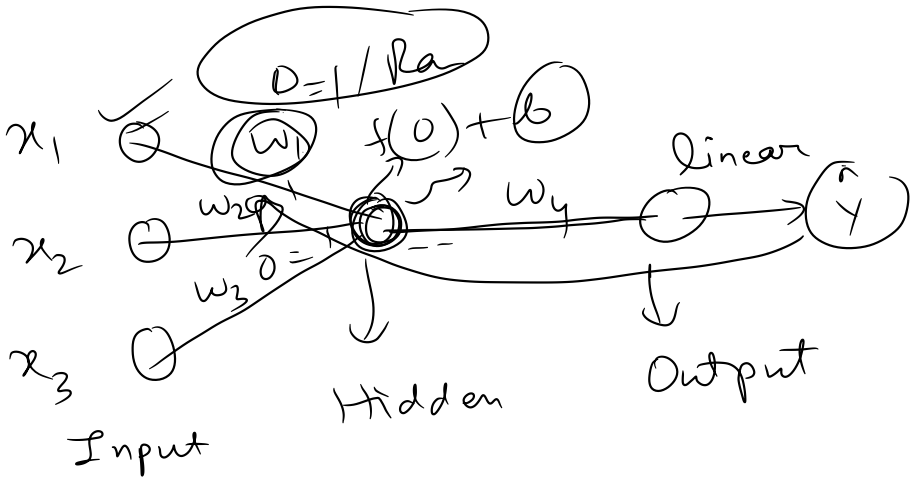
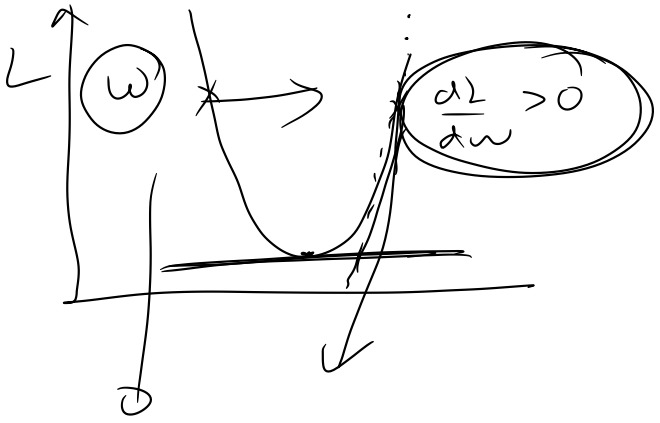


$\frac{\partial L}{\partial x} = 0$

$0 = (w_1 x_1 + w_2 x_2 + w_3 x_4) + b$

$\hat{y} = w_4 \times f(0)$

$L = \frac{(y-\hat{y})^2}{2}$



$w_n = w_n - \eta \frac{\partial L}{\partial w_n} =$

$w_n = w_n - \eta(+ve)$

$w_n = w_n - \eta(-ve)$

$= w_n + \eta(+ve)$

$\frac{\partial L}{\partial \hat{y}} = \frac{-2(y-\hat{y})}{2}$

$= -1(y-\hat{y})$

$dL = -A$

$w_4 = w_4 - \eta \frac{\partial L}{\partial w_4}$

$\frac{\partial L}{\partial w_4} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_4}$

$\hat{y} = w_4 f(0)$

$\frac{\partial \hat{y}}{\partial w_4} = w_4 f'(0)^0$

$\frac{d\hat{y}}{dw_4} = f(0)$

$\frac{\partial L}{\partial w_n}$

$w_4 = 1$

$w_4 = ?$

w_1, w_2, w_3

$L = \frac{(y-\hat{y})^2}{2}$

$w_1 = w_1 - \eta \frac{\partial L}{\partial w_1}$

$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o} \frac{\partial o}{\partial w_1}$

$\frac{\partial \hat{y}}{\partial o} = w_4 f'(o)$

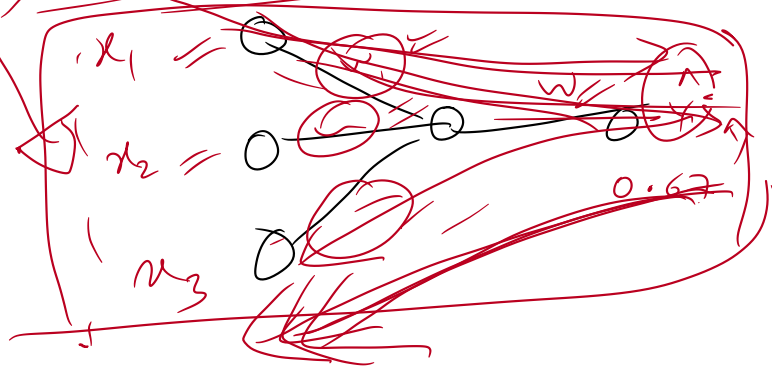
$\hat{y} = w_4 f(o)$

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

$o = w_1 x_1 + w_2 x_2 + w_3 x_3$

$\frac{\partial o}{\partial w_1} = x_1$

	Gender	Ticket Price	Age	Survived
A	M	500	40	
B				0
C				0

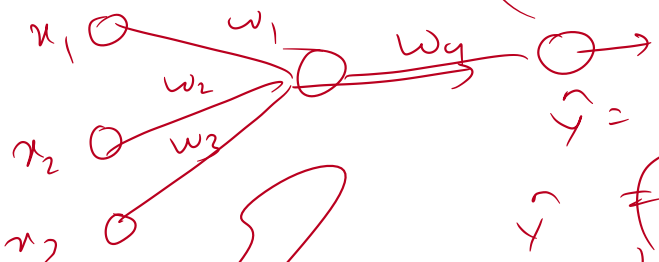


$L = -y \log \hat{y} - (1-y) \log(1-\hat{y})$

1) $L = -1 \log 0.67 - (1-1) \log$

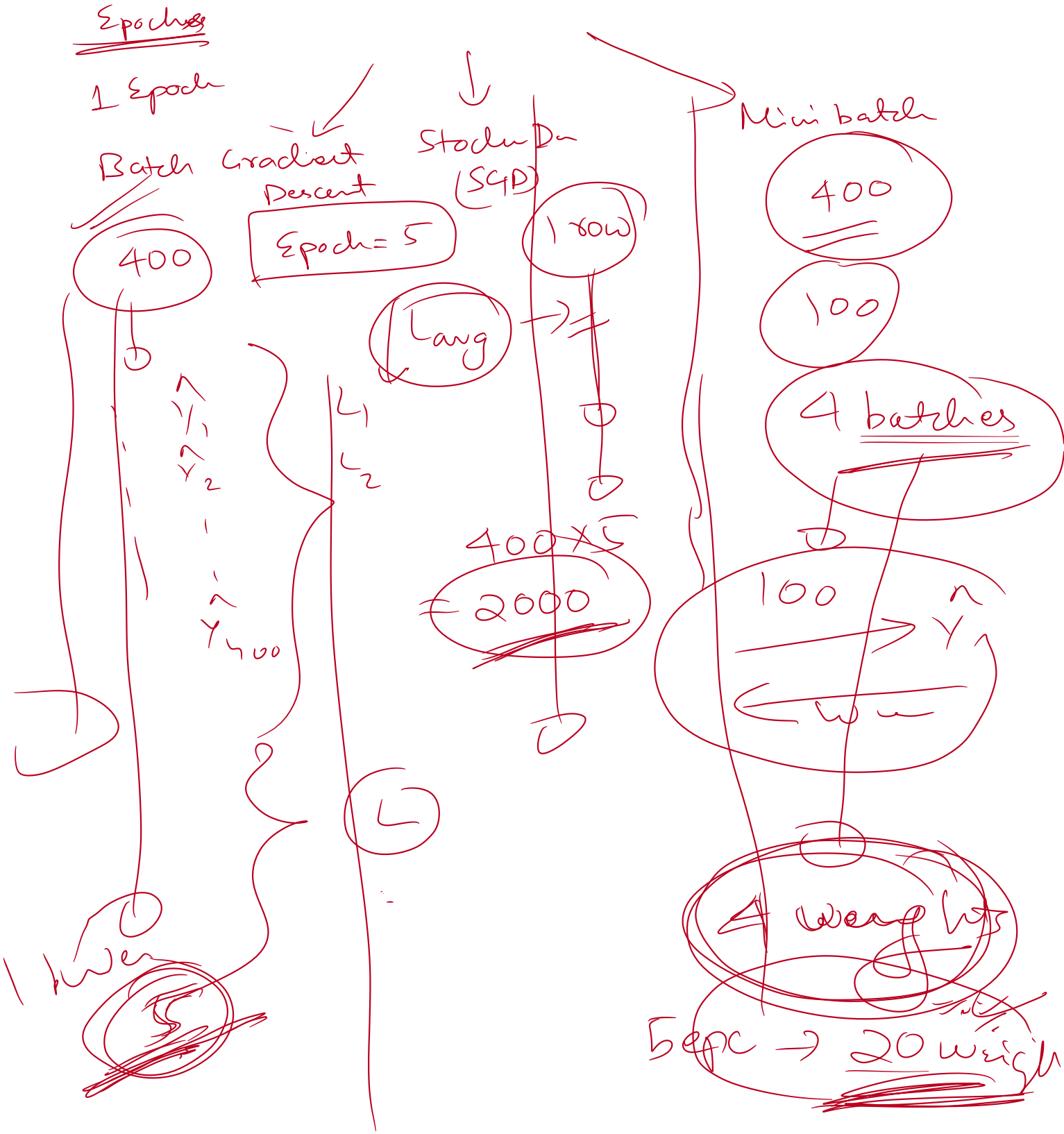
$L = -\log 0.67$

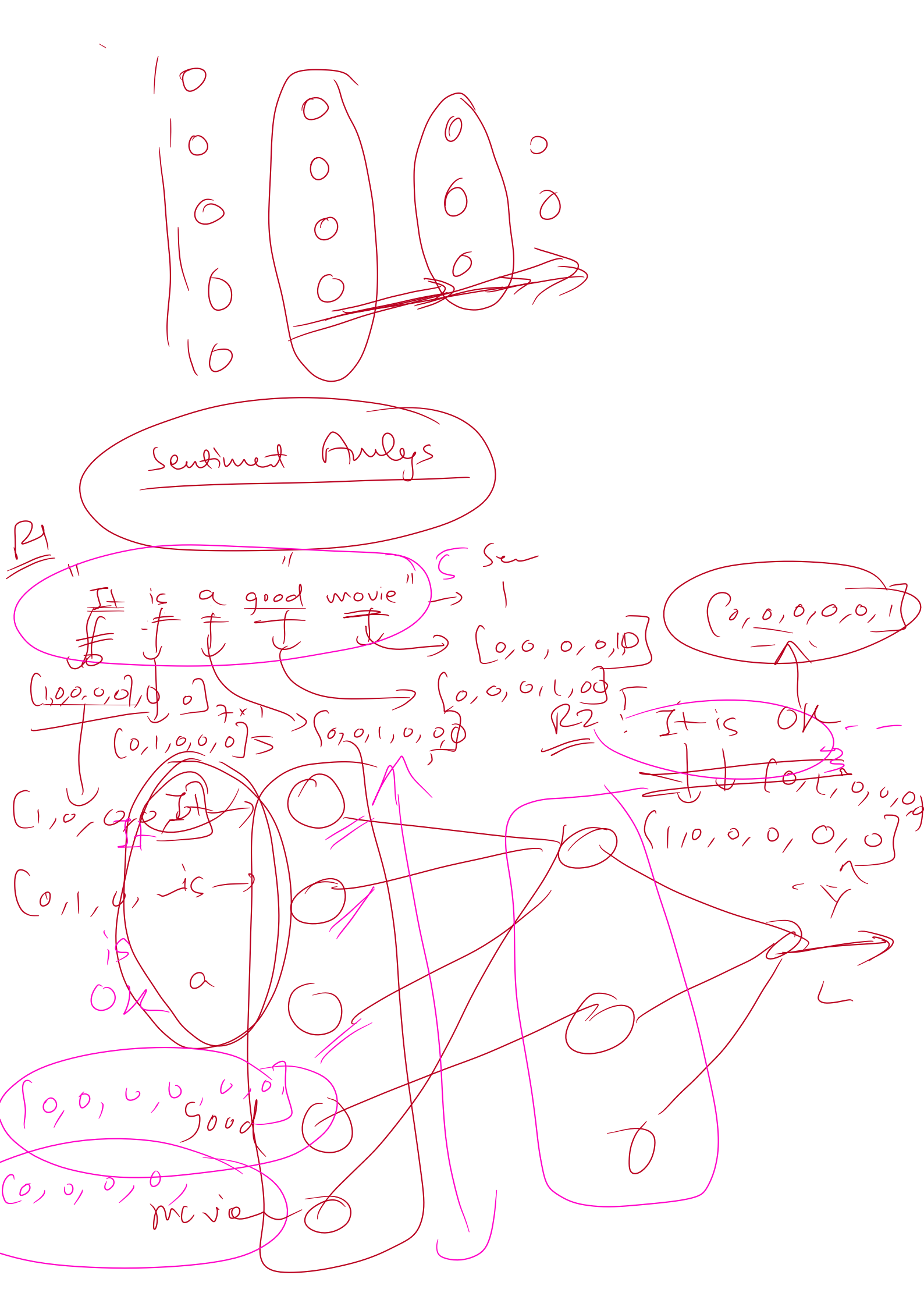
$o = (w_1 x_1 + w_2 x_2 + w_3 x_3)$

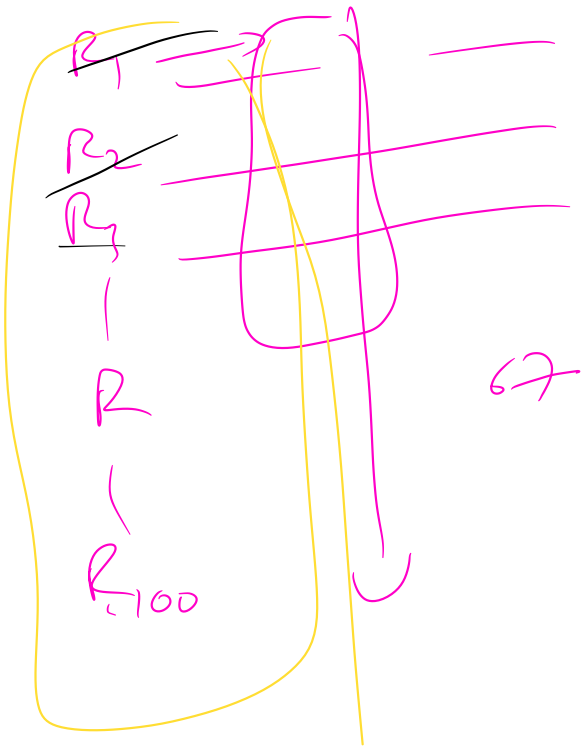


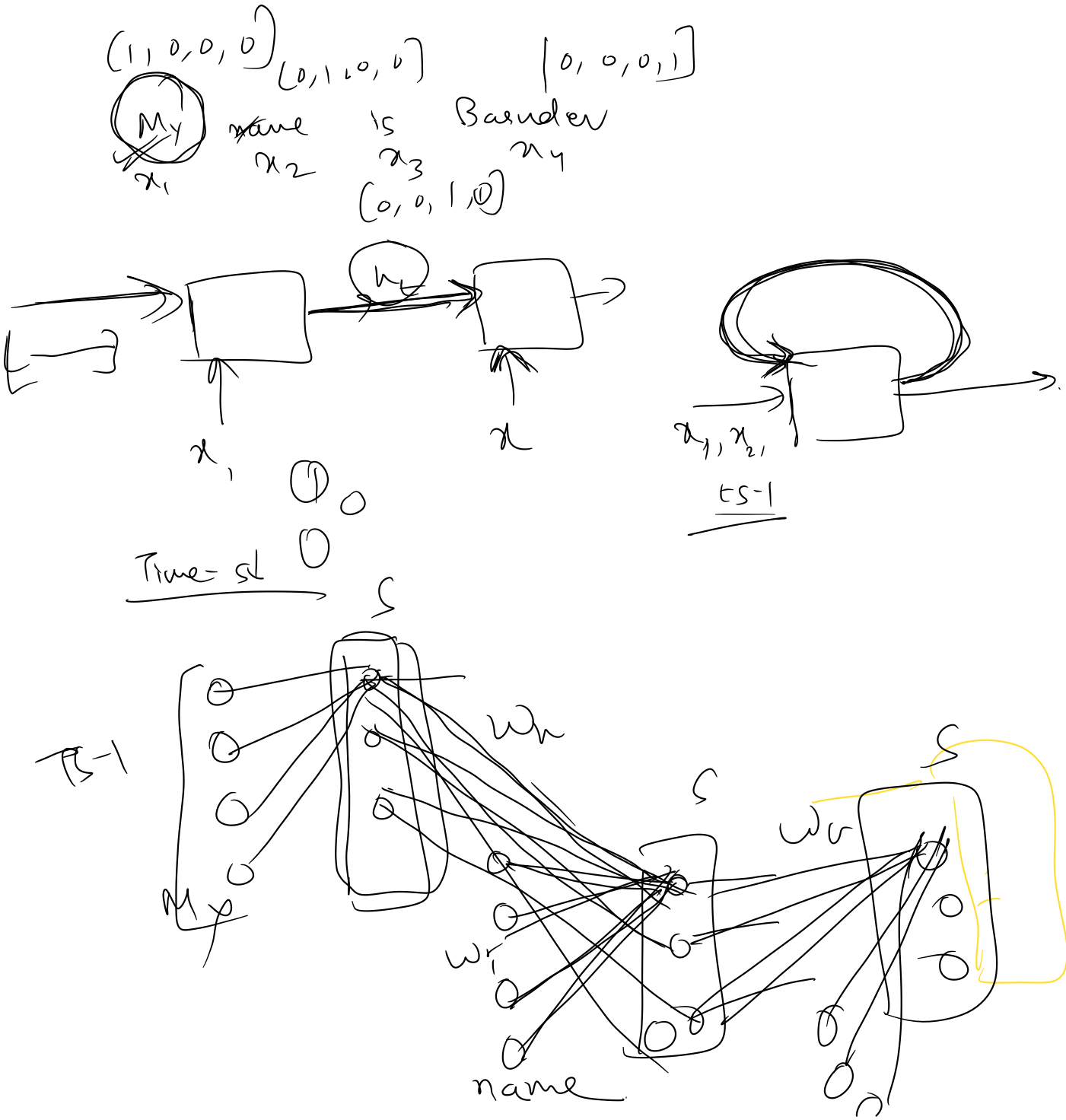
$\hat{y} = w_4 f(o)$

$\hat{y} = w_4 (w_1 x_1 + w_2 x_2 + w_3 x_3)$

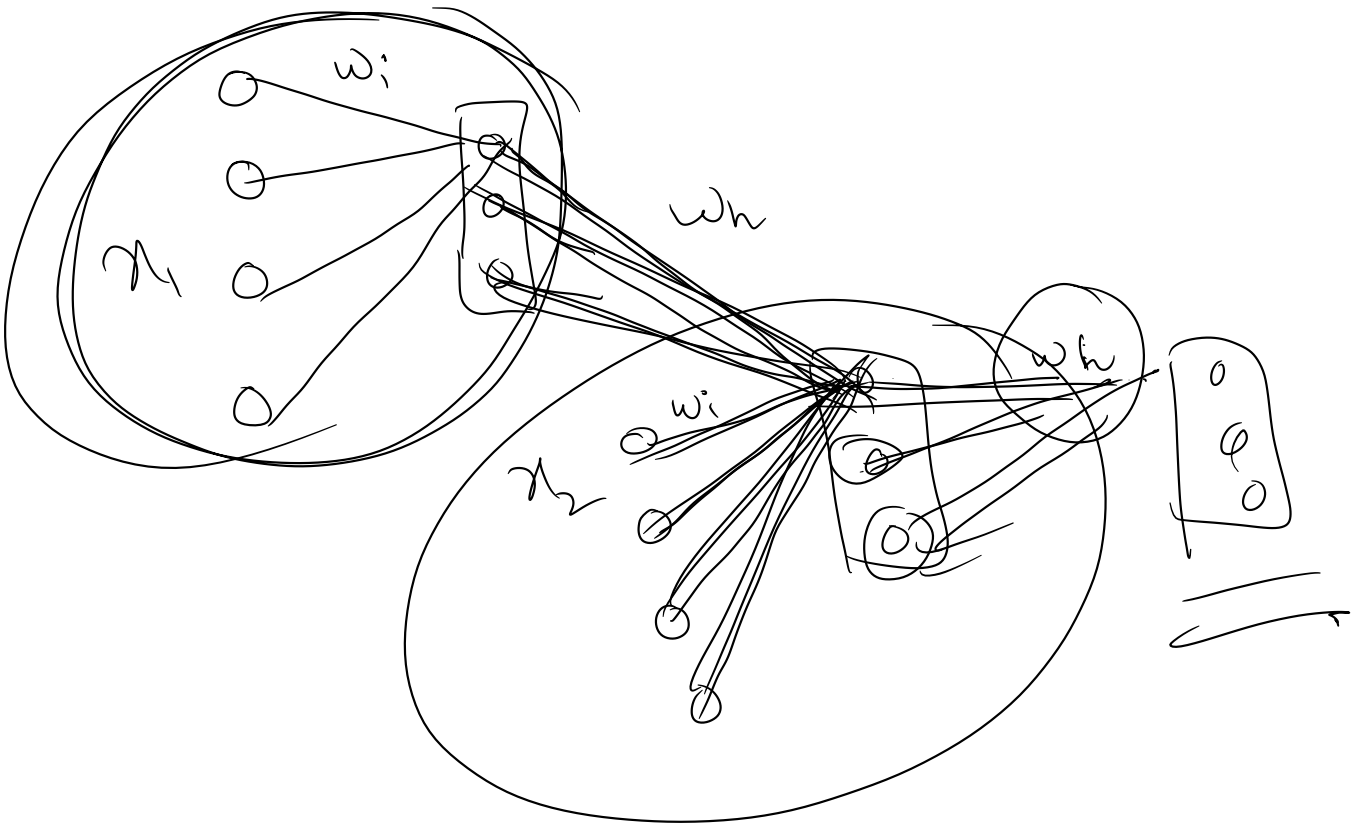

Epochs
1 Epoch
Batch
400
Gradient Descent
Epoch = 5
Stochastic (SGD)
1 row
Loss
400 × 5
= 2000
Mini batch
400
100
4 batches
100
4 weights
5 epoc → 20 weights

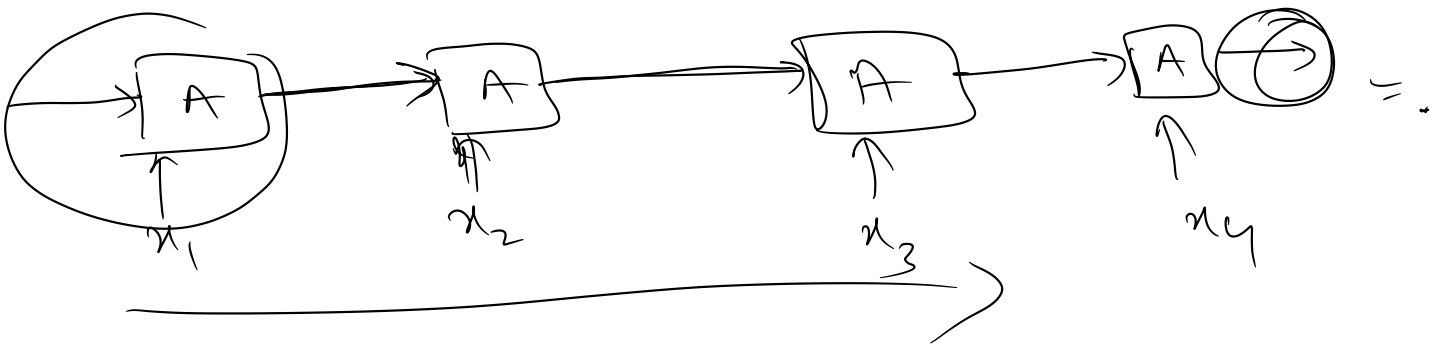

Sentiment Analysis
R1
"It is a good movie"
Sen
1
R2
It is OK
Good
movie
OK
Y


R1
R2
R3
R
R100
67


(1,0,0,0)
(0,1,0,0]
|0,0,0,1]
My
name
is
Bander
x1
x2
x3
x4
(0,0,1,0)
h
x1
x
x1, x2,
ts-1
Time= st
ts-1
My
name
wh
wr
wr

My name is
— T.S





Sequential Neural Networks

Why to avoid ANNs for sequential data??

For Sentiment Analysis, let's say the dataset has Sentences like

S_1: Basudev is studies at IITK } We assign vectors to all words.

S_2: " " " "

S_3: " " " "

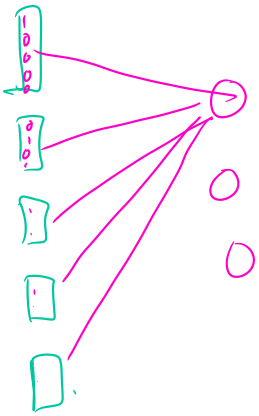
Basudev: [1 0 0 0 0]

is: [0 1 0 0 0]

studies: [0 0 1 0 0]

at: [0 0 0 1 0]

IITK: [0 0 0 0 1]



But now let's say, the next input sentence has 10 words.

The input sizes would differ

So, problem in taking inputs

We might do zero padding

But it would make it computationally expensive & sparsity in data increases

Why zero padding becomes an issue??

→ Dense layers expect dense inputs.

→ Sparsity of data ↑ : reduces effect information per sample.

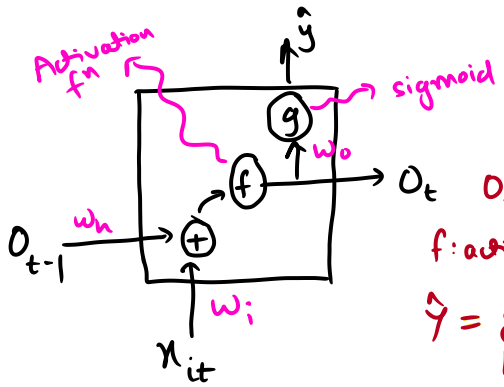
→ Zeros are treated as real data, not padding

→ Many weights never receive useful gradients.

(The corresponding weights barely get updated)

Recurrent Neural Networks

Architecture:



$O_t = f(O_{t-1} w_h + x_{it} w_i)$

f: activation fⁿ like tanh

$\hat{y} = g(O_t w_o)$

↳ Sigmoid fⁿ

After all time steps are complete

$L = -y_i \log \hat{y}_i - (1-y_i) \log (1-\hat{y}_i)$

" Recurrent" term is used because the hidden layers appear multiple times.

The RNN "unfolds" through time.

" Back propagation through time" happens here: the algo is "unrolled" across time steps, allowing gradients (errors) to flow backward not just through layers but also through the sequence's history

Why can't RNNs do machine translation?

An RNN can't produce an output, if we don't feed it an input at every time step.

Let's say we are doing English to Hindi
Both the languages have different sentence structures.

So, we can't start translating right away at every time step.

We have to capture the meaning first.

So, we have to force our RNN to generate <PAD> or some dummy token till we feed the entire sentence into it.

So, the model doesn't have actual values to BPTT.

To force our RNN;

After final <EOS> of English should output <SOS> of French.

The model has to switch from Eng to French and we have to give the French sentence as input at every time step word by word.

Major Issues with RNNs

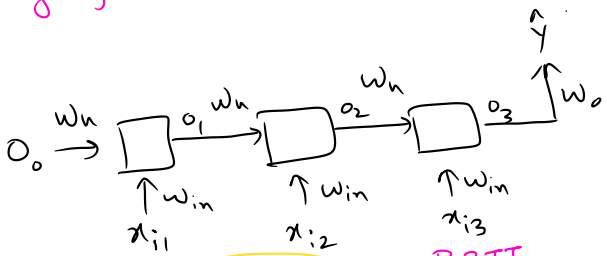
⇝ Problem of long term dependencies.

⇝ stagnated training.

Long term dependencies arise due to vanishing gradient problem.

↓

exploding gradients.



(Same RNN block produced multiple times for visualization)

BPTT

$$\frac{\partial L}{\partial w_{in}} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial o_3} \cdot \frac{\partial o_3}{\partial w_{in}} + \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial o_3} \cdot \frac{\partial o_3}{\partial o_2} \cdot \frac{\partial o_2}{\partial w_{in}} + \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial o_3} \cdot \frac{\partial o_3}{\partial o_2} \cdot \frac{\partial o_2}{\partial o_1} \cdot \frac{\partial o_1}{\partial w_{in}}$$

Short term dependency term

long term dependency

If we use activation fn like tanh or sigmoid

↓

vanishing gradient → weights may remain near their initialization

Soln: Relu or leaky Relu
Better weight init

If activation fn like relu or leaky relu

↓

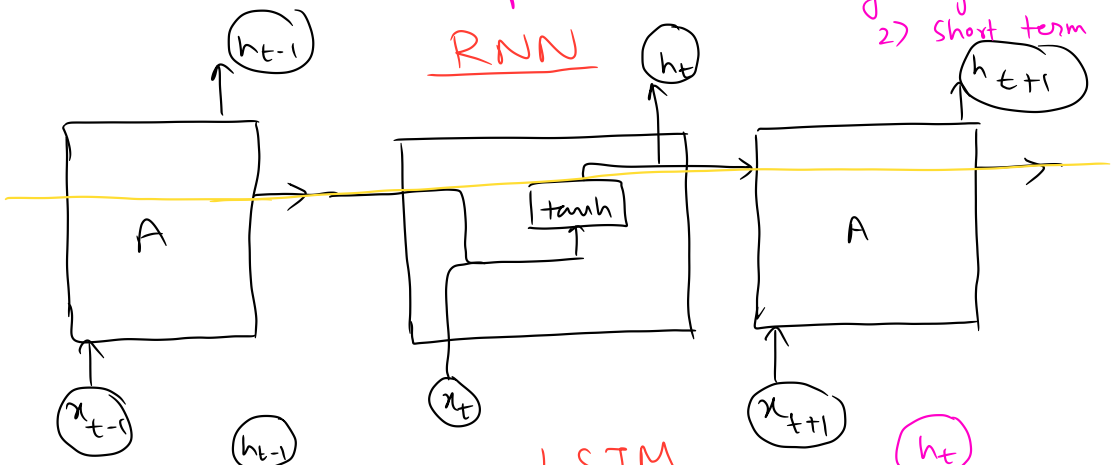
exploding gradient → chaotic gradient descent, weights become very large

Soln: Gradient Clipping (we set the max^m gradient)
Controlled Learning Rate

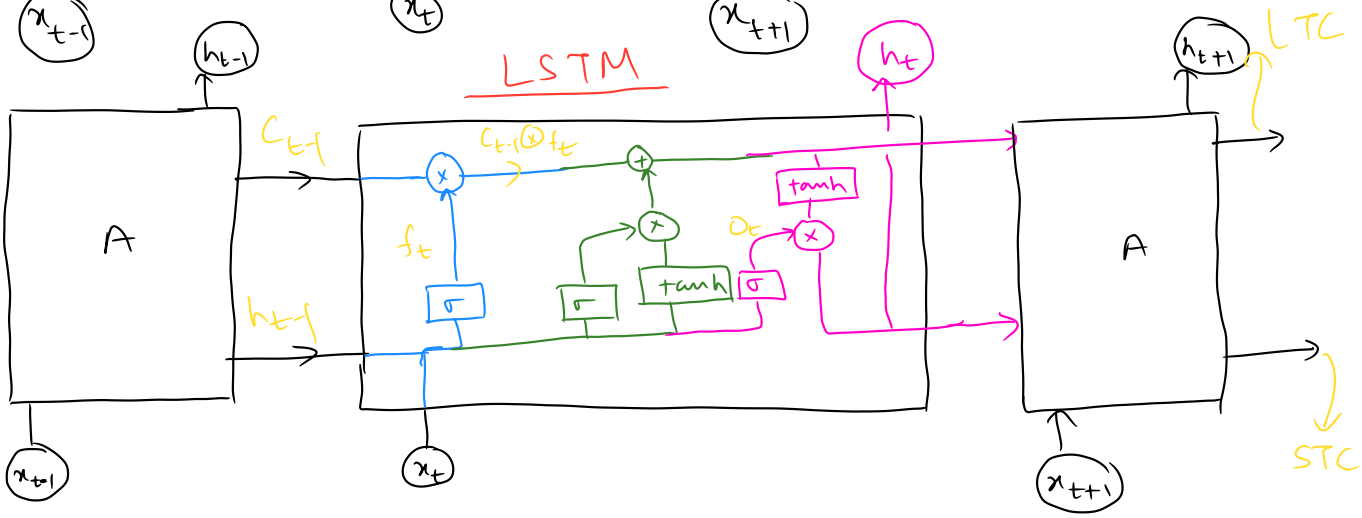
Long Short-Term Memory (LSTM)

RNN maintains memory only by the hidden state line.

It has the pressure of maintaining 1) long term context
2) Short term context



Responsible for maintaining memory.



forget gate
What to remove from LTM based on STC & current input

input gate
what to add to LTM

output gate
based on LTM & current input gives output also creates STM

LTC : C_{t-1} Cell State
STC : h_{t-1} hidden State
Input : x_t

} Total inputs

given C_t & h_t

C_t → updated
h_t → created

Terms

x_t → one word converted into vector

c_{t-1}, h_{t-1} → vectors with same dimension.

(Apart from x_t, all have same dimensions)

f_t → forget gate

i_t → input gate

$\bar{C}_t$ → candidate cell state

O_t → output gate

⊗ : point wise multiplication

⊕ : point wise addition

tanh : tanh appl^n

$[1\ 2\ 3] \otimes [3\ 4\ 5] = [3\ 8\ 15]$

$[1\ 2\ 3] \otimes [3\ 4\ 5] = [4\ 6\ 8]$

$[\tanh(1), \tanh(2), \tanh(3)]$

σ : Neural Network with sigmoid activation f^n

tanh : Neural N with tanh activation f^n

} Number of nodes same throughout Lstm

Number of nodes : hyperparameter

Number of nodes in hidden layers = dimensions of $C_{t-1}, h_{t-1}, f_t, \bar{C}_t, i_t, O_t$

Forget Gate

Decided by x_t & h_{t-1}

Let's say our x_t has dim = 4

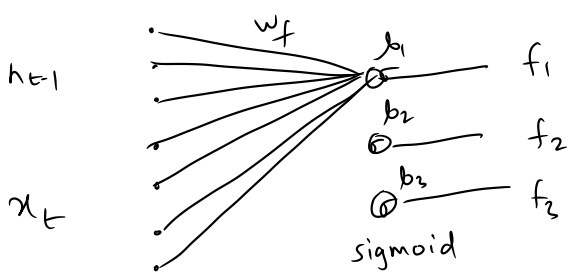
Let's say No. of nodes' dim = 3

So, dim $(h_{t-1}) = 3$

$$f_t = \sigma\left(W_f [h_{t-1}, x_t] + b_f\right)$$

3×7 $\quad 7\times1$ $\quad 3\times1$

3×1



$f_t = [f_1\ \ f_2\ \ f_3]$

↳ forget gate.

Input Gate

1) Calc Candidate Cell State $\tilde{C}_t$

2) Calc i_t
   ↳ decides which info of $\tilde{C}_t$ should be added to C_t

3) C_t

$$\tilde{C}_t = \tanh\left(w_c [h_{t-1}, x_t] + b_c\right)$$

$$i_t = \sigma\left(w_i [h_{t-1}, x_t] + b_i\right)$$

$\tilde{C}_t \otimes i_t = \tilde{C}_t^*$ (filtered candidate state)

This solves Vanishing Gradient Problem

If LSTM feels the initial information is imp; then it would set

$f_t = [1\ 1\ 1]$ Example dim = 3

$i_t = [0\ 0\ 0]$ - Example

So, $C_{t-1} = C_t$

$$C_t = f_t \otimes C_{t-1} \oplus i_t \otimes \tilde{C}_t \rightarrow \text{Cell State}$$

Output Gate

$$h_t = O_t \otimes \tanh(C_t)$$

tanh keeps a safe range of $[-1, 1]$

→ tanh is used because it keeps the hidden activations bounded
→ stabilizes training by preventing exploding gradients
→ allows gates to control information flow
→ exposes only a safe, filtered version of internal memory

Gated Recurrent Units

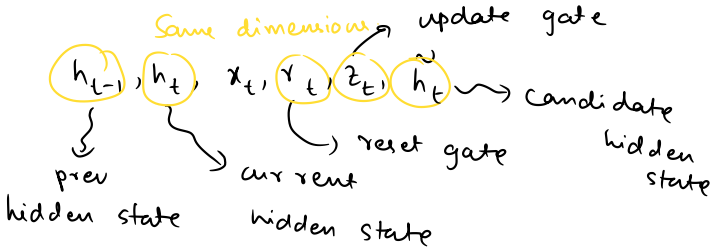
LSTM: Complex architecture, more number of parameters
↳ 3 gates (Training Time more)

GRU: simpler architecture, lesser parameters
2 gates So, reduces training time

LSTM: STC & LTC is maintained separately

GRU: Only one hidden state
↳ Reset Gate & Update Gate

We give h_{t-1} & x_t as input

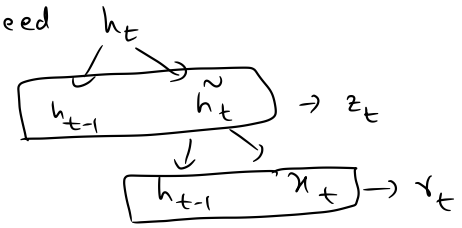


σ, tanh have same number of nodes = dim $(h_t, r_t \dots)$

Steps
1) Calc r_t
2) $\tilde{h}_t$
3) z_t
4) h_t

take
We text, we use text vectorization techniques

We need h_t



from prev hidden state (h_{t-1}) & current input (x_t) we form candidate memory $(\tilde{h}_t)$

$\tilde{h}_t$ is inclined towards to current input

h_{t-1} $\tilde{h}_t$ (Balanced) $\rightarrow h_t$

$h_{t-1} \xrightarrow{r_t} \tilde{h}_t \xrightarrow{z_t} h_t$

Overall flowchart

reset gate
$r_t = \sigma\left(W_r [h_{t-1}, x_t] + b_r\right)$

$\tilde{h}_t = \tanh\left(W_c [h_{t-1} \otimes r_t, x_t] + b_c\right)$

$h_t = (1 - z_t) \otimes h_{t-1} \oplus z_t \otimes \tilde{h}_t$

z_t is high it favours candidate hidden state
z_t is low: favours prev hidden state

Encoder - Decoder

For NLP tasks like machine translation

where input & output have varying lengths.

Let's say English to Hindi

OHE for vectorization

English: I want to really go back home (7 tokens)

Hindi: मुझे घर जाना है <start> <end> (6 tokens)