Natural Language Processing

Pushpak Bhattacharyya
CSE Dept,
IIT Patna and Bombay

Introduction to DL-NLP

Motivation for DL-NLP

Text classification using Deep NN

- Text classification at the heart of many NLP tasks
- Soft decisions needed
- A piece of text may belong to multiple classes
  - "Recent curb on H1-B in the US visas- promised in Trump's election speeches- are causing IT industries to re-orient their business plan"
- Belongs to BOTH economics and Politics- *more* to former

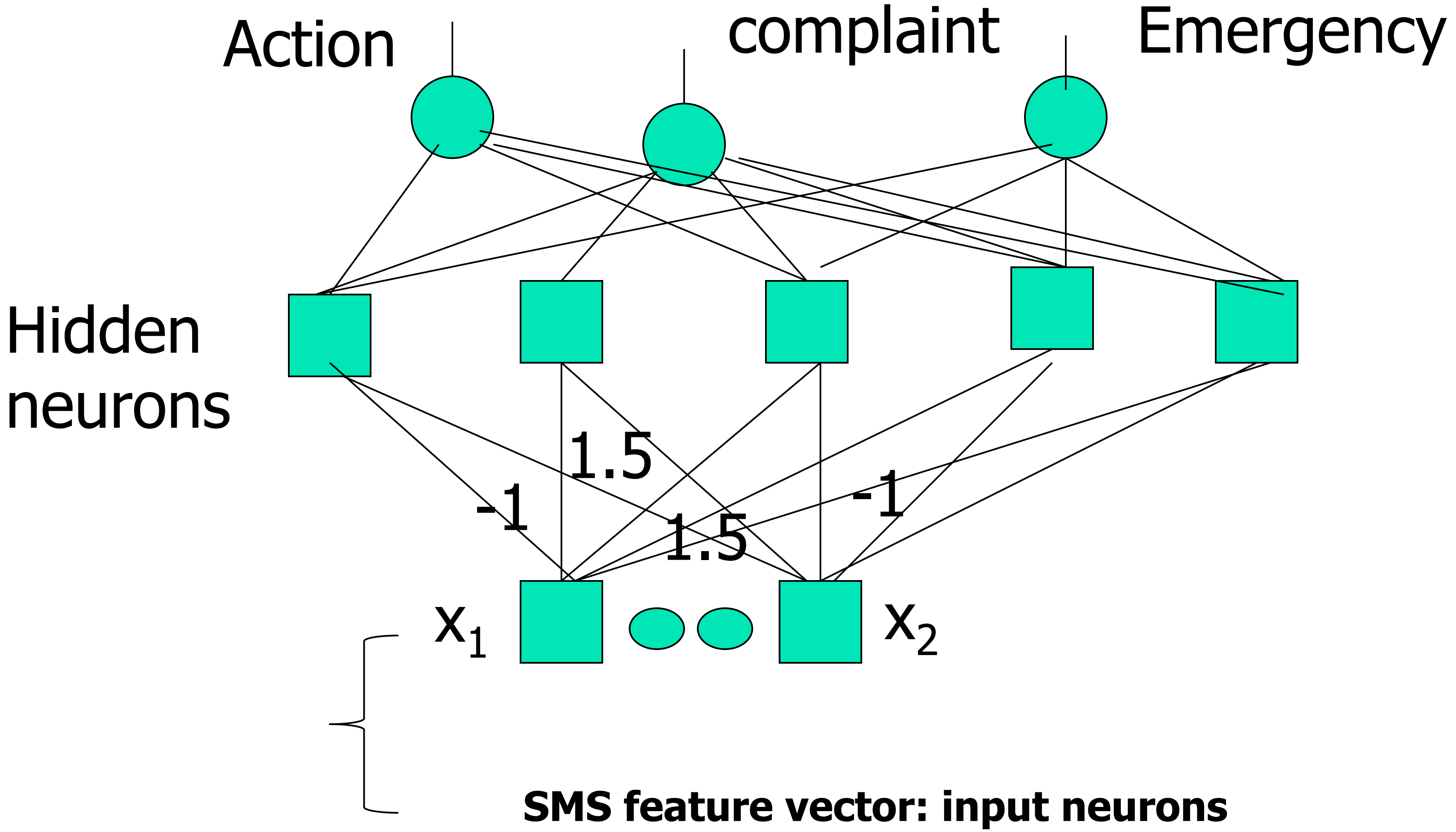
An example SMS complaint

- I have purchased a 80 litre Videocon fridge about 4 months ago when the freeze go to sleep that time compressor give a sound (khat khat khat khat ....) what is possible fault over it is normal I can't understand please help me give me a suitable answer.

Significant words (in red): after stop word removal

- I have purchased a 80 litre Videocon fridge about 4 months ago when the freeze go to sleep that time compressor give a sound (khat khat khat khat ....) what is possible fault over it is normal I can't understand please help me give me a suitable answer.

SMS classification



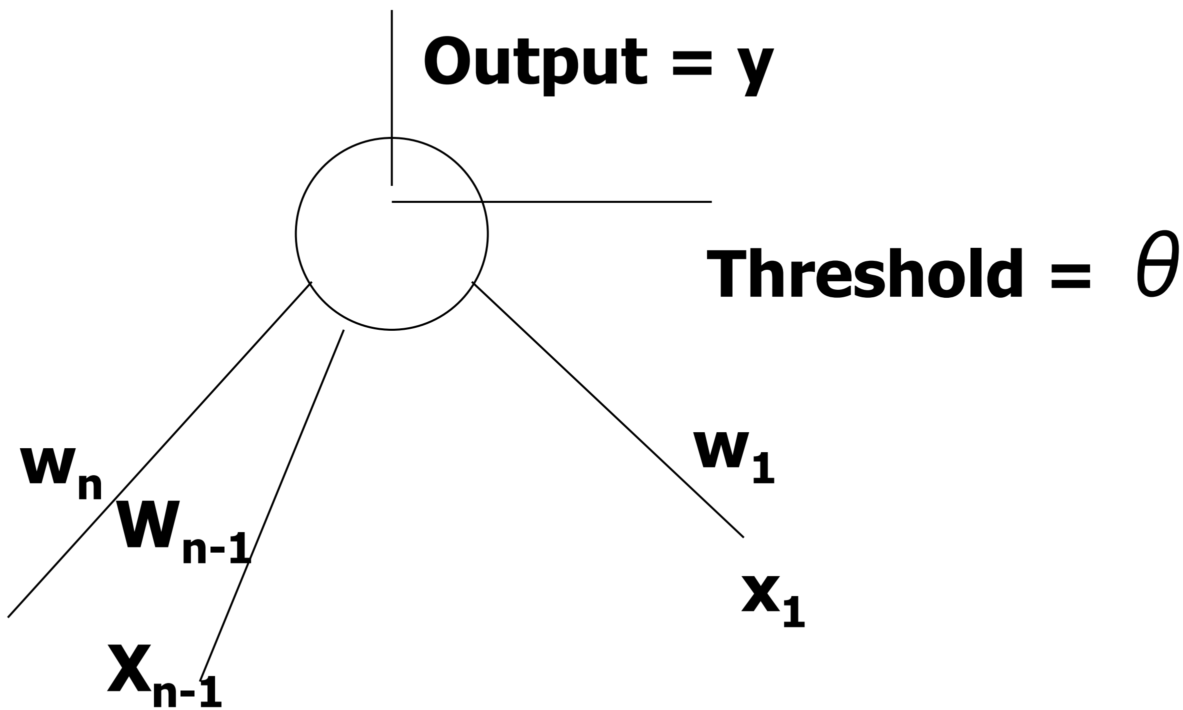
SMS feature vector: input neurons

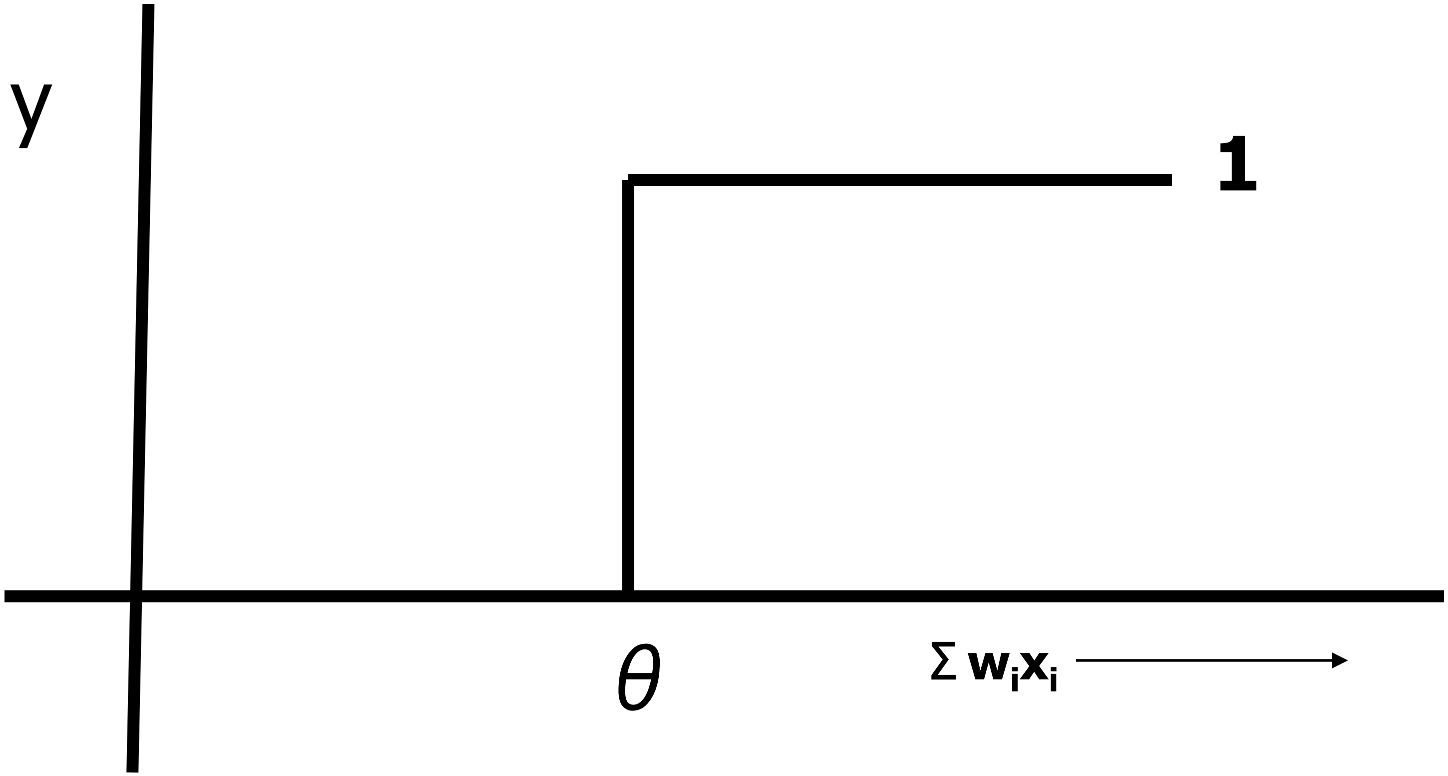
Basic Neural Models

- Precursor to Deep Learning
- Models
  - Perceptron and PTA
  - Feedforward Network and Backpropagation
  - Boltzmann Machine
  - Self Organization and Kohonen's Map
  - Neocognitron

Perceptron

The Perceptron Model





Step function / Threshold function

y = 1 for $\sum w_i x_i >= \theta$

=0 otherwise

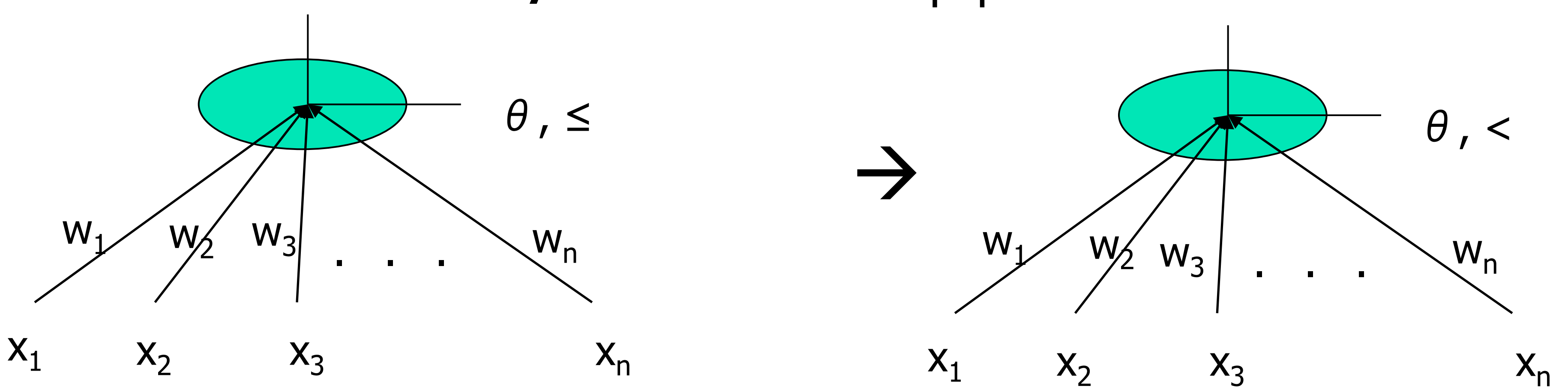
Perceptron Training Algorithm (PTA)

Preprocessing:

1. The computation law is modified to

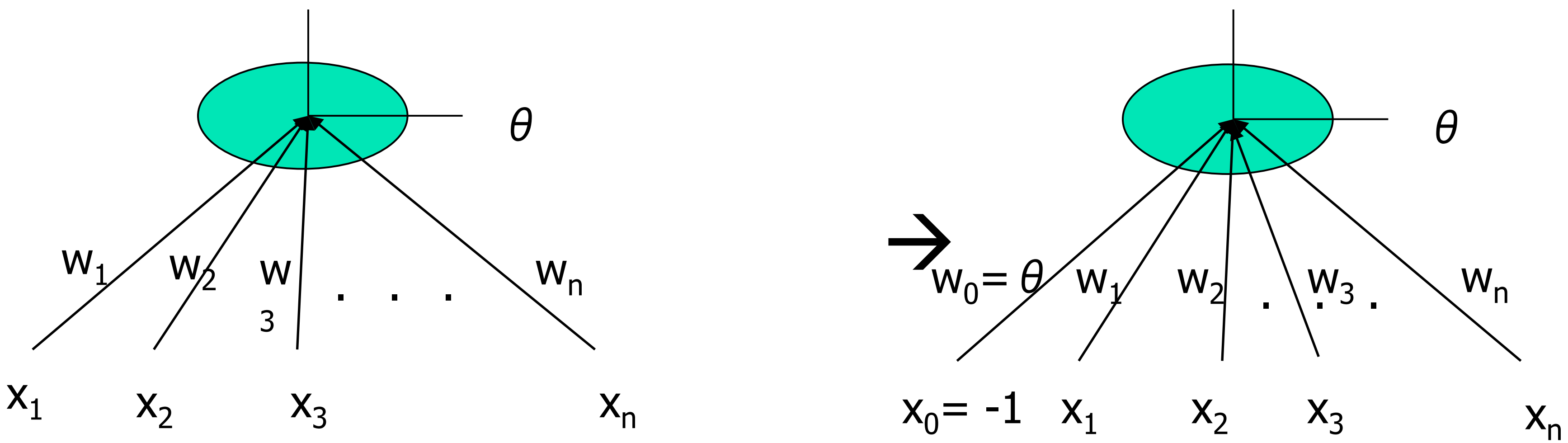
$$y = 1 \text{ if } \sum w_i x_i > \theta$$

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



PTA – preprocessing cont…

2. Absorb θ as a weight



3. Negate all the zero-class examples

Example to demonstrate preprocessing

- **OR perceptron**

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

0-class <0,0>

Augmented x vectors:-

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

0-class <-1,0,0>

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

Example to demonstrate preprocessing cont..

Now the vectors are

	x_0	x_1	x_2
X_1	-1	0	1
X_2	-1	1	0
X_3	-1	1	1
X_4	1	0	0

Perceptron Training Algorithm

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

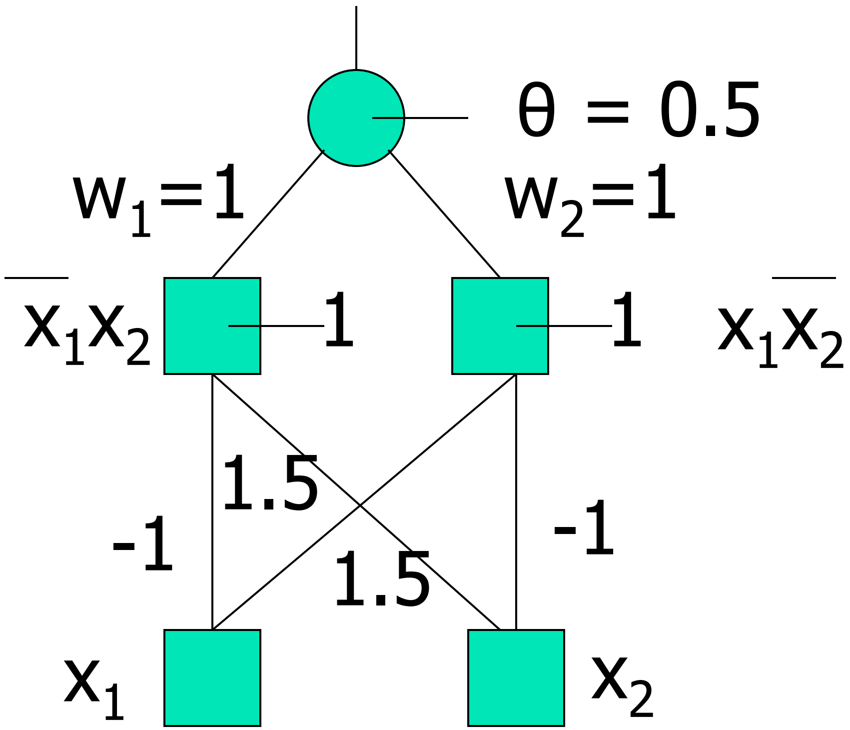
Convergence of PTA

- 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.

Feedforward Network and Backpropagation

Example - XOR



Gradient Descent Technique

- Let E be the error at the output layer

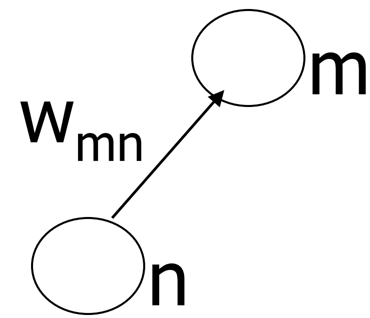
$$E = \frac{1}{2}\sum_{j=1}^{p}\sum_{i=1}^{n}(t_i - o_i)_j^2$$

- t_i = target output; o_i = observed output

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

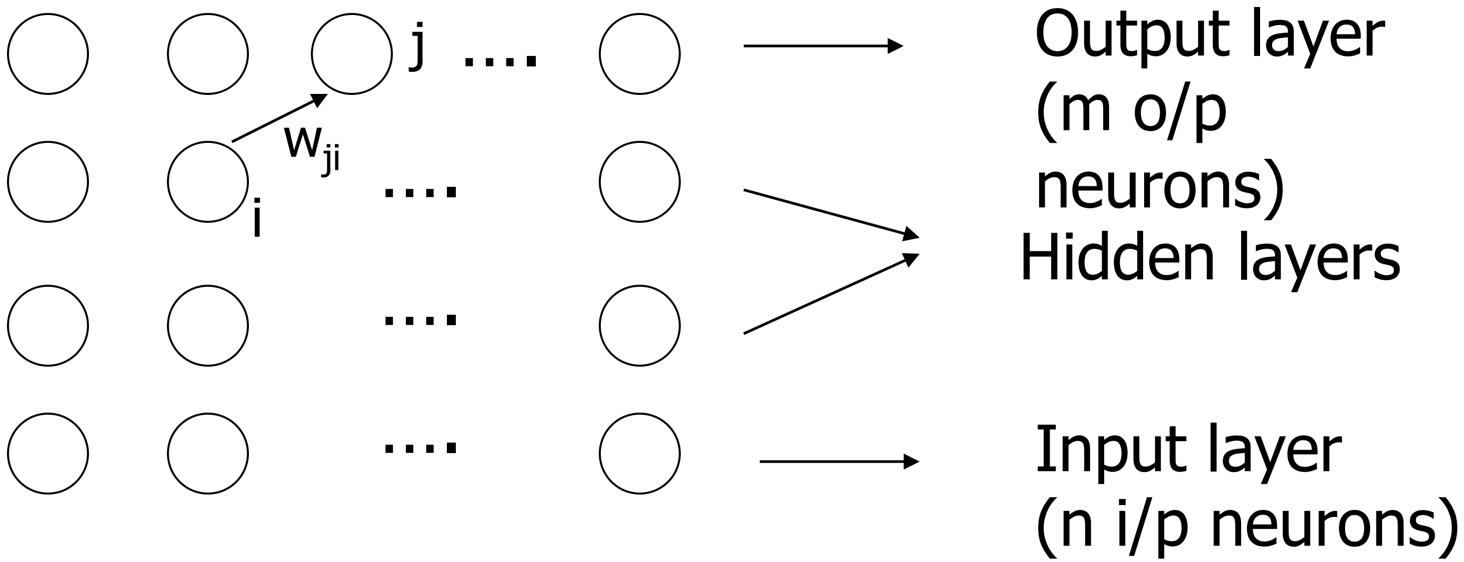
Weights in a FF NN

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



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

Backpropagation algorithm



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

Gradient Descent Equations

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

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

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

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

Backpropagation – for outermost layer

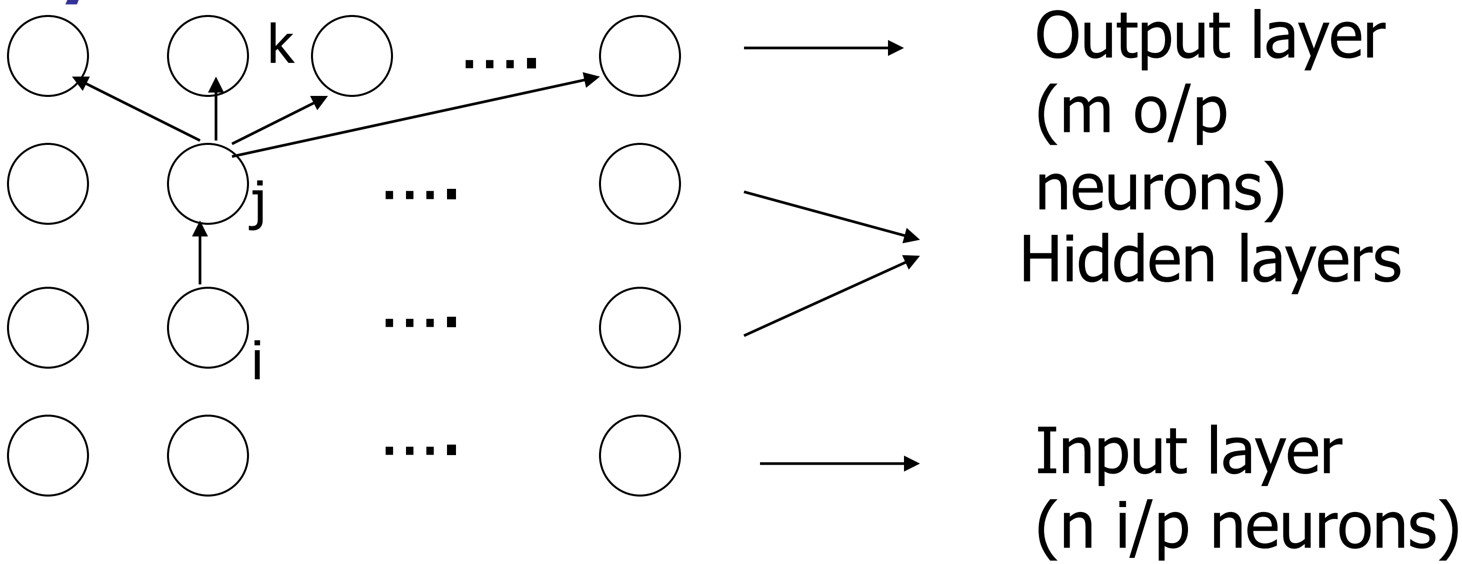
$$\delta j = -\frac{\delta E}{\delta net_j} = -\frac{\delta E}{\delta o_j} \times \frac{\delta o_j}{\delta net_j} (net_j = \text{input at the j}^{th} \text{ layer})$$

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

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

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

Backpropagation for hidden layers



δ_k is propagated backwards to find value of δ_j

Backpropagation – for hidden layers

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

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

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

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

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

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

Cause of Vanishing Gradient problem!

General Backpropagation Rule

- General weight updating rule:

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

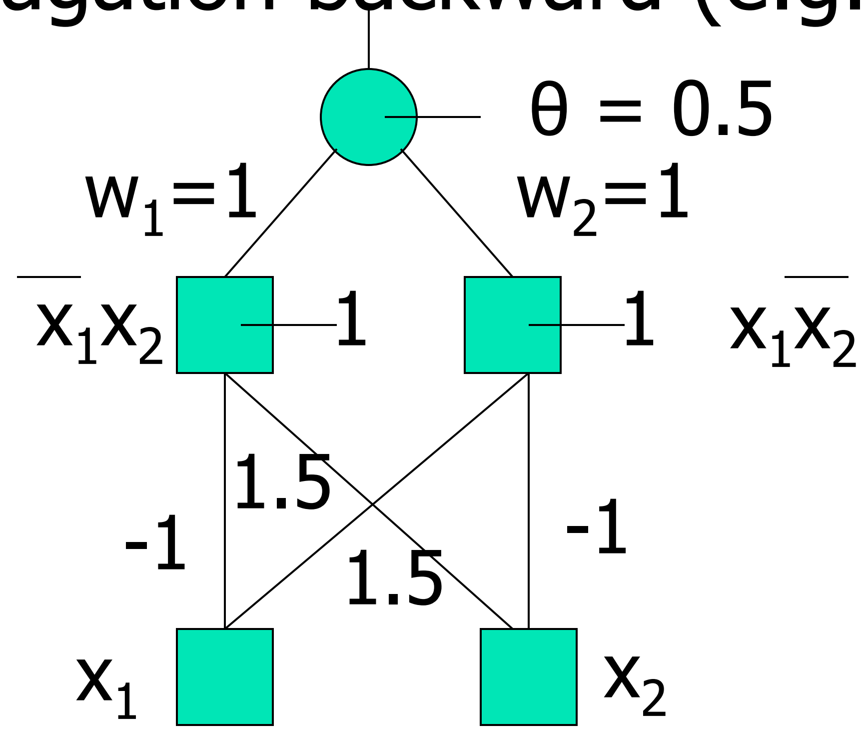
- Where

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

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

How does it work?

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



Recurrent Neural Network

Acknowledgement:

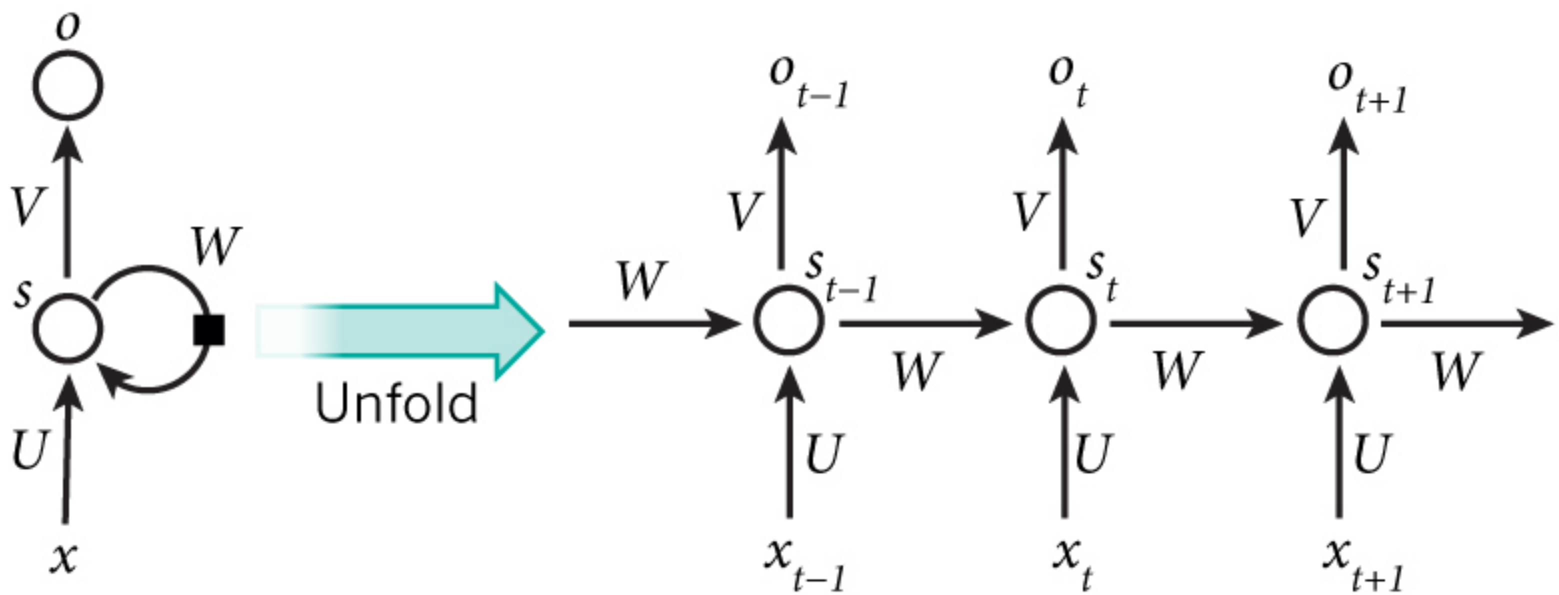
1. http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

By Denny Britz

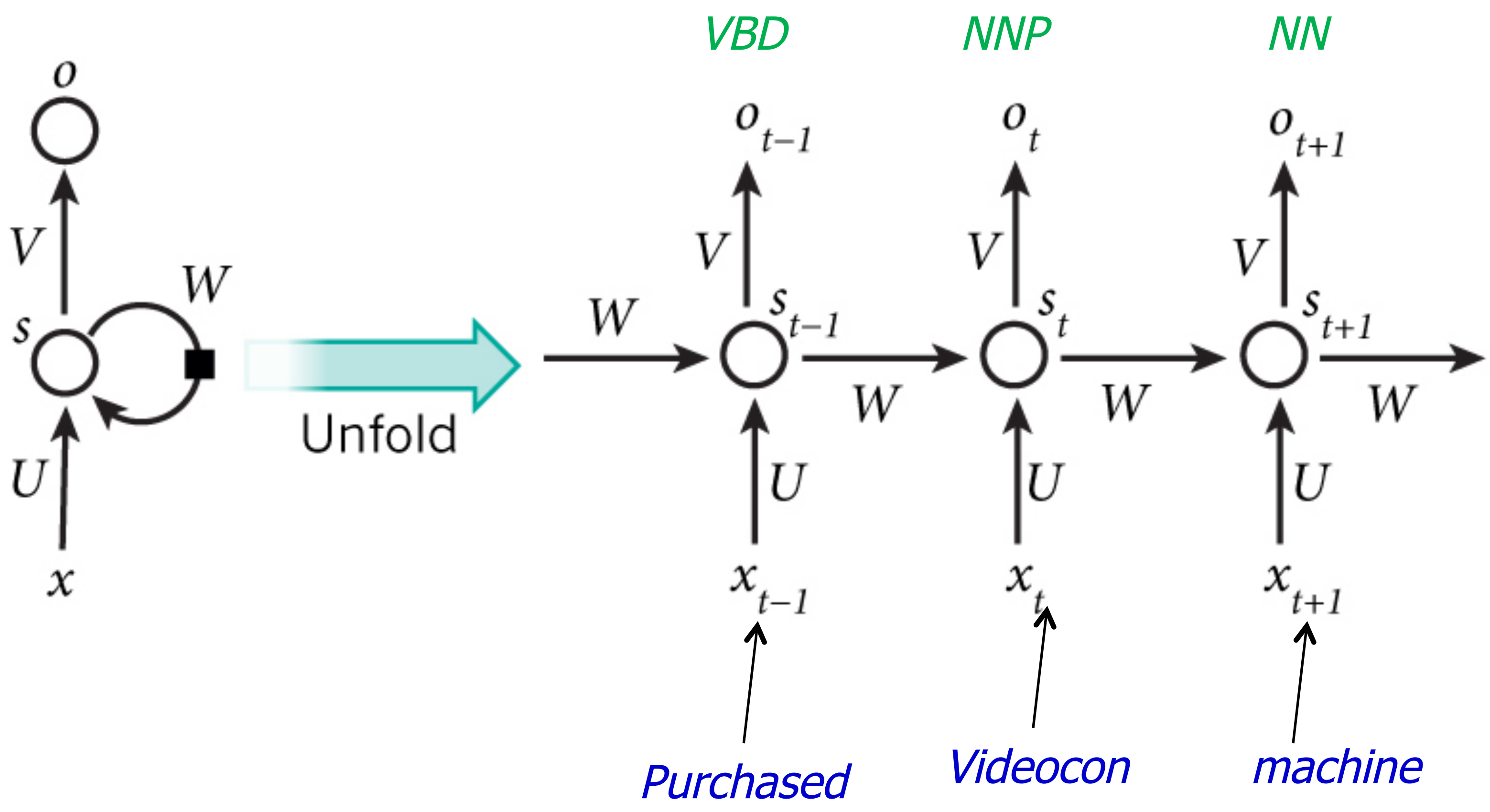
2. Introduction to RNN by Jeffrey Hinton

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

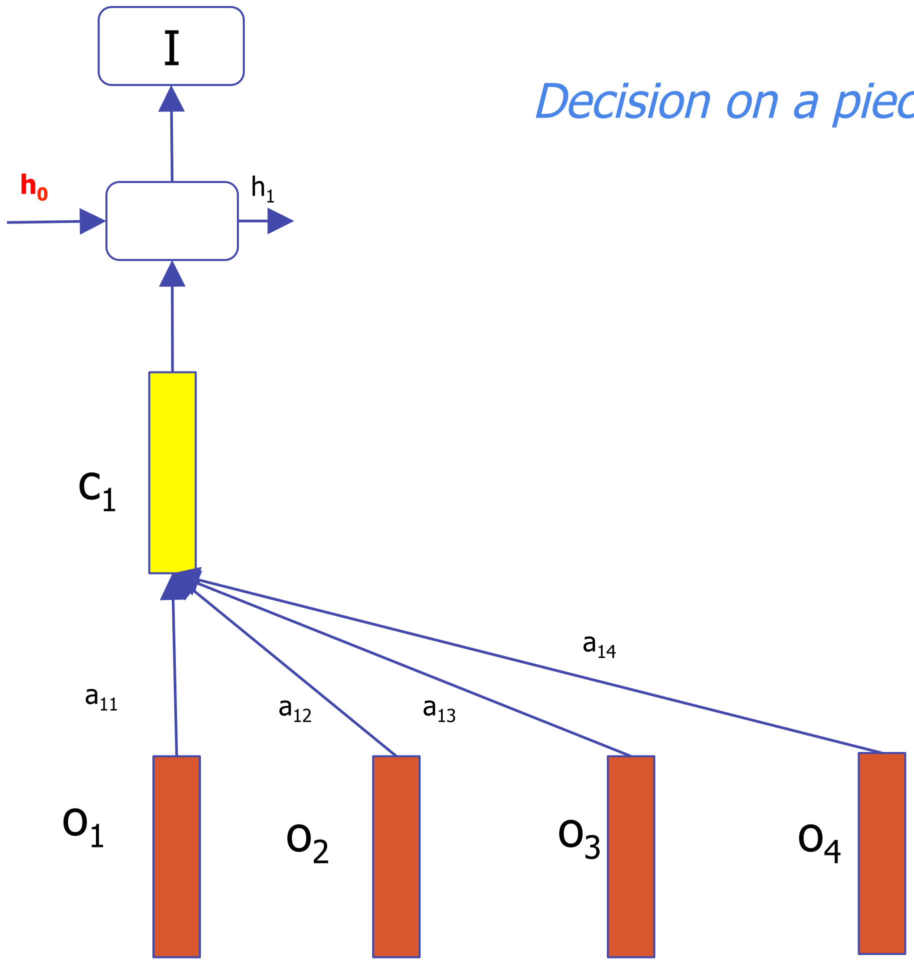
Sequence processing m/c



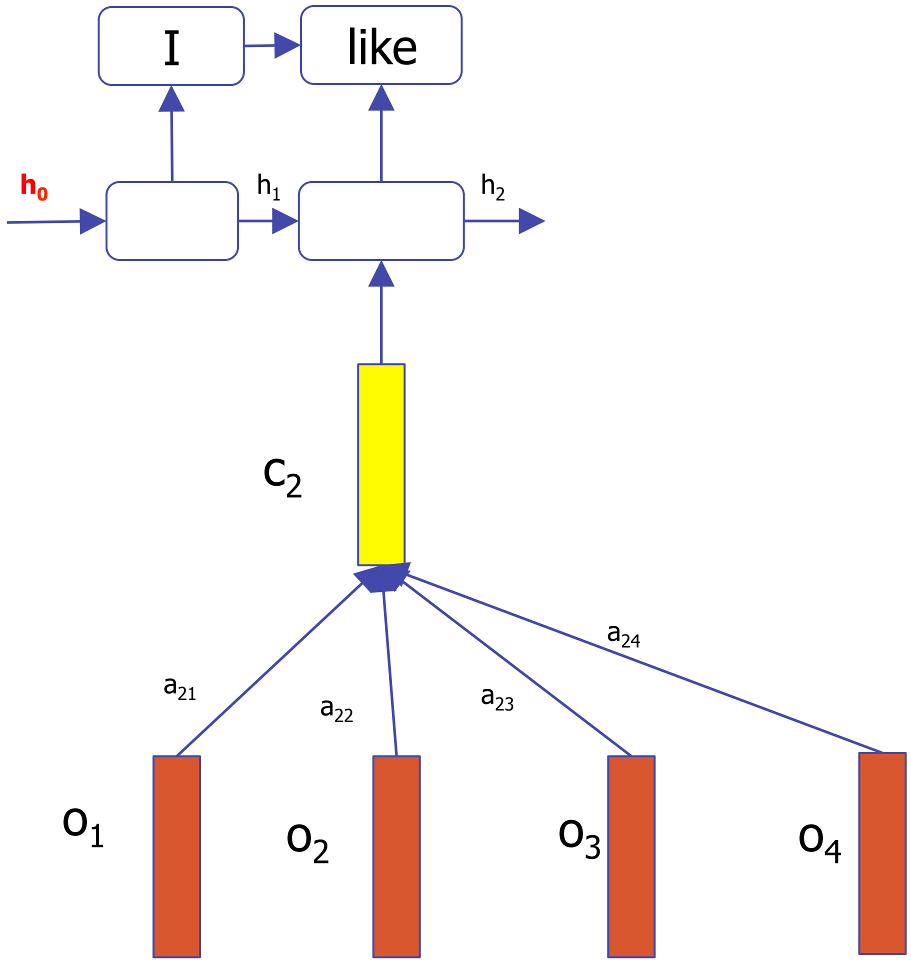
E.g. POS Tagging

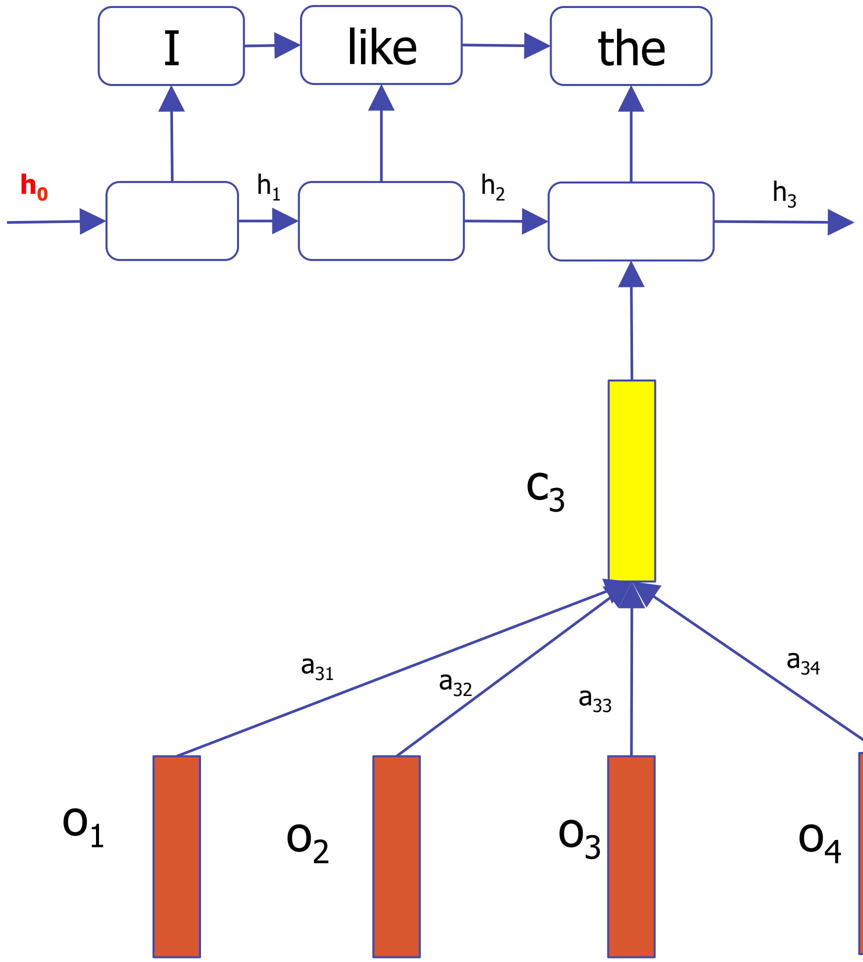


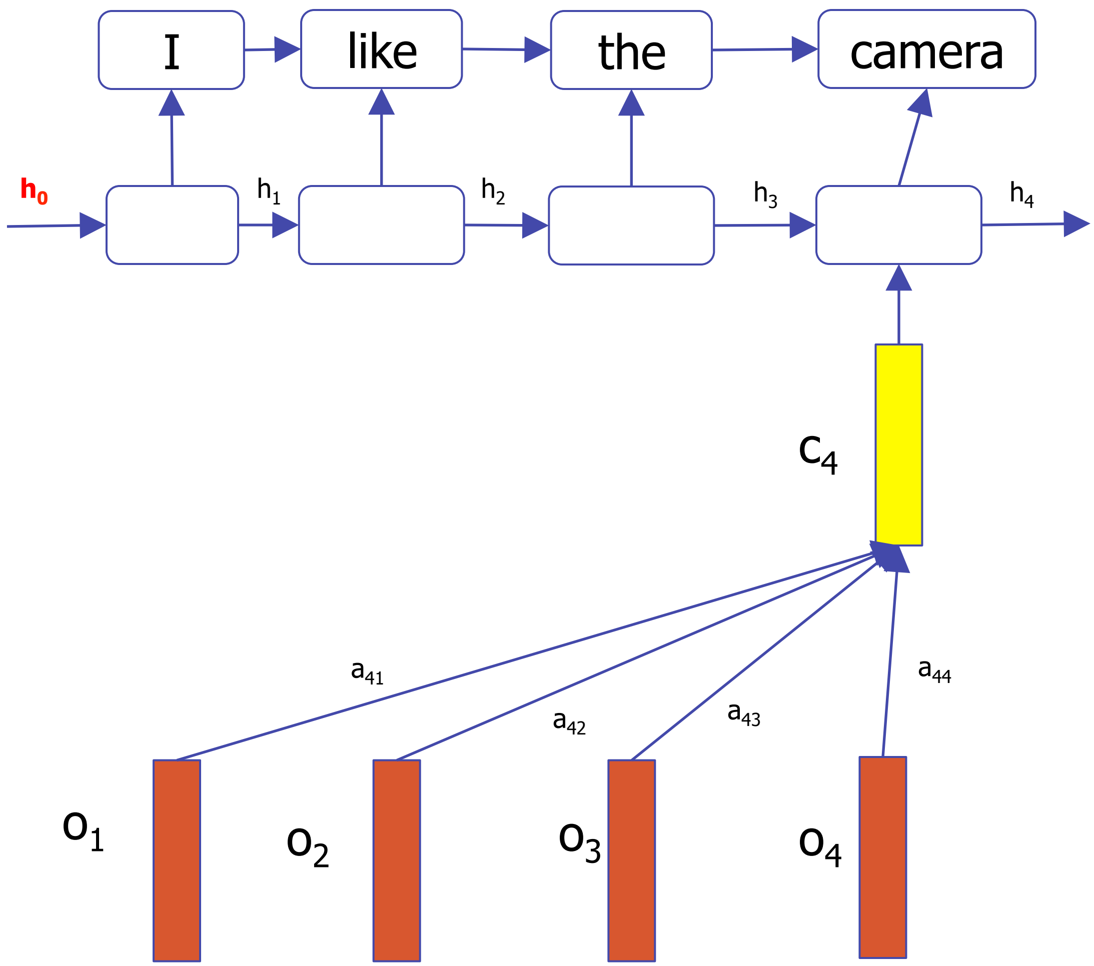
E.g. Sentiment Analysis

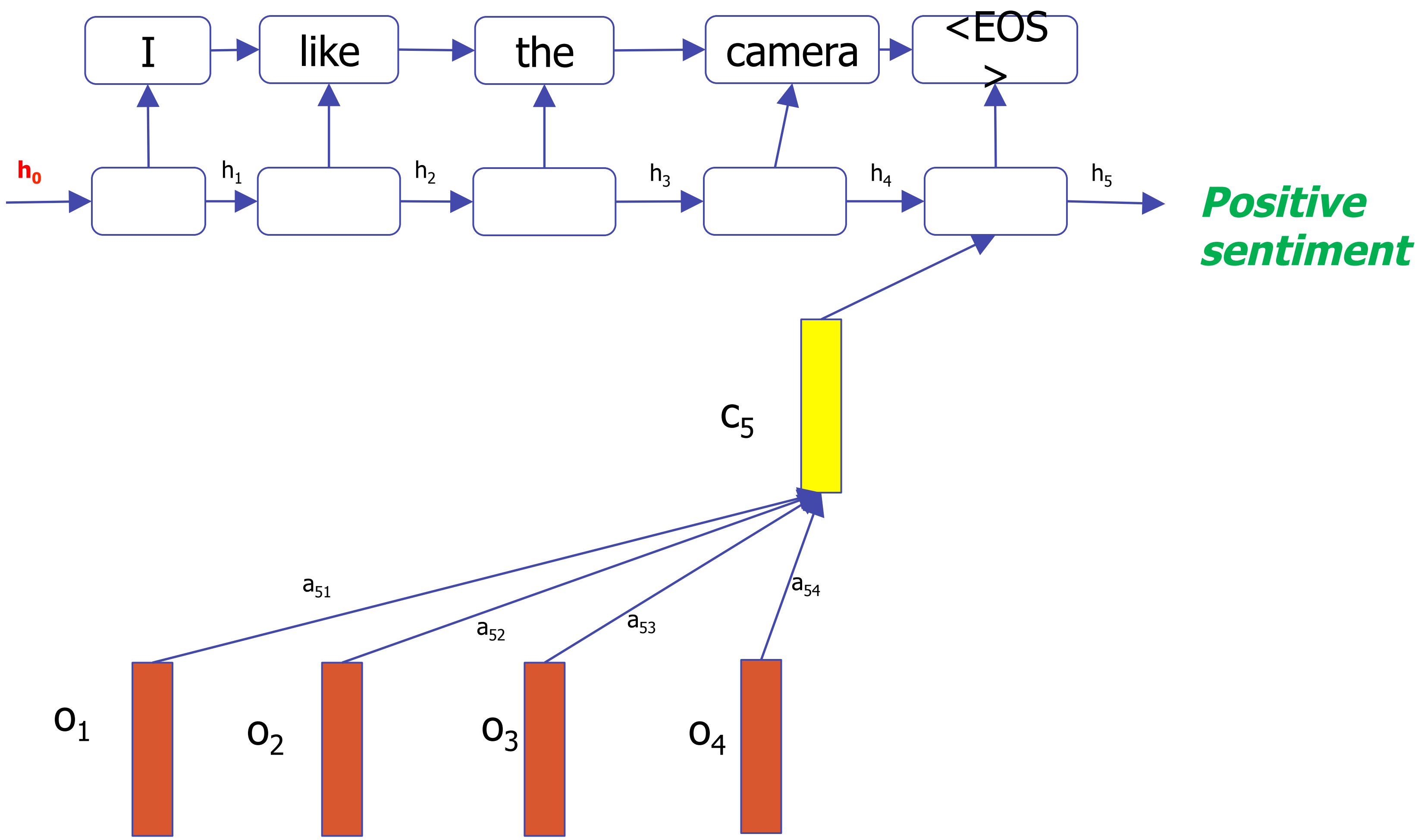


Decision on a piece of text

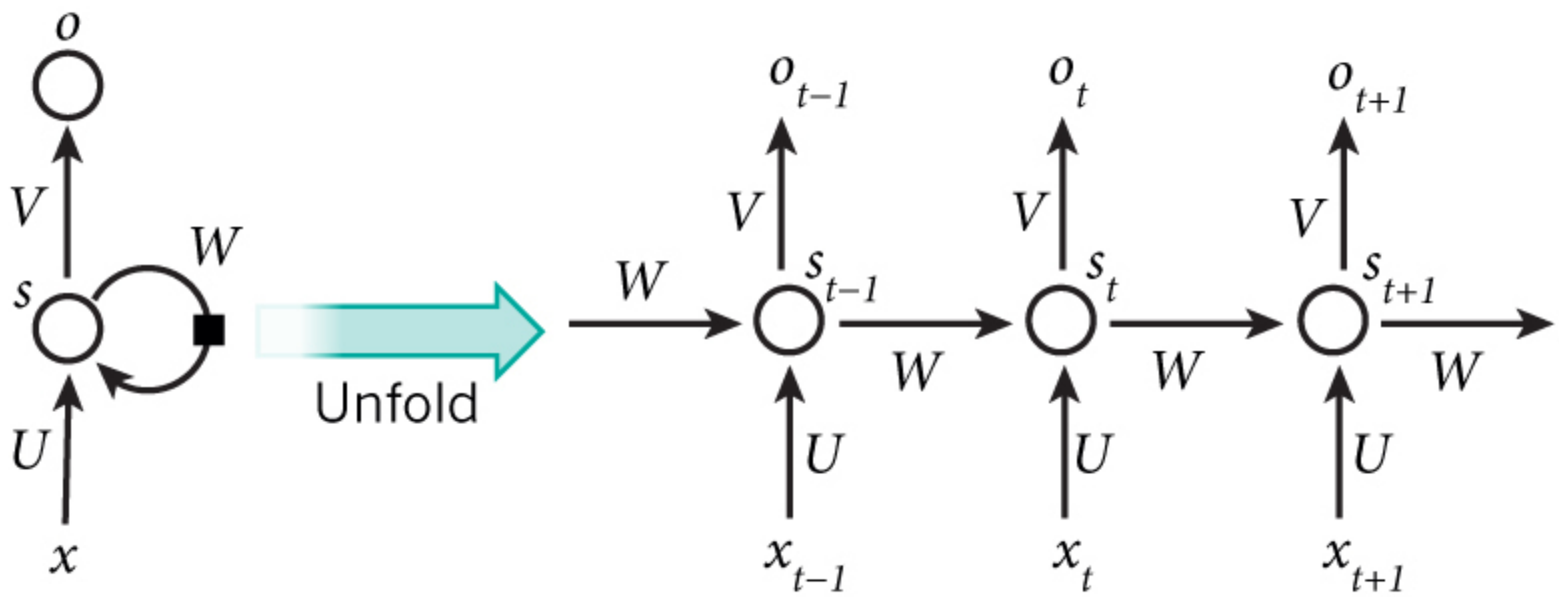

I
like
h0
h1
h2
c2
a21
a22
a23
a24
o1
o2
o3
o4


I
like
the
h_0
h_1
h_2
h_3
c_3
a_{31}
a_{32}
a_{33}
a_{34}
o_1
o_2
o_3
o_4


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

Back to RNN model



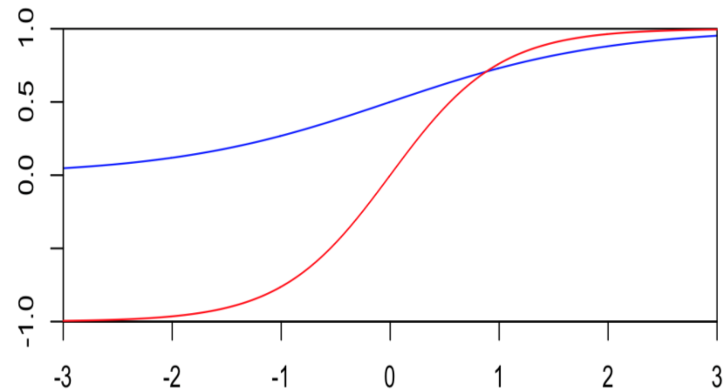
Notation: input and state

- x_t is the input at time step t. For example, could be a one-hot vector corresponding to the second word of a sentence.
- s_t is the hidden state at time step t. It is the "memory" of the network.
- $s_t = f(U.x_t + Ws_{t-1})$ ***U*** **and** ***W*** **matrices are learnt**
- f is a function of the input and the previous state
- Usually *tanh* or *ReLU* (approximated by *softplus*)

Tanh, *ReLU* (rectifier linear unit) and Softplus

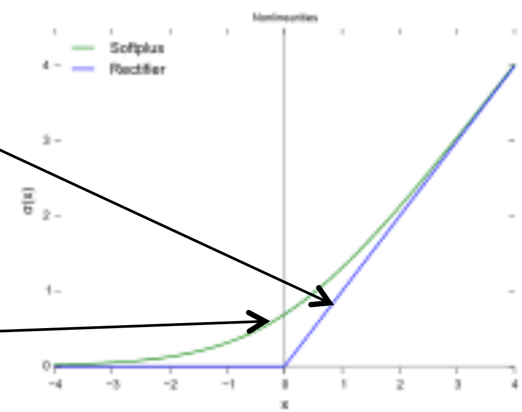
$$\tanh = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

tanh =



$$f(x) = \max(0, x)$$

$$g(x) = \ln(1 + e^{x})$$



Notation: output

- o_t is the output at step t

- For example, if we wanted to predict the next word in a sentence it would be a vector of probabilities across our vocabulary

- $o_t = softmax(V.s_t)$

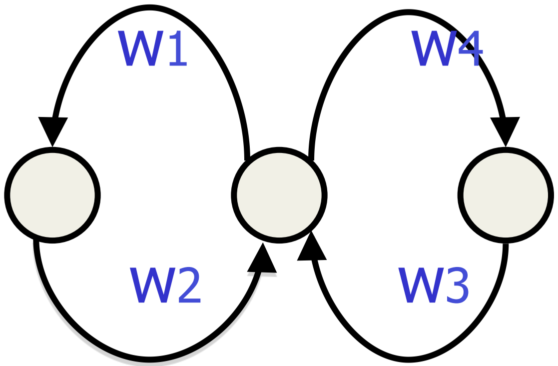
Operation of RNN

- RNN shares the same parameters (*U*, *V*, *W*) across all steps

- Only the input changes

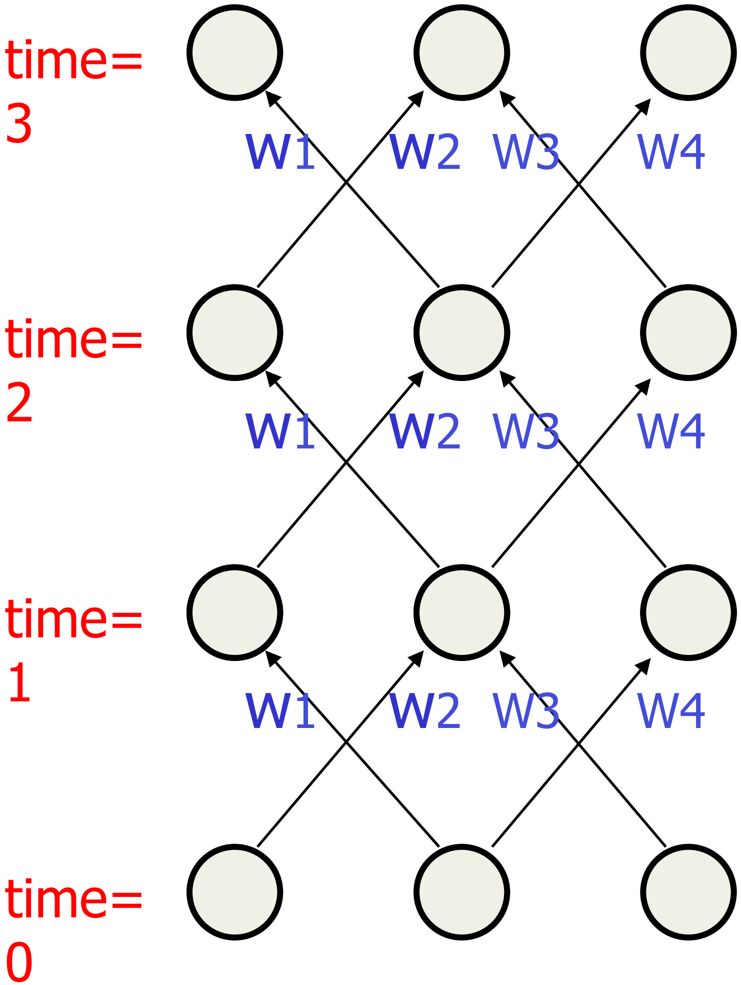
- Sometimes the output at each time step is not needed: e.g., in sentiment analysis
- Main point: the hidden states !!

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.



Reminder: Backpropagation with weight constraints

- **Linear constraints between the weights.**
- **Compute the gradients as usual**
- **Then modify the gradients so that they satisfy the constraints.**
- **So if the weights started off satisfying the constraints, they will continue to satisfy them.**

<u>Example</u>

$$To \;\; constrain: \quad w_1 = w_2$$

$$we \;\; need: \quad \Delta w_1 = \Delta w_2$$

$$compute: \quad \frac{\partial E}{\partial w_1} \quad and \quad \frac{\partial E}{\partial w_2}$$

$$use \quad \frac{\partial E}{\partial w_1} + \frac{\partial E}{\partial w_2} \quad for \; w_1 \; and \; w_2$$

Backpropagation through time (BPTT algorithm)

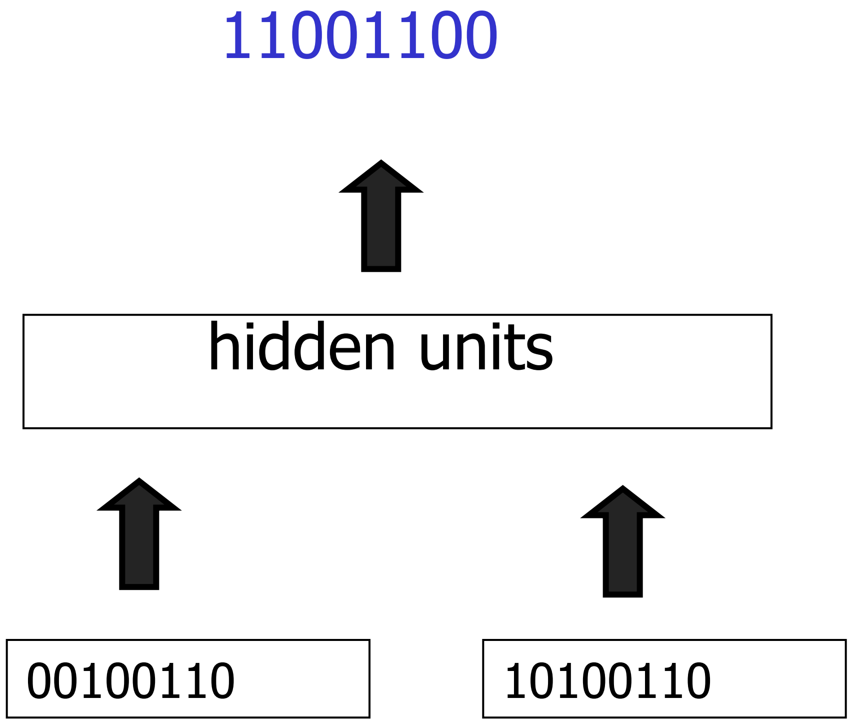
- The forward pass at each time step.
-
- The backward pass computes the error derivatives at each time step.

- After the backward pass we add together the derivatives at all the different times for each weight.

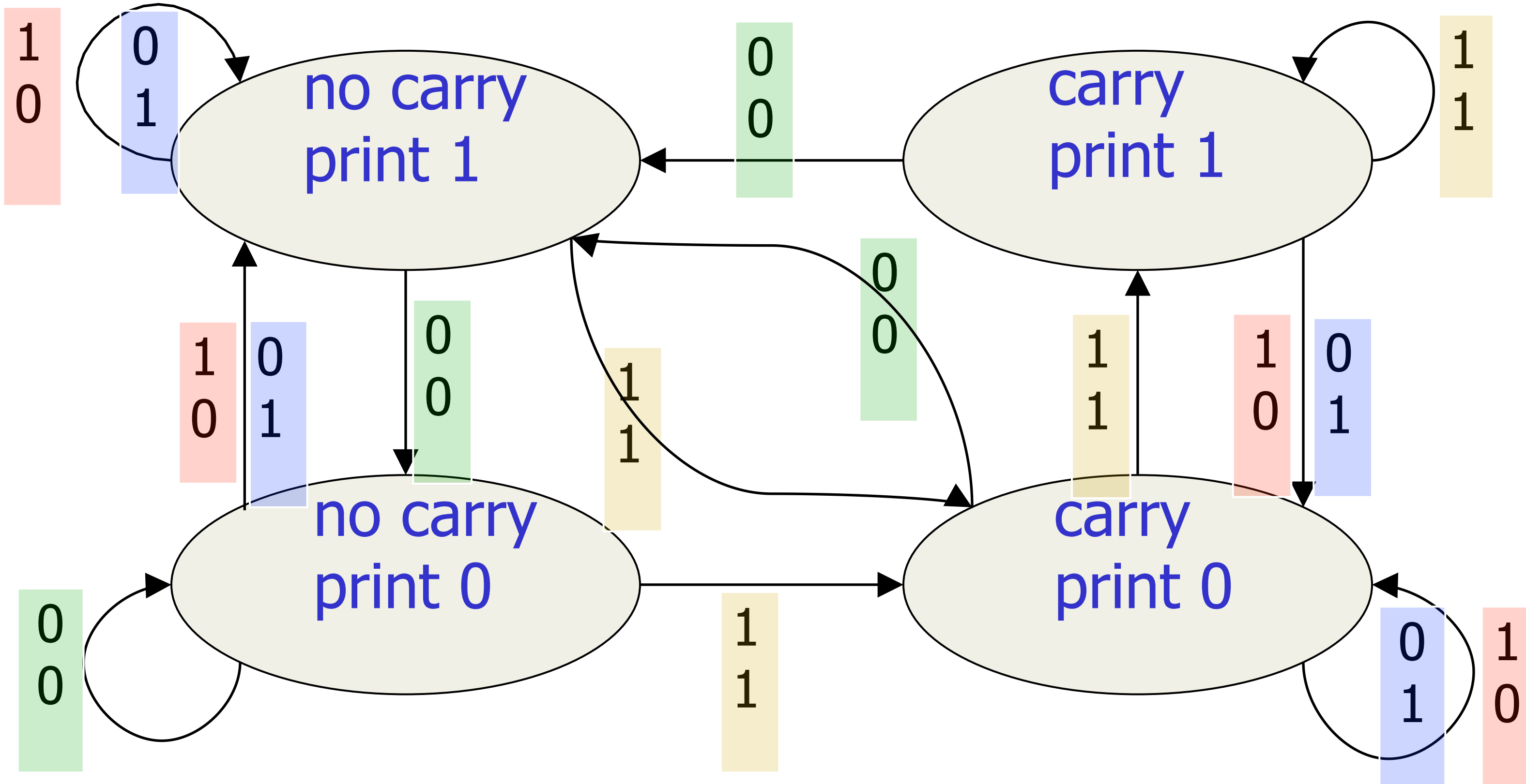
Binary addition using recurrent network (Jeffrey Hinton's lecture)

- Feed forward n/w

- But problem of variable length input



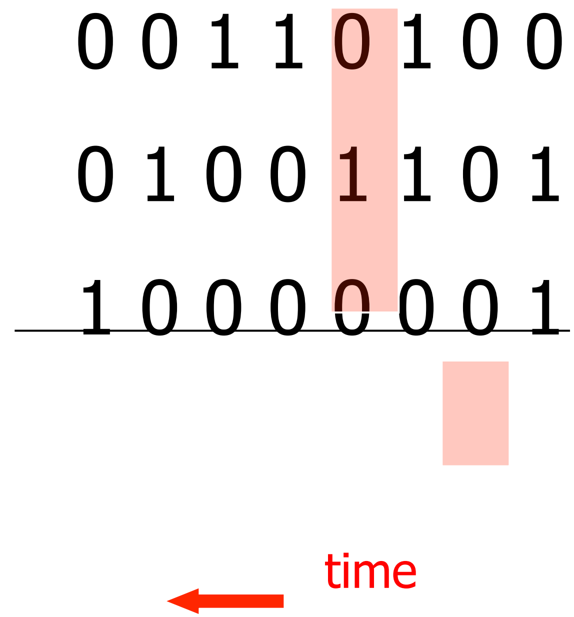
The algorithm for binary addition



This is a finite state automaton. It decides what transition to make by looking at the next column. It prints after making the transition. It moves from right to left over the two input numbers.

A recurrent net for binary addition

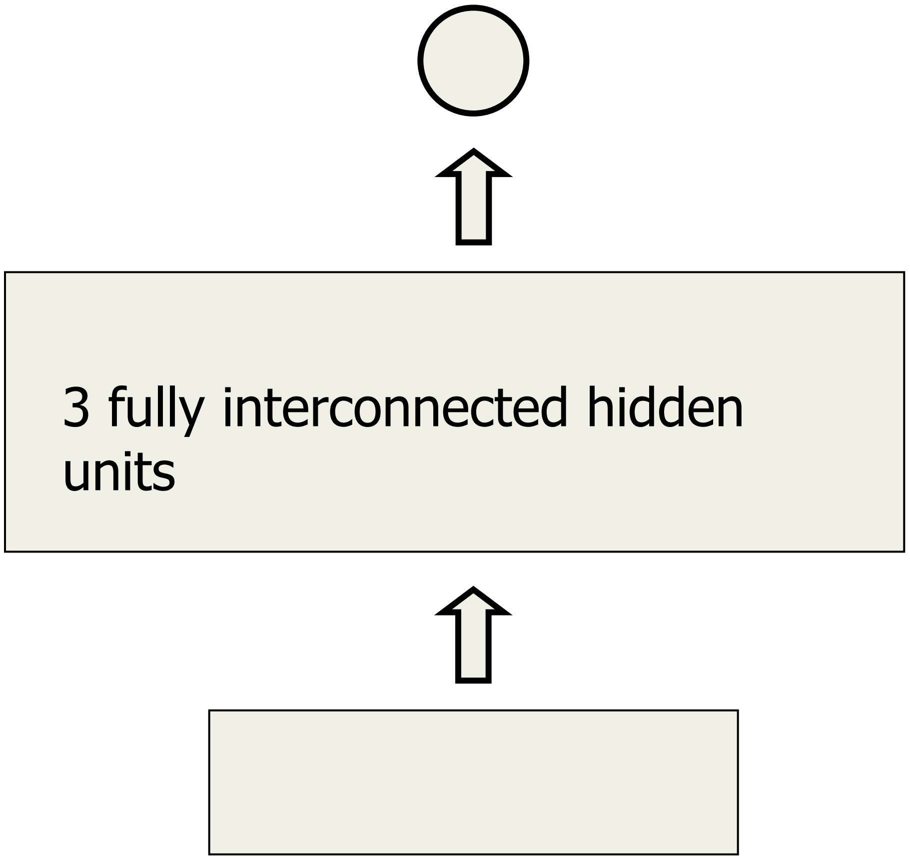
- Two input units and one output unit.
- Given two input digits at each time step.
- The desired output at each time step is the output for the column that was provided as input two time steps ago.
  - It takes one time step to update the hidden units based on the two input digits.
  - It takes another time step for the hidden units to cause the output.



The connectivity of the network

- The input units have feed forward connections

- Allow them to vote for the next hidden activity pattern.

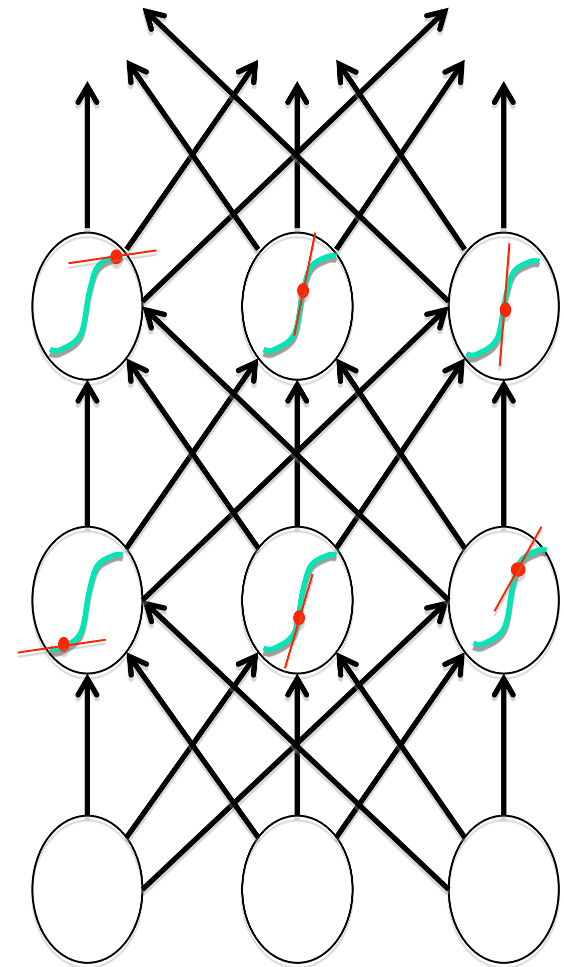


What the network learns

- Learns four distinct patterns of activity for the 3 hidden units.
- Patterns correspond to the nodes in the finite state automaton
- Nodes in FSM are like activity vectors
- The automaton is restricted to be in exactly one state at each time
- The hidden units are restricted to have exactly one vector of activity at each time.

The backward pass is linear

- The backward pass, is completely linear. If you double the error derivatives at the final layer, all the error derivatives will double.
- The forward pass determines the slope of the linear function used for backpropagating through each neuron.



Recall: Backpropagation Rule

- General weight updating rule:

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

- Where

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

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

The problem of exploding or vanishing gradients (1/2)

- If the weights are small, the gradients shrink exponentially

- If the weights are big the gradients grow exponentially.

- Typical feed-forward neural nets can cope with these exponential effects because they only have a few hidden layers.

The problem of exploding or vanishing gradients (2/2)

- In an RNN trained on long sequences (*e.g.* sentence with 20 words) the gradients can easily explode or vanish.
  - We can avoid this by initializing the weights very carefully.
- Even with good initial weights, its very hard to detect that the current target output depends on an input from many time-steps ago.
  - So RNNs have difficulty dealing with long-range dependencies.

Vanishing/Exploding gradient: solution

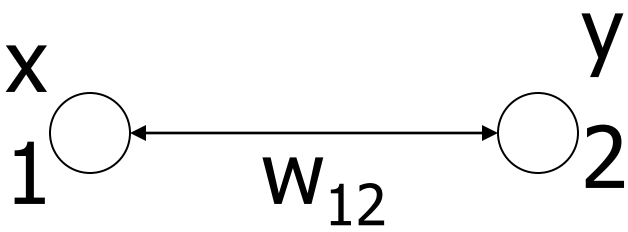
- LSTM
- Error becomes “trapped” in the memory portion of the block
- This is referred to as an "error carousel‟
- Continuously feeds error back to each of the gates until they become trained to cut off the value
- (to be expanded)

Boltzmann Machine

Illustration of the basic idea of Boltzmann Machine

- Illustrative task: To learn the identity function
- The setting is probabilistic, x = 1 or

  x = -1, with uniform probability, *i.e.*,
  - P(x=1) = 0.5, P(x=-1) = 0.5
- For, x=1, y=1 with P=0.9
- For, x=-1, y=-1 with P=0.9



x	y
1	1
-1	-1

Illustration of the basic idea of Boltzmann Machine (contd.)

- Let α = output neuron states

  β = input neuron states

  $P_{\alpha|\beta}$ = observed probability distribution

  $Q_{\alpha|\beta}$ = desired probability distribution

  Q_{β} = probability distribution on input states β

Illustration of the basic idea of Boltzmann Machine (contd.)

Precursor to Softmax Layer

- The divergence D is given as:

$D = \sum_{\alpha}\sum_{\beta} Q_{\alpha|\beta} Q_{\beta} \ln Q_{\alpha|\beta} / P_{\alpha|\beta}$

called KL divergence formula

$D = \sum_{\alpha}\sum_{\beta} Q_{\alpha|\beta} Q_{\beta} \ln Q_{\alpha|\beta} / P_{\alpha|\beta}$

$>= \sum_{\alpha}\sum_{\beta} Q_{\alpha|\beta} Q_{\beta} (1 - P_{\alpha|\beta} / Q_{\alpha|\beta})$

$>= \sum_{\alpha}\sum_{\beta} Q_{\alpha|\beta} Q_{\beta} - \sum_{\alpha}\sum_{\beta} P_{\alpha|\beta} Q_{\beta}$

$>= \sum_{\alpha}\sum_{\beta} Q_{\alpha\beta} - \sum_{\alpha}\sum_{\beta} P_{\alpha\beta}$

{$Q_{\alpha\beta}$ and $P_{\alpha\beta}$ are joint distributions}

$>= 1 – 1 = 0$

Gradient descent for finding the weight change rule

$P(S_\alpha) \propto \exp(-E(S_\alpha)/T)$

$P(S_\alpha) = (\exp(-E(S_\alpha)/T)) / (\Sigma_{\beta\ \epsilon\ all\ states} \exp(-E(S_\beta)/T)$

$\ln(P(S_\alpha)) = (-E(S_\alpha)/T) - \ln Z$

$D = \sum_\alpha \sum_\beta Q_{\alpha|\beta} Q_\beta \ln (Q_{\alpha|\beta} / P_{\alpha|\beta})$

$\Delta w_{ij} = \eta\ (\delta D/\delta w_{ij})$; gradient descent

Calculating gradient: 1/2

$$\delta D / \delta w_{ij} = \delta/\delta w_{ij} [\sum_{\alpha} \sum_{\beta} Q_{\alpha|\beta} Q_{\beta} \ln (Q_{\alpha|\beta} / P_{\alpha|\beta})]$$

$$= \delta/\delta w_{ij} [\sum_{\alpha} \sum_{\beta} Q_{\alpha|\beta} Q_{\beta} \ln Q_{\alpha|\beta}$$

$$- \sum_{\alpha} \sum_{\beta} Q_{\alpha|\beta} Q_{\beta} \ln P_{\alpha|\beta}]$$

Constant With respect To w_{ij}

$$\delta(\ln P_{\alpha|\beta}) / \delta w_{ij} = \delta/\delta w_{ij} [-E(S_{\alpha})/T - \ln Z]$$

$$Z = \sum_{\beta} exp(-E(S_{\beta}))/T$$

Calculating gradient: 2/2

$\delta [-E(S_\alpha)/T] / \delta w_{ij} = (-1/T)\, \delta/\delta w_{ij} [- \sum_i \sum_{j>i} w_{ij} s_i s_j]$

$= (-1/T)[-s_i s_j|_\alpha]$

$= (1/T)[s_i s_j|_\alpha]$

$\delta (\ln Z)/\delta w_{ij} = (1/Z)(\delta Z/\delta w_{ij})$

$Z = \sum_\beta exp(-E(S_\beta)/T)$

$\delta Z/\delta w_{ij} = \sum_\beta [exp(-E(S_\beta)/T)(\delta(-E(S_\beta/T)/\delta w_{ij})]$

$= (1/T) \sum_\beta exp(-E(S_\beta)/T).s_i s_j|_\beta$

Final formula for Δw_{ij}

$$\Delta w_{ij} = [1/T/\ [s_i s_j|_{\alpha} - (1/Z) \sum_{\beta} exp(-E(S_{\beta})/T).s_i s_j|_{\beta}$$

$$= [1/T][s_i s_j|_{\alpha} - \sum_{\beta} P(S_{\beta}).s_i s_j|_{\beta}]$$

Expectation of i^{th} and j^{th}
Neurons being on together

Issue of Hidden Neurons

- Boltzmann machines
  - can come with hidden neurons
  - are equivalent to a Markov Random field
  - with hidden neurons are like a Hidden Markov Machines
- Training a Boltzmann machine is equivalent to running the Expectation Maximization Algorithm

Use of Boltzmann machine

- Computer Vision
  - Understanding scene involves what is called “Relaxation Search” which gradually minimizes a cost function with progressive relaxation on constraints
- Boltzmann machine has been found to be slow in the training
  - Boltzmann training is NP-hard.

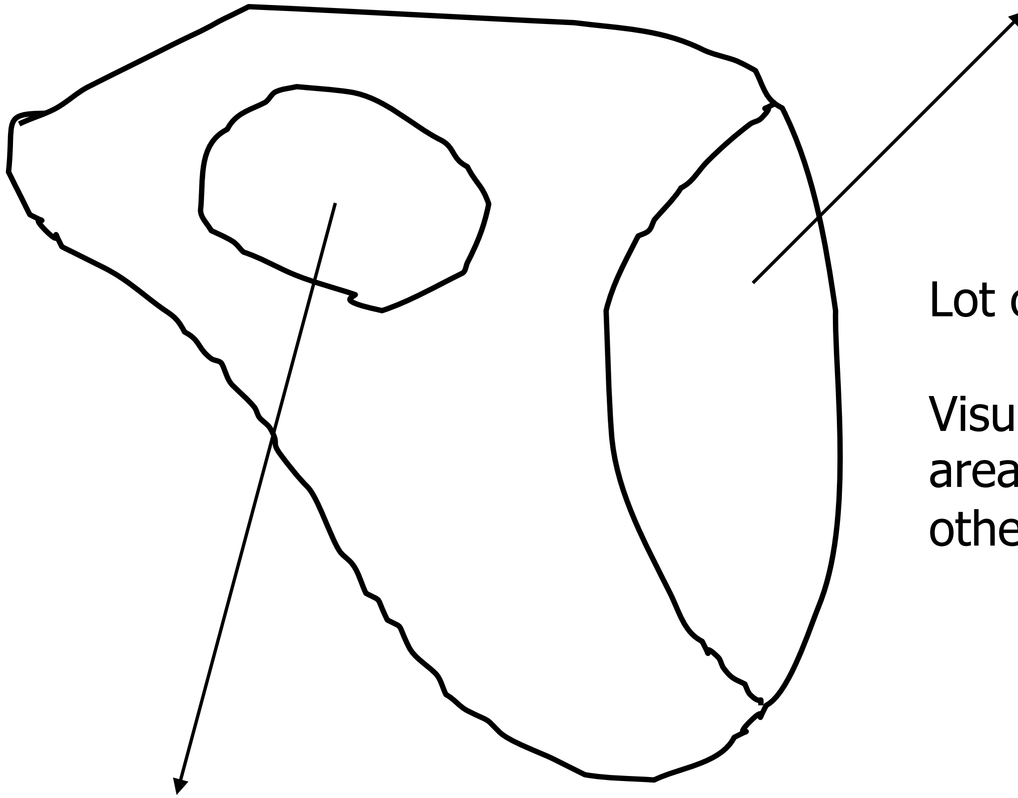
Questions

- Does the Boltzmann machine reach the global minimum? What ensures it?
- Why is simulated annealing applied to Boltzmann machine?
  - local minimum → increase T → n/w runs →gradually reduce T → reach global minimum.
- Understand the effect of varying T
  - Higher T → small difference in energy states ignored, convergence to local minimum fast.

Self Organization and Kohonen Net

Mapping of Brain

Back of brain(vision)



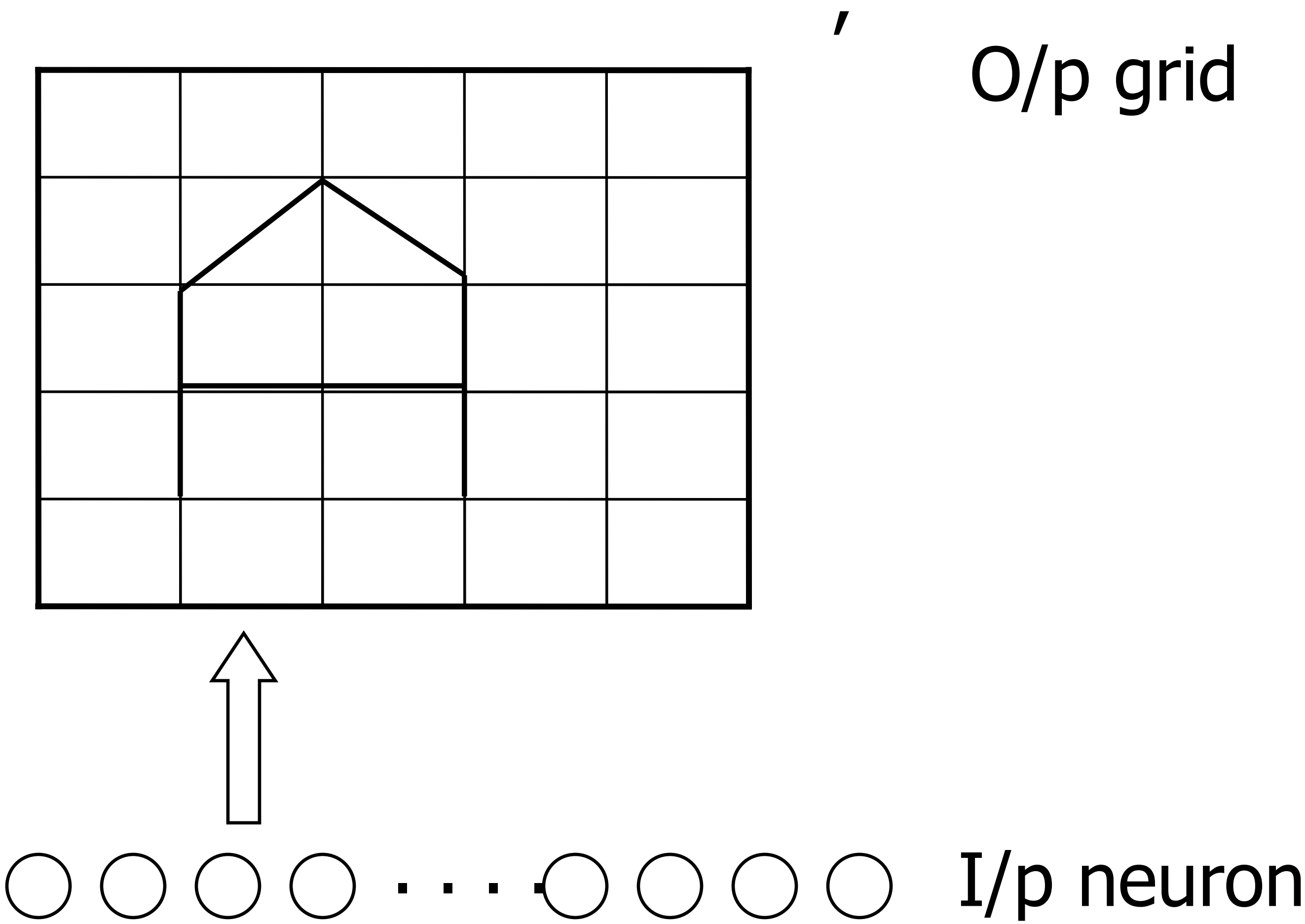
Lot of resilience:

Visual and auditory areas can do each other's job

Side areas
For auditory information processing

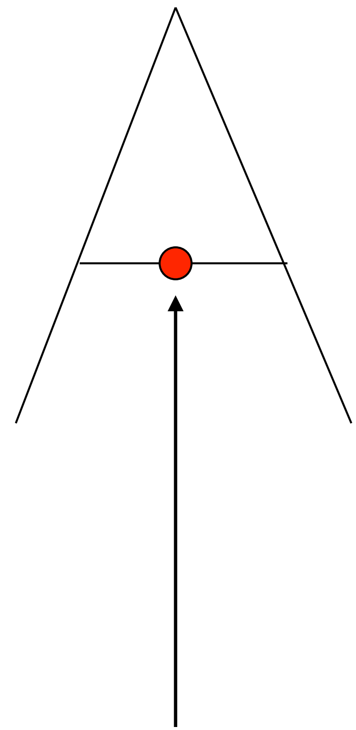
Character Recognition:

A, **A**, *A*, 



Kohonen Net

- Self Organization or Kohonen network fires a group of neurons instead of a single one.
- The group “some how” produces a “picture” of the cluster.
- Fundamentally SOM is competitive learning.
- But weight changes are incorporated on a neighborhood.
- Find the winner neuron, apply weight change for the winner and its “neighbors”.



Winner

Neurons on the contour are the "neighborhood" neurons.

Weight change rule for SOM

$$W^{(n+1)}_{P+\delta(n)} = W^{(n)}_{P+\delta(n)} + \eta^{(n)}_{P+\delta(n)} (I^{(n)} - W^{(n)})_{P+\delta(n)}$$

Neighborhood: function of n

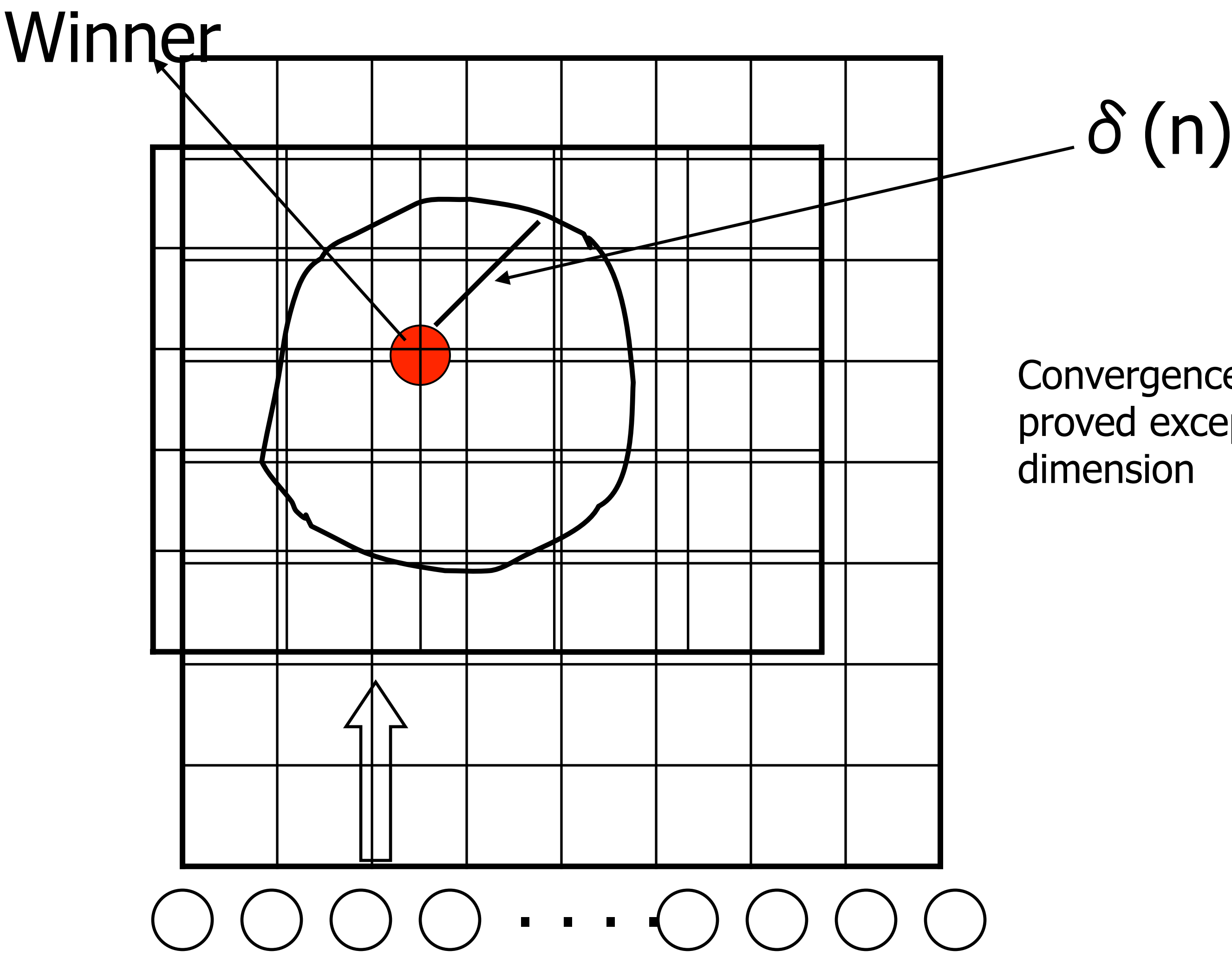
Learning rate: function of n

$\delta(n)$ is a decreasing function of n

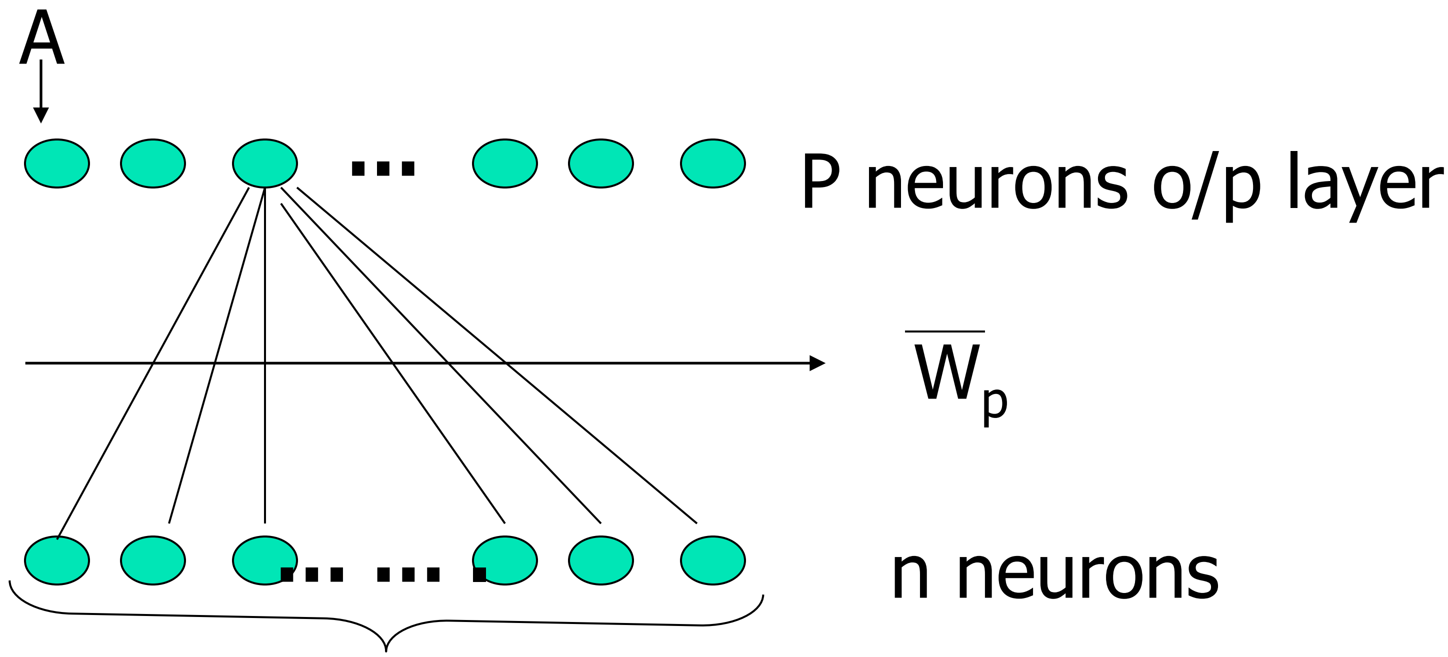
$\eta(n)$ learning rate is also a decreasing function of n

$0 < \eta(n) < \eta(n-1) <= 1$

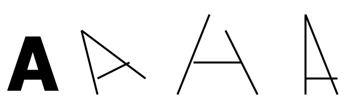
Pictorially



Convergence for kohonen not proved except for uni-dimension



Clusters:

A : 

B : ⋮

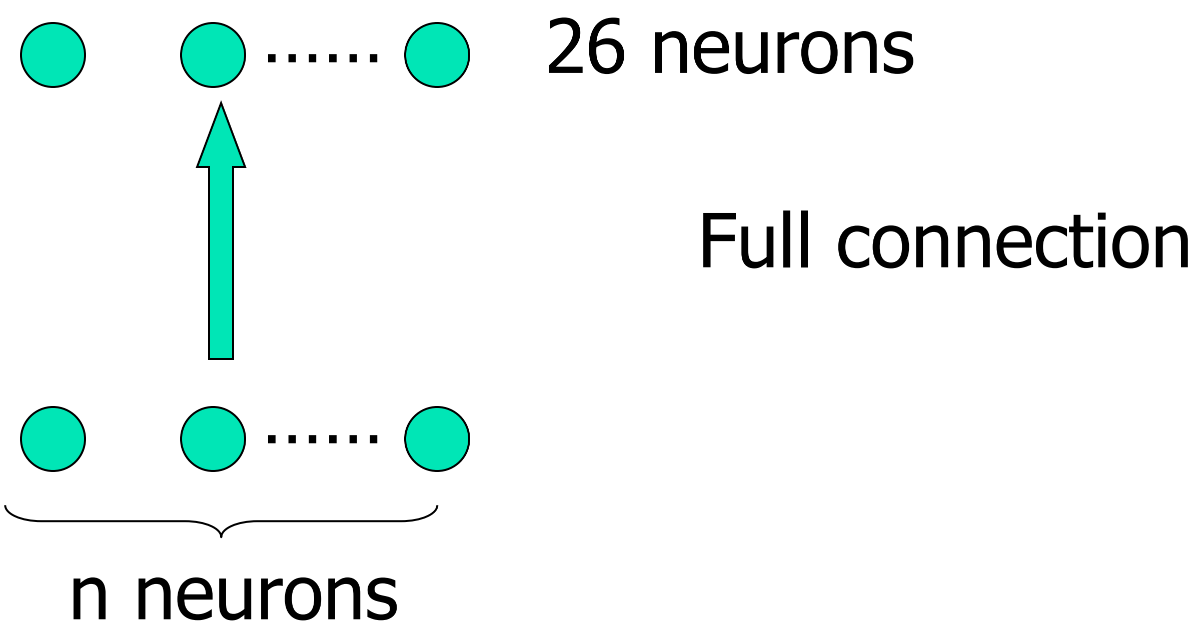
C : ⋮

Clustering Algos

1. Competitive learning
2. K – means clustering
3. Counter Propagation

K – means clustering

K o/p neurons are required from the knowledge of *K* clusters being present.



Steps

1. Initialize the weights randomly.
2. I^k is the vector presented at k^{th} iteration.

3. Find W* such that

$$|w^* - I^k| < |w_j - I^k| \text{ for all } j$$

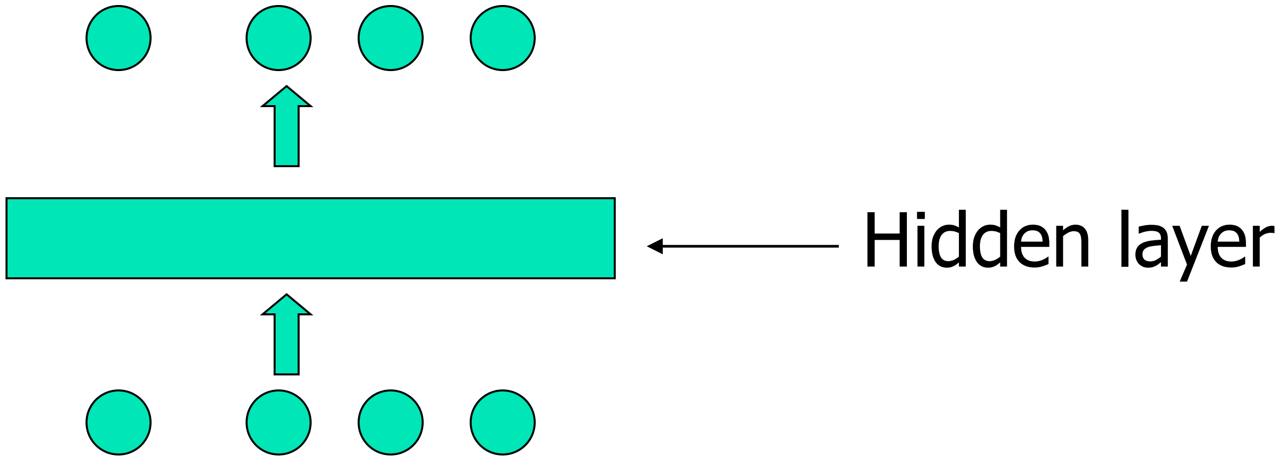
4. make $W^{*(new)} = W^{*(old)} + \eta (I^k - w^*)$.

5 $k \leftarrow k + 1$.

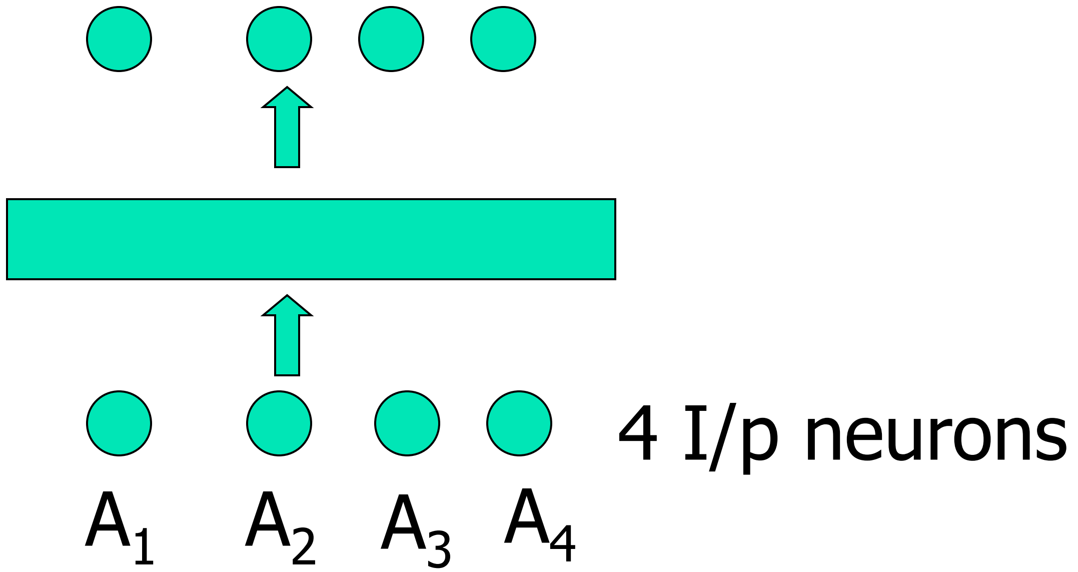
6. Go to 2 until the error is below a threshold.

Two part assignment

Supervised

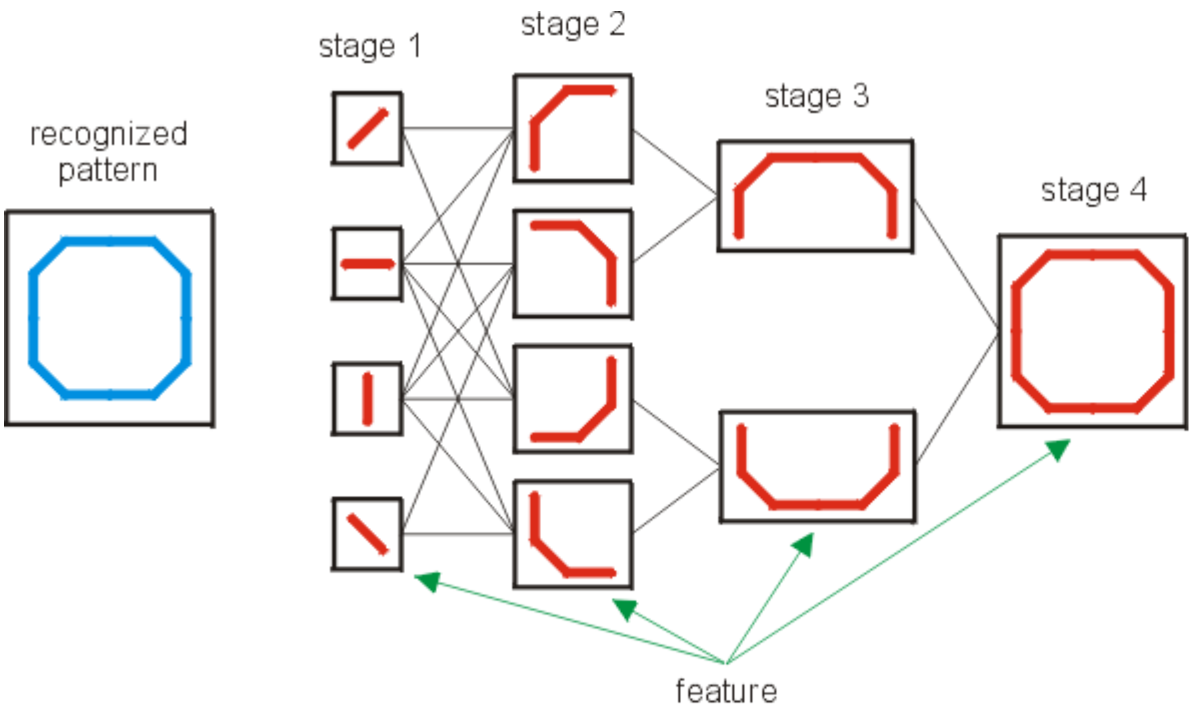


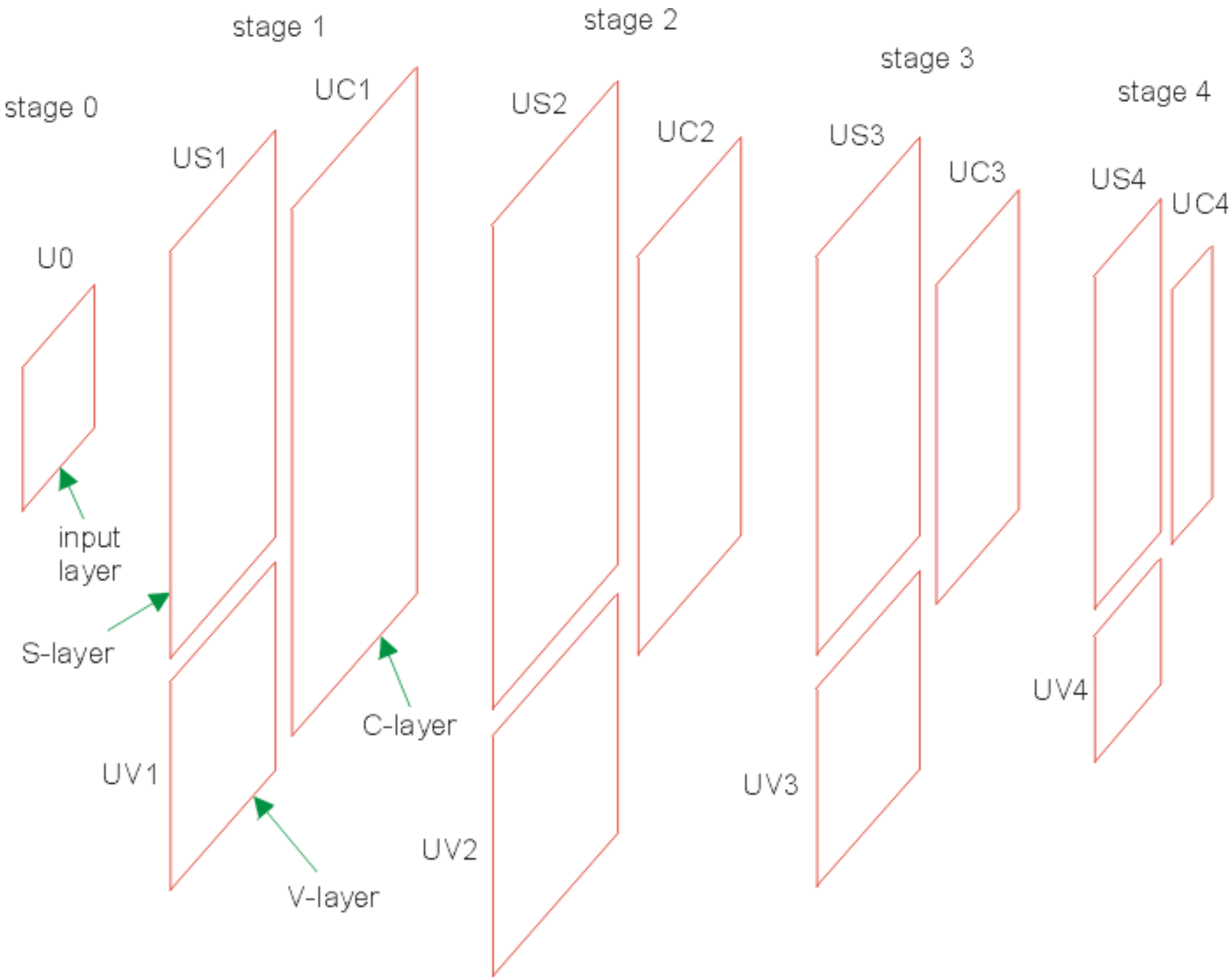
Cluster Discovery By SOM/Kohenen Net



NeoCognitron
(Fukusima et. al., 1980)

Hierarchical feature extraction based




stage 0
stage 1
stage 2
stage 3
stage 4
U0
US1
UC1
US2
UC2
US3
UC3
US4
UC4
input
layer
S-layer
C-layer
UV1
UV2
UV3
UV4
V-layer

S-Layer

- Each **S-layer** in the neocognitron is intended for **extraction of features** from corresponding stage of hierarchy.
- Particular S-layers are formed by distinct number of S-planes. Number of these S-planes depends on the number of extracted features.

V-Layer

- Each **V-layer** in the neocognitron is intended for **obtaining of informations about average activity** in previous C-layer (or input layer).
- Particular V-layers are always formed by only one V-plane.

C-Layer

- Each **C-layer** in the neocognitron is intended for **ensuring of tolerance of shifts of features** extracted in previous S-layer.
- Particular C-layers are formed by distinct number of C-planes. Their number depends on the number of features extracted in the previous S-layer.

Hands on for afternoon

- Implement the binary adder on RNN
- Create a FF-BP POS tagger