed/open”
- “lockdown” pushed this probability mass towards “river bank”
- Push further strengthened by arrival of “crowding”, “immersion” and “river” one after the other; “river” closes the case!

Vanishing/Exploding Gradient





$\delta_{x1}=[W_{11,1}\delta_{H11}+W_{12,1}\delta_{H12}]$.derivative of activation at $X_1=[W_{11,1}\delta_{H11}+W_{12,1}\delta_{H12}].1$ (convention)

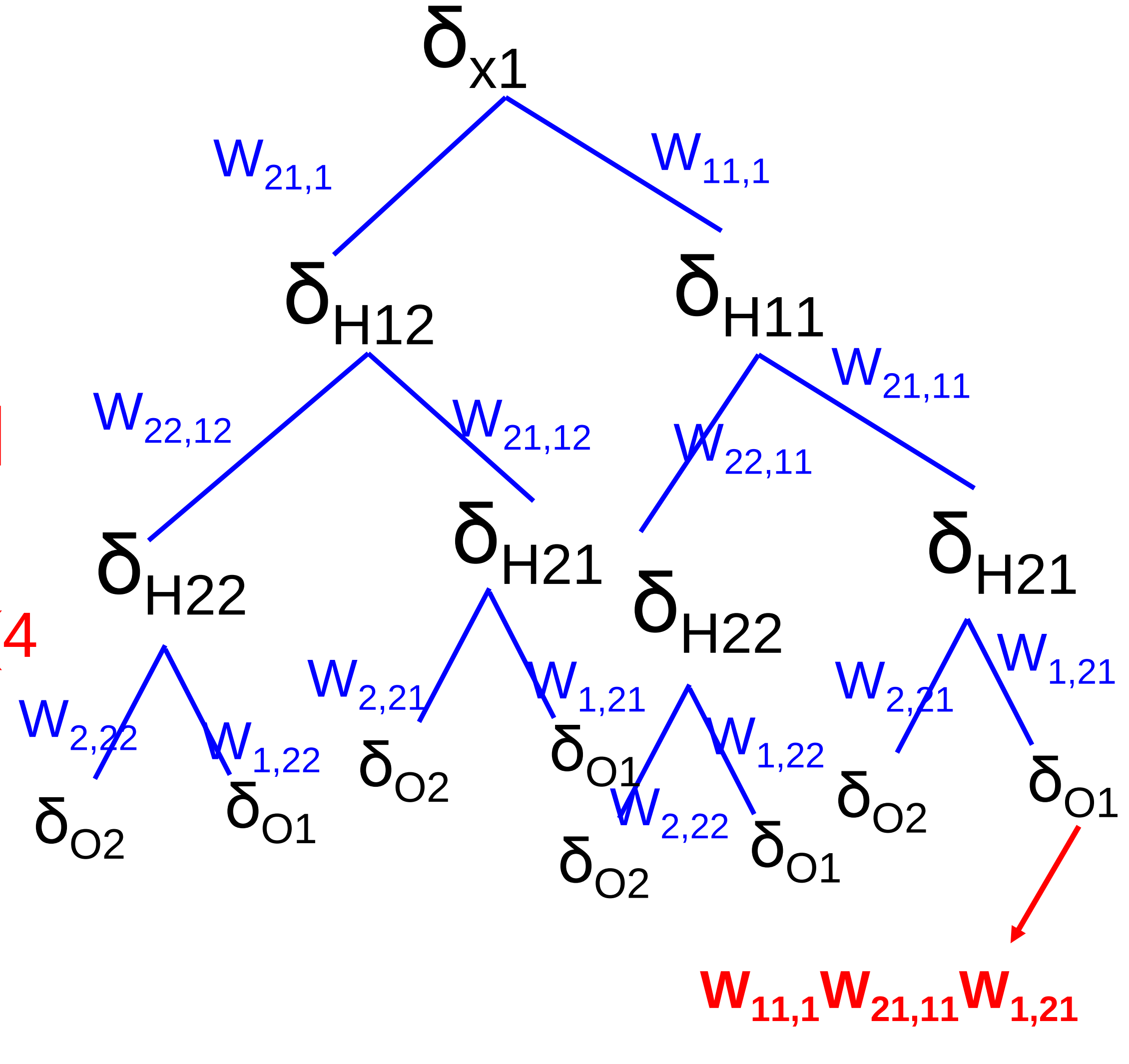
Vanishing/Exploding Gradient

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